Bauwerksmodellierung im kooperativen Planungsprozess: Mit der Objektorientierung zur Verarbeitungsorientierung

Dissertation zur Erlangung des akademischen Grades

Doktor - Ingenieur

an der Fakultät Bauingenieurwesen
der
Bauhaus-Universität Weimar

vorgelegt von

Christian Koch
geboren am 1. Mai 1979
in Anklam

Gutachter: 1. Prof. Dr.-Ing. Berthold Firmenich, Bauhaus-Universität Weimar

2. Prof. Dr.-Ing. Uwe Rüppel, Technische Universität Darmstadt

3. Prof. Dr.-Ing. habil. Reinhard Hübler, Bauhaus-Universität Weimar

Eingereicht am: 2. Juli 2008

Tag der Disputation: 9. Dezember 2008

Vorwort

Die vorliegende Arbeit entstand in der Zeit von August 2004 bis Dezember 2008 während meiner Tätigkeit als wissenschaftlicher Mitarbeiter an der Juniorprofessur CAD in der Bauinformatik an der Bauhaus-Universität Weimar. Im Arbeitsumfeld der Juniorprofessur und der Professur Informatik im Bauwesen bearbeitete ich das Thema im Rahmen der von der Deutschen Forschungsgemeinschaft (DFG) geförderten Projekte: Einzelprojekt „Eine formale Sprache für operative CAD-Modelle“ und anschließendes Transferprojekt „Operatives Modellieren im Bauwesen“.

Ich möchte mich an dieser Stelle bei all jenen Personen bedanken, die zum Gelingen dieser Arbeit beigetragen haben.

Mein besonderer Dank gilt meinem Mentor Prof. Berthold Firmenich. Er hat mich von der Idee bis hin zur fertigen Arbeit begleitet, mir die nötigen Freiräume geschaffen und mir mit seinen wissenschaftlichen und auch persönlichen Ratschlägen zur Seite gestanden.

Bei Prof. Uwe Rüppel möchte ich mich für die externe Begutachtung dieser Arbeit bedanken. Weiterhin danke ich Prof. Reinhard Hübler für die Erstellung seines Gutachtens sowie für die offenen und zielführenden Diskussionen zum Thema. Prof. Karl Beucke bin ich für seine fachlichen und auch organisatorischen Hinweise dankbar.

Darüber hinaus bedanke ich mich bei allen Kollegen am Lehrstuhl für die freundschaftliche Arbeitsatmosphäre. Besonders dankbar bin ich Jens-Uwe Wagner für seine aufbauenden und vertrauensvollen Worte der Zuversicht. Katrin Wender danke ich für das Korrekturlesen und für manchen wertvollen Hinweis. Weiterhin möchte ich mich bei Frau Dr. Heidemarie Schirmer vom Verlag der Bauhaus-Universität für die engagierte Zusammenarbeit bei der Veröffentlichung dieser Arbeit in der neuen Schriftenreihe „Informatik in Architektur und Bauwesen“ bedanken.

Nicht zuletzt danke ich meiner Familie und meinen Freunden für die persönliche Unterstützung während der vergangenen Jahre. Mein besonders herzlicher Dank gilt Julia Wolf. Sie ist immer für mich dagewesen, hat mir zugehört und es geschafft, mich in unserer Freizeit abzulenken.

Weimar, im Februar 2009
Christian Koch

„Zweifeln ist der Anfang der Wissenschaft;
wer an nichts zweifelt, prüft nichts,
wer nichts prüft, entdeckt nichts,
wer nichts entdeckt, ist blind
und muß blind bleiben.[1] *“*

[1] Johann Christian Wiegleb (1732–1800), deutscher Apotheker und Chemiker, aus [Wiegleb 1777]

Kurzfassung

Im rechnergestützten Bauplanungsprozess arbeiten verschiedene Fachplaner an der gemeinsamen Aufgabe, ein Bauwerk zu planen, zusammen. Verfügbare Kooperationsansätze beschäftigen sich mit versionierten und verteilten Bauwerksmodellen, die auf Basis der Objektorientierung virtuelle Bauwerkszustände beschreiben. Die in diesen zustandsorientierten Modellen unberücksichtigten Zustandsänderungen führen zu derzeitigen Problemen beim Austausch, beim Vergleich und bei der Zusammenführung von versionierten Bauwerksinformationen.

Gegenstand der vorliegenden Arbeit ist die Entwicklung eines verarbeitungsorientierten Ansatzes zur ganzheitlichen Betrachtung der Bauwerksmodellierung. Neben der zustandsorientierten Beschreibung eines virtuellen Bauwerks werden zusätzlich änderungsorientierte Informationen in Form von Modellieroperationen in der Modellbildung berücksichtigt. Es wird eine Modellierungssprache definiert, um Operationen formal zu beschreiben. Modellieroperationen bilden eine Verarbeitungsschnittstelle für Objektmodelle, repräsentieren Entwurfsabsichten, reichern bestehende Bauwerksmodelle mit Änderungssemantik an und tragen zur Konsistenzsicherung in diesen Modellen bei. Neuartige Kooperationskonzepte für den Austausch, den Vergleich und das Zusammenführen von versionierten Bauwerksinformationen werden auf Grundlage des vorgeschlagenen Ansatzes entwickelt.

Sowohl das Modell als auch die Sprache werden unabhängig von aktuellen Technologien formal beschrieben. Die prinzipielle Anwendbarkeit des vorgeschlagenen Ansatzes wird im Rahmen einer Pilotimplementierung auf Basis eines Open-Source-Systems im Bauwesen nachgewiesen.

Abstract

Several actors involved in the computer-supported building planning process work together towards a common goal — the design of a building. Available cooperation approaches focus on versioned and distributed building models which describe virtual building states on the basis of the object-oriented method. State changes remain unconsidered and lead to known problems when exchanging, comparing and merging versioned building information.

The work presented deals with the development of a processing-oriented approach for the integral consideration of building modeling. In addition to the state-oriented description of a virtual building, change-oriented information is provided by means of model operations. A new modeling language is defined for the formal description of operations. Model operations establish a processing interface for object models, represent design intents, enhance existing building models with change semantics and add to the consistency of these models. New enhanced concepts for cooperation are defined on the basis of the approach presented. These concepts provide functionality for cooperation when exchanging, comparing and merging versioned building information.

Both the model and the language are formally described in order to be independent of current technologies. The applicability of the approach proposed is verified in principle by a pilot implementation based on an Open Source engineering system.

Inhalt

Verzeichnis der Bilder

Verzeichnis der Tabellen

1 Einleitung

„Im Bauwesen reicht das klassische Planungsmaterial von textuellen Beschreibungen und zeichnerischen Darstellungen bis hin zu räumlichen, maßstäblichen Modellen aus geeigneten Materialien. Bei der rechnergestützten Planung wird das Planungsmaterial durch Produktmodelle repräsentiert. [...] Jedes Modell ist immer nur ein mehr oder weniger genaues Abbild eines Ausschnitts der Wirklichkeit. Es wird als Grundlage für die in der jeweiligen Domäne zu treffenden Planungsentscheidungen gewählt. So ist das Modell eines Bauwerks für einen Architekten ganz anders geartet als für den Tragwerksplaner oder den Ingenieur für die Technische Gebäudeausrüstung. Die Integration aller Informationen zur Planung, Erstellung, Verwaltung und zum Rückbau eines Gebäudes in einem einzigen Modell erscheint zwar theoretisch denkbar, praktisch aber kaum sinnvoll zu sein. Vielmehr werden objektorientierte Partialproduktmodelle entwickelt, die die jeweilige Sicht des Fachplaners abbilden können.“

aus [Firmenich u. Rank 2007, S. 121]

„Projects in civil and building engineering generally require a close cooperation between separate engineering teams from various disciplines towards a common goal. Engineering cooperation is based upon communication processes that need to support the specific needs of engineering projects. Currently, by far most projects are still relying for these purposes on a set of Technical Documents that is exchanged between engineering teams. [...] Digital technologies have not changed this approach fundamentally yet. Most often application software is used to produce the same documents as before, digital exchange formats are used to speed up the process of information interchange. This optimizes individual steps in the process but does not change the fundamental approach with all its inherent problems.“

aus [Beucke 2006, S. 74 ff.]

1.1 Problemstellung

Bauplanungsprozess: Im Bauplanungsprozess arbeiten verschiedene Fachplaner an einer gemeinsamen Aufgabe – der Planung eines Bauwerks – zusammen. Verfügbare Kommunikations- und Softwaretechnologien unterstützen diesen Prozess heutzutage vorwiegend durch digitale Medien. So wird beispielsweise zur Architekturplanung, zur Tragwerksplanung, für die TGA[1]-Planung und auch zur Bauablaufplanung eine Vielzahl von Fachsoftware eingesetzt, um Teile eines virtuellen Bauwerks im Rechner zu bearbeiten. Die erzeugten Bauwerksinformationen werden mit dem Ziel der Kooperation zwischen den einzelnen Planungsbeteiligten ausgetauscht.

Rechnergestützte Kooperation: Die rechnergestützte Kooperation wird in [Rüppel 2007a] in vier Ebenen gegliedert. In der Kommunikationsschicht werden Planungsdaten über moderne Rechnernetzwerke[2] zwischen den Fachplanern übertragen. Den Fachplanern sind in der Organisationsschicht Rollen zugeordnet, die jeweilige Hoheiten, Rechte und Fähigkeiten definieren. Die Koordinationsschicht beschreibt die Ablaufsteuerung der Planung auf der Basis von Prozessmodellen, welche Planungsaktivitäten und Planungszustände abbilden. Die Ressourcenschicht fasst Bauwerksmodelle und die in Fachapplikationen umgesetzten Verarbeitungsmethoden zusammen (s. Bild 1.1).

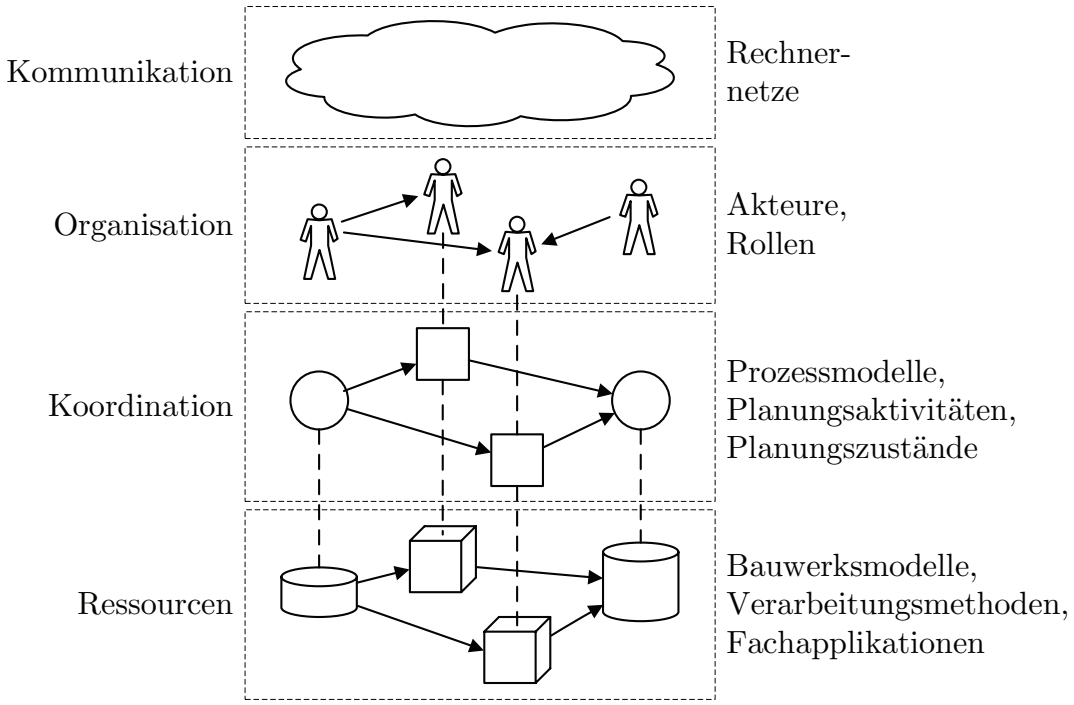


Bild 1.1: Integrative Kooperation im Bauplanungsprozess nach [Rüppel 2007a]

Kooperationskonzepte: Die Komplexität der spezialisierten Planungsaufgaben hat zu einer großen Anzahl verschiedener Datenmodelle und Fachapplikationen im Bauwe-

[1] Technische Gebäudeausrüstung
[2] z. B. das Internet

sen geführt. Traditionelle Kooperationsansätze beschäftigen sich mit deren Integration durch Standardisierung von Datenstrukturen und widmen sich darauf basierenden Konzepten bei der Versionierung, beim Austausch und der Archivierung sowie beim Vergleich und bei der Zusammenführung von Bauwerksinformationen. Den Stand der Forschung bilden verteilte Bauwerksmodelle, die auf Basis der Objektorientierung den virtuellen Bauwerkszustand beschreiben und versionieren. In diesen zustandsorientierten Modellen bleibt die Verarbeitungssemantik – änderungsorientierte Informationen, die den Planungsvorgang charakterisieren – unberücksichtigt. Dieser Umstand spiegelt sich in derzeitigen Problemen beim Austausch, beim Vergleich und bei der Zusammenführung von versionierten Bauwerksinformationen wider. Neben kumulierenden Informationsverlusten beim Austausch können Entwurfsabsichten[3] nicht abgebildet und infolgedessen auch nicht beim Vergleich und bei der Zusammenführung von Planungsinformationen herangezogen werden. Eine ausreichende Kooperationsunterstützung ist auf Grundlage von versionierten Bauwerkzuständen allein somit nicht gegeben. Hier bietet der in der vorliegenden Arbeit verfolgte Ansatz neue Möglichkeiten, diese Probleme zu lösen.

1.2 Einführende Beispiele

Im Folgenden werden zwei einfache Beispiele vorgestellt, um die Unterschiede und Vorteile des in dieser Arbeit verfolgten Ansatzes gegenüber herkömmlicher Modellbildung zu veranschaulichen.

1.2.1 Schachpartie

Modellbildung: Das Modell des Schachspiels beinhaltet das Schachbrett mit 64 Feldern, die 16 weißen und 16 schwarzen Schachfiguren sowie die Schachregeln, welche die Spielzüge und das Schlagen der Figuren festlegen. Um den Spielstand zu beschreiben, werden die Positionen einzelner Schachfiguren auf dem Brett angegeben.

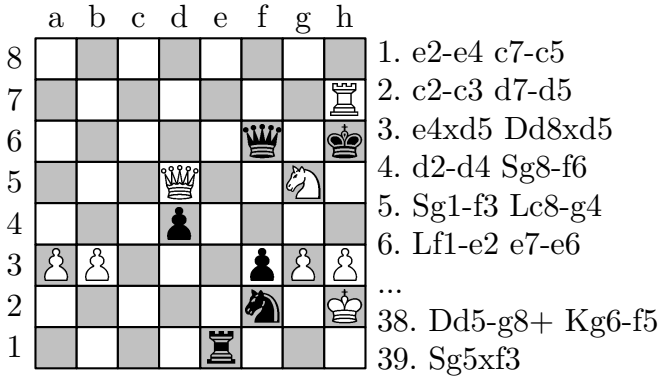


Bild 1.2: Schlussstellung und Spielzüge einer Schachpartie[4]

[3] engl. design intents

Spielstrategie: Am Ende einer Schachpartie stehen noch einzelne Figuren auf dem Schachbrett. Das Ergebnis des Spiels ist als Zustand sichtbar, wie es dazu kam jedoch nicht. Schachinteressierten genügt es nicht, das Ergebnis einer Schachpartie als Schlussstellung auf dem Brett zu sehen. Sie wollen jeden einzelnen Spielzug nachvollziehen und die Spielstrategien erkennen. Deshalb wird, wie im Bild 1.2 auf Seite 3 dargestellt, beispielsweise in Zeitungen zusätzlich zum Ergebnis die Sequenz der einzelnen Züge von Spielbeginn an in einer bestimmten Notation bzw. Sprache abgedruckt. Es wird also nicht nur das Spielergebnis als Zustand (Position der Figuren), sondern zusätzlich auch die Spielhistorie als Änderungsvorgang (Züge der Figuren) dargestellt.

1.2.2 Volumenmodellierung

Modellbildung: In der Volumenmodellierung wird unter anderem zwischen zwei Modellierungsansätzen unterschieden. Während ein B-Rep[5]-Modell die Geometrie und Topologie der Volumenoberfläche beschreibt, wird das Volumen in einem CSG[6]-Modell als Konstruktionsbaum mit boolschen Operationen auf primitive Grundkörper abgebildet. Um das Ergebnis eines Volumenentwurfs darzustellen, wird die Oberflächenbescheibung an Visualisierungssoftware weitergegeben.

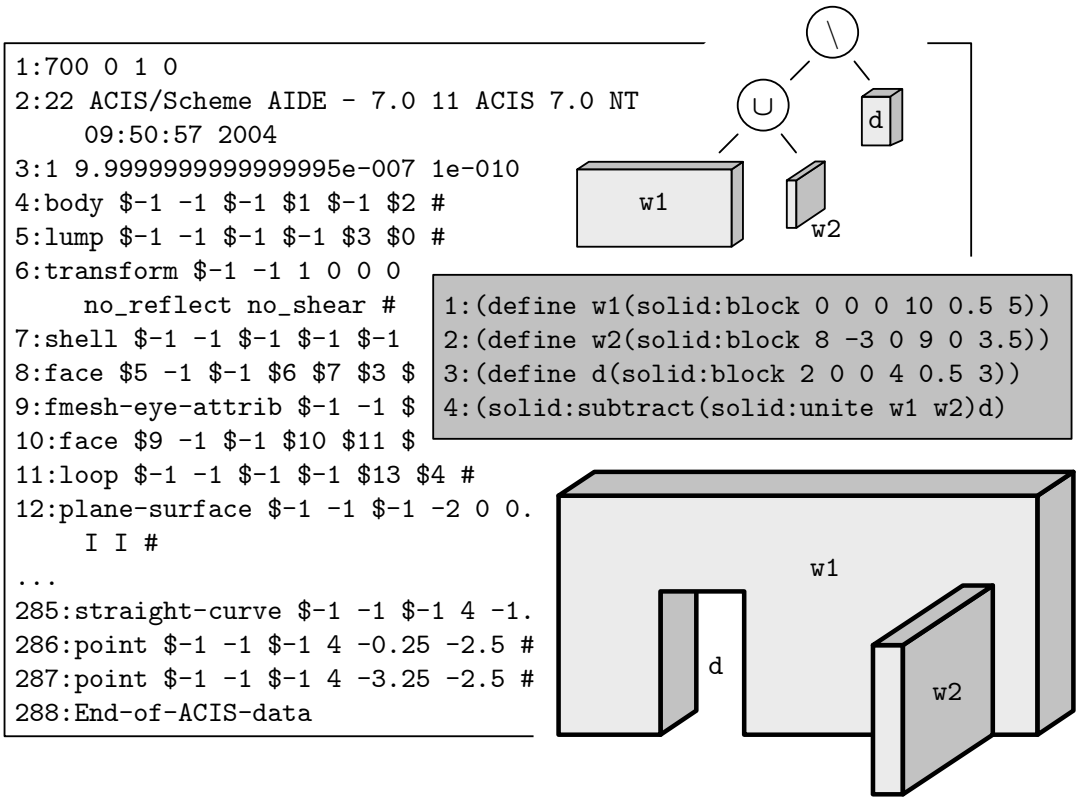


Bild 1.3: Repräsentationen eines einfachen Volumenmodells nach [Firmenich 2004]

[4]Schlussstellung der Partie: Computerprogramm Deep Blue vs. Kasparow, Spiel 1, Philadelphia 1996
[5]Boundary Representation
[6]Constructive Solid Geometry

Interpretierbarkeit und Aussagekraft: Die CSG-Volumenmodellierung entspricht im Hinblick auf den Konstruktionsvorgang der Arbeitsweise von Fachplanern, um Bauwerksgeometrien dreidimensional zu entwerfen. Das Gesamtmodell wird beispielsweise, wie im Bild 1.3 veranschaulicht, durch Vereinigung von Wandvolumina und Subtraktion des Öffnungsvolumens konstruiert. Der komplexen und für den Fachplaner unverständlichen B-Rep-Beschreibung des resultierenden Geometriemodells (288 Zeilen im SAT-Format von ACIS[7], Bildhindergrund) steht die kompakte, verständliche und aussagekräftige CSG-Darstellung in Form von Modellieroperationen (4 Zeilen in der Scheme-Sprache von ACIS, Bildvordergrund) gegenüber.

1.2.3 Diskussion

Analogie: Bereits an diesen einfachen Beispielen wird der Vorteil einer ganzheitlichen Betrachtungsweise deutlich. Neben der Abbildung des Ergebnisses als Zustand wird in der Modellbildung zusätzlich die Entstehungsgeschichte als Änderungsvorgang berücksichtigt. Übertragen auf die Modellierung von Bauwerken im Planungsprozess bedeutet dies:

- Zusätzliche, änderungsorientierte Informationen über die Entwurfsgeschichte und die verfügbaren zustandsorientierten Bauwerksinformationen bilden eine ganzheitliche Entwurfsbetrachtung.
- Modellieroperationen ermöglichen die Beschreibung der Verarbeitungs- bzw. Änderungssemantik und erlauben so die Speicherung von Entwurfsintentionen.
- Die Verarbeitung von zustandsorientierten Bauwerksmodellen durch den Fachplaner basiert in der Regel auf Kommandos. Das bedeutet, das Speichern änderungsorientierter Informationen bedarf keines zusätzlichen Bearbeitungsaufwandes.
- Eine Sprache zur Formulierung von Modellieroperationen dient einer kompakten, aussagekräftigen und vom Fachplaner interpretierbaren Darstellung von Änderungsinformationen.

1.3 Abgrenzung und Zielsetzung

Kooperationsebene: Im Rahmen dieser Arbeit wird die Kooperationsunterstützung innerhalb der Ressourcenschicht (vgl. Bild 1.1 auf Seite 2) betrachtet. Es wird untersucht, wie Planungsmethoden, Bauwerksmodelle und Fachapplikationen erweitert werden müssen, um den vorgeschlagenen Modellierungsansatz im vernetzt-kooperativen Bauplanungsprozess anzuwenden.

Fachdomäne: Innerhalb der Bauwerksplanung werden Planungsdomänen wie beispielsweise die Architekturplanung, die Tragwerksplanung, die TGA-Planung und die Bauablaufplanung unterschieden. Zur direkten Umsetzung in Fachapplikationen sind

[7] s. [Corney u. Lim 2001]

Domänenmodelle zu komplex, als dass sie umfassend verarbeitet werden können. Beispielsweise ist keine Software bekannt, die sämtliche fachlichen Planungsaufgaben wie Berechnung, Bemessung und Dokumentation innerhalb der Domäne Tragwerksplanung abdeckt. Aus diesem Grund erfolgt eine weitere Unterteilung einer Planungsdomäne in verschiedene Fachdomänen. Der Domäne Tragwerksplanung sind beispielsweise die Fachdomänen FEM[8]-Analyse, Tragwerksbemessung und Zeichnungserstellung zugeordnet. Innerhalb der einzelnen Fachdomänen sind verschiedene Fachapplikationen mit unterschiedlichen nativen Datenmodellen verfügbar. Dieser Zusammenhang ist im Bild 1.4 veranschaulicht.

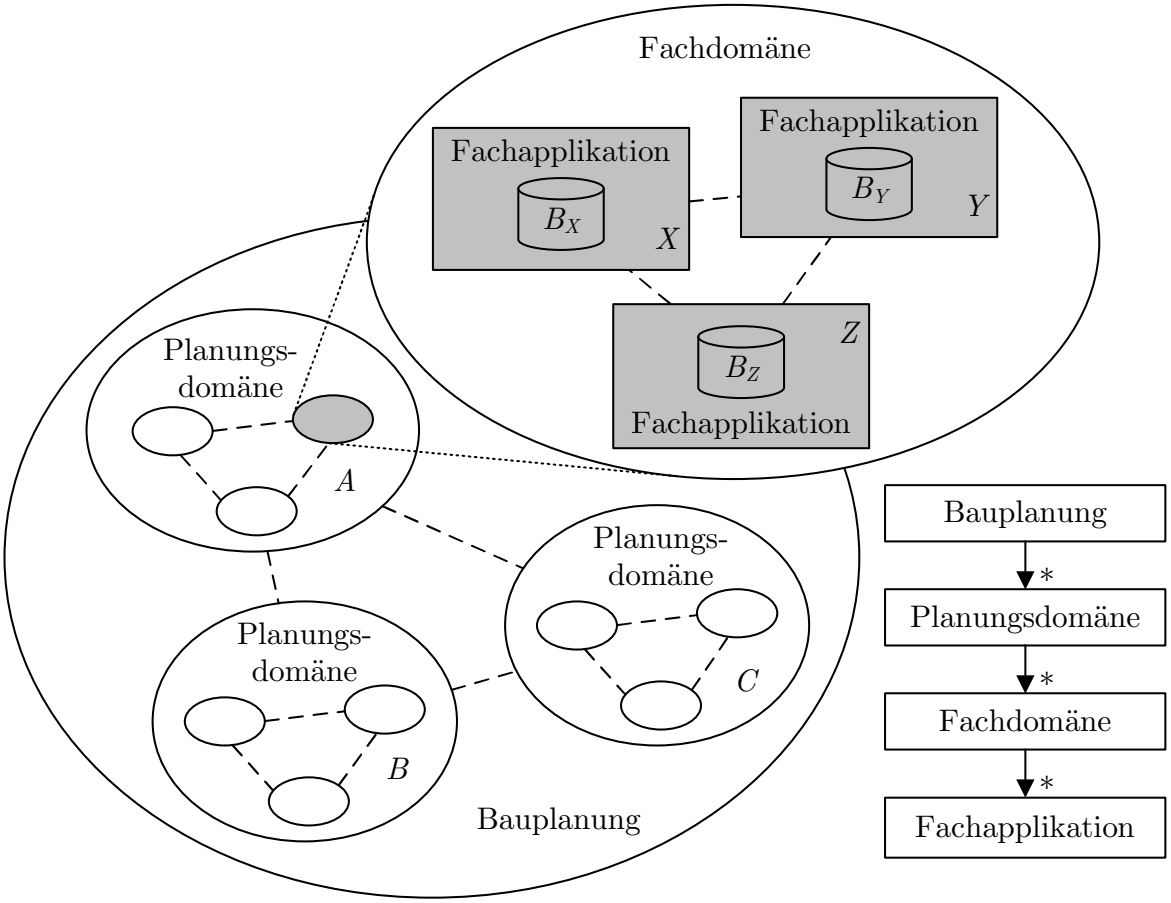


Bild 1.4: Fachdomäne in der Bauplanung

Fachdomäneninterne Kooperation: Diese Arbeit widmet sich nicht einer Kooperationsunterstützung durch die Integration verschiedener Planungs- und Fachdomänen. Vielmehr wird die fachdomäneninterne Zusammenarbeit betrachtet. Es wird untersucht, wie diverse, für eine Planungsaufgabe eingesetzte Fachapplikationen mit ihren unterschiedlich strukturierten Datenmodellen integriert werden können.

Zielsetzung: Die Ziele der vorliegenden Arbeit betreffen die Entwicklung eines Modells und einer Sprache für die verarbeitungsorientierte Modellierung sowie deren Anwen-

[8] Finite Elemente Methode

dungskonzepte bei der fachdomäneninternen Kooperation im rechnergestützten Bauplanungsprozess. Diese können wie folgt zusammengefasst werden:

- Entwicklung eines Modells zur ganzheitlichen, verarbeitungsorientierten Betrachtungsweise der Bauwerksmodellierung. Das verarbeitungsorientierte Modell soll das virtuelle Bauwerk sowohl zustandsorientiert als auch änderungsorientiert beschreiben.
- Entwicklung einer Modellierungssprache zur Beschreibung von Modellieroperationen als Grundlage der standardisierten Verarbeitung von Bauwerksmodellen. Diese Sprache ist im Hinblick auf die Anwendbarkeit sowohl in der Benutzer- und Programmierschnittstelle von Fachapplikationen als auch zur persistenten Speicherung von Änderungen zu konzipieren.
- Berücksichtigung, Wiederverwendung und Erweiterung verfügbarer Methoden, Modelle und Applikationen der derzeitigen rechnergestützten Bauplanung.
- Dauerhafte, formale und von konkreten Umsetzungstechnologien unabhängige Beschreibung des Modells und der Sprache auf der Basis der Mengenlehre und der Relationenalgebra einerseits und einer Grammatik andererseits.
- Entwicklung neuer Kooperationskonzepte zur Unterstützung der Versionierung, des Austauschs, der Archivierung, des Vergleichs und der Zusammenführung von Bauwerksinformationen innerhalb von Fachdomänen auf Grundlage des neuen Modellierungsansatzes.

1.4 Vorgehensweise

Aufbau der Arbeit: Die vorliegende Arbeit ist in die folgenden sieben Kapitel eingeteilt, wobei die Kapitel 3, 4 und 5 den Kern der Arbeit bilden.

Kapitel 2 – Stand der Technik und Forschung: Nachdem die Einleitung den Gegenstand, die Motivation und die Zielsetzung der Arbeit dargelegt hat, beschäftigt sich das zweite Kapitel mit dem aktuellen Stand der Technik und Forschung. Ausgehend von der Definition grundlegender Begriffe werden die objektorientierte Methode zur Bauwerks- und Fachapplikationsmodellierung und die darauf basierenden Konzepte bei der Versionierung, beim Austausch, bei der Archivierung, beim Vergleich und bei der Zusammenführung von Bauwerksinformationen in der vernetzt-kooperativen Bauwerksplanung beleuchtet. Begleitend dazu werden Probleme identifiziert sowie Zusammenhänge und Abgrenzungen zur vorliegenden Arbeit dargelegt.

Kapitel 3 – Modellbildung: Das dritte Kapitel beschreibt den verarbeitungsorientierten Modellierungsansatz. Aufbauend auf den Ansätzen der Objekt-, der Versions- und der Änderungsorientierung wird der vorgeschlagene Ansatz als ganzheitliche Betrachtung entwickelt. Zusätzlich zur Zustandsbeschreibung in Form von Bauwerksversionen werden Zustandsänderungen des virtuellen Bauwerks als Sequenz von Modellieroperationen betrachtet. Mit der Absicht, das vorgeschlagene Modell dauerhaft und

formal zu beschreiben, wird eine mathematische Formulierung auf Basis der Mengenlehre und der Relationenalgebra vorgestellt. Auf Grundlage des entwickelten Modells wird die Bedeutung von Modellieroperationen im Kontext der Bauwerksmodellierung herausgestellt.

Kapitel 4 – Operative Modellierungssprache: Das vierte Kapitel stellt eine Sprache zur Beschreibung von Modellieroperationen in der Verarbeitungsorientierung vor. Nach der Einführung einiger Grundlagen werden ausgehend von den Anforderungen an eine operative Modellierungssprache drei aufeinander aufbauende Sprachebenen unterschieden. Es wird untersucht, wie die Sprache zur Formulierung von Modellierprogrammen, Modellieroperationen und Änderungen angewendet werden kann. Für jede dieser Ebenen wird die Sprachsyntax formal durch eine Grammatik definiert und anhand von Beispielen verdeutlicht. Es wird großer Wert auf die detaillierte Beschreibung der vorgestellten Modellierungssprache gelegt, sodass auch fachfremde Ingenieure die formale Struktur, die Funktionsweise und die Anwendung der Sprache verstehen können.

Kapitel 5 – Anwendungskonzepte: Das fünfte Kapitel widmet sich den Anwendungskonzepten der verarbeitungsorientierten Modellierung und der operativen Modellierungssprache im Bauplanungsprozess. Es werden die Standardisierung operativer Modelle in Fachdomänen untersucht und die Umsetzung operativer Modelle in Fachapplikationen beschrieben. Darüber hinaus werden Betrachtungen zur Anwendung der Sprache in den Benutzer- und Programmierschnittstellen von Fachapplikationen sowie zu neuen Kooperationskonzepten beim Austausch, bei der Archivierung, beim Vergleich und bei der Zusammenführung von versionierten Bauwerksinformationen durchgeführt.

Kapitel 6 – Pilotimplementierung: Im sechsten Kapitel wird die prinzipielle Anwendbarkeit des verarbeitungsorientierten Ansatzes und der Modellierungssprache auf Basis eines Open-Source-Systems nachgewiesen. Es wird sowohl die sprachbasierte als auch die objektorientierte Umsetzung eines operativen Zeichnungsmodells beispielhaft gezeigt. Zur Verifikation des Ansatzes und der Sprache wird die Anwendung des operativen Zeichnungsmodells in der Benutzer- und Programmierschnittstelle des Open-Source-Systems sowie beim Vergleich und bei der Zusammenführung von Zeichnungsversionen vorgestellt.

Kapitel 7 – Zusammenfassung und Ausblick: Die Arbeit schließt im siebenten Kapitel mit einer Zusammenfassung, den Erkenntnissen aus dieser Arbeit und einem Ausblick auf künftige Forschungsarbeiten.

2 Stand der Technik und Forschung

„Zweifeln ist der Anfang der Wissenschaft;
wer an nichts zweifelt, prüft nichts, wer nichts prüft, entdeckt nichts,
wer nichts entdeckt, ist blind und muß blind bleiben.“
Johann Christian Wiegleb (1732–1800), aus [Wiegleb 1777]

Dieses Kapitel beschäftigt sich mit dem aktuellen Stand der Technik und Forschung in der rechnergestützten Bauplanung. Aufbauend auf der Definition grundlegender Begriffe werden die objektorientierte Methode zur Bauwerks- und Fachapplikationsmodellierung und die darauf basierenden Kooperationskonzepte zur Versionierung, beim Austausch, bei der Archivierung, beim Vergleich und bei der Zusammenführung von Bauwerksinformationen beleuchtet. Begleitend dazu werden Probleme identifiziert sowie Zusammenhänge und Abgrenzungen zur vorliegenden Arbeit dargelegt. Es sei darauf hingewiesen, dass dieses Kapitel keinen umfassenden Überblick über den aktuellen Forschungsstand[1] auf dem Gebiet der vernetzt-kooperativen Bauwerksplanung darstellt, sondern lediglich die für diese Arbeit relevanten Technologien und Konzepte in der fachdomäneninternen Kooperation betrachtet.

2.1 Grundlagen und Begriffe

In diesem Abschnitt werden zunächst grundlegende, relevante Begriffe vorgestellt und deren Bedeutung und Verwendung im Zusammenhang mit dieser Arbeit festgelegt.

2.1.1 Modelle

Modell und Modellbildung: Die Abstraktion eines Systems auf die für eine bestimmte Aufgabe wesentlichen Elemente mit ihren Beziehungen zueinander basiert häufig auf einer Theorie[2] und wird Modell genannt. Als System wird meistens die Wirklichkeit verstanden. Der Vorgang der Erstellung eines Modells heißt Modellbildung. Während mathematische Modelle beispielsweise Differentialgleichungssysteme darstellen, basieren physikalische Modelle zusätzlich auf physikalischen Gesetzmäßigkeiten[3]. Computermodelle im Bauwesen sind im Allgemeinen digitale Modelle zur Planung, Berechnung und Visualisierung von Bauwerken mit dem Computer.

[1] s. hierzu [Rüppel 2007b] und auch [Weise 2006]
[2] z. B. Balkentheorie bzw. Biegetheorie des Balkens (I. Ordnung: $EIw''''(x) = q(x)$)
[3] z. B. Gravitation

Fachliches Modell: Die Beschreibung des Zustands und des Verhaltens von Bauwerken oder Bauwerksteilen als System aus einer fachlichen Sicht heraus wird fachliches Modell genannt. Auf Basis der Balkentheorie wird beispielsweise ein *Biegebalkenmodell mit Gleichlast* entwickelt, welches sowohl den Zustand des Balkens durch seine Länge l und die Last q als auch das Verhalten des Balkens durch das maximale Biegemoment $M_{max} = \frac{ql^2}{8}$ beschreibt.

Fachliche Instanz: Eine konkrete Ausprägung, ein Exemplar des fachlichen Modells wird fachliche Instanz genannt. Ein Biegebalken *beam01* mit einer Länge von 10 m und einer Gleichlast von 4 kN/m stellt mit einem maximalen Biegemoment von 50 kNm eine Instanz des fachlichen Modells *Biegebalken mit Gleichlast* dar.

Computermodell/ Bauwerksmodell: Eine Datenstruktur und darauf basierende Algorithmen zur rechnerinternen Abbildung des Zustands und des Verhaltens eines fachlichen Modells heißt Computermodell. Im Rahmen dieser Arbeit wird ein Computermodell als Abstraktion eines fachlichen Modells und nicht als direkte Abstraktion der Realität verstanden. Es wird demnach davon ausgegangen, dass für jedes Computermodell ein entsprechendes fachliches Modell existiert. Dem Stand der Technik entsprechend werden im Bauplanungsprozess objektorientierte Computermodelle[4] verwendet. Das vorgestellte fachliche Biegebalkenmodell kann beispielsweise durch die Klasse *Load* mit dem Attribut *value:double* und die Klasse *Beam* mit den Attributen *length:double*, *qload:Load* und der Methode *double:getMaxMoment()* abstrahiert werden. Computermodelle werden im Folgenden Bauwerksmodelle oder vereinfachend Modelle genannt.

Modellinstanz/ Bauwerksinstanz: Zur Laufzeit einer Fachapplikation wird ein Computermodell instanziiert, indem Objekte erzeugt werden. Die konkrete Ausprägung eines Biegebalkens als Objekt *beam01:Beam* mit der Länge *length=10.0* und einem Lastobjekt *qload:Load* mit dem Attributwert *value=4.0* auf der Basis des beschriebenen Bauwerksmodells heißt Modellinstanz oder Bauwerksinstanz.

Abstrahieren und Instanziieren: Den Zusammenhang zwischen den Vorgängen des Abstrahierens und des Instanziierens in der Modellbildung ist im Bild 2.1 veranschaulicht.

Ausgewertete Modelle: Bauwerksmodelle, die ein fachliches Modell zustands- bzw. ergebnisorientiert[5] beschreiben, werden ausgewertete Modelle genannt. Sie bilden den Planungsgegenstand als Ergebnis bzw. Zustand während der Verarbeitung ab. In der Volumenmodellierung stellt das B-Rep-Modell ein Volumen in ausgewerteter Form mit Punkten, Kanten und orientierten Flächen dar und ist demnach ein ausgewertetes Modell.

Nicht-ausgewertete Modelle: Bauwerksmodelle, die ein fachliches Modell änderungs- bzw. verhaltensorientiert[6] beschreiben, werden als nicht-ausgewertete Modelle bezeichnet. Sie bilden den Planungsgegenstand als Prozess bzw. Vorgang der Verarbeitung ab.

[4] s. Abschnitt 2.2.1 auf Seite 14
[5] auch: deklarativ, das *Was* beschreibend
[6] auch: prozedural oder imperativ, das *Wie* beschreibend

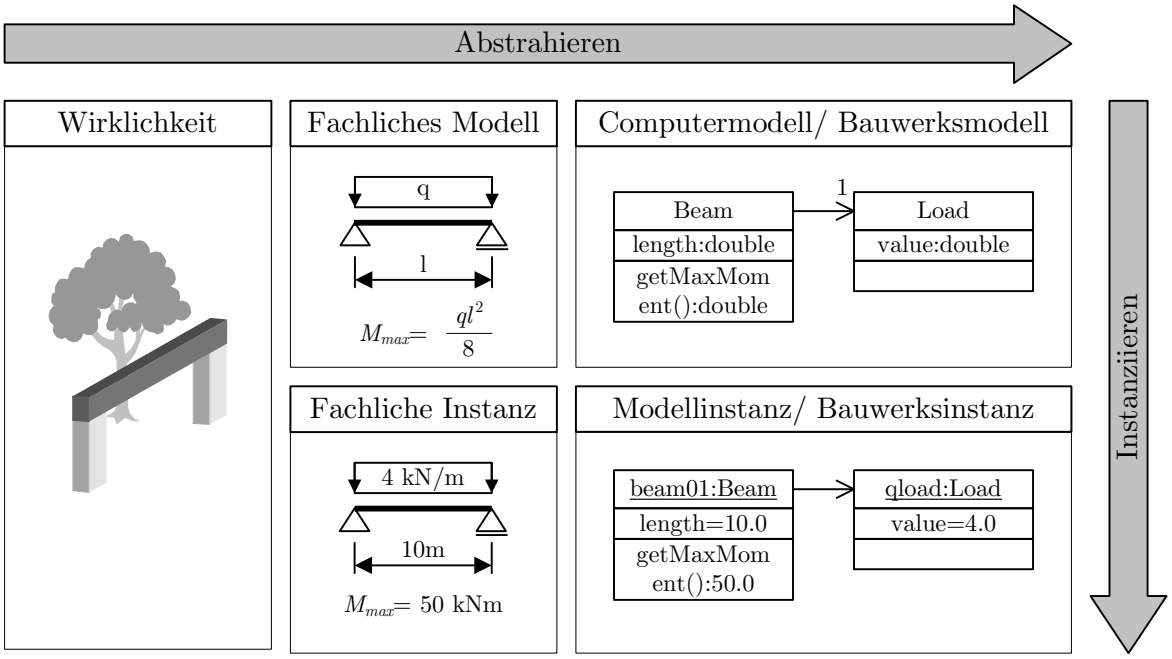


Bild 2.1: Abstrahieren und Instanziieren in der Modellbildung

Um einen Zustand zu erzeugen, werden nicht-ausgewertete Modellinstanzen zu ausgewerteten Modellinstanzen evaluiert. In der Volumenmodellierung stellt das CSG-Modell ein Volumen als Konstruktionsbaum mit primitiven Grundkörpern und boolschen Operationen dar und ist demnach ein nicht-ausgewertetes Modell. CSG-Modellinstanzen werden als Ergebnis der CSG-Operationen zu B-Rep-Modellinstanzen ausgewertet, um beispielsweise das Volumen zu bestimmen oder am Bildschirm zu visualisieren.

Hybride Modelle: Bauwerksmodelle, die ein fachliches Modell sowohl zustands- als auch änderungsorientiert beschreiben, werden als hybride Modelle bezeichnet. Ein Beispiel hierfür sind CSG-Modelle, die zusätzlich ausgewertete B-Rep-Informationen an den Knoten des Konstruktionsbaumes speichern.

2.1.2 Applikationen

Applikation: Computerprogramme bzw. Softwareanwendungen, die Informationen auf der Basis von Computermodellen im Rechner verarbeiten, heißen Applikationen. Es wird zwischen Systemsoftware (Betriebssystem, Geräteverwaltung, Übersetzer usw.) und Anwendersoftware (Fachapplikationen) unterschieden.

Fachapplikation: Computerprogramme, die von Fachplanern zur Lösung von Problemen innerhalb einer Fachdomäne verwendet werden, heißen Fachapplikationen. Im Bauplanungsprozess werden unterschiedliche Fachapplikationen für jeweils unterschiedliche Aufgaben (z. B. Fassadenentwurf, Bewehrungsplanung, FEM-Berechnungen, Kalkulation usw.) eingesetzt. Das einer Fachapplikation zugrunde liegende Computermodell basiert auf einem fachlichen Modell. So existiert beispielsweise eine Fachapplikation

zum Entwurf und zur Berechnung des maximalen Moments eines *Biegebalkens mit Gleichlast*.

2.1.3 Kooperation

Kooperation: Die Zusammenarbeit verschiedener Fachplaner im Prozess der Bauwerksplanung am gemeinsamen Planungsgegenstand zum gemeinsamen Ziel hin wird Kooperation[7] genannt. Sie basiert auf der Kommunikation der Beteiligten untereinander und der Koordination der einzelnen fachlichen Teilprozesse[8]. Die kooperative Bauwerksplanung ist durch ihren iterativen und verteilten Charakter gekennzeichnet. Auf dieser Basis hat [Bretschneider 1998] verschiedene Kooperationsarten beschrieben.

Kooperationsarten: Hinsichtlich der Zeit werden die *synchrone* und die *asynchrone* Kooperation unterschieden. Während synchrone Kooperation zur selben Zeit stattfindet, wird bei der asynchronen Kooperation zu unterschiedlichen Zeitpunkten zusammengearbeitet. Betrachtet man den Planungsgegenstand, dann wird bei synchroner Kooperation zusätzlich zwischen *wechselseitiger* und *paralleler* Kooperation differenziert. Synchron wechselseitig meint das gleichzeitige Arbeiten an denselben Teilen des Planungsmaterials, während beim synchron parallelen Zusammenarbeiten Fachplaner gleichzeitig unterschiedliche Teile des Planungsmaterials bearbeiten.

Kooperationsphasen: Die Kooperation im Bauplanungsprozess ist nach [Firmenich 2002], [Beer 2006] und [Weise 2006] durch drei sich zyklisch wiederholende Planungsphasen gekennzeichnet. Im ersten Planungsschritt wählen die Fachplaner die benötigte Teilmenge des gesamten Planungsmaterials aus. In der zweiten Phase erfolgt in einem privaten Arbeitsbereich die synchrone oder asynchrone Bearbeitung dieser Teilmenge, die im dritten Schritt in den gemeinsamen Informationsbestand zurückgeschrieben wird. Diese drei Kooperationsphasen zusammen bilden eine sogenannte *lange Transaktion*[9].

Kooperationskonzepte: Zur umfassenden Unterstützung der einzelnen Kooperationsarten und -phasen müssen die bei der Bauwerksplanung entstehenden Informationen verwaltet werden. In dieser Arbeit werden mit Informations- bzw. Datenmanagement[10] folgende Begriffe zusammengefasst:

- **Versionierung:** Im Planungsprozess wird der Zustand der Bauwerksinformationen iterativ geändert. Das Vorhalten bzw. Einfrieren bestimmter Planungsstände wird Versionierung genannt. Während der Bauwerksplanung entstehen Planungsrevisionen und auch Planungsvarianten. Beispielsweise erhalten technische Zeichnungen einen Änderungsindex und einen Zeitstempel als Versionskennzeichen. Geprüfte und somit als verbindlich bzw. gültig erklärte Versionen des Planungsmaterials bilden die Grundlage für die nachfolgenden Planungsschritte und werden als *Freigabestände*[11] bezeichnet.

[7] lat. cooperatio, deutsch: Zusammenarbeit, Mitwirkung
[8] vgl. [Rüppel 2007a]
[9] engl. long-duration transaction, vgl. [Katz 1990] und [Firmenich 2002]
[10] engl.: information management, data management
[11] s. [DIN 6789-5 1995]

- **Austausch:** Die am Planungsprozess beteiligten Fachplaner tauschen versionierte Bauwerksinformationen untereinander aus, um diese in ihren Fachapplikation weiter zu verarbeiten.
- **Archivierung:** Bei der Bauwerksplanung werden Planungsinformationen aus rechtlichen Gründen über lange Zeiträume archiviert. Der Aufbewahrung von Papierplänen in Planschränken aus den Zeiten manueller Planerstellung steht die heutige Langzeitarchivierung von digitalen Bauwerksinformationen gegenüber.
- **Vergleich:** Die während der kooperativen Bauwerksplanung entstehenden Planungsstände werden zur Änderungsermittlung bei Revisionen, zur Gegenüberstellung von Varianten und zur Konflikterkennung miteinander verglichen. Dieser Vorgang wird Vergleich (Diff) genannt. Das Ergebnis eines Informationsvergleichs sind die Unterschiede der verglichenen Versionsstände und eventuelle Konfliktsituationen.
- **Zusammenführung:** Im Zuge der kooperativen Bauwerksplanung entstehen Planungsvarianten und auch Planungskonflikte, die im Prozess einer Zusammenführung (Merge) zu einem neuen, gültigen Planungsstand vereinigt werden.

2.2 Bauwerksmodelle und Fachapplikationen

Dem Stand der Technik entsprechend wird die Objektorientierung als Methode in der Softwareentwicklung eingesetzt. Aufbauend auf einer kurzen Einführung in diese Methode werden objektorientierte Bauwerksmodelle und objektorientierte Fachapplikationen als Basis des kooperativen Bauplanungsprozesses innerhalb von Fachdomänen betrachtet.

2.2.1 Objektorientierte Methode

Objektorientierung: Nach [Claus u. Schwill 2001] stellt die Objektorientierung eine Methode aus der Informatik dar, um auf Basis von Objekten einerseits Informationen darzustellen, zu strukturieren, zu verarbeiten, zu übertragen und zu speichern und andererseits, um Softwaresysteme zu planen, zu entwerfen und umzusetzen (s. [Booch 1994] und [Balzert 2005]). Mit Hilfe dieser Methode werden sowohl Computermodelle als auch Applikationen objektorientiert entwickelt.

Objekt: Ein Objekt abstrahiert ein Element eines betrachteten Systems[12] und bildet dessen Eigenschaften und Verhalten zur Lösung einer bestimmte Aufgabe ab. In [Booch 1994] wird ein Objekt wie folgt definiert.

> *"An object has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class; [...]"*

Klasse: In der Objektorientierung werden Datenstrukturen (Zustand) und Algorithmen bzw. Verarbeitungsmethoden (Verhalten) als Einheit der Modellbildung zusammengefasst und in Klassen definiert. Ein Objekt ist die Instanz oder Ausprägung einer Klasse.

Attribut: Die Eigenschaften eines Objekts beschreiben seinen Zustand. Sie werden in Klassen als Attribute definiert und zur Laufzeit einer Fachapplikation mit Werten belegt. Attributwerte speichern die Daten des abzubildenden Systems.

Methode: Das Verhalten eines Objekts wird durch seine Methoden abgebildet. Methoden werden in Klassen definiert und zur Laufzeit einer Fachapplikation ausgeführt, um den Objektzustand zu ändern.

Objektorientiertes Modell: Eine strukturierte Klassenmenge zur Beschreibung von Objekten mit Attributen und Methoden dient der rechnerinternen Abbildung von fachlichen Modellen eines Bauwerks oder von Bauwerksteilen. Sie stellt ein objektorientiertes Modell bzw. Computermodell dar (s. Bild 2.2). Zur Laufzeit einer Applikation wird das Computermodell instanziiert und als strukturierte Objektmenge bzw. Objektmodell verarbeitet.

[12] hier ist das betrachtete System ein fachliches Modell eines Bauwerks oder Bauwerksteils

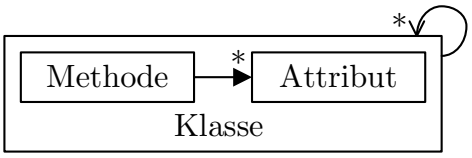


Bild 2.2: Aufbau eines objektorientierten Modells

Datenkonsistenz: Daten- bzw. Attributkapselung[13] ist ein Prinzip der Objektorientierung. Es beschreibt das Schützen von Daten vor direktem externem Zugriff. Die interne Struktur der Daten bleibt dem Entwickler vorbehalten. Als einheitliche Schnittstelle kapseln Methoden die Daten, kontrollieren den Zugriff und stellen so die Konsistenz der Daten innerhalb des Objekts sicher. Das Bild 2.3 veranschaulicht das objektorientierte Prinzip der Datenkapselung.

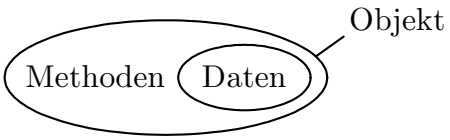


Bild 2.3: Prinzip der Datenkapselung

Grafische Notation: Die UML[14] (Unified Modeling Language) stellt als standardisierte Modellierungssprache unter anderem grafische Diagramme zur Veranschaulichung und Dokumentation objektorientierter Systeme bereit. Im Rahmen der vorliegenden Arbeit werden als grafische Notation das UML-Klassendiagramm und das UML-Zustandsdiagramm verwendet.

2.2.2 Bauwerksmodelle

DFG-Schwerpunktprogramm 694: Im DFG[15]-Schwerpunktprogramm „Objektorientierte Modellierung in Planung und Konstruktion“ wurde die Objektorientierte Methode als Modellierparadigma für Problemstellungen aus dem Bereich der Planung und der Konstruktion im Bauwesen erforscht. Die Ergebnisse und Erkenntnisse werden in [Hartmann 2000, S. 24] wie folgt zusammengefasst:

- Mit der Objektorientierung steht *„ein informatisches Kalkül zur Beschreibung, Erfassung und computergerechten Repräsentation von komplexen bzw. hochgradig komplexen Ingenieurproblemen [...] zur Verfügung. Sowohl die statische Problemstruktur als auch die dynamische Verhaltensstruktur sowie die funktionalen Aspekte eines Problems lassen sich schlüssig abbilden.“*

[13] auch Sichtbarkeits- oder Geheimnisprinzip
[14] s. [OMG 2003], [Booch u. a. 2001] und [Rumbaugh u. a. 2001]
[15] Deutsche Forschungsgemeinschaft

- *„Das objektorientierte Paradigma erlaubt es [...], Softwaresysteme zu schaffen, die nicht nur technische, informatische Eigenschaften [...] wie Effizienz, Schnelligkeit, Wiederverwendbarkeit abdecken, sondern auch menschenorientierte Aspekte wie individuelle Auslegung, gute Bedienbarkeit, interaktive Einbindung von Anwendern oder kooperative Aspekte.“*

Objektorientiertes Bauwerksmodell: Auf der Basis der Objektorientierung werden Computermodelle zur Beschreibung von Bauwerken oder Bauwerksteilen entwickelt. Diese Modelle werden objektorientierte Bauwerksmodelle genannt. Im Rahmen dieser Arbeit wird zwischen ausgewerteten, nicht-ausgewerteten und hybriden Bauwerksmodellen unterschieden.

Ausgewertete Bauwerksmodelle

Industry Foundation Classes: Die *Industry Foundation Classes*[16] (IFC) sind an dieser Stelle als bekanntestes ausgewertetes Bauwerksmodell zu nennen. Die Einordnung der IFC in die Kooperationskonzepte des Bauplanungsprozesses wird im Abschnitt 2.3 auf Seite 22 beschrieben.

Native Bauwerksmodelle: Die nativen Computermodelle verfügbarer Fachapplikationen im Bauwesen beschreiben das Bauwerk in ausgewerteter Form als Zustand. Aufgrund der Vielzahl von Fachapplikationen existiert auch eine große Anzahl von Modellen. Beispielsweise setzen die DWG-Datenbasis der AutoDesk-Produkte[17], die DGN-Datenbasis von Microstation[18] und auch die NDW-Datenbasis der Software Allplan BIM[19] ein natives, ausgewertetes Bauwerksmodell um.

Nicht-ausgewertete und hybride Bauwerksmodelle

Nicht-ausgewertetes Volumenmodell: Als nicht-ausgewertetes Bauwerksmodell ist das CSG-Modell der Volumenmodellierung im Bauwesen zu nennen. Es ist jedoch keine Fachapplikation bekannt, die ein reines CSG-Modell implementiert. Zur Auswertung werden CSG-Modellinstanzen in B-Rep-Modellinstanzen überführt, um beispielsweise das Volumen zu bestimmen.

Hybride Volumenmodelle: Hybride Bauwerksmodelle sind derzeit überwiegend bei der Geometrie- bzw. Volumenmodellierung von Bauwerken zu finden. In sogenannten parametrischen Volumenmodellierern wie beispielsweise Pro/ENGINEER[20], Solid-

[16] s. [Adachi u. a. 2003], [Eastman 1999, S. 279 ff.] und [Kiviniemi u. a. 2008]
[17] CAD- und BIM-Lösungen: AutoCAD, AutoCAD Architecture, AutoCAD Revit usw., s. http://www.autodesk.de
[18] Integrierte CAD-Plattform für Architekten und Ingenieure von Bentley, s. http://www.bentley.com/de-DE/Products/MicroStation/
[19] CAD-Lösung für Architektur und Ingenieurbau von Nemetschek, s. http://www.allplan.de
[20] Integrierte 3D-CAD/CAM/CAE-Lösung von PTC, s. http://www.ptc.com/products/proengineer

Works[21] und Generative Components[22] werden hybride Computermodelle zur Volumenbeschreibung verwendet. Neben der Abbildung des ausgewerteten Volumens speichern Konstruktionsbäume die Entwurfshistorie als nicht-ausgewertete Operationsstruktur.

Probleme von objektorientierten Bauwerksmodellen

Objektidentifikation: Zur Laufzeit einer Fachapplikation wird einem Objekt bei der Erzeugung ein temporärer oder transienter Identifikator zugeordnet. In der Regel ist ein transienter Identifikator die Adresse des Objekts im Arbeitsspeicher. Wird ein Objekt persistent gespeichert und anschließend von einer Fachapplikation geladen, erhält es einen anderen, neuen transienten Objektidentifikator. Transiente Objektidentifikatoren sind *zeitabhängig* und deshalb zur Identifikation von Objekten im kooperativen Bauplanungsprozess ungeeignet ([Beer 2006]). In [Weise 2006] wird gezeigt, dass in aktuell eingesetzten Modellen am Beispiel von IFC-Datensätzen

> *„[...] der Anteil der eindeutig identifizierbaren Objekte [...] für praxisrelevante Problemstellungen zwischen 5% und deutlich unter 1% liegt.“*

Aus diesem Grund werden in [Bilchuk 2005] und [Beer 2006] persistente Objektidentifikatoren (POIDs) eingeführt, die innerhalb einer Modellinstanz bzw. eines Dokuments eindeutig und über die gesamte Lebensdauer eines Objekts konstant sind. Im Rahmen dieser Arbeit werden die POIDs als Operanden in Modellieroperationen verwendet. Die Verfügbarkeit von POIDs stellt somit eine Anforderung an die zugrunde liegenden Bauwerksmodelle dar.

Verarbeitungssemantik: Obwohl die Objektorientierte Methode die Beschreibung von Objekten als Einheit von Attributen und Verarbeitungsmethoden vorsieht, werden letztere bei der Definition von Bauwerksmodellen oft vernachlässigt. Am Beispiel der IFC ist ersichtlich, dass die Verarbeitungssemantik, das heißt eine Beschreibung, *wie* IFC-Objektmodelle konsistent zu verarbeiten sind, fehlt. Es werden lediglich Implementierungshinweise[23] in Form detaillierter Klassenstrukturen mit Attributdefinitionen für die Entwicklung von IFC-basierten Fachapplikationen gegeben.

Erste Ansätze zur Abbildung einer Verarbeitungssemantik sind in den Konstruktionsbäumen von parametrisierten Volumenmodellen zu erkennen. Hier wird der Konstruktionsvorgang implizit in der Modellinstanz festgehalten. Im Gegensatz zum vorgeschlagenen Modellierungsansatz bleibt die Verarbeitung der geometrischen Parameter (Attribute) selbst unberücksichtigt. In der vorliegenden Arbeit wird ein Ansatz zur Definition und Standardisierung der Verarbeitungssemantik von objektorientierten Bauwerksmodellen durch Modellieroperationen vorgestellt.

[21] CAD für den Maschinenbau von SolidWorks, s. http://www.solidworks.de/pages/products/3dmech.html

[22] Parametrisches und assoziatives Entwurfssystem auf Basis von Bentley's Microstation, s. [Aish 2005] und http://www.bentley.com/en-US/Markets/Building/GenerativeComponents.htm

[23] s. [Liebich 2004]

Modellinstanzkonsistenz: Die fehlende Verarbeitungssemantik in der Definition von objektorientierten Computermodellen führt zu Problemen hinsichtlich der Konsistenz von Bauwerksinstanzen. Während der Verarbeitung von Bauwerksinformationen mit einer Fachapplikation muss sichergestellt werden, dass die in der Modellinstanz abgebildeten Informationen widerspruchsfrei sind. In [Laabs 1998] und [Hanff 2003] werden Ansätze zur Konsistenzsicherung in objektorientierten Bauwerksinstanzen vorgeschlagen. [Laabs 1998] stellt ein auf Objektmengen basierendes Beobachterkonzept vor, welches einerseits mengenstrukturierte Modelle voraussetzt und andererseits nur für kleine Teilmodelle handhabbar ist, da sehr komplexe Strukturen bei großen Modellen entstehen. Im Unterschied dazu wird in [Hanff 2003] eine verzögerte Aktualisierung der Objektmenge auf Grundlage von Abhängigkeiten vorgeschlagen. Dieser Ansatz setzt die zusätzliche Implementierung eines generalisierten Schnittstellenapparats (Zugriff auf Attribute und auf Ein- und Ausgabeparameter von Methoden, Aktualisierungsmethoden für Attribute) voraus, die durch eine separate Klassenbeschreibung in XML generiert werden kann. Beide Ansätze können nicht auf verfügbare, bereits in Fachapplikationen implementierte Computermodelle angewendet werden, sondern sind lediglich für Neuentwicklungen relevant. In der vorliegenden Arbeit wird gezeigt, wie auf Basis von Modellieroperationen die Konsistenz von verfügbaren Bauwerksinstanzen in bereits existierenden Fachapplikationen während der Verarbeitung sichergestellt werden kann.

2.2.3 Fachapplikationen

Fachapplikation: Neben den Bauwerksmodellen werden auch die Fachapplikationen auf Grundlage der Objektorientierung entwickelt. Eine objektorientierte Fachapplikation stellt ein Softwareprogramm dar, mit dem ein Fachplaner eine Bauwerksinstanz verarbeitet. In Analogie zu einem Betriebssystem[24] ist der typische Aufbau einer objektorientierten Fachapplikation wie beispielsweise AutoCAD[25] im Bild 2.4 dargestellt.

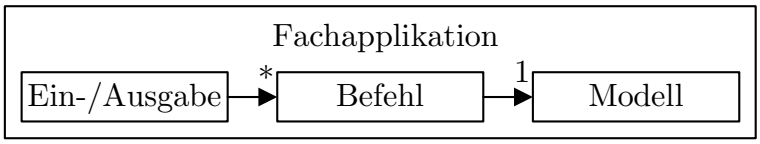


Bild 2.4: Aufbau einer objektorientierten Fachapplikation

In der Komponente *Ein-/Ausgabe* wird die grafische oder alphanumerische Benutzerinteraktion des Fachplaners mit der Fachapplikation abgebildet. In Anlehnung an das objektorientierte Kommando-Entwurfsmuster[26] werden Benutzereingaben in der Komponente *Befehl* durch Befehlsobjekte abstrahiert. Die Befehle dienen der Verarbeitung des *Modells* und sind aus diesem Grund von fundamentaler Bedeutung für die vorliegende Arbeit. Das Konzept der Verarbeitungsabstraktion durch Befehle wird hier aufgegriffen und zur Kooperationsunterstützung erweitert.

[24] Benutzerschnittstelle — Verarbeitungseinheit(Prozesse) — Dateisystem, vgl. [Bourne 1988]
[25] CAD-Fachapplikation der Firma AutoDesk, `http://www.autodesk.de/autocad`
[26] engl.: Command, s. [Gamma u. a. 2002]

Benutzerschnittstellen: Grafische Benutzerschnittstellen[27] werden dem Stand der Technik entsprechend auf Basis der Objektorientierung ereignisorientiert[28] entwickelt. Zum einen dienen sie der Präsentation des Objektmodells, zum anderen nehmen sie die Eingabe des Fachplaners entgegen. Der Fokus in dieser Arbeit liegt auf den Eingabemöglichkeiten. Die grafischen Benutzerschnittstellen verfügbarer Fachapplikationen innerhalb einer Fachdomäne können unter anderem wie folgt charakterisiert werden:

- Sie sind sehr komplex im Funktionsumfang.
- Sie unterscheiden sich stark zwischen den einzelnen Fachapplikationen.
- Sie ändern sich zum Teil auch mit jeder neuen Applikationsversion.

Die Auffassung der hohen Komplexität wird auch von Don Norman[29] in einem aktuellen Interview auf SPIEGEL ONLINE vertreten (s. [Norman 2008]):

> *„Die Kommandozeile ist schnell. Man muss einfach eingeben, was man vom Computer will. [...] Die grafische Benutzeroberfläche ist großartig, gerade weil man sich nichts merken muss. Aber sie funktioniert nur, wenn man aus wenigen Sachen auswählen muss.“*

Diese Nachteile vorhandener Benutzerschnittstellen lassen den Schluss zu, dass (a) verfügbare Fachapplikationen nur noch von geschulten Spezialisten zu bedienen bzw. zu beherrschen sind und (b) es sich deshalb lohnt, über eine einfache Kommandosprache als standardisierte, *zusätzliche* Benutzerschnittstelle zu Fachapplikationen innerhalb einer Fachdomäne nachzudenken. Darüber hinaus fördert eine alphanumerische Benutzereingabe die Makrofähigkeit von Fachapplikationen insofern, dass Eingaben als Makros aufgezeichnet und beispielsweise mit anderen Parametern wieder abgespielt werden können.

Programmierschnittstellen: Die Programmierschnittstellen[30] von objektorientierten Fachapplikationen dienen der Vergrößerung ihres Funktionsumfangs durch die Erweiterung des nativen Modells und das Hinzufügen entsprechender Verarbeitungsbefehle. Mit Hilfe von Programmierschnittstellen werden beispielsweise bauwesenspezifische Fachapplikationen auf der Basis allgemeiner CAD-Systeme entwickelt[31]. In [Beucke u. a. 1990] werden verschiedene Ebenen von Programmierschnittstellen bezüglich deren Offenheit unterschieden. Die Tabelle 2.1 auf der nächsten Seite stellt diese Ebenen dar und ordnet ihnen Anwender und Anwendungen zu, wobei sich diese Arbeit mit den hervorgehobenen Ebenen beschäftigt.

[27] engl.: Graphical User Interface (GUI)
[28] s. [Gumm u. Sommer 2002, S. 246 ff.]
[29] geboren 1935, emeritierter Professor für Kognitionswissenschaften an der University of California, San Diego und lehrender Professor für Informatik an der Northwestern University in Chicago
[30] Application Programming Interface (API)
[31] z. B. ObjektARX-Programmierschnittstelle für AutoCAD, s. `http://www.objectarx.com` und `http://www.autodesk.de/autocad`

API-Ebene	Anwender	Anwendung
Makros	Fachplaner	Standardaktionen, komfortable Erledigung fachspezifischer Anforderungen
Interpretierte Skriptsprache	Fachplaner/ Ingenieur	Parametrisierung, Ablaufsteuerung
Compilierte Programmiersprache	Softwareentwickler/ Informatiker	Modellerweiterung, Applikationsentwicklung

Tabelle 2.1: Ebenen von Programmierschnittstellen

Mit Bezug auf die vorliegende Arbeit werden folgende Probleme mit derzeit verfügbaren Programmierschnittstellen identifiziert:

- Sie sind sehr komplex in der Anwendung.
- Es existieren keine Standards.
- Der Austausch modellerweiternder Informationen ist problematisch.

Aufgrund der hohen Komplexität ist es nur professionellen Programmierspezialisten möglich, verfügbare objektorientierte Programmierschnittstellen zu verwenden. Fachplaner selbst können auf dieser Basis keine einfachen Modellierprogramme zum automatisierten Modellieren umsetzen. Infolge fehlender API-Standards ist eine Portierung von auf API-Basis erstellten Fachapplikationen bzw. einfachen Modellierprogrammen von einem Softwaresystem in ein anderes nahezu unmöglich. Ferner können Bauwerksinformationen, die auf der Grundlage von Modellerweiterungen erstellt worden sind, auch nur bei Verfügbarkeit dieser Erweiterung wiederverwendet werden.

Kommandosprachen: Nach [Claus u. Schwill 2001] stellt eine Kommandosprache[32] eine alphanumerische Schnittstelle zwischen Betriebssystem und Benutzer dar. Mit Hilfe einer Kommandosprache werden Aufträge an einen Rechner formuliert. Ein Kommandointerpreter (auch: Shell) evaluiert die Benutzereingaben und führt entsprechende Aktionen aus. Ein Kommando setzt sich in der Regel aus einem Aktionswort und einer Parameterliste zusammen.

Skriptsprachen: Die Konzepte der Kommandosprachen finden sich in Skriptsprachen wieder, wenngleich diese eine Erweiterung der Kommandosprachen darstellen. Mit Hilfe von Skriptsprachen können kleine, überschaubare Programme formuliert werden, die unter anderem vorhandene Programme oder Prozeduren nacheinander oder kooperierend ablaufen lassen. Skriptsprachen werden interpretiert und zeichnen sich durch folgende Eigenschaften aus:

- Variablen- und Prozedurenkonzept,

[32] engl.: command language

- im Allgemeinen nicht typisiert,
- Kontrollstrukturen (Sequenz, Verzweigung und Wiederholung),
- Makrofähigkeit durch Kommandosequenzen.

Eine sehr bekannte und mächtige Skriptsprache ist Tcl/ Tk[33]. Die Aktualität dieser Skriptsprache zeigt sich am State-of-the-art–FEM[34]-Framework OpenSees, welches Tcl verwendet und um Kommandos zur Modellierung, Analyse und Ausgabespezifikation von FE-Simulationen erweitert (s. [Mazzoni u. a. 2006] und [Gerold u. Koch 2007]). Weiterhin beachte man den Erfolg der Skriptsprachen JavaScript und PHP im Internet. Beispielsweise ist ein Großteil der verfügbaren web-basierten Content-Management-Systeme[35] auf der Basis dieser Skriptsprachen erfolgreich entwickelt worden. Nicht zuletzt sei die Shell-Skript-Programmierung[36] von Betriebssystemen erwähnt. Für Systemadministratoren haben Shell-Skripte eine große Bedeutung, weil sie damit (a) auf verfügbare Funktionalität, die durch grafische Benutzerschnittstellen nicht zugänglich ist, zugreifen und (b) administrative Konfigurationen automatisiert ablaufen lassen können.

Fachsprachen: Im Zusammenhang mit dieser Arbeit stellt eine Fachsprache eine kommandobasierte Sprache zur Beschreibung von Modellieroperationen innerhalb einer Fachdomäne dar. In [Dahlenburg 1996] wird eine Fachsprache für die Fachdomäne *Gestaltmodellierung in frühen Phasen des architektonischen Entwurfs* entwickelt und umgesetzt. Es wird damit nachgewiesen, dass eine Fachsprache vorteilhaft im rechnergestützten Entwurfsprozess eingesetzt werden kann: Die Sprache dient

- als Werkzeug für die Protokollierung von Objektmodellen (Entwurfsabsicht),
- als Schnittstelle für Anwendungsprogrammierer zur Schaffung spezifischer CAD-Systeme (API) und
- als formale Grundlage für verschiedenste Interaktionstechniken (UI).

Die vorliegende Arbeit greift den Ansatz einer Fachsprache zur sprachbasierten Definition von Modellieroperation in einer Fachdomäne auf, geht aber über die Anwendung in Benutzer- und Programmierschnittstellen hinaus. In dieser Arbeit wird auf der Basis einer verarbeitungsorientierten Modellbildung eine operative Modellierungssprache entwickelt und deren Anwendung im Hinblick auf die fachdomäneninterne Kooperation untersucht.

[33] Tool command language und Toolkit, deutsch: Werkzeugbefehlssprache und Werkzeugkasten, s. [Ousterhout 1995], [Harrison 1997] und [Flynt 1999]
[34] Finite Element Method, s. [Cook u. a. 1989]
[35] CMS, deutsch: Inhaltsverwaltungssystem, i. A. zur Verwaltung von Webseiteninhalten
[36] z. B. Unix-Shell, Windows PowerShell usw.

2.3 Kooperationskonzepte

In diesem Abschnitt werden die Anwendung der objektorientierten Bauwerksmodelle und Fachapplikationen und die darauf basierenden Kooperationskonzepte bei der Versionierung, beim Austausch, bei der Archivierung, beim Vergleich und bei der Zusammenführung von Bauwerksinformationen innerhalb einzelner Fachdomänen des Bauplanungsprozesses beleuchtet.

DFG-Schwerpunktprogramm 1103: Im DFG-Schwerpunktprogramm 1103 „Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau“ wurden verteilte Produktmodelle, netzwerkgerechte Prozessmodelle, verteilte Simulationen und Agentensysteme im Hinblick auf die arbeitsteilige Projektbearbeitung im Informations- und Kommunikationsverbund des Konstruktiven Ingenieurbaus erforscht. Die Ergebnisse und Erkenntnisse werden in [Rüppel 2007b] zusammengefasst. Der Themenbereich *Verteilte Produktmodelle* ist von großer Bedeutung für diese Arbeit, da sich dieser der Untersuchung der verteilten Bearbeitung einer objektorientierten Bauwerksinstanz und deren Konsistenzerhaltung als zentralen Forschungsgegenstand widmet (s. [Firmenich u. Rank 2007]).

2.3.1 Versionierung von Bauwerksinformationen

Ausgehend von der Forderung nach langen Transaktionen[37] ist die Versionierung als Basis einer verteilten Bearbeitung des gemeinsamen Planungsmaterials allgemein anerkannt. In [Katz 1990], [Firmenich 2002], [Beer 2006], [Beucke 2006] und [Weise 2006] wird gezeigt, wie versionierte Modelle die Konsistenz und die Nachvollziehbarkeit der verteilten Bauwerksinformationen im Ingenieurwesen gewährleisten. Grundsätzlich werden zwei Versionierungsansätze unterschieden: der *versionsorientierte* und der *änderungsorientierte* Ansatz.

Versionsorientierter Ansatz

Versionsstände: Beim versionsorientierten Ansatz werden die einzelnen Versionsstände als Zustände der Modellinstanz – als Modellinstanzversionen – explizit gespeichert. Im Hinblick auf die Granularität des Versionierungsgegenstandes wird zwischen *Dokumentversionierung* und *Objektversionierung* unterschieden.

Dokumentversionierung: Die von Fachapplikationen verarbeiteten Objektmodelle werden traditionell in nativen Dokumenten bzw. Dateien gespeichert. Zur Verwaltung von Dokumenten im Bauplanungsprozess werden Dokumentenmanagementsysteme (DMS) eingesetzt. Die Einordnung und die Anwendungsbereiche von DMS sind in [Klingelhöller 2001] und [Gulbins u. a. 1999] beschrieben und können wie folgt zusammengefasst werden. DMS dienen der Unterstützung des gesamten Dokumentlebenszyklus

- bei der Erstellung, Verteilung und Verwaltung,

[37] vgl. Abschnitt 2.1.3 auf Seite 12

- bei der Erfassung, Indizierung und Archivierung,
- bei der Verwaltung von Zugriffsberechtigungen und
- bei der Versionierung.

DMS sind in Bezug auf deren Anwendung im verteilten Bauplanungsprozess mit Einschränkungen verbunden, die eine nur eingeschränkte Kooperationsunterstützung ermöglichen. Einerseits bilden DMS lediglich lineare Dokumenthistorien, das heißt ausschließlich Revisionen, aber keine im Bauplanungsprozess auftretenden Varianten ab. Andererseits ist festzuhalten, dass DMS den Versionenvergleich und die Zusammenführung von Versionen nicht unterstützen können, da ihnen die Struktur und der Inhalt der versionierten, nativen Dokumente nicht bekannt sind (s. [Beer 2006] und [Firmenich u. a. 2005]).

Objektversionierung: Im Unterschied zur Dokumentversionierung wird in der Objektversionierung das Objekt als Einheit der Versionierung betrachtet. In [Firmenich 2002] wird ein Versionierungsmodell auf Basis einer strukturierten Objektversionsmenge mit Bindungen sowie Operationen zur konsistenten verteilten Bearbeitung im Bauplanungsprozess vorgestellt. Auf dieser Grundlage hat [Beer 2006] die Anwendung dieses Ansatzes im Hinblick auf verteilte Modelle und Applikationen untersucht, wobei – wie auch in dieser Arbeit – großer Wert auf die Wiederverwendbarkeit verfügbarer Bauwerksmodelle und Fachapplikationen gelegt wird. In [Nour u. a. 2006] und [Weise 2006] wird gezeigt, wie die Objektversionierung im Rahmen von IFC-Bauwerksinstanzen eingesetzt werden kann.

In [Weise 2006] wird auf Basis der Objektversionierung ein Ansatz zur Abbildung von Änderungen in der modellbasierten Objektplanung vorgestellt. Änderungen dienen dabei der Dokumentation, der Bewertbarkeit und der Korrigierbarkeit von Planungsschritten. Im Unterschied zur vorliegenden Arbeit werden (a) die Änderungssemantik im Objektkontext[38] und nicht im Modellkontext definiert und (b) die Objektänderungen zwischen Versionsständen im Nachhinein auf Basis heuristischer Verfahren berechnet. Gegenstand der vorliegenden Arbeit ist die Definition von Modellieroperationen, die direkt der Beschreibung von semantischen Änderungen zwischen Versionsständen dienen und ohnehin von Fachplanern als Befehle instanziiert werden.

Änderungsorientierter Ansatz

Änderungen: Im Unterschied zum versionsorientierten Ansatz werden im änderungsorientierten Modell nicht die einzelnen Versionstände, sondern die Änderungen zwischen diesen gespeichert. Die Versionsstände werden dann durch die Ausführung bzw. die Anwendung der Änderungen erzeugt.

Softwareentwicklung: Im Prozess der Softwareentwicklung werden zur effizienten Speicherung und Verwaltung von Quellcodeversionen in Versionskontrollsystemen[39]

[38] Erzeugen, Verändern, Löschen, Teilen, Zusammenlegen und Evolution unabhängiger Modellobjekte
[39] engl.: Version Control System (VCS)

unter anderem änderungsorientierte Ansätze verfolgt[40]. Im Unterschied zum Bauplanungsprozess werden bei der Softwareentwicklung aber fast ausschließlich Textdokumentversionen verwaltet, deren Änderungen sich aufgrund der bekannten Änderungssemantik mit verfügbaren Algorithmen[41] ermitteln lassen (s. [Ramunno 2005] und [Firmenich u. a. 2005]).

Bauplanungsprozess: Änderungsorientierte Ansätze zur Versionsmodellierung im Bauplanungsprozess sind neben [Weise 2006] nicht bekannt. Die Abgrenzung zur vorliegenden Arbeit ist bereits oben dargelegt, wird aber in den folgenden Abschnitten noch weiter konkretisiert.

Kombinierter Versionierungsansatz

Vereinheitlichter Versionierungsansatz: In [Zeller 1997] wird nachgewiesen, dass der versionsorientierte und der änderungsorientierte Ansatz im Ergebnis äquivalent sind. Um die Vorteile beider Ansätze kombinieren zu können, wird weiterhin ein vereinheitlichtes Versionierungsmodell für das Software-Configuration-Management[42] vorgeschlagen. In [Firmenich 2002] wird bereits gezeigt, wie der versionsorientierte Ansatz aus der Softwareentwicklung auf den verteilten Bauplanungsprozess übertragen werden kann. Darauf aufbauend wird im Rahmen der vorliegenden Arbeit der änderungsorientierte Ansatz untersucht, ein kombinierter verarbeitungsorientierter Modellierungsansatz vorgestellt und dessen Anwendung bei der Versionierung von Bauwerksinformationen gezeigt.

2.3.2 Austausch von Bauwerksinformationen

Standardisierung: Der Informationsaustausch beschreibt den Vorgang des Austauschs der versionierten Bauwerksinstanzen zwischen verschiedenen Fachapplikationen. Die grundlegende Idee des Austauschs basiert auf einem zu definierenden bzw. zu vereinbarenden Modellstandard für die Beschreibung der auszutauschenden Bauwerksinformationen. Dieser Standard wird unabhängig von der Art des Workflows[43] beim Austausch festgelegt. Im Rahmen dieser Arbeit wird zwischen der Standardisierung ausgewerteter, nicht-ausgewerteter und hybrider Modelle unterschieden.

ISO-STEP-Standard: Von großer Bedeutung für den Austausch von Bauwerksinformationen als Produktdaten ist der ISO-STEP[44]-Standard („Standard for the Exchange of Product Data“). Unter anderem werden in diesem Standard die Sprache EXPRESS (ISO 10303-11) als Formalismus zur Modelldefinition und die alphanumerische Repräsentation der auszutauschenden Modellinstanzen (ISO 10303-21) festgelegt.

[40] vgl. [Zeller 1997]
[41] z. B. *longest common subsequence problem*, s. [Hunt u. McIlroy 1976]
[42] SCM
[43] dokumentenbasiert oder modellserverbasiert
[44] Bezeichnung für ISO 10303

Ausgewertete Modelle

Standardisierte Bauwerksinformationsmodelle: Ein Bauwerksmodell, welches sämtliche, den Lebenszyklus eines Bauwerks betreffenden Informationen integrierend in einem Modell abbildet, heißt Bauwerksinformationsmodell[45] (BIM) oder auch Produktdatenmodell. Dem Stand der Forschung entsprechend sind die *Industry Foundation Classes*[46] (IFC) als bekanntestes standardisiertes BIM zu nennen. Die IFC stellen einen Versuch der IAI[47] dar, ein sehr komplexes, integriertes Bauwerksinformationsmodell mit semantischen Objekten wie Wänden, Stützen, Decken, Türen, Fenstern, Dächern, Treppen, Geländer usw. zu entwickeln. Es zielt auf die Integration aller Planungsdomänen (Architekturplanung, Tragwerksplanung, TGA-Planung (HVAC), Bauablaufplanung usw.) in ein Modell ab. Der Austausch von IFC-Bauwerksinstanzen zwischen den Planungsbeteiligten basiert (a) traditionell auf Dokumenten im STEP-Format oder (b) auf einen sogenannten IFC-Modellserver (s. [Kiviniemi u. a. 2005]), welcher auch den Zugriff auf Teilmodelle erlaubt. Gegenstand der vorliegenden Arbeit ist nicht der Austausch von ausgewerteten Bauwerksinformationen zwischen verschiedenen Planungs- oder Fachdomänen, sondern der Austausch nicht-ausgewerteter Bauwerksinstanzen innerhalb einer Fachdomäne.

Standardisierte Fachdomänenmodelle: Zur Komplexitätsreduktion in BIMs werden Partialmodelle für die jeweiligen Fachdomänen (hier: Fachdomänenmodelle) eingeführt. Diese werden in [Willenbacher 2002, S. 53] wie folgt definiert:

> *„Ein Partialmodell spezifiziert ein phasen- bzw. disziplinspezifisches objektorientiertes konzeptuelles Schema, demnach eine abgeschlossene Menge von Objektbeschreibungen und deren Relationen untereinander als Teilmenge des Bauwerksmodells. Die Objektmengen der Partialmodelle sind nicht notwendigerweise disjunkt bzw. redundanzfrei.“*

In den IFC sind diese Partialmodelle im sogenannten Domain Layer als Architecture Domain, Structural Analysis Domain, HVAC Domain, Construction Management Domain und Facilities Management Domain vorgesehen, mit der Version IFC2x Edition 2 allerdings noch nicht vollständig verfügbar[48]. Als weitere Beispiele für Fachdomänenmodelle werden nachfolgende Standards aufgeführt:

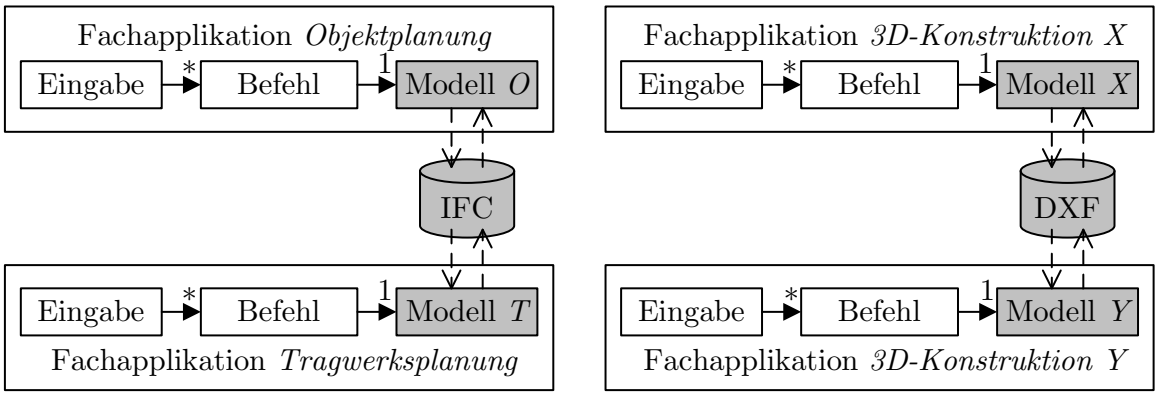
- **Stahlbau:** CIMsteel[49] als integrierte Modellspezifikation für Stahlbaukonstruktionen; entspricht dem ISO-STEP Part 230 „Application protocol: Building structural frame: Steelwork“ und ist Teilmenge von ISO-STEP Part 214 „Application protocol: Core data for automotive mechanical design processes“(CIMsteel Integration Standard CIS/2, s. [Crowley u. Watson 2000])

[45] engl. Building Information Model (BIM)
[46] s. [Adachi u. a. 2003], [Eastman 1999, S. 279 ff.] und [Kiviniemi u. a. 2008]
[47] International Alliance for Interoperability, s. http://www.iai-international.org/
[48] s. [Adachi u. a. 2003]
[49] Computer Integrated Manufacturing for Constructional Steelwork

- **3D-Zeichnungsmodell:** DXF[50] als Quasi-Standard von AutoDesk für ein geometrieorientiertes 3D-Konstruktionsmodell (s. [Rudolph u. a. 1993] und [AutoDesk 2008])

- **2D-Zeichnungsmodell:** STEP-CDS[51]: Dem Standard ISO-STEP Part 202 „Application protocol: Associative draughting“ entsprechendes Modell für 2D-Konstruktionszeichnungen (s. [Haas 1998])

Informationsaustausch: Der Informationsaustausch auf der Basis standardisierter ausgewerteter Modelle ist im Bild 2.5 dargestellt. Während das Bild 2.5a den Austausch von Bauwerksinformationen zwischen verschiedenen Fachdomänen mittels IFC darstellt, wird im Bild 2.5b der Austausch innerhalb einer Fachdomäne am Beispiel von DXF veranschaulicht.



(a) Bauwerksinformationsmodell (b) Fachdomänenmodell

Bild 2.5: Informationsaustausch auf Basis standardisierter ausgewerteter Modelle

Informationsverluste: Als grundlegendes Problem beim Austausch von Bauwerksinstanzen auf der Basis ausgewerteter Modelle wird der Informationsverlust bei der Datentransformation sowohl zwischen nativen Modellen (B_X, B_Y, B_Z) direkt als auch zwischen nativen Modellen und Standardmodellen (S) identifiziert (vgl. [Beucke 2002] und [Firmenich 2004]). Aufgrund der unterschiedlichen Informationsmengen und -strukturen werden bei der Transformation (trf) die Informationen des einen Modells nur unvollständig im anderen Modell abgebildet. Das Bild 2.6 veranschaulicht diesen Sachverhalt. Eine verlustfreie Informationsübertragung ist hier nur innerhalb der Schnittmenge $S \cap B_X \cap B_Y \cap B_Z$ möglich.

[50] Drawing Exchange Format
[51] Construction Drawing Subset, s. http://www.step-cds.de/

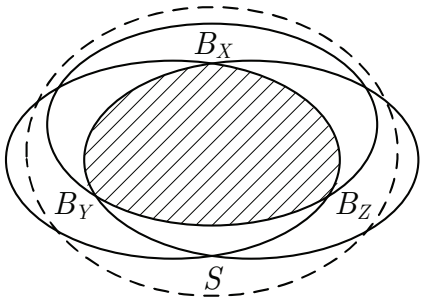


Bild 2.6: Informationsmengen von Modellen nach [Beucke 2002]

Das Bild 2.7 verdeutlicht, dass der so entstehende Informationsverlust im verteilten Planungsprozess sogar kumuliert. Die Informationsverluste werden als Differenzen von Informationsmengen beschrieben.

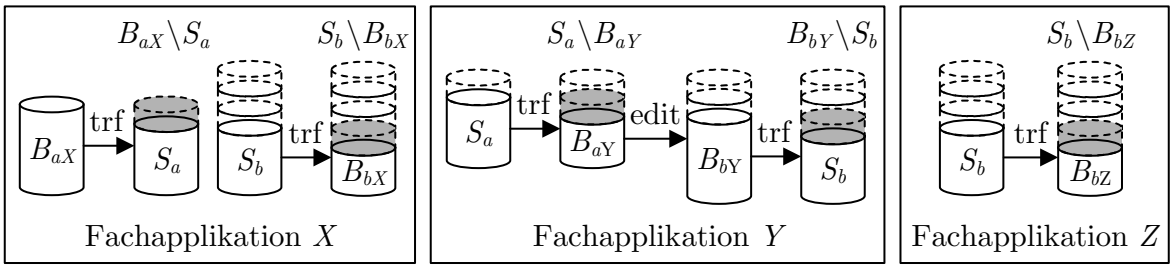


Bild 2.7: Kumulierender Informationsverlust nach [Firmenich 2004]

In der Fachapplikation X wird die Modellinstanz B in der Version B_{aX} erzeugt und auf Basis eines Standardmodells S in die Version S_a transformiert. Der dabei auftretende Informationsverlust wird mit $B_{aX} \setminus S_a$ beschrieben. Zur Weiterverarbeitung mit der Fachapplikation Y wird die Version S_a zunächst in die native Modellinstanzversion B_{aY} überführt, was wiederum mit dem Informationsverlust $S_a \setminus B_{aY}$ einhergeht. Nach der Weiterverarbeitung der Modellinstanz zur Version B_{bY} wird diese mit dem Informationsverlust $B_{bY} \setminus S_b$ ins Standardformat S_b transformiert. Die bis dahin verlorengegangenen Informationen können nicht kompensiert werden, summieren sich demnach auf. Zu Beginn der Verarbeitung der Modellinstanzversion B_{bZ} mit der Fachapplikation Z werden die kumulierenden Informationsverluste mit $(B_{aX} \setminus S_a) \cup (S_a \setminus Z_{aY}) \cup (B_{bY} \setminus S_b) \cup (S_b \setminus B_{bZ})$ beschrieben.

Beschreibt man die Informationsverluste ∇, unabhängig von einem bestimmten nativen Modell, auf Basis zustandsorientierter Informationsmengen im Standardformat S, dann ergibt sich der im Bild 2.8 auf der nächsten Seite dargestellte kumulierende Informationsverlust $\nabla_{a,c} = \nabla_{a,b} \cup \nabla_{b,c}$ beim sequentiellen Austausch von Bauwerkszuständen.

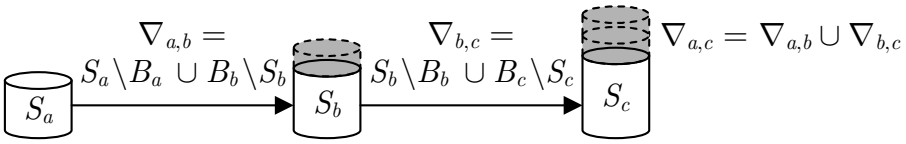


Bild 2.8: Informationsverluste in Bezug auf zustandsorientierte Informationsmengen im Standardformat S

Im Gegensatz zum standardisierten Austausch von ausgewerteten Bauwerkszuständen verwendet der vorgeschlagene verarbeitungsorientierte Modellierungsansatz nicht-ausgewertete, änderungsorientierte Informationen beim Datenaustausch. Diese Art des Informationsaustauschs ist durch nicht-kumulierende Informationsverluste gekennzeichnet.

Entwurfsabsichten: Ein weiteres Problem betrifft die Übertragbarkeit von Entwurfsabsichten[52]. Ausgewertete Modelle beschreiben lediglich den Zustand der Modellinstanz, nicht aber wie dieser im Entwurfsprozess entstanden ist. Aus diesem Grund können auf der Basis standardisierter ausgewerteter Modelle keine Entwurfsabsichten zwischen den Fachplanern übertragen werden. Im Rahmen der vorliegenden Arbeit werden die Entwurfsintentionen als Änderungen im verarbeitungsorientierten Modell berücksichtigt, beim Austausch von Bauwerksinformationen übertragen und vorteilhaft für die Archivierung, den Vergleich und die Zusammenführung eingesetzt.

Nicht-ausgewertete Modelle

Standardisiertes historienbasiertes parametrisches Modell: In [Mun u. a. 2003] und [Choi u. a. 2002] wird der Informationsaustausch auf Basis von standardisierten Modellierkommandos als makro-parametrischer Ansatz vorgeschlagen. Ausgehend von der verfügbaren Makrofähigkeit[53] von CAD-Systemen werden native CAD-Makrodateien in Standardmakros und umgekehrt übersetzt und so nicht-ausgewertete Bauwerksinstanzen ausgetauscht (s. Bild 2.9). Als Vorteil dieses Ansatzes wird die übertragbare Entwurfshistorie herausgestellt. Die Anwendbarkeit wird am Beispiel der CAD-Systeme CATIA[54] und SolidWorks[55] nachgewiesen. Die Arbeiten von [Mun u. a. 2003] und [Choi u. a. 2002] fließen zum Teil in den im folgenden Abschnitt beschriebenen ISO-STEP-Standard ein.

Abgrenzung: Der in dieser Arbeit verfolgte Ansatz greift die Idee des Austauschs standardisierter Befehle als nicht-ausgewertete Informationen auf und verallgemeinert diesen in dem neuen verarbeitungsorientierten Modell. Dieses vorgeschlagene Modell findet nicht nur beim Austausch, sondern auch bei der Versionierung, der Archivierung, beim Vergleich und bei der Zusammenführung von Bauwerksinstanzen Anwendung.

[52] engl.: design intents
[53] Funktionalität zum Aufzeichnen und Abspielen nativer Befehlen in Textform
[54] CAD/CAM/CAE-Softwaresuite von Dassault Systèmes, s. http://www.3ds.com/products/catia/catia-discovery/
[55] CAD-System für den Maschinenbau von Dassault Systèmes, s. http://www.solidworks.de/pages/products/products.html

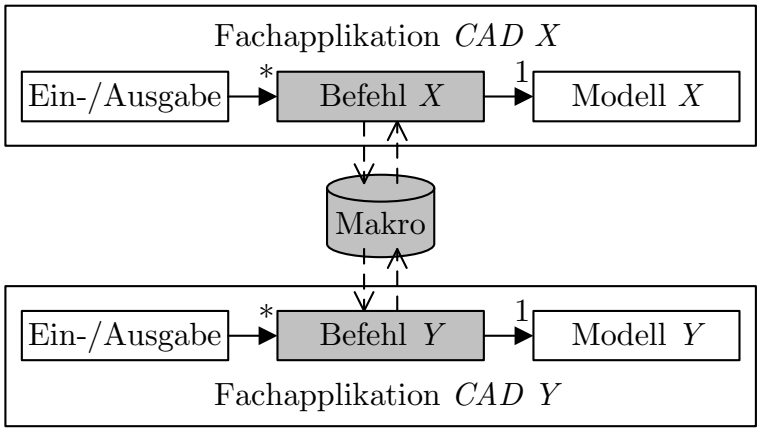


Bild 2.9: Informationsaustausch auf Basis standardisierter Makrodateien

Hybride Modelle

ISO-STEP-Standard: In [Pratt 2003], [Pratt u. a. 2005] und [Pratt u. Kim 2006] wird vorgeschlagen, sowohl die Entwurfshistorie als prozeduralen Konstruktionsvorgang als auch das Entwurfsergebnis als explizite Repräsentation in einem hybriden oder sogenannten dualen Modell zu beschreiben und für den Informationsaustausch zu verwenden. Diese Arbeiten sind in den ISO-STEP-Standard, speziell in dessen Teile [ISO 10303-55 2002], [ISO 10303-111 2003] und [ISO 10303-112 2004] eingeflossen. Das Bild 2.10 zeigt die schematische Darstellung des Informationsaustauschs auf der Basis standardisierter hybrider Modelle.

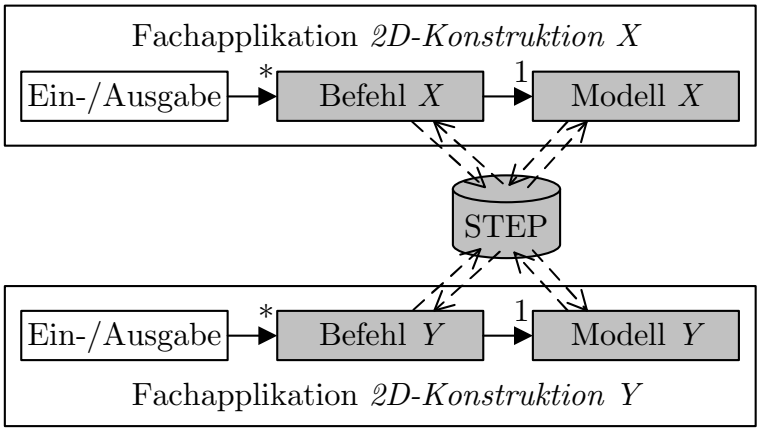


Bild 2.10: Informationsaustausch auf Basis standardisierter hybrider Modelle

Beispiel 2.1: Hybride Modellinstanz nach ISO 10303-112

Der im Bild 2.11 auf der nächsten Seite veranschaulichte Vorgang einer 2D-Konstruktion am Beispiel einer abgerundeten Ecke wird im anschließenden Listing 2.1 als hybride Modellinstanz im STEP-Format nach [ISO 10303-112 2004] dargestellt. In Zeile 2 ist

die prozedurale Sequenz der in den Zeilen 21, 22, 6, 7 und 5 definierten Operationen ersichtlich.

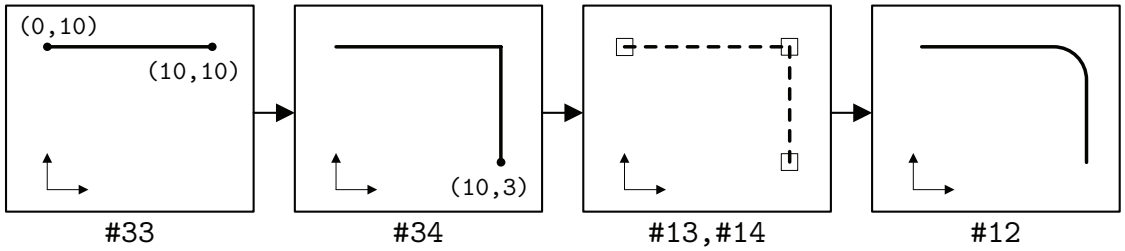


Bild 2.11: Beispiel für hybride Modellinstanz aus [ISO 10303-112 2004, S. 63 ff.]

```
...
#10=PROCEDURAL_SHAPE_REPRESENTATION_SEQUENCE('Example',
    (#33,#34,#13,#14,#12,...),$,'sketch1');
...
#12=SKETCH_CREATE_FILLET('Fillet1',#17,#18,2.,.T.);
#13=SELECTED_ITEMS_IN_PROCEDURAL_SHAPE_REP('',(#17));
#14=SELECTED_ITEMS_IN_PROCEDURAL_SHAPE_REP('',(#18));
...
#17=TRIMMED_CURVE('',#21,(#37),(#38),.T.,.CARTESIAN.);
#18=TRIMMED_CURVE('',#22,(#39),(#40),.T.,.CARTESIAN.);
#21=LINE('',#37,#25);
#22=LINE('',#39,#26);
...
...
#25=VECTOR('',#29,10.);
#26=VECTOR('',#30,7.);
...
#29=DIRECTION('',(1.,0.,0.));
#30=DIRECTION('',(0.,-1.,0.));
...
#33=SKETCH_CREATE_LINE_2POINTS('line1',#37,#38);
#34=SKETCH_CREATE_LINE_2POINTS('line2',#39,#40);
...
#37=CARTESIAN_POINT('point1',(0.,10.,0.));
#38=CARTESIAN_POINT('point2',(10.,10.,0.));
#39=CARTESIAN_POINT('point3',(10.,10.,0.));
#40=CARTESIAN_POINT('point4',(10.,3.,0.));
...
```

Listing 2.1: Hybride Modellinstanz nach ISO 10303-112

Abgrenzung: Wie auch der historienbasierte parametrische Ansatz wird der hybride Modellierungsansatz nach [ISO 10303-112 2004] nur im Hinblick auf die Anwendbarkeit beim Informationsaustausch untersucht. Darüber hinaus beschränken sich diese

Ansätze lediglich auf geometrische und topologische Bauwerksinformationen. Gegenstand der vorliegenden Arbeit ist ein allgemeingültiger hybrider Modellierungsansatz, der (a) in allen Fachdomänen der Bauwerksplanung eingesetzt werden kann und der (b) neben der Anwendung beim Austausch auch die Versionierung, die Archivierung, den Vergleich und die Zusammenführung von Bauwerksinformationen in der kooperativen Planung unterstützt.

2.3.3 Archivierung von Bauwerksinformationen

Bei der Bauwerksplanung müssen Planungsinformationen aus rechtlichen Gründen über lange Zeiträume archiviert werden. Während in den Zeiten manueller Planerstellung die Aufbewahrung von Papierplänen in Planschränken im Vordergrund stand, besteht heute die Aufgabe, digitales Planungsmaterial lesbar und interpretierbar im Sinne einer Langzeitarchivierung vorzuhalten. Dem Stand der Technik entsprechend werden Dokumentenmanagementsysteme[56] mit angeschlossenen Langzeitarchivierungssystemen eingesetzt. Im Folgenden wird die Archivierung von Bauwerksinformationen im Hinblick auf die Art und die Modellstruktur der zu archivierenden Informationen betrachtet.

Multimediaformat PDF: Wie in [Butke 2006] beschrieben, gewinnt das PDF/A[57]-Format als Dokumentenformat für die Langzeitarchivierung im Bauwesen immer mehr an Bedeutung. Auf der Basis von Texten, Bildern und Grafiken wird der Planungsstand in Form eines multimedialen, fachapplikationsunabhängigen Dokuments beschrieben und anschließend archiviert. Die direkte Weiterverarbeitung von PDF-Dokumenten in einer Fachapplikation ist im Allgemeinen nicht möglich. Im Rahmen dieser Arbeit steht die Archivierung von in Fachapplikationen verarbeitbaren Bauwerksinformationen im Vordergrund.

Ausgewertete Modelle

Standardisierte und native Modelle: Von Fachapplikationen werden standardisierte oder native Objektmodelle alphanumerisch oder binär in Dokumenten gespeichert und als Dokumentversionen in Dokumentenmanagementsystemen archiviert. Aufgrund neuer Anforderungen an die Modelle ändern diese oft ihre Struktur[58]. Dieser Umstand wirkt sich nachteilig auf die Interpretierbarkeit langzeitarchivierter Daten aus. Aktuelle Fachapplikationen können „alte“ Modellinstanzen unter Umständen überhaupt nicht mehr oder nur mit großem Konvertierungsaufwand interpretieren. Dass die Interpretierbarkeit jedoch von größter Bedeutung bei der digitalen Langzeitarchivierung ist, wird am Beispiel der ursprünglichen Intention des DXF-Standards in [Rudolph u. a. 1993, S. 38 ff.] deutlich:

[56] s. [Klingelhöller 2001] und [Gulbins u. a. 1999]

[57] Portable Dokument Format/ Archive, plattformübergreifendes Dateiformat für Dokumente von der Firma Adobe Systems, s. http://www.adobe.com/devnet/pdf/pdf_reference.html, standardisiert in ISO 19005-1 (2005)

[58] auch als Schemaevolution bezeichnet

> *„Selbst wenn es in 20 Jahren kein Programm mehr gibt, das DXF-Dateien versteht, so ist dieses Format doch so einfach und dokumentiert, dass die Inhalte nachvollziehbar sind.*[59] *“*

Nicht-ausgewertete und hybride Modelle

ISO-STEP-Standard: In [Pratt 2007] wird darauf hingewiesen, dass der Standard ISO-STEP *empfiehlt*, sowohl die Konstruktionshistorie als auch explizite Repräsentation einer Modellinstanz zu archivieren. Der Vorschlag, sowohl nicht-ausgewertete als auch ausgewertete Informationen zur Langzeitarchivierung zu speichern, wird auch in dieser Arbeit aufgegriffen.

Sprachbasierter Standard: Im vorgeschlagenen Modellierungsansatz erfolgt die Archivierung von Bauwerksinformationen zum Teil auf der Basis sprachbasiert definierter Modellieroperationen. In dieser Arbeit wird gezeigt, wie die eingeführte operative Modellierungssprache vorteilhaft zur Langzeitarchivierung von Bauwerksinformationen eingesetzt werden kann.

2.3.4 Vergleich von Planungsständen

Die im kooperativen Bauplanungsprozess entstehenden unterschiedlichen Versionen des Planungsmaterials werden miteinander verglichen, um (a) bei Planungsrevisionen Änderungen zu ermitteln, (b) im Falle von Planungsvarianten alternative Entwürfe gegenüberzustellen und (c) mögliche Planungskonflikte zu erkennen und zu visualisieren. Im Folgenden wird der Vergleich von Planungsversionen auf der Basis von ausgewerteten, nicht-ausgewerteten und hybriden Bauwerksmodellen untersucht.

Ausgewertete Modelle

Dokumentversionen: Der Vergleich von Dokumentversionen im Bauplanungsprozess ist neben herkömmlicher Standardsoftware (Textverarbeitung, Tabellenkalkulation usw.) ausschließlich aus dem Bereich von CAD-Konstruktionssoftware bekannt. In [Richter u. Beucke 2006] und [Ramunno 2005] werden die verfügbaren Werkzeuge *PlanDiffViewer* von WeltWeitBau[60] sowie *compareDWG* von Furix[61] im Hinblick auf deren Praxisrelevanz untersucht. Der PlanDiffViewer führt den Zeichnungsvergleich auf applikationsunabhängiger HP-GL[62]-Grafikebene durch, was bedeutet, dass die nativen, ursprünglichen Objektinformationen[63] beim Vergleich unberücksichtigt bleiben. Das Werkzeug compareDWG basiert hingegen auf dem direkten Vergleich von Modellobjekten der DWG-Datenbasis[64], ist aber fest an das DWG-Datenformat gebunden.

[59] Das DXF-Format in der aktuellen Version [AutoDesk 2008] ist zwar gut dokumentiert, aber dennoch sehr komplex und ohne Weiteres nicht einfach nachvollziehbar
[60] WeltWeitBau Ingenieurgesellschaft für angewandte Bauinformatik mbH, http://www.wwbau.de/
[61] Furix AutoCAD Tools, http://www.furix.com/
[62] Hewlett Packard Graphic Language, Grafikformat für Drucker bzw. Plotter
[63] z. B. Layer
[64] s. Abschnitt 2.2.2 auf Seite 15

Die Untersuchungen in [Richter u. Beucke 2006] und [Ramunno 2005] kommen zu dem Schluss, dass die verfügbaren Werkzeuge zum Vergleich von CAD-Dokumentversionen nur eingeschränkt einsetzbar sind. Die Ursachen werden hauptsächlich auf das Fehlen persistenter Objektidentifikatoren zurückgeführt.

Objektversionen: Um die Probleme beim Vergleich von Dokumentversionen zu beheben, wird in [Richter u. Beucke 2006], [Weise 2006] und [Schäfer 2006] der Vergleich auf der Basis von Objektversionen vorgeschlagen. Dieser setzt prinzipiell die Verfügbarkeit persistenter Objektidentifikatoren voraus, wenngleich auch in [Weise 2006] ein heuristisches Verfahren zur Objektidentifikation vorgestellt und für den Vergleich herangezogen wird. Im Unterschied zur Dokumentversionierung werden bei der Objektversionierung hinzugefügte, modifizierte und gelöschte Objekte gekennzeichnet, was den Aufwand beim Versionenvergleich erheblich reduziert. Dennoch ist dieser Ansatz mit folgenden Problemen verbunden. Die unterschiedlichen Zustände eines Objekts werden in den Attributwerten der entsprechenden Objektversionen gespeichert. Als Ergebnis des Vergleichs werden dem Fachplaner Unterschiede auf Basis von Objektattributen präsentiert, obwohl dem Planer weder die Attributbedeutung noch die Semantik der Unterschiede bzw. der native Modellkontext bekannt sind. Beispielsweise führt eine kleine Änderung am Planungsgegenstand zu Änderungen mehrerer Attributwerte in verschiedenen Objekten. Der Vergleich auf der Basis von Objektversionen ergibt hier mehrere einzelne Unterschiede, die kontextfrei und unabhängig voneinander betrachtet, keine verwertbare Aussagekraft besitzen. Im Rahmen dieser Arbeit wird gezeigt, wie auf der Basis des vorgeschlagenen verarbeitungsorientierten Modellierungsansatzes diese fehlende Änderungssemantik abgebildet und vorteilhaft für den Versionenvergleich und die Unterschiedpräsentation herangezogen werden kann.

Nicht-ausgewertete und hybride Modelle

Bauplanungsprozess: Der Vergleich von Planungsständen auf der Grundlage nicht-ausgewerteter oder hybrider Modelle im Bauwesen ist derzeit nicht bekannt und wird in der vorliegenden Arbeit als neuartiger Ansatz untersucht.

2.3.5 Zusammenführung von Planungsständen

Das im kooperativen Planungsprozess verarbeitete versionierte Planungsmaterial beinhaltet neben Revisionen auch Planungsvarianten und -konflikte. Ziel der Zusammenführung sind das Verschmelzen von Planungsvarianten und das notwendige Auflösen von Planungskonflikten, indem nach einem Vergleich in der Regel zwei Versionen zu einer neuen, gültigen Version zusammengeführt werden. Im Folgenden wird das Zusammenführen von versionierten Bauwerksinformationen auf Grundlage ausgewerteter, nicht-ausgewerteter und hybrider Bauwerksmodelle vorgestellt.

Ausgewertete Modelle

Dokumentversionen: Wie in [Richter u. Beucke 2006] und [Ramunno 2005] dargelegt, existieren neben in Standardsoftware implementierter Funktionalität derzeit keine

Werkzeuge zur Unterstützung der Zusammenführung von Dokumentversionen im Bauplanungsprozess. Wie oben beschrieben, widmen sich verfügbare Werkzeuge lediglich dem Dokumentenvergleich. Erste Ansätze zur Zusammenführung sind dahingehend zu erkennen, dass beispielsweise ermittelte Unterschiede als separate Dokumente gespeichert und so eventuell für eine Zusammenführung genutzt werden können.

Objektversionen: In [Beer 2006], [Richter u. Beucke 2006] und [Schäfer 2006] wird neben dem Vergleich auch die Zusammenführung von Bauwerksinformationen auf der Basis von Objektversionen vorgeschlagen. Analog zum Vergleich erfolgt die Zusammenführung von Planungsständen durch das Betrachten der die Zustände beschreibenden Objektattribute. Im Falle der Zusammenführung ist nicht nur der lesende, sondern der weitaus problematischere schreibende Zugriff auf die Objektattribute notwendig. Sind, wie in vielen nativen Objektmodellen, keine geeigneten Zugriffsmethoden dafür vorgesehen, dann steht dieser Ansatz im Widerspruch mit dem objektorientierten Prinzip der Datenkapselung[65] und die Datenkonsistenz ist nicht nur auf Modellinstanzebene, sondern sogar auf Objektebene gefährdet. Die Zusammenführung auf der Basis von einzelnen Objektattributen ist für den Fachplaner aufgrund der fehlenden Attribut- und Unterschiedsemantik ohne Weiteres nicht möglich. Zusätzlich stellt die in Objektzuständen nicht abbildbare, jedoch für die Zusammenführung notwendige Änderungssemantik einen Mangel dar, weil Entwurfsintentionen nicht berücksichtigt werden können.

Nicht-ausgewertete und hybride Modelle

Objektänderungen: Das derzeit einzige bekannte Zusammenführungsverfahren von versionierten Bauwerksinformationen auf Basis von Änderungen ist in [Weise 2006] beschrieben. Die auf Grundlage von Planungsversionen berechneten Objektänderungen werden im Zusammenführungsprozess wiederverwendet, indem diese in Kombination angewendet oder bei Konflikten einzeln übernommen oder verworfen werden. Im Unterschied zur vorliegenden Arbeit wird die Änderungssemantik im Objektkontext definiert. Der fehlende Modellkontext beim Zusammenführen kann auch hier zu inkonsistenten Modellzuständen führen. Darüber hinaus werden Objektänderungen zur Abbildung von fachlichen Entwurfsintentionen als unzureichend erachtet. In der vorliegenden Arbeit werden Modellieroperationen auf Modellebene definiert, um diese zur Wiederverwendung von Entwurfsabsichten und zur Konsistenzsicherung während der Zusammenführung heranzuziehen.

[65] s. Abschnitt 2.2.1 auf Seite 14

3 Modellbildung

„Der menschliche Geist ist unfähig, anders als in Modellen zu denken.“
Christian Wissel, aus [Wissel 1989]

Dieses Kapitel stellt den verarbeitungsorientierte Modellierungsansatz vor. Aufbauend auf den Ansätzen der Objekt-, der Versions- und der Änderungsorientierung wird der vorgeschlagene Modellierungsansatz als ganzheitliche Betrachtung entwickelt und an einem begleitenden Beispiel erläutert. Mit der Absicht, das verarbeitungsorientierte Modell dauerhaft und formal zu beschreiben, wird eine mathematische Formulierung auf Basis der Mengenlehre und der Relationenalgebra vorgestellt. Auf Grundlage des entwickelten Modells wird die Bedeutung von Modellieroperationen im Kontext der Bauwerksmodellierung herausgestellt.

3.1 Modellierungsansatz

Gegenstand des vorgeschlagenen *verarbeitungsorientierten Modellierungsansatzes* ist nicht die Beschreibung eines Objektmodells einer Fachapplikation als *Ergebnis* der Planung (*objektorientierter Ansatz*), sondern die Beschreibung der Modellverarbeitung mit einer Fachapplikation als *Prozess* der Planung. Dieser Prozess wird nicht wie im herkömmlichen Sinne durch in Nachfolgerbeziehung stehende Versionen (*versionsorientierter Ansatz*) allein, sondern zusätzlich durch die im Verarbeitungsprozess entstehenden Änderungen (*änderungsorientierter Ansatz*) beschrieben.

Begleitendes Beispiel: In den folgenden Abschnitten veranschaulicht ein kleines Beispiel den vorgeschlagenen Ansatz. Dazu wird angenommen, dass eine fachliche Bauwerksinstanz aus den Elementen *Wand* x, *Last* y und *Öffnung* z besteht (s. Bild 3.1). Eine fiktive Fachapplikation dient der Dimensionierung und der Bemessung der Wand in Bezug auf die Last und die Öffnung.

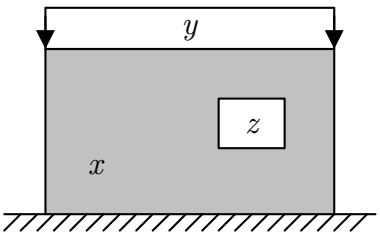


Bild 3.1: Beispielhafte fachliche Instanz

3.1.1 Objektorientierter Ansatz

Objekt: Dem objektorientierten Modellierungsansatz folgend werden die Elemente der fachlichen Instanz durch Objekte abstrahiert. Der Zustand der Objekte wird durch ihre Attributwerte definiert, ihr Verhalten wird durch die Methoden beschrieben.

Beispiel 3.1: Objekte

Die Elemente der fachlichen Instanz werden durch die im Bild 3.2 dargestellten Objekte x, y und z abstrahiert. Der Zustand der Öffung ist beispielsweise durch ihre Abmessungen bestimmt. Der Zustand der Wand wird durch ihre Abmessungen und die Position der Öffnung beschrieben und der Zustand der Last ist durch ihren Wert definiert.

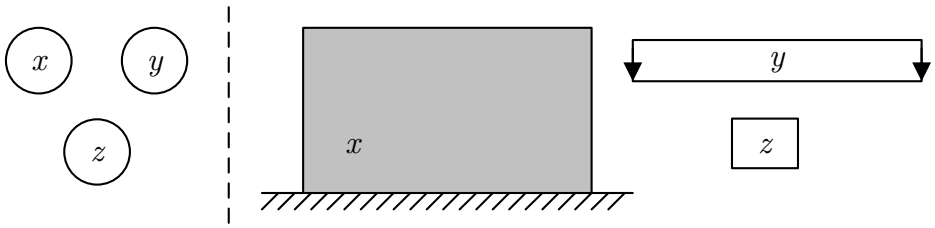


Bild 3.2: Objekte als abstrahierte Elemente der fachlichen Instanz

Objektorientiertes Modell: Gleichartige Objekte werden in Klassen zusammengefasst und strukturiert. Ein objektorientiertes Modell ist eine strukturierte Menge von Klassen zur Beschreibung eines vollständigen fachlichen Modells. Die Beziehungen zwischen Elementen des fachlichen Modells werden auf das objektorientierte Modell übertragen, indem die Klassen Beziehungen zwischen Objekten definieren.

Beispiel 3.2: Objektorientiertes Modell

Die Objekte x, y und z sind unterschiedlichen Typs und werden jeweils durch ihre Klasse *Wand*, *Last* bzw. *Öffnung* beschrieben. Das Bild 3.3 veranschaulicht das objektorientierte Modell als UML-Klassendiagramm. Die Beziehungen zwischen Wand und Last und zwischen Wand und Öffnung sind durch gerichtete, offene Assoziationspfeile dargestellt.

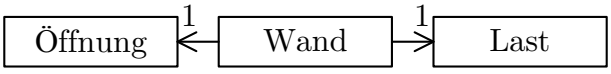


Bild 3.3: Objektorientiertes Modell zur Abstraktion des fachlichen Modells

Modellinstanz: Eine strukturierte Menge von Objekten heißt Instanz des objektorientierten Modells oder Modellinstanz und wird mit einer Fachapplikation erzeugt und verarbeitet. In einer Modellinstanz haben alle Objekte einen Zustand. Der Zustand der Modellinstanz wird als Gesamtheit der einzelnen Objektzustände verstanden.

Beispiel 3.3: Modellinstanz

Die Modellinstanz B besteht aus den Objekten x, y und z und wird durch das im Bild

3.3 dargestellte objektorientierte Modell definiert. Das Bild 3.4 zeigt die Modellinstanz zur Abstraktion der fachlichen Instanz.

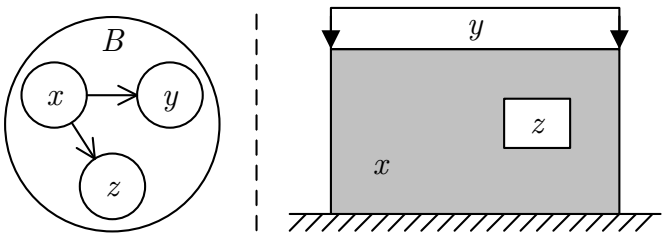


Bild 3.4: Modellinstanz als abstrahierte fachliche Instanz

Zusammenfassung: Beim objektorientierten Ansatz wird das fachliche Modell durch ein objektorientiertes Modell abstrahiert. Elemente des fachlichen Modells werden durch Klassen abgebildet. Eine Fachapplikation instanziiert das objektorientierte Modell zur Modellinstanz. Die Modellinstanz stellt eine strukturierte Menge von Objekten dar. Während der Verarbeitung der Modellinstanz werden Objekte erzeugt, von einem Zustand in den nächsten überführt und gelöscht.

3.1.2 Versionsorientierter Ansatz

Verarbeitung: Bei der Verarbeitung einer Modellinstanz mit einer Fachapplikation wird der Zustand von Objekten geändert, was gleichzeitig auch eine Zustandsänderung der Modellinstanz bedeutet. Beim versionsorientierten Ansatz steht als Versionsmodell der Zustand der Modellinstanz im Mittelpunkt der Betrachtung.

Objektversionen: Eine Objektversion beschreibt das Objekt in einem ausgezeichneten Zustand. Da ein Objekt mehrere Zustände annehmen kann, werden mehrere Objektversionen einem Objekt zugeordnet. Der Zustand eines Objekts wird in der Regel durch die Methoden des Objekts geändert.

Virtuelle Objektversionen: In [Firmenich 2002] werden virtuelle Objektversionen als besondere Objektversionen identifiziert. Während die Ursprungsobjektversion den virtuellen Objektzustand vor der Instanziierung definiert, beschreiben die gelöschten Objektversionen den virtuellen Objektzustand nach dem Entfernen einer Objektversion.

Objektversionsgraph: Im Objektversionsgraphen werden nach [Firmenich 2002] die Änderungsbeziehungen zwischen einzelnen Objektversionen eines Objekts veranschaulicht. Die Knoten des Graphen sind die Objektversionen, die Kanten bilden die Zustandsänderungen des Objekts ab.

Beispiel 3.4: Objektversionen

Dem Objekt z, welches die Öffnung beschreibt, sind die Objektversionen z_β, z_0, z_1, z_2 und z_3 zugeordnet. Dabei ist z_β die Ursprungsobjektversion. Das Bild 3.5 auf der nächsten Seite veranschaulicht den Objektversionsgraphen des Objekts z und die entsprechenden Objektzustände 0, 1, 2 und 3.

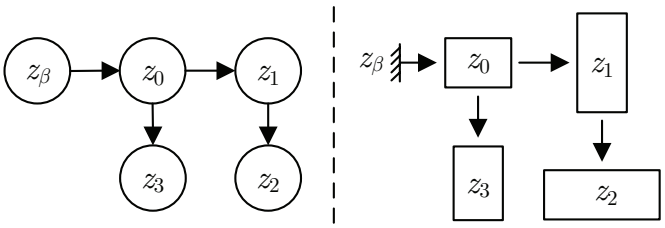


Bild 3.5: Objektversionen

Modellinstanzversion: Eine strukturierte Menge von Objektversionen heißt Modellinstanzversion. Eine Modellinstanzversion beschreibt die Modellinstanz in einem ausgezeichneten Zustand. Dieser Zustand ist durch die Gesamtheit der einzelnen Objektzustände – der Objektversionen – definiert. In einer Modellinstanzversion darf pro Objekt maximal eine ihm zugeordnete Objektversion enthalten sein.

Beispiel 3.5: Modellinstanzversion

Die Modellinstanzversion B_b besteht aus den Objektversionen x_1, y_0 und z_1. Das Bild 3.6 zeigt die Modellinstanzversion zur Abstraktion der fachlichen Instanz in dem ausgezeichneten Zustand b.

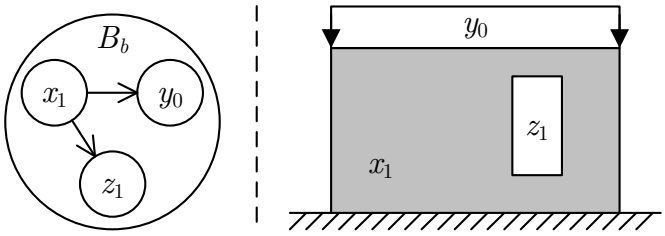


Bild 3.6: Modellinstanzversion

Versionsgraph: Im versionsorientierten Ansatz wird der Versionsgraph zur Abbildung der Nachfolgerbeziehungen zwischen den Versionen einer Modellinstanz eingeführt. Die Knoten des Graphen sind die einzelnen Modellinstanzversionen, die durchgezogenen Kanten bilden die einzelnen Nachfolgerbeziehungen ab. Für eine Modellinstanz existiert genau ein Versionsgraph.

Beispiel 3.6: Vereinfachter Versionsgraph

Der Modellinstanz B sind die Modellinstanzversionen B_a, B_b und B_c zugeordnet. Das Bild 3.7 veranschaulicht einen Auszug aus dem Versionsgraphen von B und die entsprechenden Zustände der fachlichen Instanz.

Revision und Variante: Beim versionsorientierten Ansatz bietet der Versionsgraph die Möglichkeit, Revisions- und Variantenbeziehungen zwischen einzelnen Modellinstanzversionen zu erkennen. Zwischen zwei Modellinstanzversionen besteht eine Revisionsbeziehung, wenn eine Version durch Nachfolgerbeziehungen im Versionsgraph aus einer anderen Version hervorgegangen ist. Die hervorgegangene Modellinstanzversion

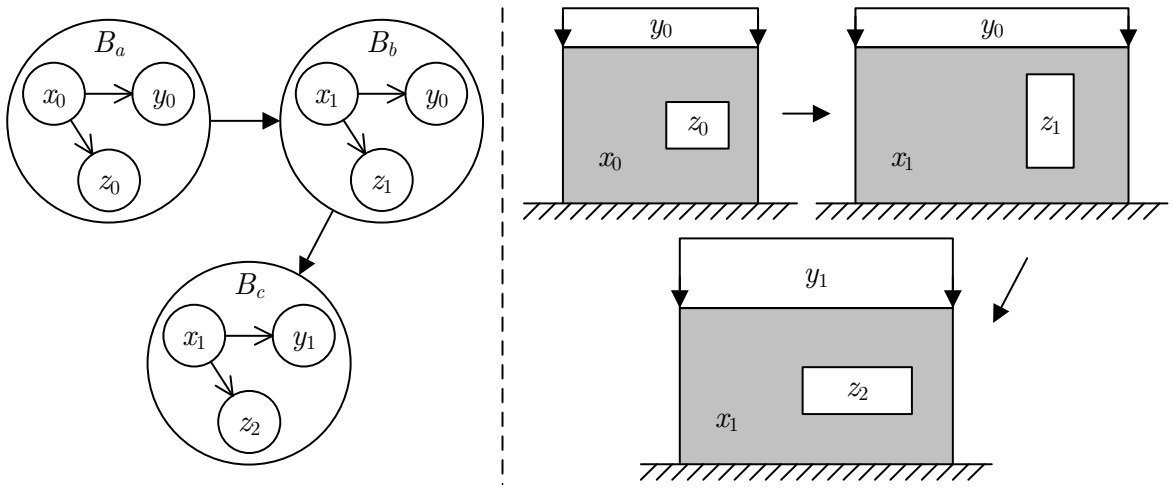


Bild 3.7: Auszug aus dem Versionsgraphen der Modellinstanz

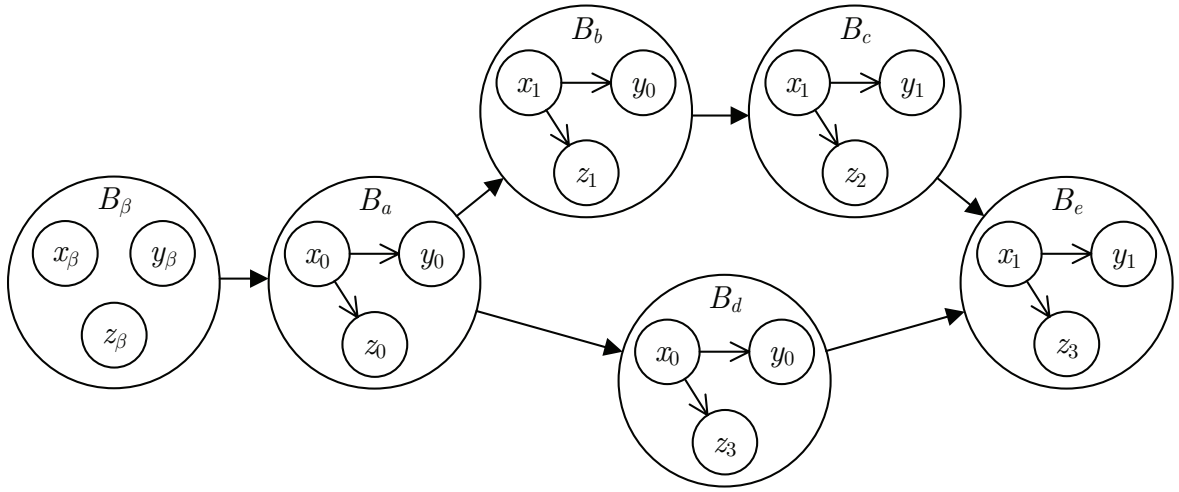
wird dann als Revision der anderen Modellinstanzversion bezeichnet. Zwischen zwei Versionen besteht eine Variantenbeziehung, wenn zwei Modellinstanzversionen nicht jeweils auseinander hervorgegangen, aber dennoch aus einer gemeinsamen Vorgängerversion entstanden sind. Die gemeinsame Vorgängerversion heißt Ursprungsversion, die entstandenen Modellinstanzversionen werden als Varianten oder Alternativen bezeichnet.

Zusammenführung: Die im kooperativen Planungsprozess entstehenden Modellinstanzversionen bilden mitunter einzelne Teillösungen oder Varianten innerhalb der gesamten Planungsaufgabe ab. Enthalten die Teillösungen oder Varianten wichtige Informationen zur Beschreibung der Gesamtlösung, dann müssen einzelne Modellinstanzversionen in einem Abstimmungsprozess zu einer Gesamtversion zusammengeführt werden. Eine Zusammenführung ist im Versionsgraphen dadurch zu erkennen, dass die zusammengeführte Version mehr als eine Vorgängerversion aufweist bzw. die zusammenzuführenden Versionen eine gemeinsame Nachfolgerversion haben.

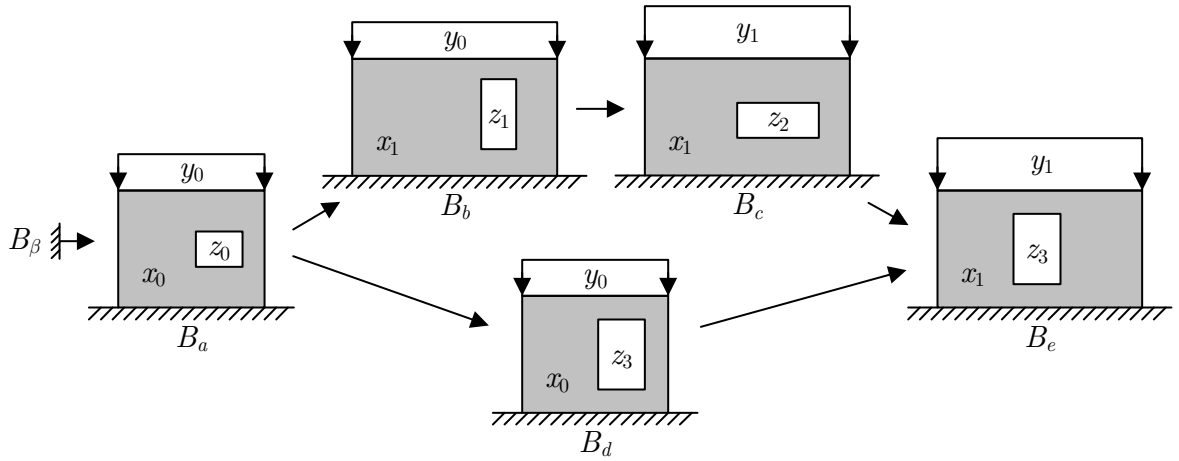
Virtuelle Modellinstanzversion: Eine Modellinstanzversion, die keine Vorgängerversion hat, heißt leere oder virtuelle Modellinstanzversion. Diese Modellinstanzversion ist virtuell, weil sie die virtuellen Ursprungsobjektversionen enthält. Die virtuelle Modellinstanzversion beschreibt den Zustand der Modellinstanz vor deren Instanziierung in einer Fachapplikation, wird aber formal als Knoten im Versionsgraphen dargestellt.

Beispiel 3.7: Versionsgraph mit virtueller Modellinstanzversion, Revision, Variante und Zusammenführung

Im Bild 3.8 auf der nächsten Seite sind der vollständige Versionsgraph der Modellinstanz B (3.8a) und die entsprechende Entwicklung der fachlichen Instanz (3.8b) dargestellt. Die virtuelle Modellinstanzversion ist B_β. Die Version B_c ist gleichzeitig eine Revision von B_b und eine Variante zu B_d. Die Varianten B_c und B_d werden zur Modellinstanzversion B_e zusammengeführt, indem der Zustand 1 der Wand x und der Zustand 1 der Last y aus der Version B_c und der Zustand 3 der Öffnung z aus der Modellinstanzversion B_d den Zustand der Version B_e bilden.



(a) Versionsgraph der Modellinstanz



(b) Entwicklung der fachlichen Instanz

Bild 3.8: Versionsgraph der Modellinstanz und Entwicklung der fachlichen Instanz

Zusammenfassung: Beim versionsorientierten Ansatz wird als Erweiterung des objektorientierten Ansatzes die Modellinstanz versioniert, indem Objektversionen eingeführt werden. Eine Objektversion beschreibt ein Objekt in einem ausgezeichneten Zustand. Eine strukturierte Menge von Objektversionen wiederum stellt einen ausgezeichneten Zustand der Modellinstanz – die Modellinstanzversion – dar. Die Nachfolgerbeziehungen zwischen einzelnen Modellinstanzversionen, Revisionen, Varianten und Zusammenführungen werden im Versionsgraphen der Modellinstanz veranschaulicht. Bei Zusammenführungen können im Versionsgraphen sowohl die zusammenzuführenden als auch die zusammengeführten Versionen identifiziert werden. Der Zusammenführungsvorgang selbst ist nicht abbildbar.

3.1.3 Änderungsorientierter Ansatz

Verarbeitung: Beim änderungsorientierten Ansatz steht weniger der Zustand der zu verarbeitenden Modellinstanz, sondern vielmehr der Prozess der Änderung der Modellinstanz im Mittelpunkt der Betrachtung. Eine Fachapplikation verarbeitet die Modellinstanz, indem sie eine Modellinstanzversion durch eine fachtypische Änderung in die nächste Modellinstanzversion überführt.

Operation: Die kleinste Verarbeitungseinheit wird Operation genannt. Eine Operation definiert einen fachtypischen Verarbeitungsschritt, der für eine ausgewählte Menge von Objekten die Zustände der Objekte in die jeweils nachfolgenden Objektzustände überführt. Wie diese Überführung innerhalb der Objekte umgesetzt wird, ist Gegenstand des Abschnitts 3.3 in diesem Kapitel.

Beispiel 3.8: Operationen

Bezüglich der beispielhaften fachlichen Instanz sind aufgeführte Operationen als Verarbeitungsschritte vorstellbar:

- *Wand erzeugen*
- *Wandabmessungen ändern*
- *Wand bemessen*
- *Last erzeugen*
- *Lastwert ändern*
- *Öffnung erzeugen*
- *Öffnungsabmessungen ändern*
- *Öffnung verschieben*

Operatives Modell: Während das objektorientierte Modell ein fachliches Modell als Zustand beschreibt, abstrahiert das operative Modell die Verarbeitung. Das operative Modell besteht aus einer Menge von fachtypischen Operationen zur umfassenden Verarbeitung einer Modellinstanz in einer Fachapplikation.

Beispiel 3.9: Operatives Modell

Die Gesamtheit der im Beispiel 3.8 aufgelisteten Operationen bildet das operative Modell zur Verarbeitung des im Beispiel 3.2 auf Seite 36 dargestellten objektorientierten Modells.

Operationsinstanz: Die in einer Fachapplikation instanziierten Operationen heißen Operationsinstanzen. Sie stellen konkrete Verarbeitungsschritte zur Laufzeit dar und operieren auf der Modellinstanz.

Beispiel 3.10: Operationsinstanzen

In der Tabelle 3.1 auf der nächsten Seite sind für die Operationen des operativen Modells exemplarische Operationsinstanzen mit Identifikatoren (ID) angegeben.

Operation	Operationsinstanz	ID
Wand erzeugen	*Wand x mit Höhe 3 m und Länge 4 m erzeugen*	$o1$
Wandabmessungen ändern	*Wand x auf Länge 6 m ändern*	$o2$
Wand bemessen	*Wand x bemessen*	$o3$
Last erzeugen	*Last y mit Wert 5 kN/m erzeugen*	$o4$
Lastwert ändern	*Last y auf Wert 10 kN/m ändern*	$o5$
Öffnung erzeugen	*Öffnung z mit Höhe 0.8 m, Breite 1.2 m und Position (2 m, 1 m) erzeugen*	$o6$
Öffnungsabmessungen ändern	*Öffnung z auf Höhe 2 m und Breite 0.8 m ändern*	$o7$
	Öffnung z auf Höhe 0.8 m und Breite 2 m ändern	$o8$
	Öffnung z auf Höhe 2 m und Breite 1.2 m ändern	$o9$
Öffnung verschieben	*Öffnung z um Vektor (2 m, -0.2 m) verschieben*	$o10$
	Öffnung z um Vektor (-2 m, 0.2 m) verschieben	$o11$
	Öffnung z um Vektor (0 m, -0.2 m) verschieben	$o12$

Tabelle 3.1: Operationen und Operationsinstanzen

Änderung: Eine Folge von Operationsinstanzen wird Änderung δ genannt. Eine Änderung beschreibt die Überführung der Modellinstanz von einer Version in eine andere Version.

Beispiel 3.11: Änderung

Das Bild 3.9 verdeutlicht die Änderung $\delta_{a,b}$ der Version B_a zur Version B_b. Diese Änderung wird durch die Folge der Operationsinstanzen $\langle o2, o7, o10\rangle$ beschrieben (vgl. Tabelle 3.1).

Operative Modellinstanzversion: Im änderungsorientierten Ansatz wird eine Modellinstanzversion auf Basis von Änderungen operativ beschrieben. Im Gegensatz zum versionsorientierten Ansatz, bei dem eine Version als Zustand der Modellinstanz ausgedrückt wird, heißt eine von der virtuellen Version ausgehende Folge von Änderungen operative Modellinstanzversion.

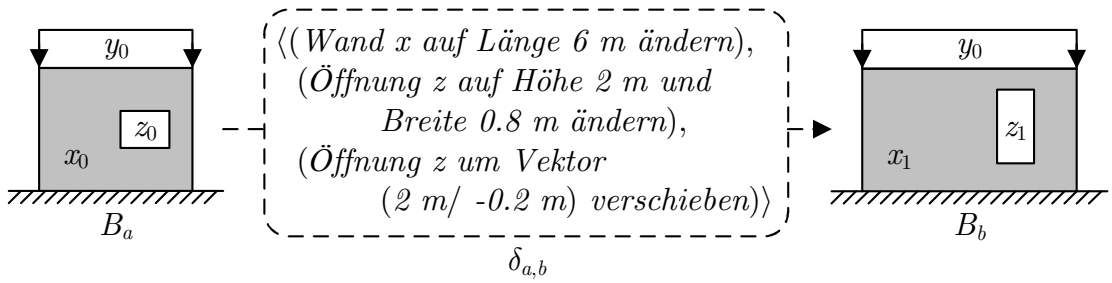


Bild 3.9: Änderung der Modellinstanz

Beispiel 3.12: Operative Modellinstanzversionen
Die operative Modellinstanzversion $M(B_c)$ wird durch die Folge von Änderungen $\langle \delta_{\beta,a}, \delta_{a,b}, \delta_{b,c} \rangle$ beschrieben. Die Änderungssequenz $\langle \delta_{\beta,a}, \delta_{a,d} \rangle$ definiert die operative Modellinstanzversion $M(B_d)$. Das Bild 3.10 verdeutlicht diesen Sachverhalt unter Verwendung der Operationsinstanzen der Tabelle 3.1 auf Seite 42.

$$\delta_{\beta,a} = \langle o1, o4, o6 \rangle,\ \delta_{a,b} = \langle o2, o7, o10 \rangle$$
$$\delta_{b,c} = \langle o5, o8, o11 \rangle,\ \delta_{a,d} = \langle o9, o12 \rangle$$

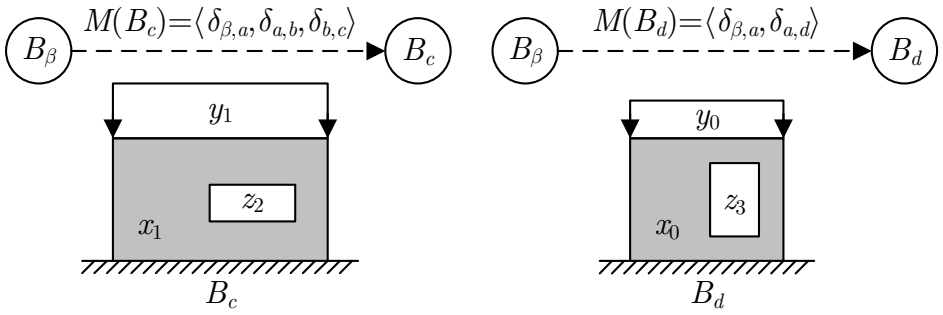


Bild 3.10: Operative Modellinstanzversionen

Änderungsbaum: Im änderungsorientierten Ansatz wird ein Graph zur Abbildung der Änderungsbeziehungen zwischen den Versionen einer Modellinstanz eingeführt. Die Knoten des Graphen sind die einzelnen Modellinstanzversionen, die gestrichelten Kanten bilden die einzelnen Änderungen ab. Jede Version ist durch genau eine Änderung aus einer anderen Version hervorgegangen. Diese Eigenschaft führt zu dem Schluss, dass der betrachtete Graph ein Baum ist. Er wird im Folgenden als Änderungsbaum bezeichnet. Jeder Knoten im Änderungsbaum hat im Gegensatz zum Versionsgraphen genau eine Eingangskante. Für eine Modellinstanz existiert genau ein Änderungsbaum.

Zusammenführung: Dem änderungsorientierten Ansatz folgend wird im Zusammenführungsprozess genau eine Ausgangsversion durch Änderung in eine neue Modellinstanzversion überführt. Eine Zusammenführung wird im Änderungsbaum demnach durch eine Änderung beschrieben. Aus diesem Grund sind im Änderungsbaum die

zusammenzuführenden Versionen und die zusammengeführte Version nicht identifizierbar. Jedoch bietet der änderungsorientierte Ansatz die Möglichkeit, den Zusammenführungsprozess auf Basis der in den zusammenzuführenden operativen Modellinstanzversionen gespeicherten Änderungen durchzuführen. Das bedeutet, die Änderungen, die zu den zusammenzuführenden Zuständen der Modellinstanz geführt haben, können im Zusammenführungsprozess als Verarbeitungsschritte wiederverwendet werden.

Beispiel 3.13: Änderungsbaum und Zusammenführung

Für die im Bild 3.8a des Beispiels 3.7 auf Seite 40 dargestellten Modellinstanzversionen B_β, B_a, B_b, B_c, B_d und B_e der Modellinstanz B wird im Bild 3.11 der Änderungsbaum mit entsprechenden Änderungen dargestellt. Die Zusammenführung der Varianten B_c und B_d zur Modellinstanzversion B_e ist im Änderungsbaum der Modellinstanz B durch die Änderung $\delta_{a,e}$ beschrieben. Diese Zusammenführung ist dadurch gekennzeichnet, dass die Last im Zustand y_1 aus der Version B_c und die Öffnung im Zustand z_3 aus der Version B_d in die Version B_e zu übernehmen sind. Diejenigen Änderungen, die zu diesen Zuständen führen, bilden die Basis für die neue Änderung $\delta_{a,e}$. Auf Grundlage der in den operativen Modellinstanzversionen $M(B_c)$ und $M(B_d)$ gespeicherten Operationsinstanzen $o2, o5$ bzw. $o12$ wird die neue Änderung $\delta_{a,e}$ erzeugt.

$$\delta_{a,e} \quad = \quad \langle o2, o5, o12 \rangle$$

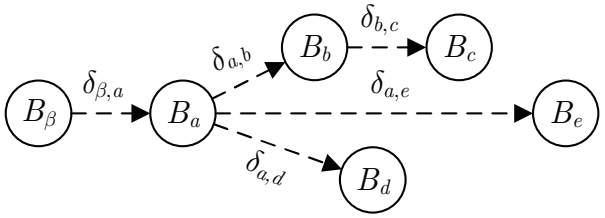


Bild 3.11: Änderungsbaum der Modellinstanz

Zusammenfassung: Beim änderungsorientierten Ansatz wird im Gegensatz zum versionsorientierten Ansatz die Verarbeitung der Modellinstanz als Sequenz von Änderungen gespeichert. Eine Änderung überführt eine Modellinstanzversion in eine andere Modellinstanzversion und wird dabei als Folge von Verarbeitungsschritten – den Operationsinstanzen – formuliert. Eine Folge von Änderungen, die auf der virtuellen Version operiert, heißt operative Modellinstanzversion. Eine Operationsinstanz stellt eine konkrete Ausprägung einer Operation dar. Die Menge aller verfügbaren fachtypischen Operationen abstrahiert die Verarbeitung der Modellinstanz und wird operatives Modell genannt. Die Änderungsbeziehungen zwischen einzelnen Versionen werden im Änderungsbaum der Modellinstanz veranschaulicht. Im Unterschied zum Versionsgraphen können zusammenzuführende und zusammengeführte Versionen im Änderungsbaum nicht identifiziert werden. Dennoch ist die Wiederverwendung gespeicherter Änderungen im Zusammenführungsprozess von Vorteil. Die Zusammenführung selbst wird als Vorgang wiederum durch eine Änderung beschrieben.

3.1.4 Verarbeitungsorientierter Ansatz

Verarbeitung: Beim verarbeitungsorientierten Ansatz ist, wie der Name bereits angibt, die ganzheitliche Verarbeitung der Modellinstanz Gegenstand der Betrachtung. Diese Verarbeitung wird als Kombination des versionsorientierten und des änderungsorientierten Ansatzes modelliert, sodass Informationen sowohl über die Zustände der Modellinstanz – die Versionen – als auch über die Verarbeitungsschritte der Modellinstanz – die Änderungen – abgebildet werden können.

Verarbeitungsgraph: Für eine Modellinstanz werden die Versionen, die Nachfolgerbeziehungen aus dem Versionsgraphen und die Änderungsbeziehungen aus dem Änderungsbaum im Verarbeitungsgraphen dargestellt. Die Knoten des Graphen sind die Versionen, eine Kantenmenge bildet die Nachfolgerbeziehungen ab und eine weitere Kantenmenge beschreibt die Änderungsbeziehungen zwischen den einzelnen Modellinstanzversionen.

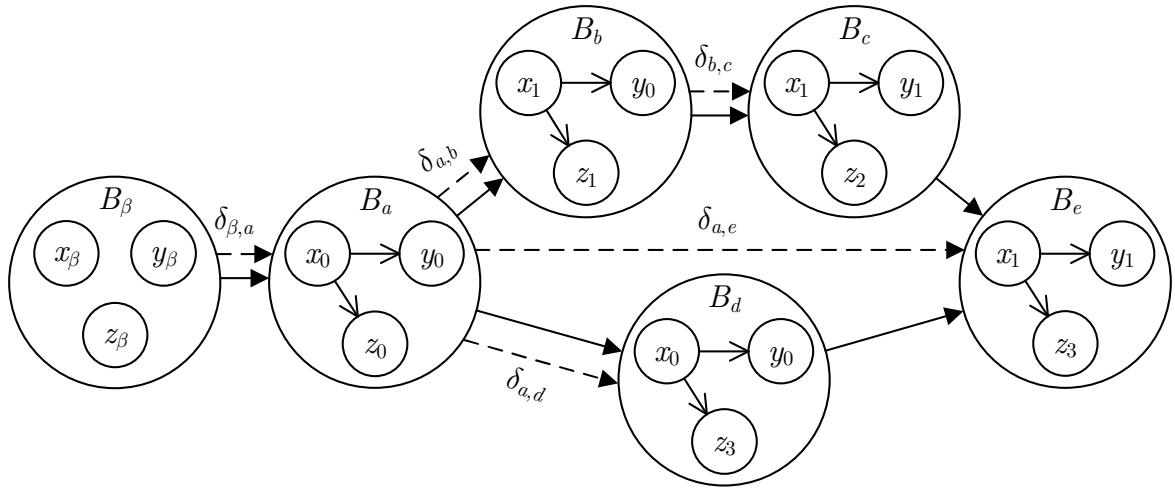
Zusammenführung: Im verarbeitungsorientierten Ansatz ist eine Zusammenführung von Modellinstanzversionen im Verarbeitungsgraphen dadurch gekennzeichnet, dass sowohl die zusammenzuführenden und zusammengeführten Versionen identifizierbar sind, aber auch die im Zusammenführungsprozess ausgeführten Änderungen gespeichert werden.

Beispiel 3.14: Verarbeitungsgraph

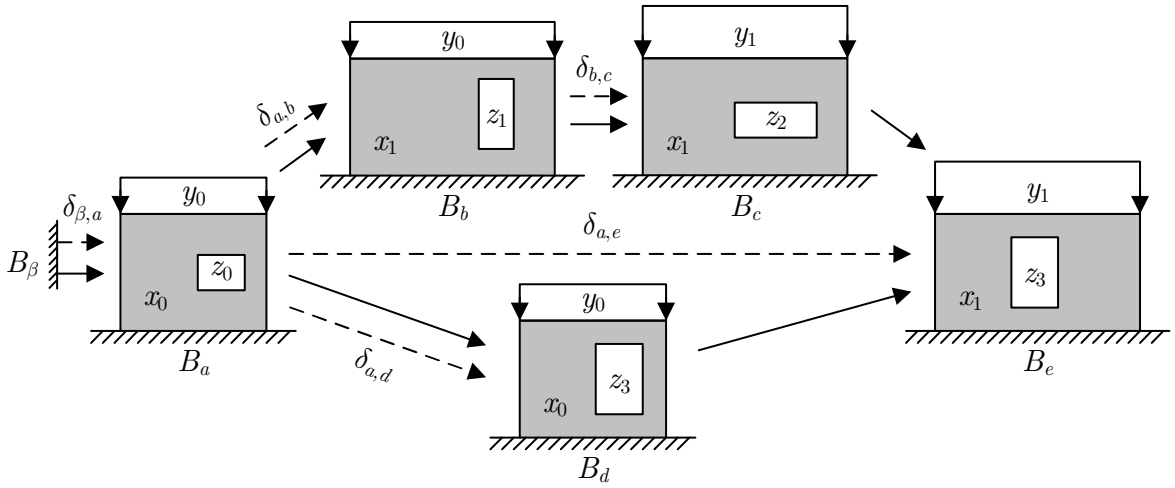
Im Bild 3.12 auf der nächsten Seite wird die Verarbeitung der fachlichen Instanz (3.12b) durch den Verarbeitungsgraphen der Modellinstanz B (3.12a) dargestellt. Es werden die einzelnen Modellinstanzversionen und sowohl ihre Nachfolgerbeziehungen als auch Änderungsbeziehungen veranschaulicht. Die die Änderungen beschreibenden Operationsinstanzen sind der Tabelle 3.1 auf Seite 42 zu entnehmen.

$$\delta_{\beta,a} = \langle o1, o4, o6 \rangle,\ \delta_{a,b} = \langle o2, o7, o10 \rangle$$
$$\delta_{b,c} = \langle o5, o8, o11 \rangle,\ \delta_{a,d} = \langle o9, o12 \rangle$$
$$\delta_{a,e} = \langle o2, o5, o12 \rangle$$

Zusammenfassung: Beim verarbeitungsorientierten Ansatz steht die Verarbeitung der Modellinstanz im Vordergrund. Als Kombination des versionsorientierten und des änderungsorientierten Ansatzes werden die Modellinstanzzustände als Versionen und die Verarbeitungsschritte als Änderungen gespeichert. Im Hinblick auf den Zusammenführungsprozess werden die jeweiligen Möglichkeiten dieser beiden Ansätze dazu genutzt, die an der Zusammenführung beteiligten Versionen und Änderungen abzubilden.



(a) Verarbeitungsgraph der Modellinstanz



(b) Entwicklung der fachlichen Instanz

Bild 3.12: Verarbeitungsgraph der Modellinstanz und Entwicklung der fachlichen Instanz

Verarbeitungsorientierung: Das Bild 3.13 verdeutlicht die Beziehungen und Zusammenhänge zwischen dem objektorientierten, dem versionsorientierten und dem änderungsorientierten Ansatz. Die auf Basis der Objektorientierung beschriebenen Informationen werden mit Hilfe der Versionsorientierung versioniert. Der änderungsorientierte Ansatz modelliert die Verarbeitung dieser Bauwerksinformationen auf der Grundlage von Änderungen. Die ganzheitliche Betrachtung dieser Ansätze wird in dieser Arbeit als *verarbeitungsorientierte Modellierung* verstanden. In der Versionsorientierung wird das Versionieren des Modellzustandes betrachtet, in der Änderungsorientierung steht das Modellverhalten in Form von Änderungen im Vordergrund. Die Stellung der Verarbeitungsorientierung wird durch die im Bild 3.14 dargestellte Matrix der Modellierungsansätze deutlich.

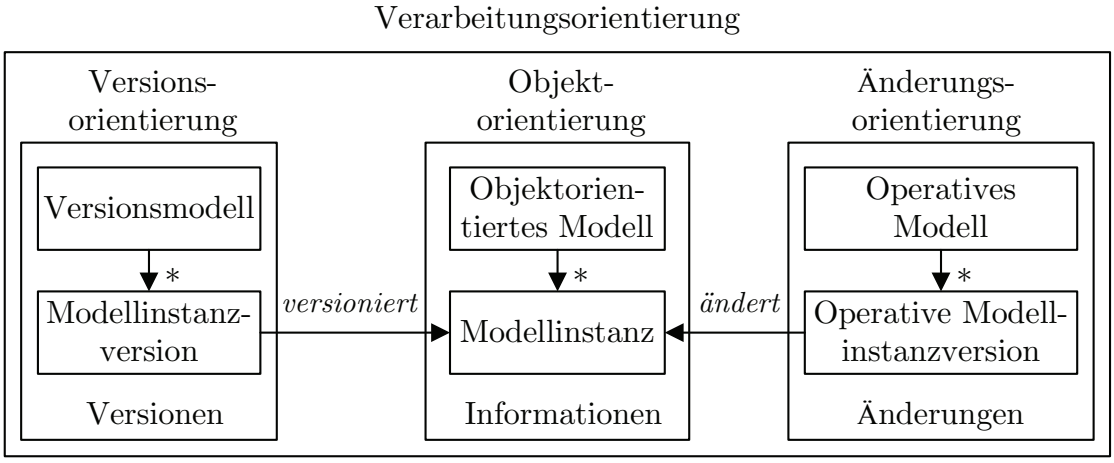


Bild 3.13: Verarbeitungsorientierung

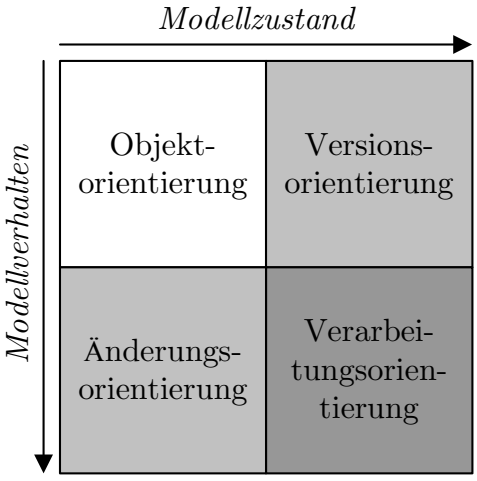


Bild 3.14: Matrix der Modellierungsansätze

3.2 Mathematische Beschreibung

In diesem Abschnitt wird eine allgemeingültige und übertragbare Formulierung des verarbeitungsorientierten Modells auf Basis der Mengenlehre und der Relationenalgebra vorstellt.

3.2.1 Grundlagen

Objekt: In der objektorientierten Modellierung abstrahiert ein Objekt ein Element der fachlichen Instanz, indem es dessen Zustand in Attributen und dessen Verhalten in Methoden nachbildet.

Modellinstanz: Das von einer Fachapplikation zur Verarbeitung instanziierte objektorientierte Modell heißt Modellinstanz B und wird durch eine strukturierte Menge von Objekten beschrieben.

$$B := \{b \mid b \text{ ist ein Objekt der Modellinstanz}\} \tag{3.1}$$

Operationsinstanzmenge: Eine Operationsinstanz verarbeitet die Modellinstanz, indem sie auf derselben operiert. Dabei wird zwischen lesenden Operationsinstanzen und schreibenden Operationsinstanzen unterschieden. Lesende Operationsinstanzen stellen Abfragen dar, welche bestimmte Informationen über die Modellinstanz ermitteln. Schreibende Operationsinstanzen fügen der Modellinstanz Objekte hinzu, verändern oder löschen vorhandene Objekte. Die Menge aller Operationsinstanzen, die die Modellinstanz verarbeiten, heißt Operationsinstanzmenge Ω.

$$\Omega := \{\omega \mid \omega \text{ ist eine Operationsinstanz auf der Modellinstanz } B\} \tag{3.2}$$

Operationsmenge: Gleichartige Operationsinstanzen werden durch Operationen typisiert und zusammengefasst. Die Menge aller Operationen heißt Operationsmenge T. Das operative Modell wird durch die Operationsmenge T beschrieben.

$$T := \{t \mid t \text{ ist eine Operation des operativen Modells}\} \tag{3.3}$$

Operationsabbildung: Jede Operationsinstanz $\omega \in \Omega$ ist genau einer Operation $t \in T$ zugeordnet. Diesen Sachverhalt beschreibt die Operationsabbildung τ.

$$\tau : \Omega \to T := \{(\omega, t) \in \Omega \times T \mid \omega \text{ ist eine Operationsinstanz der Operation } t\} \tag{3.4}$$

Beispiel 3.15: Operationsabbilung

Das Bild 3.15 zeigt eine Operationsabbildung τ mit einer Operationsinstanzmenge Ω

und einer Operationsmenge T.

$$
\begin{aligned}
T &= \{erzeugeWand, erzeugeFenster, verschiebe, loesche\} \\
\Omega &= \{loesche(b), erzeugeWand(0,0,\ldots,2,a), verschiebe(2,1,1,a), \\
&\quad erzeugeWand(1,2,..,1,b)\} \\
\tau &= \{(erzeugeWand(0,0,\ldots,2,a), erzeugeWand), \\
&\quad (erzeugeWand(1,2,..,1,b), erzeugeWand), \\
&\quad (verschiebe(2,1,1,a), verschiebe), (loesche(b), loesche)\}
\end{aligned}
$$

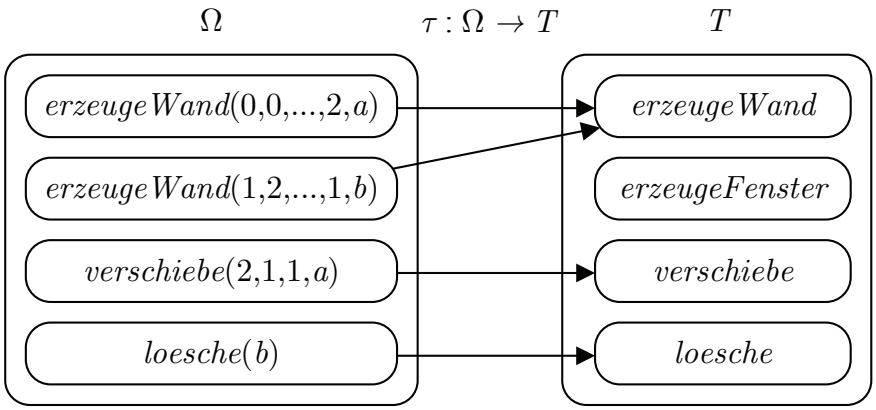


Bild 3.15: Operationsabbildung

Objektversion: Durch die Verarbeitung der Modellinstanz mit Operationsinstanzen verändern Objekte ihren Zustand. Ein ausgezeichneter Zustand eines Objekts wird Objektversion genannt. Der Zustand vor der Instanziierung eines Objekts heißt Ursprungsobjektversion, der Zustand nach dem Löschen einer Objektversion wird als gelöschte Objektversion bezeichnet. Die Ursprungsobjektversion und die gelöschten Objektversion werden virtuelle Objektversionen genannt, weil das Objekt eigentlich keinen Zustand hat. Versionsbeziehungen zwischen einzelnen Objektversionen einer Modellinstanz werden durch die Objektversionsrelation in [Firmenich 2002] beschrieben, hier aber nicht weiter betrachtet.

Modellinstanzversion: Im Bauplanungsprozess wird die Modellinstanz B von mehreren beteiligten Planern iterativ und zum Teil auch zeitgleich bearbeitet, sodass verschiedene Versionen der Modellinstanz B entstehen. Eine Version der Modellinstanz heißt Modellinstanzversion B_x, beschreibt die Modellinstanz B im Zustand x und wird durch eine strukturierte Menge von Objektversionen beschrieben. Zu Beginn der Planung existiert eine virtuelle Version β der Modellinstanz, welche die virtuellen Ursprungsobjektversionen enthält.

$$
\begin{aligned}
B_x &:= \{b \mid b \text{ ist eine nicht-virtuelle Objektversion} \\
&\quad \text{der Modellinstanz } B \text{ im Zustand } x\} \qquad (3.5) \\
\beta &: \text{virtuelle Modellinstanzversion} \qquad (3.6)
\end{aligned}
$$

Änderung: Eine Folge von auf eine Modellinstanzversion angewendeten Operationsinstanzen $\omega \in \Omega$ wird als Änderung δ bezeichnet.

$$\delta \quad := \quad \langle \omega_i, \ldots, \omega_k \rangle \quad \text{mit} \quad \omega_i, \ldots, \omega_k \in \Omega \tag{3.7}$$

Das Bild 3.16 zeigt eine Änderung $\delta_{x,y}$, die auf die Modellinstanzversion B_x angewendet wird. Es entsteht die Version B_y. Die Ausführung einer Änderung wird durch eine gestrichelte Linie mit ausgefülltem Pfeil dargestellt.

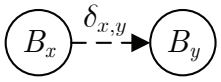


Bild 3.16: Änderung

Gerichter Graph: Ein Gebilde $G = (K; R)$ wird gerichteter Graph genannt, wenn K die Knotenmenge und $R \subseteq K \times K$ die Kantenmenge des Graphen ist. Eine gerichtete Kante vom Knoten $a \in K$ zum Knoten $b \in K$ wird als geordnetes Paar $(a, b) \in R$ formuliert (vgl. [Pahl u. Damrath 2000, S. ?]).

$$\begin{aligned} G \quad &:= \quad (K; R) \\ K \quad &: \quad \text{Knotenmenge} \\ R \subseteq K \times K \quad &: \quad \text{Kantenmenge der geordneten Knotenpaare} \end{aligned} \tag{3.8}$$

Pfad im Graphen: Eine Folge von Kanten im gerichteten Graphen $G = (K; R)$ heißt Pfad p, wenn der Endknoten jeder Kante mit Ausnahme der letzten Kante dem Anfangsknoten der folgenden Kante entspricht (vgl. [Pahl u. Damrath 2000, S. 568]). Eine Kante $(a, b) \in R$ eines Pfades p wird als Element des Pfades $(a, b) \in p$ definiert.

$$p_x \quad := \quad \langle (x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n) \rangle \quad : \bigwedge_{j=1}^{n} ((x_{j-1}, x_j) \in R) \tag{3.9}$$

Kettung von Pfaden: Gegeben seinen zwei Pfade $p_x = \langle (x_0, x_1), \ldots, (x_{l-1}, x_l) \rangle$ und $p_y = \langle (y_0, y_1), \ldots, (y_{m-1}, y_m) \rangle$ in einem gerichteten Graphen $G = (K; R)$. Ist der Endknoten $x_l \in K$ der letzten Kante $(x_{l-1}, x_l) \in R$ von p_x gleich dem Anfangsknoten y_0 der ersten Kante $(y_0, y_1) \in R$ von p_y, dann existiert ein Pfad p_z als Kettung $p_x \circ p_y$.

$$\begin{aligned} p_z = p_x \circ p_y \quad := \quad & \langle (z_0, z_1), \ldots, (z_{i-1}, z_i), (z_i, z_{i+1}), \ldots, (z_{n-1}, z_n)) \rangle \\ & : \bigwedge_{j=1}^{n} ((z_{j-1}, z_j) \in R) \wedge \bigwedge_{j=1}^{i} ((z_{j-1}, z_j) \in p_x) \wedge \bigwedge_{j=i}^{n} ((z_j, z_{j+1}) \in p_y) \end{aligned} \tag{3.10}$$

Beispiel 3.16: Kettung von Pfaden

Das Bild 3.17 veranschaulicht die Pfade p_x und p_y in einem gerichteten Graphen mit

$$p_x = \langle (a, b), \ldots, (d, e) \rangle \text{ und } p_y = \langle (e, f), \ldots, (h, i) \rangle \ .$$

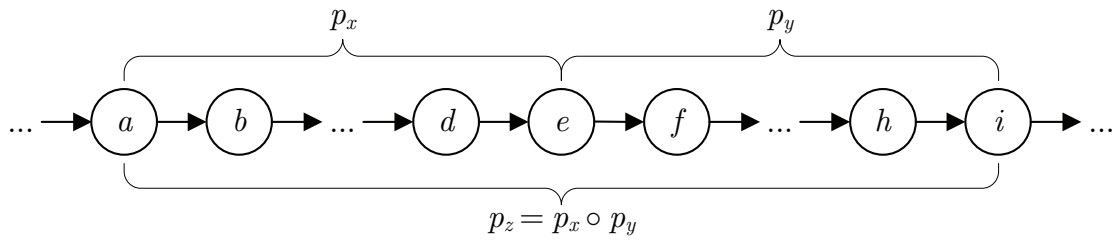


Bild 3.17: Kettung von Pfaden in einem gerichteten Graphen

Die Kettung der Pfade p_x und p_y ergibt sich zum Pfad $p_z = p_x \circ p_z$ mit

$$p_z = \langle (a,b), \ldots, (d,e), (e,f), \ldots, (h,i) \rangle \ .$$

Durchschnitt von Pfaden: Gegeben seinen zwei Pfade $p_x = \langle (x_0, x_1), \ldots, (x_{l-1}, x_l) \rangle$ und $p_y = \langle (y_0, y_1), \ldots, (y_{m-1}, y_m) \rangle$ in einem gerichteten Graphen $G = (K; R)$. Ist eine Folge von Kanten $\langle (z_0, z_1), \ldots, (z_{n-1}, z_n) \rangle$ in beiden Pfaden p_x und p_y enthalten, dann wird diese Folge mit Pfad p_z als Durchschnitt $p_x \cap p_y$ bezeichnet.

$$\begin{aligned} p_z = p_x \cap p_y \quad := \quad & \langle (z_0, z_1), \ldots, (z_{n-1}, z_n) \rangle \\ & : \bigwedge_{j=1}^{n} \left((z_{j-1}, z_j) \in R \wedge (z_{j-1}, z_j) \in p_x \wedge (z_{j-1}, z_j) \in p_y \right) \end{aligned} \tag{3.11}$$

Beispiel 3.17: Durchschnitt von Pfaden
Das Bild 3.18 veranschaulicht die Pfade p_x und p_y in einem gerichteten Graphen mit

$$p_x = \langle (a,b), \ldots, (d,e), \ldots, (g,h) \rangle \text{ und } p_y = \langle (d,e), \ldots, (g,h), \ldots, (j,k) \rangle \ .$$

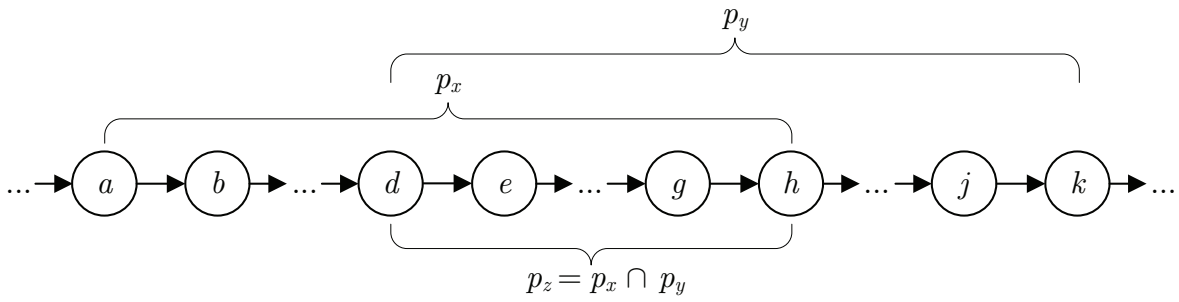


Bild 3.18: Durchschnitt von Pfaden in einem gerichteten Graphen

Der Durchschnitt der Pfade p_x und p_y ergibt sich zum Pfad $p_z = p_x \cap p_y$ mit

$$p_z = \langle (d,e), \ldots, (g,h) \rangle \ .$$

3.2.2 Versionsgraph

Menge der Modellinstanzversionen: Die Menge aller Modellinstanzversionen inklusive der virtuellen Modellinstanzversion β heißt V.

$$V := \{v \mid v \text{ ist eine Modellinstanzversion}\} \tag{3.12}$$

Nachfolgerrelation: Die Nachfolgerbeziehungen zwischen Modellinstanzversionen werden durch die Nachfolgerrelation E_N beschrieben. Ist die Version $b \in V$ Nachfolgerversion von $a \in V$, dann ist das geordnete Paar (a, b) Element der Nachfolgerrelation E_N. Die virtuelle Version β ist keine Nachfolgerversion einer anderen Version.

$$E_N := \{(a, b) \in V \times V \mid b \text{ ist die Nachfolgerversion von } a \ \wedge \ b \neq \beta\} \tag{3.13}$$

Versionsgraph: Die Versionen der Modellinstanz und ihre Nachfolgerbeziehungen werden im sogenannten Versionsgraphen G_N der Modellinstanz veranschaulicht. Die Knotenmenge des Graphen G_N ist die Menge V der Modellinstanzversionen und die Kantenmenge entspricht der Nachfolgerrelation E_N. Der Versionsgraph ist ein gerichteter, zyklenfreier Wurzelgraph mit der virtuellen Modellinstanzversion B_β als Wurzelknoten.

$$G_N := (V; E_N) \tag{3.14}$$

Grafisch werden die Knoten des Versionsgraphen durch Kreise und die Kanten durch durchgezogene Linien mit ausgefülltem Pfeil dargestellt.

Beispiel 3.18: Versionsgraph der Modellinstanz

Das Bild 3.19 zeigt den Versionsgraphen G_N der Modellinstanz B mit B_β als virtuelle Modellinstanzversion, den Versionen B_a, B_b, B_c, B_d und B_e sowie den Nachfolgerbeziehungen.

$$\begin{aligned} \beta &= B_\beta \\ V &= \{B_\beta, B_a, B_b, B_c, B_d, B_e\} \\ E_N &= \{(B_\beta, B_a), (B_a, B_b), (B_b, B_c), (B_a, B_d), (B_c, B_e), (B_d, B_e)\} \end{aligned}$$

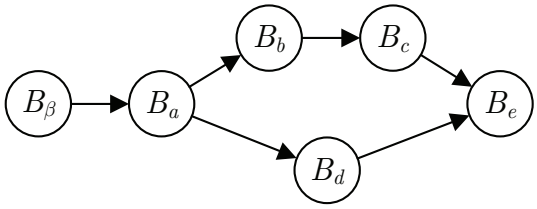


Bild 3.19: Versionsgraph G_N der Modellinstanz B

Pfad im Versionsgraphen: Eine Folge von Kanten im gerichteten, zyklenfreien Versionsgraphen G_N heißt Pfad p im Versionsgraphen, wenn der Endknoten jeder Kante mit Ausnahme der letzten Kante dem Anfangsknoten der folgenden Kante entspricht.

$$p_x := \langle (x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n) \rangle : \bigwedge_{j=1}^{n} ((x_{j-1}, x_j) \in E_N) \tag{3.15}$$

Beispiel 3.19: Pfad im Versiongraphen

Das Bild 3.20 veranschaulicht den Pfad p_x von der Version B_a zur Version B_g im Versionsgraphen.

$$p_x = \langle (B_a, B_b), \ldots, (B_f, B_g) \rangle$$

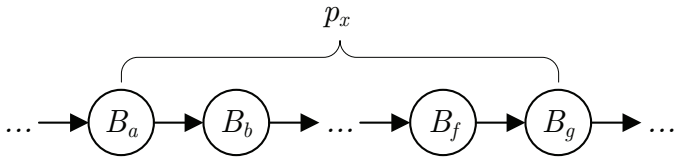


Bild 3.20: Pfad im Versionsgraphen

Transitive Hülle der Nachfolgerrelation: Die Relation $H(E_N) \subseteq V \times V$ heißt transitive Hülle der Nachfolgerrelation E_N. Das geordnete Paar (a, c) ist Element von $H(E_N)$, wenn im Versionsgraphen G_N ein Pfad von $a \in V$ nach $c \in V$ existiert.

$$H(E_N) := \{(a, c) \in V \times V \mid \text{ es existiert ein Pfad von } a \text{ nach } c\} \tag{3.16}$$

Beispiel 3.20: Transitive Hülle

Für den im Bild 3.19 dargestellten Versionsgraphen ergibt sich folgende transitive Hülle $H(E_N)$ der Nachfolgerrelation E_N.

$$\begin{aligned} H(E_N) = \{&(B_\beta, B_a), (B_\beta, B_b), (B_\beta, B_c), (B_\beta, B_d), (B_\beta, B_e), (B_a, B_b), (B_a, B_c), \\ &(B_a, B_d), (B_a, B_e), (B_b, B_c), (B_b, B_e), (B_c, B_e), (B_d, B_e)\} \end{aligned}$$

Revision und Variante: Im Versionsgraphen werden sowohl Revisionen als auch Varianten abgebildet. Existiert im Versionsgraphen ein Pfad von $a \in V$ nach $c \in V$, dann ist c eine Revision von a, andernfalls ist c eine Variante zu a.

$$c \text{ ist Revision von } a :\Leftrightarrow (a, c) \in H(E_N) \tag{3.17}$$

$$c \text{ ist Variante zu } a :\Leftrightarrow (a, c) \notin H(E_N) \tag{3.18}$$

Beispiel 3.21: Revision und Variante

Im Versionsgraphen des Bildes 3.19 ist B_c eine Revision von B_b und gleichzeitig eine Variante zu Version B_d (vgl. Beispiel 3.20).

Zusammenführung von Versionen im Versionsgraphen: Die durch das parallele und iterative Arbeiten entstehenden Versionen beschreiben jeweilige Teillösungen der gesamten Planungsaufgabe. Enthalten die Teillösungen wichtige Informationen zur Beschreibung der Gesamtlösung, dann müssen unterschiedliche Versionen, das heißt Varianten und auch Revisionen in einem Abstimmungsprozess zu einer Gesamtversion zusammengeführt werden. Die Menge der zusammenzuführenden Modellinstanzversionen heißt Zusammenführungsmenge S. Die bei der Zusammenführung entstehende Modellinstanzversion wird Zusammenführungsversion $\gamma \in V$ der Zusammenführungsmenge S genannt.

$$S := \{a \in V \mid \text{Die Modellinstanzversion } a \text{ wird zusammengeführt}\} \subset V \tag{3.19}$$

$$\gamma \in V : \text{Zusammenführungsversion der Zusammenführungsmenge } S \tag{3.20}$$

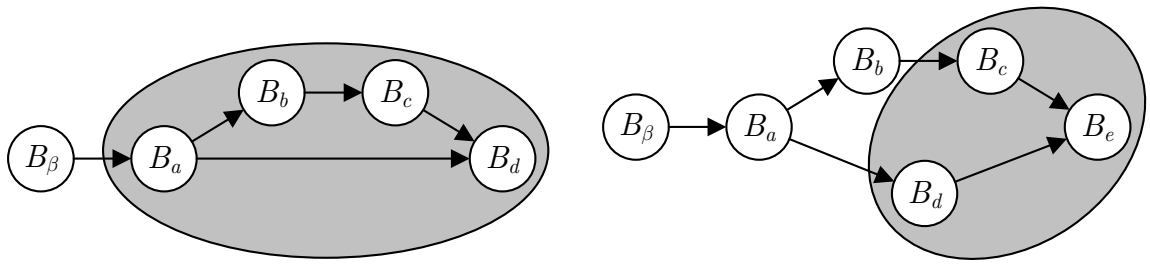
Die Zusammenführung der Modellinstanzversionen $a, \ldots, c \in V$ zur Zusammenführungsversion $\gamma \in V$ wird durch eine Menge von Kanten $(a, \gamma), \ldots, (c, \gamma) \in S \times \{\gamma\} \subset E_N$ im Versionsgraphen G_N beschrieben. Diese Menge heißt Menge Σ der Zusammenführungskanten. Nach einer Zusammenführung haben die Versionen $a, \ldots, c$ die gemeinsame Nachfolgerversion γ.

$$\Sigma := \{(a, \gamma) \in S \times \{\gamma\} \mid \text{Die Modellinstanzversion } a \text{ wird zur Zusammenführungsversion } \gamma \text{ zusammengeführt } \} \subset E_N \tag{3.21}$$

Beispiel 3.22: Zusammenführung von Versionen im Versionsgraphen

Im Bild 3.21a werden die Versionen B_a und B_c zur Version B_d zusammengeführt. Die Version B_c ist eine Revision der Version B_a. Im Bild 3.21b werden die Versionen B_c und B_d zur Version B_e zusammengeführt. Die Version B_d ist eine Variante zur Version B_c. Die Zusammenführungen werden jeweils durch die Zusammenführungsmenge S, die Zusammenführungsversion γ und die Menge Σ der Zusammenführungskanten beschrieben.

$$\begin{aligned} S^a &= \{B_a, B_c\} \\ \gamma^a &= B_d \\ \Sigma^a &= \{(B_a, B_d), (B_c, B_d)\} \end{aligned} \qquad \begin{aligned} S^b &= \{B_c, B_d\} \\ \gamma^b &= B_e \\ \Sigma^b &= \{(B_c, B_e), (B_d, B_e)\} \end{aligned}$$



(a) Zusammenführung B_a mit einer Revision B_c (b) Zusammenführung von Varianten B_c und B_d

Bild 3.21: Versionsgraphen mit Zusammenführungen

Zusammenführungsinformationen im Versionsgraphen: Der Versionsgraph bildet die Versionen und ihre Nachfolgerbeziehungen ab. Somit werden Informationen darüber gespeichert, welche Version aus einer Zusammenführung hervorgegangen ist und welche Versionen zusammengeführt wurden.

3.2.3 Änderungsbaum

Änderungsrelation: Die Änderungsbeziehungen zwischen den Modellinstanzversionen werden durch die Änderungsrelation E_A beschrieben. Eine Version $a \in V$ der Modellinstanz wird zur Version $b \in V$, wenn auf a eine Änderung angewendet wird. Die virtuelle Version β entsteht nicht durch die Anwendung einer Änderung auf eine andere Version.

$$E_A := \{(a,b) \in V \times V \mid \text{die Version } b \text{ entsteht durch die Anwendung einer Änderung auf die Version } a \ \wedge \ b \neq \beta\} \tag{3.22}$$

Änderungsmenge: Die Menge aller auf die Modellinstanz B angewendeten Änderungen heißt Änderungsmenge Δ.

$$\Delta := \{\delta \mid \delta \text{ ist eine Änderung auf der Modellinstanz } B\} \tag{3.23}$$

Änderungsabbildung: Jedem geordneten Paar $(a,b) \in E_A$ ist die Änderung $\delta \in \Delta$ zugeordnet. Diesen Sachverhalt beschreibt die Änderungsabbildung α.

$$\alpha : E_A \to \Delta := \{((a,b),\delta) \in E_A \times \Delta \mid \text{der Kante } (a,b) \text{ ist die Änderung } \delta \text{ zugeordnet}\} \text{ mit } \alpha((a,b)) = \delta \tag{3.24}$$

Änderungsbaum: Die Versionen der Modellinstanz und ihre Änderungsbeziehungen werden im Graphen G_A dargestellt. Die Knotenmenge des Graphen G_A ist die Menge V der Versionen und die Kantenmenge entspricht der Änderungsrelation E_A. Jeder Kante $(a,b) \in E_A$ ist eine Änderung $\delta \in \Delta$ zugeordnet. Die Version $b \in V$ entsteht, wenn genau eine Änderung $\delta \in \Delta$ auf genau der Version $a \in V$ ausgeführt wird. Daraus folgt, dass jeder Knoten $v \in V$ des Graphen G_A, mit Ausnahme des Knotens der virtuellen Version β, genau einen Vorgängerknoten hat. Der Graph G_A ist somit ein Wurzelbaum mit der virtuellen Version β als Wurzelknoten und wird Änderungsbaum G_A der Modellinstanz genannt.

$$G_A := (V, \Delta; E_A, \alpha) \tag{3.25}$$

Grafisch werden die Knoten des Änderungsbaumes durch Kreise und die Kanten durch gestrichelte Linien mit ausgefülltem Pfeil dargestellt. Die den Kanten durch die Änderungsabbildung zugeordneten Änderungen erscheinen als Texte auf den Kanten.

Beispiel 3.23: Änderungsbaum der Modellinstanz

Das Bild 3.22 auf der nächsten Seite zeigt den Änderungsbaum G_A der Modellinstanz B

mit der virtuellen Version β, den Versionen B_a, B_b, B_c, B_d und B_e und den ausgeführten Änderungen $\delta_{\beta,a}, \delta_{a,b}, \delta_{b,c}, \delta_{a,c}$ und $\delta_{a,e}$ (vgl. Versionsgraph G_N der Modellinstanz B im Bild 3.19 auf Seite 52).

$$
\begin{aligned}
\beta &= B_\beta \\
V &= \{B_\beta, B_a, B_b, B_c, B_d, B_e\} \\
E_A &= \{(B_\beta, B_a), (B_a, B_b), (B_b, B_c), (B_a, B_d), (B_a, B_e)\} \\
\Delta &= \{\delta_{\beta,a}, \delta_{a,b}, \delta_{b,c}, \delta_{a,d}, \delta_{a,e}\} \\
\alpha &= \{((B_\beta, B_a), \delta_{\beta,a}), ((B_a, B_b), \delta_{a,b}), ((B_b, B_c), \delta_{b,c}), ((B_a, B_d), \delta_{a,d}), ((B_a, B_e), \delta_{a,e})\}
\end{aligned}
$$

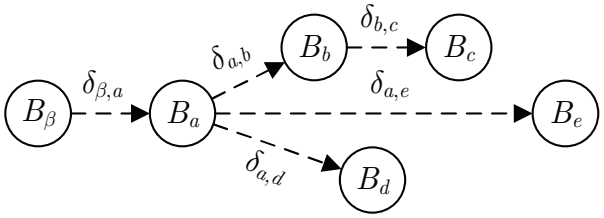


Bild 3.22: Änderungsbaum G_A der Modellinstanz B

Wurzelpfad: Gegeben sei $x_0 \in V$ als Wurzelknoten des Änderungsbaumes G_A. Dann existiert genau ein Wurzelpfad $r(x_n)$ vom Wurzelknoten x_0 bis zum entsprechenden Versionsknoten $x_n \in V$ im Änderungsbaum G_A.

$$
r(x_n) := \langle (x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n) \rangle : \bigwedge_{j=1}^{n} ((x_{j-1}, x_j) \in E_A) \tag{3.26}
$$

Beispiel 3.24: Wurzelpfad im Änderungsbaum
Für die Modellinstanzversion B_c im Änderungsbaum des Bildes 3.22 wird der Wurzelpfad $r(B_c)$ angegeben.

$$
r(B_c) = \langle (B_\beta, B_a), (B_a, B_b), (B_b, B_c) \rangle
$$

Operative Modellinstanzversion: Gegeben sei die Modellinstanzversion $x_n \in V$. Dann wird die operative Modellinstanzversion $M(x_n)$ durch die Folge der den im Wurzelpfad $r(x_n)$ des Änderungsbaumes G_A enthaltenen Kanten zugeordneten Änderungen beschrieben. Diese Folge von Änderungen stellt eine operative Beschreibung der Modellinstanzversion x_n dar.

$$
\begin{aligned}
M(x_n) := {} & \langle \delta_{0,1}, \delta_{1,2}, \ldots, \delta_{n-1,n} \rangle \\
& : \bigwedge_{j=1}^{n} (\delta_{j-1,j} = \alpha((x_{j-1}, x_j)) \wedge (x_{j-1}, x_j) \in E_A)
\end{aligned} \tag{3.27}
$$

Beispiel 3.25: Operative Modellinstanzversion

Es wird die Version B_c im Änderungsbaum des Bildes 3.22 auf Seite 56 betrachtet. Die operative Beschreibung der Version B_c entspricht der operativen Modellinstanzversion $M(B_c)$.

$$M(B_c) \quad = \quad \langle \delta_{\beta,a}, \delta_{a,b}, \delta_{b,c} \rangle$$

Zusammenführung von Versionen im Änderungsbaum: Die Modellinstanzversionen $x_l, \ldots, y_m \in V$ werden zur Zusammenführungsversion $\gamma \in V$ zusammengeführt. Die Zusammenführungsmenge S ist demnach

$$S \quad = \quad \{x_l, \ldots, y_m\} \subset V \ .$$

Dann ergeben sich die im Änderungsbaum des Bildes 3.23 veranschaulichten Wurzelpfade $r(x_l), \ldots, r(y_m)$ zu

$$\begin{aligned} r(x_l) \quad &= \quad \langle (x_0, x_1), \ldots, (x_{l-1}, x_l) \rangle \\ &\vdots \\ r(y_m) \quad &= \quad \langle (y_0, y_1), \ldots, (y_{m-1}, y_m) \rangle \ . \end{aligned}$$

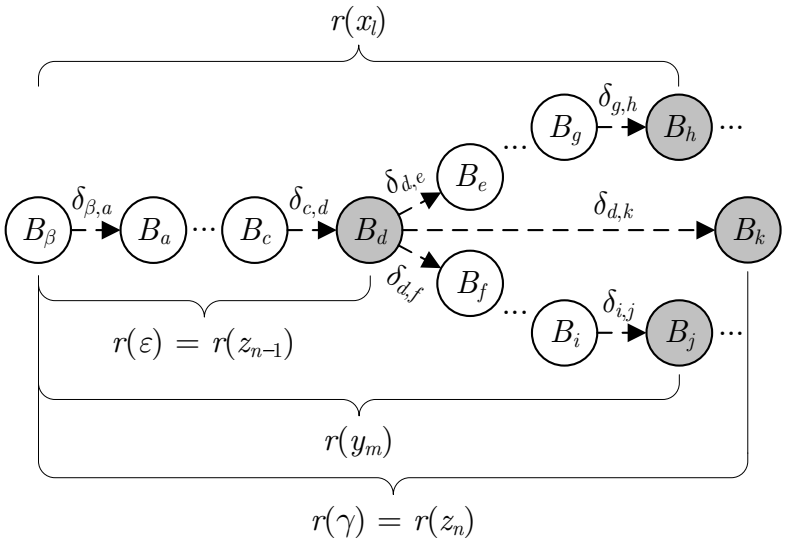


Bild 3.23: Wurzelpfade bei einer Zusammenführung im Änderungsbaum

Die Folge der gemeinsamen Kanten der Wurzelpfade $r(x_l), \ldots, r(y_m)$ wird als Durchschnitt $r(x_l) \cap \ldots \cap r(y_m)$ bestimmt und heißt Ursprungspfad $r(z_{n-1})$.

$$r(z_{n-1}) \quad := \quad r(x_l) \cap \ldots \cap r(y_m) = \langle (z_0, z_1), \ldots, (z_{n-2}, z_{n-1}) \rangle$$

Der Endknoten $z_{n-1} \in V$ des Ursprungspfades $r(z_{n-1})$ wird Ursprungsversion $\varepsilon \in V$ der Zusammenführungsmenge S genannt.

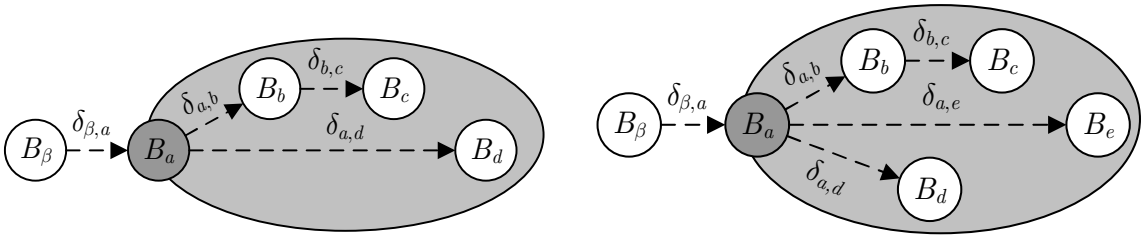
$$\varepsilon \in V \quad : \quad \text{Ursprungsversion der Zusammenführungsmenge } S \tag{3.28}$$

Die Zusammenführung der Versionen $x_l, \ldots, y_m \in V$ zur Zusammenführungsversion $\varepsilon \in V$ wird durch die Zusammenführungsänderungskante $\sigma \in E_A$ von der Ursprungsversion ε zur Zusammenführungsversion $\gamma \in V$ im Änderungsbaum G_A beschrieben. Die bei einer Zusammenführung ausgeführten Änderungen $\delta_{\varepsilon,\gamma} \in \Delta$ werden der Zusammenführungsänderungskante $(\varepsilon, \gamma) \in E_A$ zugeordnet. Der Wurzelpfad $r(\gamma)$ der Zusammenführungsversion γ wird als Kettung des Wurzelpfads $r(\varepsilon)$ der Ursprungsversion ε mit dem Pfad der Zusammenführungsänderungskante $\sigma \in E_A$ formuliert.

$$\sigma := (\varepsilon, \gamma) \in E_A \tag{3.29}$$
$$\alpha(\sigma) := \delta_{\varepsilon,\gamma} \in \Delta \tag{3.30}$$
$$r(\gamma) := r(\varepsilon) \circ \langle \sigma \rangle \tag{3.31}$$

Beispiel 3.26: Zusammenführung von Versionen im Änderungsbaum
Im Bild 3.24a wird die Zusammenführung der Versionen B_a und B_c zur Version B_d dargestellt. Die Version B_c ist eine Revision der Version B_a. Im Bild 3.24b werden die Versionen B_c und B_d zur Version B_e zusammengeführt. Die Version B_d ist eine Variante zur Version B_c. In den Änderungsbäumen ist zu erkennen, dass sowohl im Bild 3.24a als auch im Bild 3.24b die Version B_e durch die Anwendung der Änderung $\delta_{a,e}$ entstanden ist. Die Information, welche Versionen zusammengeführt wurden, wird durch die Änderungsbäume nicht beschrieben. Die im Bild 3.24 dargestellten Zusammenführungen werden jeweils durch die Zusammenführungsmenge S, die Zusammenführungsversion v_γ, die Ursprungsversion v_ε und die Zusammenführungsänderungskante σ beschrieben (vgl. Beispiel 3.22 auf Seite 54).

$$\begin{aligned} S^a &= \{B_a, B_c\} \\ \gamma^a &= B_d \\ \varepsilon^a &= B_a \\ \sigma^a &= (B_a, B_d) \\ r(\gamma^a) &= \langle (B_\beta, B_a), (B_a, B_d) \rangle \end{aligned} \qquad \begin{aligned} S^b &= \{B_c, B_d\} \\ \gamma^b &= B_e \\ \varepsilon^b &= B_a \\ \sigma^b &= (B_a, B_d) \\ r(\gamma^b) &= \langle (B_\beta, B_a), (B_a, B_e) \rangle \end{aligned}$$



(a) Zusammenführung B_a mit einer Revision B_c (b) Zusammenführung von Varianten B_c und B_d

Bild 3.24: Änderungsbäume mit Zusammenführungen

Zusammenführungsinformationen im Änderungsbaum: Der Änderungsbaum bildet die Modellinstanzversionen und ihre Änderungsbeziehungen ab. Im Gegensatz

zum Versionsgraphen werden keine Informationen darüber gespeichert, welche Versionen zusammengeführt werden und welche Version dabei entsteht. Jedoch wird die bei der Zusammenführung ausgeführte Änderung als Kante im Änderungsbaum berücksichtigt.

3.2.4 Verarbeitungsgraph

Verarbeitungsgraph: Zur ganzheitlichen Modellierung der Modellinstanzversionen (V), der Änderungen (Δ), der Nachfolger- und Änderungsbeziehungen (E_N, E_A) sowie der Änderungsabbildung (α) wird der Verarbeitungsgraph G_V eingeführt.

$$G_V := (V, \Delta; E_N, E_A, \alpha) \tag{3.32}$$

Beispiel 3.27: Verarbeitungsgraph der Modellinstanz

Das Bild 3.25 zeigt den Verarbeitungsgraphen G_V der Modellinstanz B mit der virtuellen Modellinstanzversion B_β, den Versionen B_a, B_b, B_c, B_d und B_e, der Nachfolgerrelation E_N, der Änderungsrelation E_A, der Änderungsmenge Δ und der Änderungsabbildung α. Darüber hinaus stellt der Graph eine Zusammenführung mit der Zusammenführungsmenge S, der Zusammenführungsversion γ, der Menge der Zusammenführungskanten Σ, der Zusammenführungsänderungskante σ und der Ursprungsversion ε dar.

$$\begin{aligned}
\beta &= B_\beta \\
V &= \{B_\beta, B_a, B_b, B_c, B_d, B_e\} \\
E_N &= \{(B_\beta, B_a), (B_a, B_b), (B_b, B_c), (B_a, B_d), (B_c, B_e), (B_d, B_e)\} \\
E_A &= \{(B_\beta, B_a), (B_a, B_b), (B_b, B_c), (B_a, B_d), (B_a, B_e)\} \\
\Delta &= \{\delta_{\beta,a}, \delta_{a,b}, \delta_{b,c}, \delta_{a,d}, \delta_{a,e}\} \\
\alpha &= \{((B_\beta, B_a), \delta_{\beta,a}), ((B_a, B_b), \delta_{a,b}), ((B_b, B_c), \delta_{b,c}), ((B_a, B_d), \delta_{a,d}), ((B_a, B_e), \delta_{a,e})\} \\
S &= \{B_c, B_d\} \\
\gamma &= B_e, \quad \Sigma = \{(B_c, B_e), (B_d, B_e)\} \\
\varepsilon &= B_a, \quad \sigma = (B_a, B_e), \quad \alpha(\sigma) = \delta_{a,e}
\end{aligned}$$

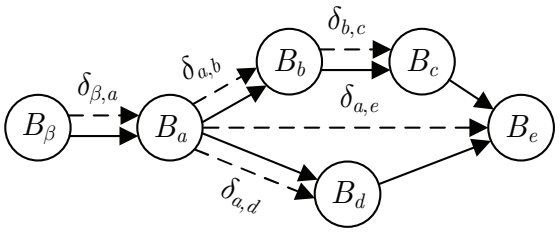


Bild 3.25: Verarbeitungsgraph G_V der Modellinstanz B

Unversionierte Modellinstanz: In einer unversionierten Umgebung wird auf der virtuellen Version β genau eine Änderung δ_β ausgeführt, welche die Modellinstanz

B als einzige Version erzeugt. Die Änderung δ_β wird durch die Operationsinstanzen $\omega_0, \ldots, \omega_n \in \Omega$ beschrieben und heißt operative Modellinstanz $M(B)$.

$$M(B) \;:=\; \delta_\beta \;\text{ mit }\; \delta_\beta = \langle \omega_0, \ldots, \omega_n \rangle \;\wedge\; \omega_0, \ldots, \omega_n \in \Omega \;\wedge\; \omega_0 \text{ operiert auf } \beta \tag{3.33}$$

Das Bild 3.26 veranschaulicht diesen Sachverhalt.

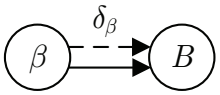


Bild 3.26: Unversionierte Modellinstanz B

3.3 Bauwerksmodelle

Dem Stand der Technik[1] entsprechend werden für den Prozess der rechnergestützten Bauwerksplanung fachliche Modelle in objektorientierte Computermodelle überführt. In diesem Abschnitt wird der verarbeitungsorientierte Ansatz im Hinblick auf die Bauwerksmodellierung beleuchtet. Dabei wird sowohl auf die Stellung als auch die Bedeutung der Operationen im Kontext von Bauwerksmodellen eingegangen.

3.3.1 Identifikation

Identität: In der Objektorientierung haben Objekte eine Identität, eine Eigenschaft, welche sie eindeutig von anderen Objekten unterscheidet. Diese Eigenschaft wird zur Identifizierung von Objekten bei der Verarbeitung eines objektorientierten Bauwerksmodells durch Operationen verwendet.

Transienter Objektidentifikator: Zur Laufzeit einer Fachapplikation wird einem Objekt bei der Erzeugung ein temporärer oder transienter Identifikator zugeordnet. In der Regel ist ein transienter Identifikator die Adresse des Objekts im Arbeitsspeicher. Eine Fachapplikation identifiziert zur Laufzeit ein Objekt auf Basis seines transienten Identifikators. Wird ein Objekt gespeichert und anschließend von einer Fachapplikation geladen, erhält es einen anderen, neuen transienten Objektidentifikator. Transiente Objektidentifikatoren sind *zeitabhängig* und deshalb zur Identifikation von Objekten im verarbeitungsorientierten Modell unbrauchbar.

Persistenter Objektidentifikator: Ein persistenter Objektidentifikator – POID[2] – ist über die gesamte Lebensdauer eines Objekts konstant und demnach *nicht zeitabhängig*. Im verarbeitungsorientierten Modellierungsansatz wird genau eine Modellinstanz betrachtet. Aus diesem Grund ist es erforderlich, dass die POIDs innerhalb einer Modellinstanz eindeutig sind.

Operanden: Operationen verarbeiten eine Modellinstanz, indem sie auf Modellobjekten operieren. Die Liste von Objekten, die von einer Operation verarbeitet werden soll, heißt Operandenliste. Zur Kennzeichnung eines Objekts als Operand werden POIDs verwendet.

Beispiel 3.28: Operanden einer Operation

Gegeben sind je eine Operation zum *Erzeugen*, zum *Modifizieren* und zum *Löschen* eines Wandobjekts. Beim Erzeugen des Objekts wird der Operand `<objA@x0fe7c>` als POID spezifiziert. In der Modifikationsoperation und beim Löschen dient diese POID als Operand in der Operandenliste `[<objA@x0fe7c>,...]`. Die nachfolgende Sequenz von Operationen verdeutlicht die Verwendung von POIDs als Operanden.

Erzeuge Wand `<objA@x0fe7c>` *mit ...;*
Modifiziere Objekte `[<objA@x0fe7c>, ...]` *durch ...;*
Lösche Objekte `[<objA@x0fe7c>, ...]`*;*

[1] vgl. Kapitel 2 auf Seite 9
[2] Persistent Object Identifier

3.3.2 Semantik

Attribute: In objektorientierten Modellen speichern Attribute die Daten. Jedem Attribut kommt dabei eine bestimmte Bedeutung oder Aussagekraft[3] zu. Das Wissen über die Bedeutung und die Verwendung der Attribute hat im Allgemeinen nur der Entwickler bzw. Programmierer des Modells.

Beispiel 3.29: Attributsemantik

Zur geometrischen Repräsentation einer Wand werden vier private Attribute zur Speicherung von Koordinaten (`x1,y1,x2,y2`) definiert. Mit dem Ziel, eine gleichbleibende Genauigkeit im gesamten Koordinatensystem der geometrischen Modellinstanz zu gewährleisten, speichert der Entwickler die Koordinaten in einem *Weltkoordinatensystem*. Die Semantik der privaten Attribute ist nach außen nicht sichtbar.

Methoden: Zur Verarbeitung der Attribute stellen Objekte Methoden als Objektschnittstelle nach außen bereit. In [Laabs 1998] werden neben *Managementmethoden* und privaten *Hilfsmethoden* weitere Arten von Methoden unterschieden:

- **Zugriffsmethoden** sorgen für den lesenden (get) und schreibenden (set) Zugriff auf die Attribute.
- **Anwendungsmethoden** modellieren das Verhalten der Objekte.

Zugriffsmethoden: Anwendungsprogrammierer verwenden Zugriffsmethoden, um in einem fest definierten Kontext lesend und schreibend auf die als privat deklarierten Objektattribute zuzugreifen. Dieser Zugriffskontext legt die Semantik der Zugriffsmethoden fest. Nicht zwangsläufig geben die `get`-Methoden den tatsächlichen Attributwert zurück, sondern setzen diesen zunächst in einen vereinbarten Kontext. Ferner werden `set`-Methoden dazu verwendet, eine Überprüfung des Attributwerts im Kontext eines Wertebereichs vorzunehmen, bevor ein Wert im Attribut gespeichert wird. Im Hinblick auf ihre Verwendung besitzen Zugriffsmethoden eine größere Aussagekraft als die semantisch niederwertigen Attributwerte.

Beispiel 3.30: Semantik von Zugriffsmethoden

Mit Bezug auf das Beispiel 3.29 werden Zugriffsmethoden für die Koordinatenattribute der Wand definiert. Es wird vereinbart, dass an der Schnittstelle der Zugriffsmethoden sog. *Nutzerkoordinaten* verwendet werden.Das bedeutet eine Transformation von Welt- in Nutzerkoordinaten beim lesenden und von Nutzer- in Weltkoordinaten beim schreibenden Zugriff. Eine solche Vereinbarung definiert den semantischen Kontext der Zugriffsmethoden[4]. Das Bild 3.27 stellt diesen Sachverhalt dar.

Anwendungsmethoden: Anwendungsmethoden bilden das Verhalten von Objekten ab. Sie dienen nicht der gezielten Verarbeitung einzelner Attribute, sondern vielmehr der Verarbeitung des Objekts als Ganzes. Die Semantik der Anwendungsmethoden entspricht somit den Verhaltensweisen eines Objekts.

[3] auch Semantik

[4] Der Kontext wird bspw. in der Dokumentation des Quellcodes angegeben

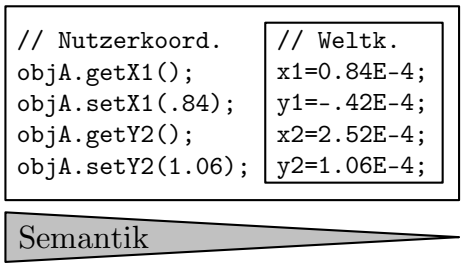


Bild 3.27: Semantik von Zugriffsmethoden

Beispiel 3.31: Semantik von Anwendungsmethoden

Mit Bezug auf die vorangegangenen Beispiele 3.29 und 3.30 wird ein Transformationsverhalten der Wand in der Anwendungsmethode `transf(m00,m10,m01,m11,m02,m12)` definiert. Die Methode führt eine affine 2D-Transformation der Wand auf Basis der durch die Übergabeparameter definierten Transformationsmatrix aus. Dieses Verhalten kennzeichnet die Bedeutung der Anwendungsmethode. Das Bild 3.28 veranschaulicht die Semantik der Anwendungsmethode `transform()`.

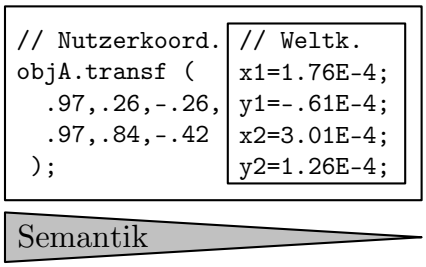


Bild 3.28: Semantik von Anwendungsmethoden

Operationen: Während Anwendungsmethoden das Verhalten einzelner Objekte definieren, dienen Operationen der Beschreibung des Verhaltens einer gesamten Modellinstanz. Operationen verarbeiten die Modellinstanz, indem sie auf Basis der Methoden auf die einzelnen Modellobjekte lesend oder schreibend zugreifen. Die Semantik der Operationen ist im Kontext der Verarbeitung einer Modellinstanz definiert. Analog zu Methoden, die die Objektschnittstelle darstellen, sind Operationen als Modellschnittstelle zu verstehen. Im Verarbeitungskontext einer Modellinstanz haben Operationen eine höhere Aussagekraft als die Methoden der einzelnen Modellobjekte.

Beispiel 3.32: Semantik von Operationen

In den vorangegangenen Beispielen 3.29, 3.30 und 3.31 werden die Eigenschaften und das Verhalten eines Wandobjekts dargestellt. Die Wand ist aber nur ein Teil des komplexen Objektmodells. In diesem Beispiel wird das Verhalten einer Modellinstanz als Ganzes betrachtet. Es wird die Operation *rotate* zur Rotation von geometrischen Modellobjekten, zu denen auch Wandobjekte zählen, definiert. Das Bild 3.29 auf der nächsten Seite verdeutlicht das Verhalten der Modellinstanz bei Rotation der beiden

Wandobjekte `objA` und `objB` um den Punkt (`2,3`) bei einem Winkel von `15°`. Zur Ausführung dieser Operationsinstanz werden die entsprechenden Anwendungsmethoden der Wandobjekte gerufen. Es wird deutlich, dass die Operation `rotate` eine größere Aussagekraft besitzt als eine Objektmethode. Sie ist semantisch hochwertiger als die einzelnen Anwendungsmethoden `transform()`, da sie zum einen eine spezielle Transformation beschreibt und zum anderen die *gemeinsame* Rotation beider Wände als Verarbeitungseinheit ausdrückt.

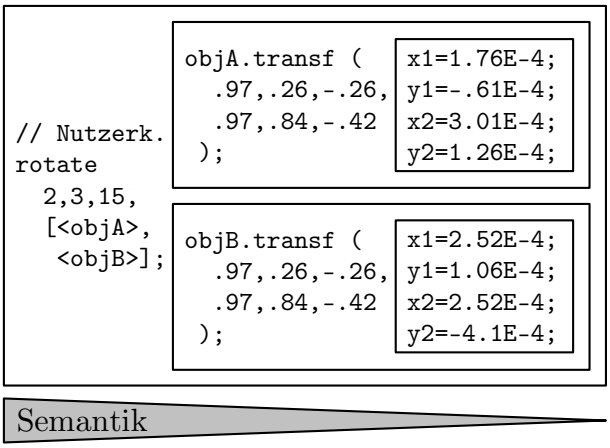


Bild 3.29: Semantik von Operationen

Zusammenfassung: Zur Verarbeitung der in einem Objektmodell als *Attribut*werte gespeicherten Daten sieht die Objektorientierung *Methoden* vor. Methoden bilden die Objektschnittstelle und beschreiben das Verhalten eines Objekts. Das Verhalten einer Modellinstanz, bestehend aus mehreren Modellobjekten, wird durch *Operationen* definiert. Diese Operationen bilden die Schnittstelle zum objektorientierten Bauwerksmodell – die Modellschnittstelle. Im Hinblick auf die Verarbeitung der Daten sind Methoden semantisch hochwertiger als die Attribute selbst. Des Weiteren besitzen Operationen im Sinne der Verarbeitung einer Modellinstanz eine höhere Aussagekraft als Objektmethoden. Das Bild 3.30 veranschaulicht diesen Zusammenhang.

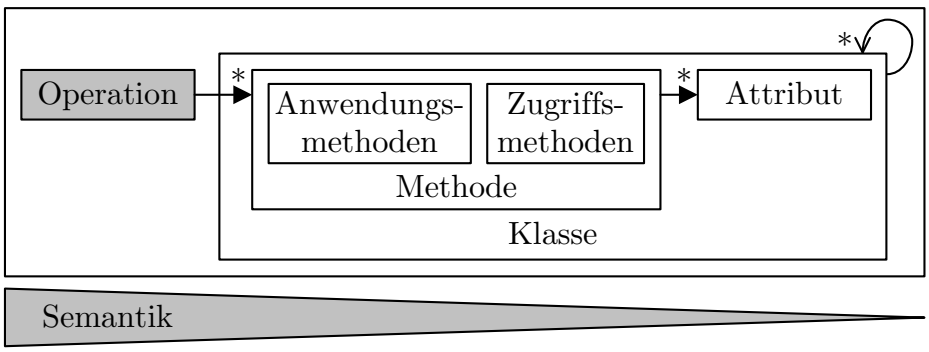


Bild 3.30: Semantik im objektorientierten Modell

3.3.3 Konsistenz

Konsistenz: Im Rahmen der Betrachtungen in dieser Arbeit wird mit Konsistenz die *Widerspruchsfreiheit* innerhalb eines Systems verstanden. Das System ist in diesem Fall die Modellinstanz. Es wird zwischen struktureller und inhaltlicher Konsistenz sowie zwischen lokaler und globaler Konsistenz unterschieden. Strukturelle Konsistenz meint die Widerspruchsfreiheit innerhalb der Datenstruktur, während inhaltliche Konsistenz auf die Widerspruchsfreiheit in der Bedeutung der Daten abzielt. Die lokale Konsistenzbedingung ist erfüllt, wenn jeweils die Eigenschaften eines Modellobjekts widerspruchsfrei sind. Globale Konsistenz hingegen meint die Widerspruchsfreiheit aller Objektzustände untereinander innerhalb einer Modellinstanz.

Beispielhaftes Modell: Im Bild 3.31 ist das UML-Klassendiagramm eines einfachen Modells zur Beschreibung eines *Balkens* mit einer *Gleichlast* dargestellt. Die nichtnegative Länge l des Balkens wird durch die Methode *setL(s)*, der Wert w der Gleichlast wird durch die Methode *setW(w)* geändert. Zur Bemessung des Balkens speichert die *Statik* das maximale Biegemoment des Balkens als Attribut m. Die Methode *updateM()* aktualisiert den Wert des Biegemoments, indem die Methode *getM()* des Balkens zur Berechnung gerufen wird. Dieses einfache Beispiel dient als Grundlage für die darauffolgenden Konsistenzbetrachtungen.

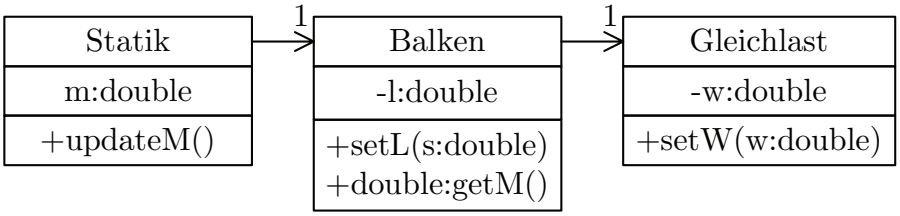


Bild 3.31: Einfaches objektorientiertes Modell

Datenebene: In objektorientierten Modellen werden zur Speicherung von *Daten* Attribute verwendet. Jedes Attribut ist einem bestimmten Datentyp zugeordnet. Bei der Übersetzung eines objektorientierten Programmcodes sorgt der Compiler für die strukturelle Konsistenzsicherung auf Datenebene, indem er eine Typüberprüfung vornimmt.

Beispiel 3.33: Konsistenz auf Datenebene

Das Bild 3.32 veranschaulicht die Datenkonsistenz des Balkens. Der Zustand 1 des Balkenattributes ist konsistent. Der fiktive Attributzustand 2 stellt dagegen auf Datenebene einen inkonsistenten Zustand dar, da der Wert des Attributes nicht dem definierten Datentyp (double) entspricht.

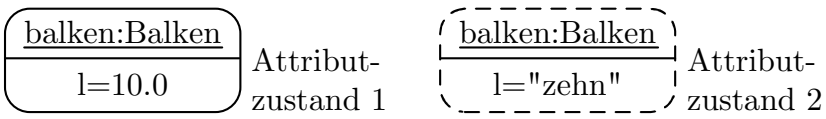


Bild 3.32: Konsistenz auf Datenebene

Objektebene: Dem objektorientierten Prinzip der Datenkapselung[5] entsprechend werden die Attribute vor unerlaubtem Zugriff geschützt. Der innere Aufbau und die Bedeutung der Attribute bleiben in der Regel dem Entwickler vorbehalten. Die Verwendung der Attribute erfolgt im Allgemeinen über *Methoden*. Innerhalb der Methoden werden Attribute in einer Transaktion verarbeitet, indem die Werte auf Gültigkeit überprüft und abhängige Attribute innerhalb des Objekts aktualisiert werden. Ein Objekt ist vor und nach der Ausführung einer Methode konsistent, kann innerhalb einer Methode jedoch kurzzeitig einen inkonsistenten Zustand einnehmen. Die Methoden eines Objekts stellen somit die lokale, inhaltliche Konsistenz innerhalb des Objekts – auf Objektebene – sicher.

Beispiel 3.34: Konsistenz auf Objektebene

Das Bild 3.33 zeigt ein UML-Zustandsdiagramm für das Balkenobjekt *balken*. Nach der Instanziierung (Objektzustand 1) und nach der Skalierung der Länge (Objektzustand 2) befindet sich das Balkenobjekt in einem konsistenten Zustand. Der Methodenaufruf *setL(-2)* ändert die Länge des Balkens auf einen negativen Wert (Objektzustand 3), was vereinbarungsgemäß einen inkonsistenten Zustand darstellt.

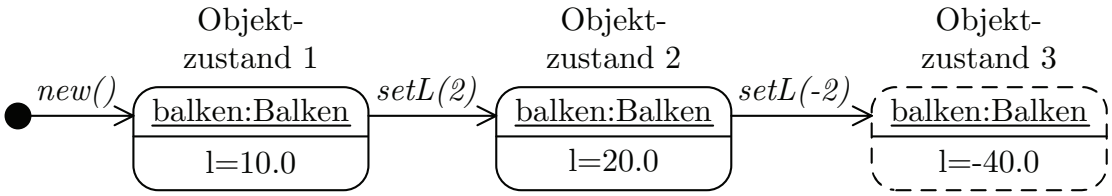


Bild 3.33: Konsistenz auf Objektebene

Diese Inkonsistenz auf Objektebene im Objektzustand 3 lässt den Schluss zu, dass der Methode *setL(s)* die Aufgabe zukommen muss, den Wert der Länge l zu überprüfen, bevor dieser als Attributwert gespeichert wird, um die inhaltliche Konsistenz des Balkenobjekts zu sichern.

Transaktionskonzept: Das aus der Datenbanktechnologie bekannte ACID-Transaktionskonzept[6] wird auf das verarbeitungsorientierte Modell übertragen. Die Ausführung einer Operationsinstanz stellt als kleinste Verarbeitungseinheit eine Transaktion mit entsprechenden Eigenschaften dar:

Atomicity (Atomarität): Eine Operationsinstanz wird entweder vollständig oder gar nicht ausgeführt.

Consistency (Konsistenz): Eine Operationsinstanz verarbeitet die Modellinstanz, indem sie diese von einem konsistenten Zustand in den nächsten konsistenten Zustand überführt, ohne jedoch zu jedem dazwischenliegenden Zeitpunkt die Konsistenz zu gewährleisten.

[5] auch Attributkapselung oder Sichtbarkeits- bzw. Geheimnisprinzip, s. [Booch 1994] und [Balzert 2005]

[6] Atomicity, Consistency, Isolation, Durability, deutsch: Atomarität, Konsistenz, Isolation, Dauerhaftigkeit, s. [Date 2000, S. 459]

Isolation (Isolation): Operationsinstanzen werden voneinander unabhängig – isoliert – betrachtet. Das bedeutet, dass sich in Ausführung befindliche Operationsinstanzen nicht gegenseitig beeinflussen.

Durability (Dauerhaftigkeit): Das Ergebnis einer erfolgreich ausgeführten Operationsinstanz ist dauerhaft. Im Falle des Systemabsturzes einer Fachapplikation ist zwar nicht der Zustand der Modellinstanz, aber die Operationsinstanz als Teil einer Änderung persistent gespeichert. So kann bei Bedarf ein Zustand der Modellinstanz wiederhergestellt werden.

Modellinstanzebene: Im vorgeschlagenen verarbeitungsorientierten Modellierungsansatz wird innerhalb einer Transaktion die Modellinstanz von *Operation*sinstanzen verarbeitet, indem diese die Zustände der entsprechenden Modellobjekte ändern. Eine Modellinstanz befindet sich vor und nach der Ausführung einer Operationsinstanz in einem konsistenten Zustand, kann aber während der Ausführung zeitweise inkonsistent werden. Dem Prinzip der Datenkapselung folgend, kapseln die Operationsinstanzen die Modellobjekte und sorgen somit für die Sicherstellung der globalen, inhaltlichen Konsistenz auf Modellinstanzebene. Den Operationen kommt dabei die Aufgabe zu, die Modellinstanz als Ganzes widerspruchsfrei zu halten, indem Abhängigkeiten innerhalb der Modellinstanz aktualisiert werden. Das Prinzip der Modellkapselung zur Konsistenzsicherung wird im Bild 3.34 veranschaulicht.

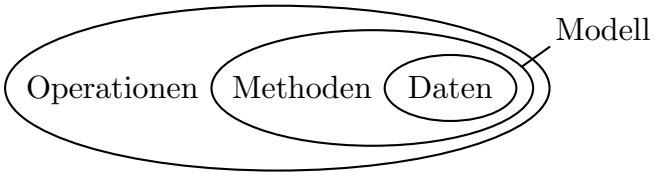


Bild 3.34: Daten- und Modellkapselung zur Konsistenzsicherung

Beispiel 3.35: Konsistenz auf Modellinstanzebene

Das Bild 3.35 auf der nächsten Seite zeigt ein UML-Zustandsdiagramm für die Modellinstanz. Nach der Erzeugung befindet sich die Modellinstanz in einem konsistenten Zustand (Modellinstanzzustand 1). Die Operationsinstanz –*Wert der Gleichlast auf 4 kN/m setzen*– ruft die Methode *setW(4)* des Gleichlastobjekts, um den Wert der Last auf 4.0 zu setzen. Das Gleichlastobjekt *last* befindet sich in einem lokal, inhaltlich konsistenten Zustand. Die gesamte Modellinstanz ist jedoch inhaltlich inkonsistent, da der Wert des Biegemoments *m* im Statikobjekt *statik* nicht dem aktuellen Modellinstanzzustand entspricht (Modellinstanzzustand 2).
Das Beispiel verdeutlicht, dass die Operationsinstanz für die globale, inhaltliche Konsistenz der Modellinstanz verantwortlich sein muss. Innerhalb der Operationsinstanz – *Wert der Gleichlast auf 4 kN/m setzen* – kann nicht nur die Methode *setW(4)* gerufen werden, sondern zusätzlich muss auch der Zustand des Statikobjekts *statik* durch Aufruf der Methode *updateM()* aktualisiert werden. Erst dann befindet sich die Modellinstanz global in einem konsistenten Zustand.

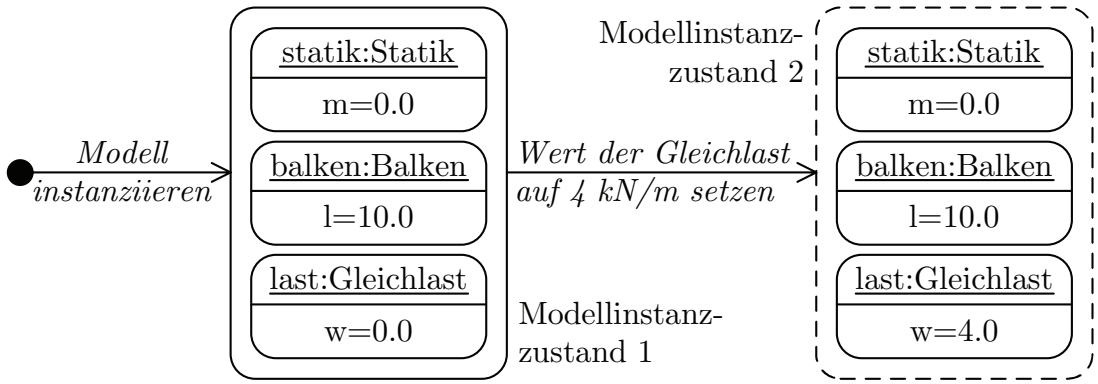


Bild 3.35: Konsistenz auf Modellinstanzebene

Fachliche Ebene: Eine Modellinstanz befindet sich aus fachlicher Sicht in einem konsistenten Zustand, wenn ingenieurtechnische Nachweise[7] auf Basis der abstrahierten fachlichen Instanz erfüllt werden. Die Sicherung der fachlichen Konsistenz erfolgt – zum Teil auch rechnergestützt[8] – auf Grundlage von Ingenieurwissen. Diese Form der Konsistenzsicherung wird in dieser Arbeit nicht weiter betrachtet.

Beispiel 3.36: Konsistenz auf fachlicher Ebene

Für den Nachweis der Gebrauchstauglichkeit darf ein Balken eine maximale Durchbiegung nicht überschreiten. Die fachliche Instanz befindet sich demnach in einem inkonsistenten Zustand, wenn die auf Basis des maximalen Biegemoments ermittelte Durchbiegung des Balkens größer ist als der in einer bauartspezifischen Norm festgelegter Grenzwert.

Zusammenfassung: In der Objektorientierung stellen *Methoden* die inhaltliche Konsistenz auf *Objektebene* sicher, indem sie die in Attributen gespeicherten Daten kapseln. Die in der Änderungsorientierung eingeführten *Operationen* erweitern das objektorientierte Prinzip der Datenkapselung. Sie dienen der Sicherung der inhaltlichen Konsistenz auf *Modellinstanzebene*, indem sie die Modellobjekte kapseln. Dieses Prinzip wird als *Modellkapselung* bezeichnet. In der Tabelle 3.2 sind zusammenfassend die Konsistenzebenen und die zugeordneten Konsistenzsicherer aufgeführt.

Konsistenzebene	**Konsistenzsicherung**
Daten/ Attribute	Compiler
Objekt	Methoden
Modellinstanz	Operationen
Fachliche Instanz	Ingenieurtechnische Nachweise

Tabelle 3.2: Konsistenzebenen und Konsistenzsicherung

[7] bspw. Nachweise der Tragfähigkeit und der Gebrauchstauglichkeit (vgl. [Schneider 2002])

[8] [Hartmann 2007] beschäftigt sich mit der Nachweisführung auf der Grundlage wissensbasierter Modellierungsansätze

3.4 Fachapplikationen

Dem Stand der Technik entsprechend[9] werden im Prozess der rechnergestützten Bauwerksplanung objektorientierte Fachapplikationen eingesetzt, die Bauwerksmodelle instanziieren und verarbeiten. In diesem Abschnitt wird der verarbeitungsorientierte Ansatz im Hinblick auf die Fachapplikationen beleuchtet. Dabei wird sowohl auf die Stellung als auch die Bedeutung der Operationen im Kontext von Fachapplikationen eingegangen.

3.4.1 Komponenten

Modellkomponente: Eine Fachapplikation instanziiert das objektorientierte Modell innerhalb der Modellkomponente. Dem verarbeitungsorientierten Ansatz folgend werden die Zustände der Modellinstanz B – die *Modellinstanzversionen*[10] – vorgehalten. Das Laden und das persistente Speichern von Modellinstanzversionen werden der Modellkomponente zugeordnet. Verfügbare Fachapplikationen besitzen im Allgemeinen Funktionalität zum Laden und zum Speichern von Dokumenten im nativen Dateiformat. Eine Modellinstanzversion entspricht somit einem Dokument, was einen ausgezeichneten Zustand der Modellinstanz beschreibt.

Verarbeitungskomponente: Die Verarbeitungskomponente einer Fachapplikation verarbeitet die Modellinstanz, indem sie Operationen instanziiert und auf die Modellinstanz anwendet. Der Prozess der Verarbeitung wird auch *Modellieren* genannt. Aus diesem Grund werden im Folgenden die Begriffe Operation und *Modellieroperation* synonym verwendet. Ein Merkmal des verarbeitungsorientierten Ansatzes ist das Vorhalten von *Änderungen* $\delta \in \Delta$[11] der Modellinstanz B. Das Aufzeichnen von Änderungen im Modellierungsprozess und das Einspielen bzw. Anwenden von Änderungen sind Aufgabe der Verarbeitungskomponente. Persistente Änderungen werden in Änderungsdateien gespeichert. Die Operationen bilden den Kern der Verarbeitungskomponente und stellen die Schnittstelle zwischen der Modellkomponente und der Ein-/Ausgabekomponente einer Fachapplikation dar.

Ein-/Ausgabekomponente: Die Benutzerschnittstelle einer Fachapplikation nimmt die Eingabe des Benutzers entgegen, zeigt den Modellinstanzzustand in der Ausgabe an und wird im Rahmen dieser Arbeit als Ein-/Ausgabekomponente bezeichnet. Die Eingabe des Benutzers wird zu einer Operationsinstanz ausgewertet und auf die Modellinstanz angewendet.

Fachapplikation: Aus Sicht des verarbeitungsorientierten Modellierungsansatzes besteht eine Fachapplikation aus drei Komponenten: *Ein-/Ausgabe*, Verarbeitung und *Modell*. Durch die *Operationen* kommt der Verarbeitungskomponente eine besondere Bedeutung zu. Sie dienen (a) der Verarbeitung/ Modellierung und (b) der Beschreibung von Änderungen einer Modellinstanz. In einer Fachapplikation bilden die Operationen die Schnittstelle zwischen der den Benutzer repräsentierenden Eingabe und der

[9] vgl. Kapitel 2 auf Seite 9
[10] vgl. Abschnitte 3.1.2 S. 38 und 3.2.1 S. 49
[11] vgl. Abschnitte 3.1.3 S. 42 und 3.2.1 S. 50

zu verarbeitenden Modellinstanz. Das Bild 3.36 veranschaulicht die Architektur einer Fachapplikation und hebt die Stellung der Operationen als Verarbeitungskomponente heraus.

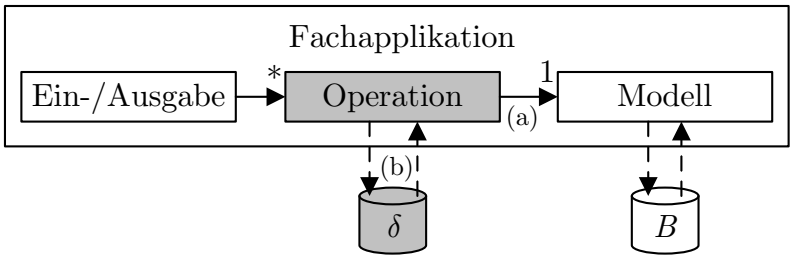


Bild 3.36: Operationen in einer Fachapplikation

3.4.2 Modellieroperationen

Modellieren: Eine Operation stellt die kleinste fachtypische Verarbeitungseinheit bei der Manipulation einer Modellinstanz in einer Fachapplikation dar. Der Benutzer einer Fachapplikation instanziiert eine Operation, um damit zu *modellieren*. Für eine bestimmte fachliche Planungsaufgabe stehen dem Planer in einer Fachapplikation festgelegte *Modellieroperationen* zur Verfügung. Die Anwendung einer instanziierten Modellieroperation auf die Modellinstanz ist Aufgabe der Verarbeitungskomponente.

Operationsgruppierung: Modellieroperationen können generell in modellverändernde und nicht-modellverändernde Operationen unterschieden werden. Die modellverändernden Modellieroperation werden wie folgt gruppiert:

Hinzufügen (ADD)**:** Modellobjekte werden in einem bestimmten Verarbeitungskontext erzeugt und der Modellinstanz hinzugefügt.

Modifizieren (MODIFY)**:** Selektierte Modellobjekte werden modifiziert, das heißt, ihr Zustand wird verändert.

Löschen (REMOVE)**:** Selektierte Modellobjekte werden aus der Modellinstanz entfernt.

Konfigurieren (SET)**:** Der Verarbeitungskontext wird konfiguriert.

Sonstige (MISC)**:** Modellieroperationen werden rückgängig gemacht (undo), wiederhergestellt (redo) oder wiederholt (again).

Nicht-modellverändernde Modellieroperationen manipulieren die Modellinstanz nicht direkt, dienen jedoch zum einen dazu, Modellobjekte für anschließende Modifikationen auszuwählen bzw. zu selektieren und zum anderen, Information über den Zustand der Modellinstanz und die Zustände der Modellobjekte abzufragen.

Selektieren/ Deselektieren (SELECT/UNSELECT)**:** Modellobjekte werden zur Verarbeitung selektiert und nach dem Abschluss von Modifikationen deselektiert.

Abfragen (GET)**:** Der Zustand der Modellinstanz oder der Zustand der selektierten Modellobjekte wird abgefragt.

Verarbeitungsprozess: Der Prozess der Verarbeitung einer Modellinstanz wird vom Benutzer gesteuert. Folgende Eigenschaften kennzeichnen die Arbeitsweisen eines Fachplaners:

- Der Planer arbeitet immer in einem fest definierten, aber modifizierbaren Verarbeitungskontext. Beim Hinzufügen eines Modellobjekts werden nicht alle Eigenschaften des Objekts vom Fachplaner spezifiziert. Alle nicht spezifizierten Eigenschaften werden dem aktuellen Verarbeitungskontext entnommen.
- Zur gezielten Modifikation einzelner Modellobjekte geht der Planer wie folgt vor. Er selektiert zunächst die zu modifizierenden Modellobjekte. Dann ändert er ihre Eigenschaften. Ist die Modifikation abgeschlossen, hebt er die Selektion auf.

Verarbeitungsregeln: Die beschriebene Vorgehensweise unterliegt gewissen Verarbeitungsregeln. Modellobjekte stehen nur dann für eine Modifikation, das Löschen und eine Abfrage zur Verfügung, wenn sie zuvor hinzugefügt und anschließend selektiert worden sind. Weiterhin müssen Kontexteigenschaften verfügbar sein, um sie bei Konfigurationen verwenden zu können. Das Bild 3.37 zeigt die Gruppen von Modellieroperationen und veranschaulicht gleichzeitig ihre *Anwendungsreihenfolge* im Hinblick auf die Verarbeitung von Modellobjekten.

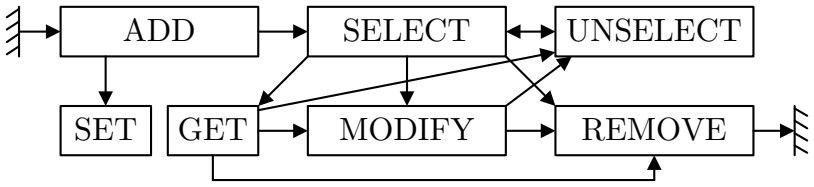


Bild 3.37: Gruppierung und Anwendungsreihenfolge von Modellieroperationen

Beispiel 3.37: Verarbeitungskontext und Modellobjekte im Verarbeitungsprozess

Mit einer fiktiven Fachapplikation werden ebene Stabwerke berechnet. *Knoten*, *Stäbe*, *Auflager* und *Lasten* sind Modellobjekte mit entsprechenden Attributen. Der Verarbeitungskontext der Fachapplikation ist durch die aktuell eingestellten Eigenschaften für das *Material*, den *Querschnitt* und die *Stabendgelenke* festgelegt. Ein beispielhafter Verarbeitungsprozess zur Modellierung eines Dreigelenkrahmens wird im Bild 3.38 abgebildet und anschließend beschrieben.

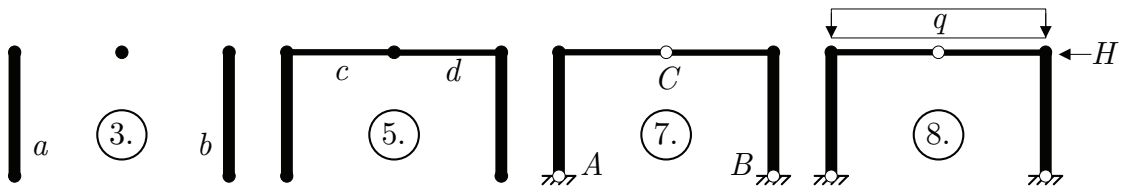


Bild 3.38: Verarbeitungsprozess

1. Die Fachapplikation wird gestartet, Material- und Querschnittsbibliotheken werden geladen. (ADD) – Es findet eine Initialisierung des Verarbeitungskontexts statt, indem ein Standardmaterial, ein Standardquerschnitt und ein Standardstabendgelenk eingestellt werden. (SET)

2. Der Fachplaner konstruiert die Knoten des Stabwerks. (ADD)

3. Der Fachplaner erzeugt zwei vertikale Stäbe a und b, indem er die Stabendknoten spezifiziert. Das Material, der Querschnitt und die Stabendgelenke werden dem aktuellen Kontext entsprechend den Stäben zugeordnet. (SELECT, ADD, UNSELECT)

4. Der Fachplaner ändert den Verarbeitungskontext, indem er einen anderen aktuellen Querschnitt einstellt. (SET)

5. Der Fachplaner erzeugt auf Basis der vorhandenen Knoten zwei horizontale Stäbe c und d, die entsprechend des geänderten Verarbeitungskontexts jeweils ein Material, einen (anderen) Querschnitt und Stabendgelenke erhalten. (SELECT, ADD, UNSELECT)

6. Nach Auswahl der Knoten fügt der Fachplaner je ein Gelenklager am Anfang A des Stabes a und am Ende B des Stabes b ein. (SELECT, ADD, UNSELECT)

7. Der Fachplaner selektiert den Stab c. (SELECT) – Anschließend ändert er das rechte Stabendgelenk C in eine gelenkige Verbindung zum angeschlossenen Stab. (MODIFY) – Der Stab c wird deselektiert. (UNSELECT)

8. Der Fachplaner fügt eine vertikale Gleichlast q und eine horizontale Einzellast H hinzu. (ADD) – Der Dreigelenkrahmen ist modelliert und der Planer startet die Berechnung. (GET)

3.4.3 Persistente Änderungen

Änderungen: Eine Folge von im Verarbeitungsprozess instanziierten und angewendeten Modellieroperationen wird als Änderung der Modellinstanz betrachtet. Änderungen werden von der Verarbeitungskomponente einer Fachapplikation in persistenter Form aufgezeichnet und auch wieder ausgeführt bzw. angewendet. Diese Vorgänge werden im Folgenden *Journaling* bzw. *Patching* genannt und sind im Bild 3.39 veranschaulicht.

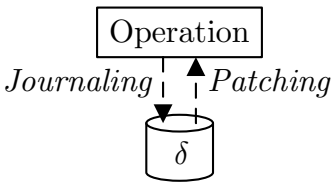


Bild 3.39: Journaling und Patching

Journaling: Das Aufzeichnen persistenter Änderungen im Verarbeitungsprozess einer Modellinstanz mit einer Fachapplikation wird als Journaling[12] definiert. Eine persistente Änderung ist eine Folge von Operationsinstanzen, welche die Modellinstanz von einer Version in die darauffolgende Version überführt. Während der Planer die Modellinstanz mit einer Fachapplikation verarbeitet, werden Operationsinstanzen entsprechend des operativen Modells persistent gespeichert. Die Modellobjekte werden durch ihre persistenten Objektidentifikatoren (POIDs) als Operanden gekennzeichnet.

Patching: Das Einspielen bzw. Anwenden von gespeicherten Änderungen auf eine Modellinstanz in einer Fachapplikation wird als Patching[13] definiert. Die in den persistenten Änderungen gespeicherten Operationsinstanzen werden in der Verarbeitungskomponente der Fachapplikation sequenziell auf die Modellinstanz angewendet, um einen bestimmten Zustand der Modellinstanz – eine Modellinstanzversion – zu erzeugen.

[12] engl.: journal, deutsch: Protokoll; vgl. Journal-Dateien zum Abspeichern nativer Befehlsfolgen in CAD-Systemen, s. [Beucke 2002]

[13] engl.: patch, deutsch: reparieren, flicken; Analogie: Die Aktualisierung eines Betriebssystems durch Softwareupdates wird auch Patching genannt.

4 Operative Modellierungssprache

"Language is a purely human and noninstinctive method of communicating ideas, emotions, and desires by means of a system of voluntarily produced symbols."
Edward Sapir (1884–1939), aus [Sapir 1921]

In diesem Kapitel wird eine Sprache zur Beschreibung von Modellieroperationen in der verarbeitungsorientierten Bauwerksmodellierung vorgestellt. Nach der Einführung einiger Grundlagen werden, ausgehend von den Anforderungen an eine operative Modellierungssprache, drei aufeinander aufbauende Sprachebenen unterschieden. Es wird untersucht, wie die Sprache zur Formulierung von Modellierprogrammen, Modellieroperationen und Änderungen angewendet werden kann. Für jede dieser Ebenen wird die Sprachsyntax formal durch eine Grammatik definiert und anhand von Beispielen verdeutlicht.

4.1 Grundlagen

4.1.1 Sprachen

Sprachen werden in natürliche Sprachen und künstliche Sprachen eingeteilt. Während Deutsch und Englisch natürliche Sprachen darstellen, zählen Programmiersprachen wie Java oder HTML zu den künstlichen Sprachen. Formal beschreibbare Sprachen werden formale Sprachen genannt. Formale Sprachen sind eine Teildisziplin der Mathematik und stellen ein eigenständiges Wissensgebiet in der theoretischen Informatik dar. Um Sprachen beschreiben zu können, werden zunächst nachfolgende Begriffe eingeführt und deren Bedeutung erklärt.

Lexik: Die Gesamtheit der Wörter einer natürlichen Sprache bzw. die Gesamtheit der Symbole einer formalen Sprache wird Vokabular oder Lexik genannt (*lex*).

Syntax: Die Syntax[1] ist die Lehre vom Satzbau einer natürlichen Sprache bzw. vom Wortbau einer formalen Sprache (*syn*).

Semantik: Die Lehre von der inhaltlichen Bedeutung einer natürlichen oder formalen Sprache ist die Semantik[2] (*sem*).

Die nachfolgenden Definitionen zur Beschreibung einer formalen Sprache lehnen sich an [Broy 1998a, S. 26 ff.] an.

[1] griech.: syntaxis, deutsch: die Zusammenstellung
[2] altgriech.: semantikos, deutsch: anzeigend

Symbol: Ein Symbol ist ein definiertes Zeichen zur Darstellung einer Information in einer Sprache.

Alphabet: Ein Alphabet C beschreibt eine endliche Menge von Symbolen.

$$C := \{c \mid c \text{ ist ein Symbol}\} \tag{4.1}$$

Wort: Eine endliche Folge $\langle c_1, c_2, \ldots, c_n \rangle$ von Symbolen eines Alphabets C heißt Wort w^n der Länge n. Das Wort der Länge 0 heißt leeres Wort ε (nicht zu verwechseln mit der Ursprungsversion ε der Zusammenführungsmenge S aus Gleichung 3.28 auf Seite 57).

$$w^n := \langle c_1, c_2, \ldots, c_n \rangle : \bigwedge_{i=1}^{n} (c_i \in C) \tag{4.2}$$

$$\varepsilon := w^0 \tag{4.3}$$

Wortmenge: Die Menge aller endlichen Wörter über einem Alphabet C heißt Wortmenge[3] C^*. Alle Wörter der Länge n über einem Alphabet C werden in der Wortmenge C^n zusammengefasst. Dabei gilt $C^0 = \{\varepsilon\}$.

$$C^* := \bigcup_{k \in \mathbb{N}_0} C^k = C^0 \cup C^1 \cup C^2 \cup \ldots \cup C^k \tag{4.4}$$

Formale Sprache: Jede Teilmenge $L \subset C^*$ von Wörtern über einem Alphabet C heißt formale Sprache über C.

$$L := \{w \mid w \text{ ist ein Wort der Sprache } L \text{ über dem Alphabet } C\} \subset C^* \tag{4.5}$$

Für eine formale Sprache wird ein Alphabet definiert, welches die Menge der Symbole darstellt. Durch eine Syntax wird die Aneinanderreihung dieser Symbole zu Wörtern beschrieben. Haben die Symbole und die syntaktischen Regeln einer formalen Sprache eine Bedeutung, dann können formale Aussagen formuliert werden. Das Bild 4.1 verdeutlicht diesen Zusammenhang.

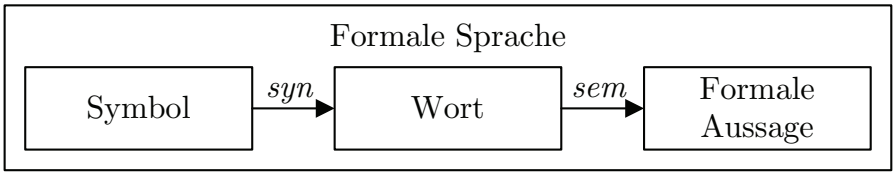


Bild 4.1: Formale Sprache

Beispiel 4.1: Formale Sprache

Für eine formale Sprache wird das Alphabet $C = \{\texttt{'3'}, \texttt{'44'}, \texttt{'='}\}$ definiert. Unter Verwendung der Syntax der Formelsprache der Mathematik können nun beispielsweise

[3] oder Wortschatz

die nachfolgenden zwei Wörter gebildet werden, die jeweils eine Aussage darstellen. Aussagen können wahr oder falsch sein.

Wort 1 : `3 = 44`
Wort 2 : `3 = 3`

Natürliche Sprache: Zur Beschreibung einer natürlichen Sprache sind im Grunde zwei formale Sprachen notwendig. Eine formale Sprache beschreibt die Lexik der natürlichen Sprache als Menge der Buchstabenfolgen (Wörter) über dem Buchstabenalphabet der natürlichen Sprache. Die zweite formale Sprache betrachtet die Syntax der natürlichen Sprache, indem sie die Menge an gültigen Wortfolgen (Sätzen) über dem Wortalphabet beschreibt. Im Allgemeinen existiert für eine natürliche Sprache ein Buchstabenalphabet, welches die Menge der Buchstaben definiert. Aus diesen Buchstaben werden Wörter gebildet. Das Vokabular oder die Lexik einer natürlichen Sprache ist meistens in Wörterbüchern zu finden. Die syntaktischen Regeln einer natürlichen Sprache legen den Satzbau fest, um mit Wörtern korrekte Sätze zu bilden. Wird sowohl den einzelnen Wörtern als auch der Aneinanderreihung von Wörtern eine Bedeutung zugeordnet, dann können Menschen mit Sätzen kommunizieren. Diesen Sachverhalt veranschaulicht das Bild 4.2.

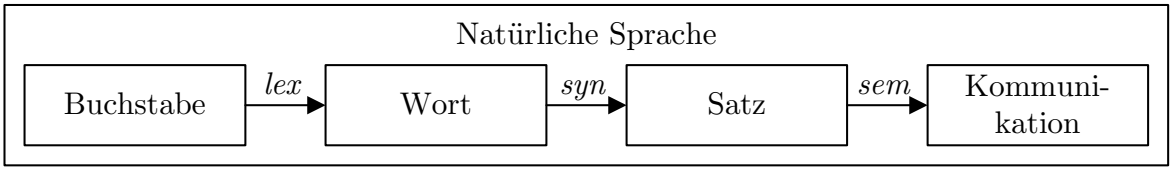


Bild 4.2: Natürliche Sprache

Programmiersprache: Eine Programmiersprache ist eine formale Sprache und dient dazu, mit Hilfe von alphanumerischen Zeichen ein Computerprogramm zu formulieren. Zur Beschreibung der Lexik einer Programmiersprache werden gültige alphanumerische Zeichen definiert, die nach lexikalischen Regeln zu Terminalsymbolen (s. Abschnitt 4.1.2) der Programmiersprache zusammengesetzt werden. Auf der Basis syntaktischer Regeln werden anschließend mit Terminalsymbolen Programme formuliert. Im Sinne einer formalen Sprache ist ein Programm ein Element der Programmiersprache und entspricht somit einem Wort (vgl. Gleichung 4.5 auf Seite 76). Sind die Bedeutung der Terminalsymbole und die Bedeutung syntaktischer Regeln festgelegt, kann ein Programm vom Computer ausgeführt werden und Informationen verarbeiten. Diesen Sachverhalt verdeutlicht Bild 4.3.

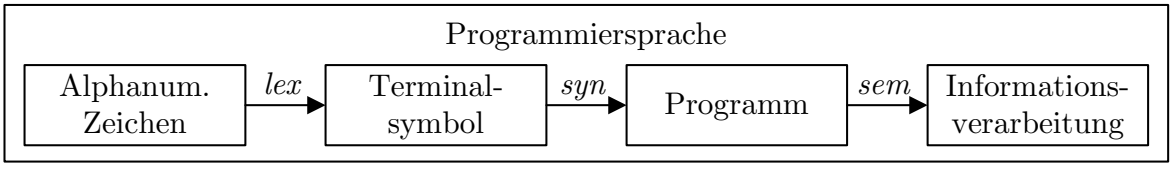


Bild 4.3: Programmiersprache

Grammatik einer Programmiersprache: Die formale Beschreibung der Lexik und der Syntax einer Programmiersprache durch lexikalische bzw. syntaktische Regeln wird Grammatik genannt. Die meisten Programmiersprachen sind der Klasse der sogenannten kontextfreien[4] Sprachen zugeordnet und besitzen dementsprechend kontextfreie Grammatiken. Zur einfachen Darstellung kontextfreier Grammatiken wird die BNF-Notation herangezogen.

4.1.2 BNF-Notation

Für die formale Beschreibung von Grammatiken kontextfreier Sprachen wird neben Syntaxdiagrammen[5] auch die BNF-Notation[6] (Backus-Naur-Form) verwendet. Die BNF stellt einen Formalismus dar, mit dem auf Basis *regulärer Ausdrücke* sowohl lexikalische als auch syntaktische Regeln einer Programmiersprache formuliert werden.

Regulärer Ausdruck: Ein regulärer Ausdruck ist eine Notation, mit der Mengen von Symbolfolgen beschrieben werden können. Während bei der Definition von lexikalischen Regeln Zeichenfolgen betrachtet werden, definieren syntaktische Regeln Folgen von Symbolen. Nachfolgend aufgeführte reguläre Ausdrücke werden in dieser Arbeit verwendet.

`a	b	c`	: Entweder Symbol `a` oder `b` oder `c`
`~[a]`	: Nicht Symbol `a`
`[a - z]`	: Symbole `a` bis `z`
`( a )?`	: Entweder 0 oder 1 mal Symbol `a`
`( a )+`	: Mindestens 1 mal Symbol `a`
`( a )*`	: Beliebig oft Symbol `a`

Beispiel 4.2: Regulärer Ausdruck

Gegeben ist eine Symbolmenge $C = \{$`'x'`, `'y'`, `'z'`$\}$. Zur Beschreibung von gültigen Symbolfolgen wird nachfolgender regulärer Ausdruck definiert.

```
~[z] ( x )+ ( y | z )?
```

Auf der Basis können nun auszugsweise folgende Symbolfolgen gebildet werden:

Symbolfolge 1 : `xx`
Symbolfolge 2 : `yxxxz`
Symbolfolge 3 : `yxy`

Terminalsymbol: Ein Terminalsymbol (kurz: Terminal) wird mit Hilfe von lexikalischen Regeln auf der Basis von regulären Ausdrücken mit alphanumerischen Zeichen definiert und stellt ein syntaktisches Symbol der zu beschreibenden Programmiersprache dar. Terminalsymbole einer Programmiersprache sind Schlüsselwörter, Kommentare, Bezeichner, Konstanten, Zahlen, Zeichenketten, Wahrheitswerte und Operatoren. In BNF-Notation werden Terminalsymbole wie folgt definiert.

[4] auch Typ-2-Sprache in der Chomsky-Hierarchie, s. [Broy 1998b, S. 212 ff.]
[5] s. [Gumm u. Sommer 2002, S. 121 ff.] und [Broy 1998a, S. 82 ff.]
[6] s. [Broy 1998a, S. 78 ff.] und [Aho u. a. 1988, S. 26 ff.]

```
< TERMINAL : ... >
```

Nichtterminalsymbol: Ein Nichtterminalsymbol (kurz: Nichtterminal oder Nonterminal) ist eine syntaktische Hilfseinheit oder eine Variable, die eingeführt wird, um syntaktische Regeln zu formulieren.

Startsymbol: In einer BNF-Grammatik wird ein Nichtterminalsymbol als sogenanntes Startsymbol ausgezeichnet.

Produktionsregel: Eine Regel zur Beschreibung eines eingeführten Nichtterminalsymbols wird Produktionsregel (kurz: Produktion) genannt. Bei einer Produktion steht auf der linken Seite das Nichtterminalsymbol und auf der rechten Seite ein regulärer Ausdruck, der eine Symbolfolge aus Terminalsymbolen und anderen Nichtterminalsymbolen definiert. Produktionsregeln stellen die syntaktischen Regeln bei der Definition einer Grammatik dar. In BNF-Notation sieht eine Produktionsregel beispielsweise wie folgt aus.

```
Nonterminal -> <TERMINAL> | OtherNonterminal
```

Grammatik: Als Grammatik zur Beschreibung der Syntax einer Programmiersprache wird ein Gebilde aus Terminalsymbolen, Nichtterminalsymbolen, Produktionen und einem Startsymbol verstanden. Ausgehend von dem Startsymbol können alle Programme einer Programmiersprache durch Anwendung der Produktionsregeln abgeleitet werden.

Beispiel 4.3: Einfache Grammatik in BNF

Es wird eine Grammatik zur Formulierung einer einfachen Gleichung definiert. Beispielsweise besteht die Gleichung der Form $a + b - c * d + e : f \ldots + y = z$ aus einer beliebigen Anzahl von Operanden, den Operatoren für Addition, Subtraktion, Multiplikation und Division, dem Gleichheitsoperator und einem Ergebnis. Dazu werden die Terminalsymbole `<ZAHL>`, `<ADD>`, `<SUB>`, `<MUL>`, `<DIV>` und `<GLEICH>` durch lexikalische Regeln als Zeichenfolgen definiert. Darüber hinaus werden die Nichtterminalsymbole `Gleichung`, `Operand`, `SumOper`, `MulOper` und `Ergebnis` zur Formulierung der Produktionsregeln verwendet. Das Nichtterminalsymbol `Gleichung` bildet das Startsymbol der Grammatik.

```
// Lexikdefinition (Zeichenfolgen): Terminalsymbole
< ZAHL   : ["1"-"9"] ( ["0"-"9"] )* >
< ADD    : "+" >
< SUB    : "-" >
< MUL    : "*" >
< DIV    : ":" >
< GLEICH : "=" >

// Syntaxdefinition (Symbolfolgen): Nichtterminalsymbole
Gleichung -> Operand ( SumOper Operand )*
             <GLEICH> Ergebnis
Operand   -> <ZAHL> | Produkt
```

```
SumOper   -> <ADD > | <SUB>
MulOper   -> <MUL>  | <DIV>
Produkt   -> <ZAHL> MulOper <ZAHL>
Ergebnis  -> <ZAHL>
```

Auf Basis dieser Grammatik kann beispielsweise die nachfolgende Gleichung formuliert werden. Diese Gleichung ist ein Element der durch die Grammatik beschriebenen Sprache. Wird diese Sprache als Programmiersprache betrachtet, dann stellt die Gleichung ein gültiges Programm dar.

$$34 - 3 * 6 + 13 = 29$$

4.2 Anforderungen an die Modellierungssprache

Grundsätzlich wird zwischen zwei Anforderungen im Hinblick auf die Verwendung der operativen Modellierungssprache unterschieden. Modellieroperationen werden einerseits in der alphanumerischen Schnittstelle[7] einer Fachapplikation vom Planer instanziiert und auf die Modellinstanz angewendet, um diese zu bearbeiten (*Eingabe*). Andererseits dienen Modellieroperationen der persistenten Beschreibung von Änderungen (*Persistenz*), die während der Verarbeitung einer Modellinstanz mit einer Fachapplikation vorgenommen werden.

4.2.1 Eingabe

Beschreibung von Modellieroperationen: Für eine Fachapplikation stehen eine Reihe von Modellieroperationen zur Verfügung, um die Modellinstanz zu verarbeiten. Innerhalb der Benutzerschnittstelle (UI[8]) instanziiert der Fachplaner sprachbasiert die Modellieroperationen, um Modellobjekte hinzuzufügen, zu selektieren, zu verändern, zu löschen, um den Verarbeitungskontext zu konfigurieren oder Abfragen zu stellen. Die Sprache zur Beschreibung dieser Modellieroperationen wird im Kontext dieser Arbeit als **OML**[9] definiert. Modellieroperationen sind Gegenstand der OML-Ebene der Modellierungssprache. Die Anwendungskonzepte der OML in der Benutzerschnittstelle einer Fachapplikation werden im Abschnitt 5.2 dieser Arbeit behandelt.

Beschreibung von Modellierprogrammen: Die Erweiterung der OML um ein Variablenkonzept, ein Prozedurenkonzept sowie Programmkontrollstrukturen bildet die operative Programmiersprache, die im Kontext dieser Arbeit als **OPL**[10] definiert wird. Modellierprogramme sind der OPL-Ebene der Modellierungssprache zugeordnet. Die OPL wird vom Fachplaner bzw. Programmierer in der Programmierschnittstelle (API[11]) einer Fachapplikation verwendet, um parametrisierte Modellierprogramme zu formulieren. Bei der Ausführung eines Modellierprogramms werden Variablen ausgewertet, Prozeduren und Kontrollstrukturen so aufgelöst, dass eine Sequenz von alphanumerisch instanziierten Modellieroperationen der OML entsteht. Diese OML-Anweisungen operieren dann auf der Modellinstanz. Die Anwendungskonzepte der OPL in der Programmierschnittstelle einer Fachapplikation werden im Abschnitt 5.3 dieser Arbeit behandelt.

Eingabesprache: Die Zielsetzung, die operative Modellierungssprache als Eingabesprache zu verwenden, führt zur Anforderung, die Sprache *imperativ*[12] und möglichst *einfach* zu gestalten. Modellieroperationen der OML sollen vom Fachplaner selbst instanziiert und zur Modellinstanzverarbeitung verwendet werden. Für diese Aufgabe eignen sich *Kommando- bzw. Skriptsprachen* (vgl. Abschnitt 2.2.3 auf Seite 18), mit

[7] zusätzlich zur grafischen Benutzerschnittstelle, vgl. Abschnitt 2.2.3 auf Seite 18
[8] engl. User Interface
[9] Operative Modellierungssprache, engl. Operative Modeling Language
[10] Operative Programmiersprache, engl. Operative Programming Language
[11] engl. Application Programming Interface
[12] auch: anweisungsorientiert, vgl. imperative Programmiersprache, [Gumm u. Sommer 2002, S. 112]

denen Anweisungen einfach formuliert werden können. Aus diesem Grund lehnt sich die Syntax der vorgeschlagenen operativen Modellierungssprache stark an die Syntax von bekannten Skriptsprachen an.

Anwendungsschema: Die Verwendung der operativen Modellierungssprache als Eingabesprache für Modellierprogramme (OPL) und Modellieroperationen (OML) verdeutlicht das Bild 4.4.

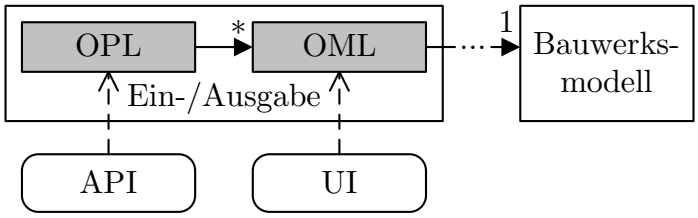


Bild 4.4: Modellierungssprache als Eingabesprache

4.2.2 Persistenz

Beschreibung von Änderungen: Neben der Verwendung der operativen Modellierungssprache als Eingabesprache in Fachapplikationen dient die Sprache auch dazu, Änderungen von Modellinstanzen als Sequenz von Operationsinstanzen *persistent* zu beschreiben[13]. Für Änderungen werden persistente Modellieroperationen definiert und während der Verarbeitung einer Modellinstanz geprägt. Persistente Modellieroperationen definieren keine Variablen, Prozeduren oder Kontrollstrukturen. Die Sprache zur Formulierung von Änderungen wird im Kontext dieser Arbeit als **POML**[14] definiert. Änderungen sind Gegenstand der POML-Ebene der Modellierungssprache. Um einzelne Modellinstanzversionen zu erzeugen, werden die in entsprechenden Änderungen gespeicherten Operationsinstanzen auf die Modellinstanz angewendet.

Persistente Identifikatoren: In Änderungen wird das Hinzufügen, das Modifizieren sowie das Löschen von Modellobjekten beschrieben. Um die Modellobjekte zu unterscheiden, werden ihnen in der POML eindeutige persistente Namen, sogenannte persistente Objektidentifikatoren (POIDs[15]), zugeordnet. In persistenten Modellieroperationen dienen POIDs als Operanden.

Anwendungsschema: Die Verwendung der Modellierungssprache zur Beschreibung von persistenten Änderungen verdeutlicht das Bild 4.5.

[13] vgl. Abschnitt 3.2.1 auf Seite 48
[14] Persistente operative Modellierungssprache, engl. Persistent Operative Modeling Language
[15] engl. Persistent Object Identifier

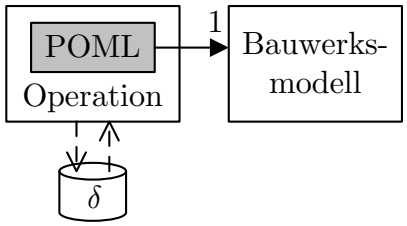


Bild 4.5: Modellierungssprache zur Beschreibung von persistenten Änderungen

4.2.3 Fachapplikationen

Der Einsatz der Modellierungssprache unter Berücksichtigung der einzelnen Sprachebenen wird im Kontext einer objektorientierten Fachapplikation im Bild 4.6 zusammenfassend veranschaulicht. Die OPL und die OML fungieren als Eingabesprache und sind aus diesem Grund der Fachapplikationskomponente *Ein-/Ausgabe* zugeordnet. Die POML ist Bestandteil der Verarbeitungskomponente *Operation*, da diese Operationen zur Beschreibung von persistenten Änderungen einer Modellinstanz umsetzt.

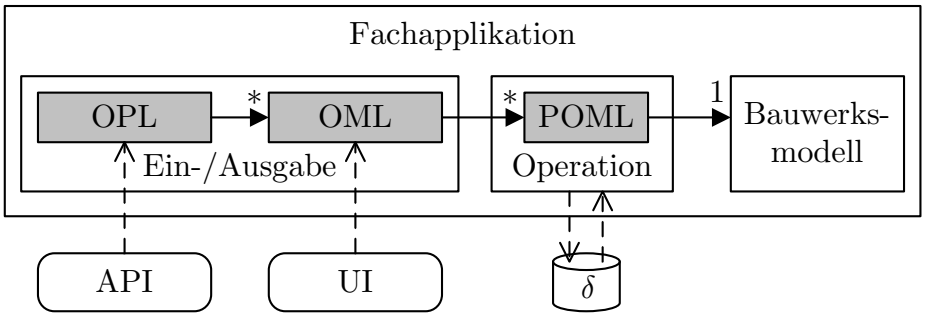


Bild 4.6: Modellierungssprache in einer Fachapplikation

4.2.4 Formale Eigenschaften

Wortmengen der Modellierungssprache: Die operative Modellierungssprache ist durch die drei vorgestellten Sprachebenen OPL, OML und POML gekennzeichnet. Jede dieser Ebenen entspricht einer eigenständigen formalen Sprache und wird somit jeweils durch eine Wortmenge definiert[16]. Die Elemente dieser Mengen sind gültige Folgen von Anweisungen in der jeweiligen Sprache.

$$\text{OPL} := \{x \mid x \text{ ist ein gültiges OPL-Programm gemäß OPL-Syntax}\} \quad (4.6)$$
$$\text{OML} := \{x \mid x \text{ ist eine gültige Folge von OML-Operationsinstanzen}\} \quad (4.7)$$
$$\text{POML} := \{x \mid x \text{ ist eine gültige Folge von POML-Operationsinstanzen}\} \quad (4.8)$$

Abbildungseigenschaft: Ein Modellierprogramm als Element der Sprache OPL mit Variablen, Prozeduren und Kontrollstrukturen wird auf eine Sequenz von Operati-

[16] vgl. Abschnitt 4.1.1 auf Seite 75

onsinstanzen als Element der Sprache in OML abgebildet. Dieser Folge von OML-Operationsinstanzen wird wiederum eine persistente Änderung in Form einer Sequenz von POML-Operationsinstanzen als Element der Sprache POML zugeordnet. Diese Eigenschaften werden durch die Abbildungen $s : \text{OPL} \rightarrow \text{OML}$ und $t : \text{OML} \rightarrow \text{POML}$ beschrieben.

$$s : \text{OPL} \rightarrow \text{OML} := \{(x, y) \in \text{OPL} \times \text{OML} \mid \text{das OPL-Programm } x \text{ wird einer Folge } y \text{ von OML-Operationsinstanzen zugeordnet}\} \quad (4.9)$$

$$t : \text{OML} \rightarrow \text{POML} := \{(y, z) \in \text{OML} \times \text{POML} \mid \text{die Folge } y \text{ von OML-Operationsinstanzen wird einer persistenten Änderung } z \text{ in POML zugeordnet}\} \quad (4.10)$$

Beispiel 4.4: Abbildungseigenschaft

Im Bild 4.7 ist ein OPL-Modellierprogramm dargestellt, das mit Variablen und einer Verzweigung die Erzeugung und die Rotation eines Wandobjekts beschreibt. Dieses Programm wird auf eine Sequenz von OML-Operationsinstanzen abgebildet (s). Den OML-Anweisungen kann wiederum eine persistente Änderung in POML zugeordnet werden (t).

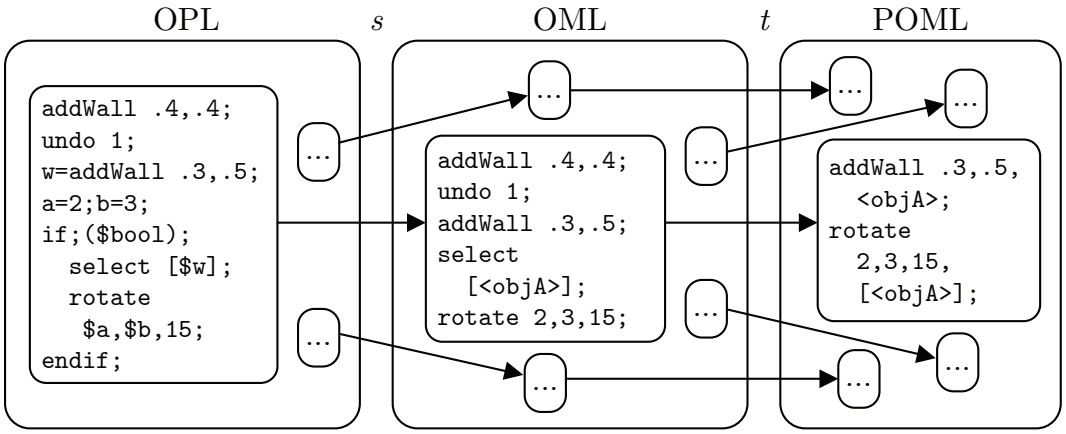


Bild 4.7: Abbildungseigenschaft der Sprachebenen

Teilmengeneigenschaft: Setzt man die einzelnen Sprachen im Hinblick auf die jeweiligen Wortmengen in Bezug, dann ist die POML eine Teilmenge der OML und die OML eine Teilmenge der OPL. Das bedeutet, persistente Änderungen in POML genügen sowohl der Syntax der OML als auch der Syntax der OPL. Darüber hinaus sind Folgen von Operationsinstanzen in OML auch korrekte OPL-Programme. Diese Eigenschaft wird formal wie folgt beschrieben und im Bild 4.8 veranschaulicht.

$$\text{POML} \subset \text{OML} \subset \text{OPL} \quad (4.11)$$

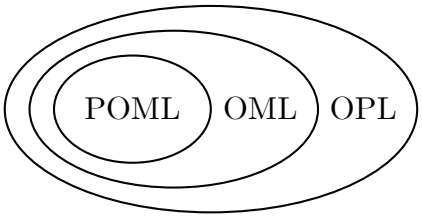


Bild 4.8: Teilmengeneigenschaft der Sprachebenen

4.3 Modellierprogramme

4.3.1 Symbole und Ausdrücke

Im folgenden Abschnitt wird die Grammatik der Modellierungssprache der OPL-Sprachebene vorgestellt. Die OPL dient der Formulierung von Modellierprogrammen, beispielsweise mit Systemkommandos für Kontrollstrukturen oder zur Erzeugung grafischer Komponenten. Es werden die Symbole, verwendete Datentypen und Ausdrücke der Sprache beschrieben. Modellieroperationen werden in der OPL-Syntax allgemein berücksichtigt, jedoch nicht weiter konkretisiert. Die vollständige Grammatik in BNF-Notation ist im Anhang A.1 zu finden.

Schlüsselwörter sind Bestandteil einer jeden Sprache. Die in der operativen Modellierungssprache verwendeten Schlüsselwörter sind als Terminalsymbole in der Grammatik definiert. Die Tabelle 4.1 listet die eingeführten Schlüsselwörter, deren Bedeutung, deren Terminalsymbole und Verweise auf deren Verwendung in alphabetischer Reihenfolge auf.

Schlüsselwort	Bedeutung	Terminal	Abschnitt
`BOTTOM`	Layoutkonstante: unten	`<BOTTOM>`	4.3.5, S. 94
`button`	Definition eines Schaltknopfes	`<BUTTON>`	4.3.5, S. 93
`buttonaction`	Definition einer Aktion für einen Schaltknopf	`<BUTTONACTION>`	4.3.5, S. 94
`call`	Aufruf einer Prozedur	`<CALL>`	4.3.3, S. 92
`CENTER`	Layoutkonstante: mittig	`<CENTER>`	4.3.5, S. 94
`concat`	Verknüpfen von Zeichenketten	`<CONCAT>`	4.3.1, S. 86
`checkbox`	Definition einer Auswahlbox	`<CHECKBOX>`	4.3.5, S. 93
`checkget`	Abfrage, ob eine Auswahlbox selektiert ist	`<CHECKGET>`	4.3.5, S. 94
`listbox`	Definition einer Auswahlliste	`<LISTBOX>`	4.3.5, S. 93
`listget`	Abfrage des selektierten Elements einer Auswahlliste	`<LISTGET>`	4.3.5, S. 94
`const`	Kennzeichner zur Definition einer Konstanten	`<CONST>`	4.3.2, S. 90
`dialog`	Definition eines grafischen Dialogs	`<DIALOG>`	4.3.5, S. 93
`dialogadd`	Hinzufügen grafischer Komponenten zu einem Dialog	`<DIALOGADD>`	4.3.5, S. 94
`dialogshow`	Anzeige eines grafischen Dialogs	`<DIALOGSHOW>`	4.3.5, S. 94
`else`	Alternative Zweig einer Verzweigung	`<ELSE>`	4.3.4, S. 92
`endif`	Ende einer Verzweigung	`<ENDIF>`	4.3.4, S. 92
`endproc`	Ende der Definition einer Prozedur	`<ENDPROC>`	4.3.3, S. 91
`endwhile`	Ende einer Wiederholung	`<ENDWHILE>`	4.3.4, S. 92

Schlüsselwort	Bedeutung	Terminal	Abschnitt
`false`	Wahrheitswert: falsch	`<FALSE>`	4.3.1, S. 89
`if`	Beginn einer Verzweigung	`<IF>`	4.3.4, S. 92
`import`	Laden einer Prozedurdatei	`<IMPORT>`	4.3.3, S. 91
`label`	Definition einer Beschriftung	`<LABEL>`	4.3.5, S. 93
`LEFT`	Layoutkonstante: links	`<LEFT>`	4.3.5, S. 94
`listadd`	Eintragen von Elementen in eine Liste	`<LISTADD>`	4.3.1, S. 90
`listcontains`	Abfrage, ob ein Element Eintrag einer Liste ist	`<LISTCONTAINS>`	4.3.1, S. 90
`listremove`	Austragen von Elementen aus einer Liste	`<LISTREMOVE>`	4.3.1, S. 90
`null`	Nullwert	`<NULL>`	4.3.1, S. 86
`print`	Ausgabe auf dem Bildschirm	`<PRINT>`	4.3.1, S. 86
`proc`	Definition einer Prozedur	`<PROC>`	4.3.3, S. 91
`return`	Zurückkehren aus einer Prozedur	`<RETURN>`	4.3.3, S. 91
`RIGHT`	Layoutkonstante: rechts	`<RIGHT>`	4.3.5, S. 94
`text`	Definition eines Textfelds	`<TEXT>`	4.3.5, S. 93
`textget`	Abfrage des Textes eines Textfelds	`<TEXTGET>`	4.3.5, S. 94
`TOP`	Layoutkonstante: oben	`<TOP>`	4.3.5, S. 94
`true`	Wahrheitswert: wahr	`<TRUE>`	4.3.1, S. 89
`while`	Beginn einer Wiederholung	`<WHILE>`	4.3.4, S. 92

Tabelle 4.1: Schlüsselwörter der operativen Programmiersprache OPL

Kommentare: Zur Dokumentation des Programmcodes, beispielsweise in Prozedurdateien, werden Kommentarzeilen mit vorangestelltem Doppel-Slash `"//"` zur Verfügung gestellt.

```
< COMMENT : "//" ... >
```

Bezeichner stellen Identifikatoren (`<ID>`) dar, die als Terminalsymbole in der Grammatik definiert sind. Sie dienen der Bezeichnung von Variablen (s. Abschnitt 4.3.2 auf Seite 90), Prozeduren und Prozedurdateien (s. Abschnitt 4.3.3 auf Seite 91), grafischen Komponenten (s. Abschnitt 4.3.5 auf Seite 93) und Modellieroperationen (s. Abschnitt 4.4 auf Seite 96).

```
< LETTER : ["a"-"z","A"-"Z"] >
< ID     : ("_")* <LETTER> ... >
```

Zahlen werden in Ganzzahlen (`<INTEGER>`) und Fließkommazahlen (`<DOUBLE>`) unterteilt.

```
< INTEGER  : "0" | ( ["1"-"9"] ( ["0"-"9"] )* ) >
< EXPONENT : ["e","E"] ( ["+","-"] )? ( ["0"-"9"] )+ >
< DOUBLE   : ( "0" | ["1"-"9"] ( ["0"-"9"] )* )?
             "." ( ["0"-"9"] )* ( <EXPONENT> )? >
```

Zeichenketten (`<STRING>`) beginnen und enden mit einem doppelten Anführungsstrich (") und stellen eine beliebige Folge von Zeichen dar, in der jedoch keine Steuerzeichen wie beispielsweise das Tabulartorzeichen auftreten dürfen. Steuerzeichen werden als Literale mittels Backslash-Editing[17] formuiert.

```
< STRING : "\"" ... "\"" >
```

Wahrheitswerte können genau zwei Zustände annehmen: entweder wahr (`TRUE`) oder falsch (`FALSE`). Diese Eigenschaft wird durch das Nichtterminalsymbol `Boolean` in der Grammatik ausgedrückt.

```
Boolean -> <TRUE> | <FALSE>
```

Persistente Identifikatoren: Zur eindeutigen und persistenten Identifikation von Modellobjekten werden persistente Identifikatoren (`<POID>`) eingeführt. Sie werden in spitze Klammern gesetzt (`< >`), beginnen mit einem Buchstaben und dürfen unter anderem keine Steuerzeichen enthalten.

```
< POID : "<" <LETTER> ... ">" >
```

Beispiel 4.5: Persistente Identifikatoren

Während `'<wall37>'` und `'<obj49@a846e21e56d37d:5cd4729:115cbf2f088:-7fff>'` gültige persistente Identifikatoren darstellen, sind `'<wall\13">'` und `'<24obj>'` keine gültigen POIDs.

Operatoren: Die in der operativen Modellierungssprache definierten Operatoren sind als Terminalsymbole in der Grammatik definiert. In der Tabelle 4.2 sind die eingeführten Operatoren, deren Bedeutung und Terminalsymbole aufgelistet.

Operator	Bedeutung	Terminalsymbol
+	Additionsoperator	<PLUS>
-	Subtraktionsoperator	<MINUS>
*	Multiplikationsoperator	<TIMES>
/	Divisionsoperator	<DIVIDEDBY>
**	Potenzieroperator	<EXP>
==	Vergleichsoperator: gleich	<EQUALS>
!=	Vergleichsoperator: ungleich	<NOTEQUALS>
<	Vergleichsoperator: kleiner-als	<LESSTHAN>

[17] z. B. wird das Tabulatorzeichen durch `"\t"` beschrieben

Operator	Bedeutung	Terminalsymbol
>	Vergleichsoperator: größer-als	<GREATERTHAN>
!	Logischer Negationsoperator	<NOT>
&&	Logischer Operator: und	<AND>
||	Logischer Operator: oder	<OR>
=	Zuweisungsoperator	<ASSIGN>

Tabelle 4.2: Operatoren der Modellierungssprache

Ausdrücke: Werte oder Verknüpfungen zwischen Werten mit Hilfe von Operatoren heißen Ausdrücke (`Expression`). Dabei wird zwischen logischen (`BoolExpr`) und nichtlogischen (`NonBoolExpr`) Ausdrücken unterschieden. Der Nullwert (<NULL>), Werte von Bezeichnern (<ID>), persistente Identifikatoren (<POID>) und Zeichenketten (<STRING>) sowie Variablenbezeichner (`Value`, s. Abschnitt 4.3.2 auf Seite 90), Listen (`List`, s. Abschnitt 4.3.1 auf Seite 90) und mathematische Terme (`Sum`) zählen zu den nichtlogischen Ausdrücken. Nachfolgende Syntax gilt für Ausdrücke.

```
Expression   -> BoolExpr
              | NonBoolExpr
BoolExpr     -> Boolean
              | "(" (<NOT>)? BoolExpr
                ( (<AND> | <OR>) (<NOT>)? BoolExpr )? ")"
              | "(" (<NOT>)? NonBoolExpr ( ( <EQUALS> | <NOTEQUALS>
                | <LESSTHAN> | <GREATERTHAN> )
                (<NOT>)? NonBoolExpr )? ")"

NonBoolExpr  -> <NULL> | <ID> | <POID> | <STRING>
              | Value
              | List
              | Sum

Sum          -> Prod ( (<PLUS> | <MINUS>) Prod )?

Prod         -> Exp ( (<TIMES> | <DIVIDED_BY>) Exp )?

Exp          -> Unary ( <EXP> Exp )*

Unary        -> ( <PLUS> )? Number
              | <MINUS> Number

Number       -> <INTEGER> | <DOUBLE>
              | Value
              | "(" Sum ")"
```

Listen: Eine geordnete Folge von Ausdrücken wird durch eine Liste (`List`) beschrieben. Die Ausdrücke werden dabei durch ein Komma (`,`) getrennt. Mittels der Anweisungen `listadd` und `listremove` werden Ausdrücke in eine existierende Liste ein- bzw. ausgetragen. Während die Anweisung `listsize` die Größe einer existierenden Liste bestimmt, prüft die Anweisung `listcontains`, ob ein angegebener Ausdruck Element einer existierenden Liste ist. Der Zugriff auf einzelne Elemente in einer Liste wird im Abschnitt 4.3.2 auf Seite 90 beschrieben. Für die Definition von Listen gilt nachfolgende Syntax.

```
List -> "[" Expression ( "," Expression )* "]"
```

Anweisungen: Ein in der operativen Modellierungssprache formuliertes Programm (Startsymbol: `Start`) besteht aus einer Liste von Anweisungen (`StmtList`). Anweisungen (`Statement`) werden durch ein Semikolon (`;`) abgeschlossen. Sie stellen entweder eine Zuweisung (`Assignment`) oder ein Kommando (`Command`) dar. Nachfolgende Syntax gilt für Anweisungen.

```
Start     -> StmtList
StmtList  -> Statement ( ";" Statement )*
Statement -> Assignment | Command
```

Kommandos: Es wird zwischen Systemkommandos (`SysCommand`) und Modellieroperationen (`ModelOperation`, s. Abschnitt 4.4) unterschieden. Systemkommandos werden durch Schlüsselwörter (s. Abschnitt 4.3.1) der Modellierungssprache ausgedrückt. Die Tabelle 4.1 auf Seite 86 enthält alle Systemkommandos und deren Bedeutung.

Modellieroperationen: Neben Systemkommandos stellen auch Modellieroperationen (`ModelOperation`) Kommandos (`Command`) dar. Die Syntax und die Verwendung von Modellieroperationen werden in den Abschnitten 4.4 auf Seite 96 und 4.5 auf Seite 102 vorgestellt.

Für Kommandos gilt nachfolgende Syntax.

```
Command -> SysCommand | ModelOperation
```

4.3.2 Variablen

Variablen haben einen Bezeichner (`<ID>`) und dienen der Speicherung von Werten.

Variablendefinition: Die Definition einer Variable ist eine Anweisung (`Statement`) bestehend aus der Deklaration und der Zuweisung (`Assignment`). Bei der Deklaration werden lediglich der Bezeichner (`<ID>`) der Variablen und die Konstanteneigenschaft (`<CONST>`) festgelegt. Aus Gründen der Einfachheit der Modellierungssprache muss kein Variablentyp angegeben werden. Dieser ergibt sich aus dem der Variablen zugewiesenen Wert. Die Zuweisung erfolgt mittels des Zuweisungsoperators `<ASSIGN>`. Der zugewiesene Wert ist das Ergebnis eines Kommandos (`Command`) oder eines Ausdrucks (`Expression`, s. Abschnitt 4.3.1). Die Definition von Variablen unterliegt nachfolgender Syntax.

```
Assignment -> ( <CONST> )? <ID> <ASSIGN> ( Command | Expression )
```

Beispiel 4.6: Variablendefinition
Gültige Variablendefinitionen sind `'a = (1+2);'`, `'_list = ["din_blau", false];'`, `'wall = addWall;'` und `'const PI = 3.1416;'`. Beispiele für ungültige Variablendefinitionen stellen die Anweisungen `'8abc = "Text";'` und `'x = 8abc;'` dar.

Variablenzugriff: Variablen müssen definiert sein, um auf deren Werte (`Value`) zuzugreifen. Beim Zugriff auf eine Variable wird dem Variablenbezeichner ein Dollarzeichen (`$`) vorangestellt. Der Zugriff auf einen Eintrag einer Liste erfolgt durch das Hinzufügen des nullbasierten Indexes des Listeneintrags in eckigen Klammern (`[i]`).

```
Value -> "$" <ID> ( "[" <INTEGER> "]" )?
```

Beispiel 4.7: Variablenverwendung
Der Zugriff auf die im Beispiel 4.6 definierten Variablen `a`, `_list`, `wall` und `PI` erfolgt durch die Ausdrücke `'$a'`, `'$_list`, `'$wall'`, `'$PI'`. Auf den zweiten Eintrag der Liste `_list` wird mit `'$_list[1]'` zugegriffen.

Variablengültigkeit: Variablen sind nur in einem bestimmten Gültigkeitsbereich[18] verwendbar. Dabei wird zwischen globalen und lokalen Variablen unterschieden. Im Programm definierte Variablen sind global, d.h. in allen Unterprogrammen bzw. Prozeduren sichtbar. Variablen, die im Gegensatz dazu innerhalb von Prozeduren definiert werden, sind auch nur innerhalb der Prozeduren – also lokal – verwendbar.

4.3.3 Prozeduren

Prozeduren haben einen Bezeichner (`<ID>`) und dienen der Zusammenfassung einer Folge von Anweisungen.

Prozedurdefinition: Die Definition einer Prozedur beginnt mit dem Systemkommando `proc`, dem Bezeichner der Prozedur (`<ID>`) und einer Liste von Bezeichnern (`IdList`) der Übergabeparameter. Es folgt die Sequenz der zusammenzufassenden Anweisungen (`StmtList`), welche das Systemkommando `return` (`Return`) beinhalten kann, um die Prozedur zu beenden und einen Wert zurückzugeben. Abgeschlossen wird die Definition einer Prozedur mit dem Systemkommando `endproc`. Nachfolgende Syntax gilt zur Definition von Prozeduren.

```
Proc   -> <PROC> <ID> IdList StmtList <ENDPROC>
IdList -> <ID> ( "," <ID> )*
Return -> <RETURN> ( Expression )?
```

Prozedurdateien: Mehrere Prozedurdefinitionen können in einer Prozedurdatei mit einem Bezeichner `<ID>` und der Dateiendung `.pro` zusammengefasst werden. Mit dem Systemkommando `import` werden Prozedurdateien geladen. Der Import (`Import`) von Prozedurdateien unterliegt nachfolgender Syntax.

[18] engl. scope

```
Import -> <IMPORT> <ID>
```

Prozeduraufruf: Prozeduren müssen definiert sein, um sie zu verwenden bzw. aufzurufen. Das bedeutet, sie wurden entweder im Programm selbst definiert oder sind Bestandteil einer geladenen Prozedurdatei. Das Systemkommando `call` ruft Prozeduren anhand ihres Bezeichners (`<ID>`) auf und übergibt entsprechende Parameter als Argumentenliste (`ArgList`). Die Identifikation einer innerhalb einer Prozedurdatei definierten Prozedur erfolgt durch die Verkettung des Bezeichners (`<ID>`) der Datei ohne Dateiendung, eines Punktes (`.`) und des Bezeichners (`<ID>`) der Prozedur. Rückgabewerte einer Prozedur können in Variablen gespeichert werden. Nachfolgende Syntax gilt für Prozeduraufrufe (`ProcCall`).

```
ProcCall -> <CALL> <ID> ( "." <ID> )? ( ArgList )?
ArgList  -> Expression ( "," Expression )*
```

4.3.4 Kontrollstrukturen

Ablaufsteuerung: Kontrollstrukturen dienen der Ablaufsteuerung von Programmen. Es wird zwischen der Sequenz, der Verzweigung und der Wiederholung unterschieden.

Sequenz: Eine Folge von nacheinander auszuführenden Anweisungen wird Sequenz genannt. Sie spiegelt sich als Nichtterminalsymbol `StmtList` (s. Abschnitt 4.3.1 auf Seite 90) in der Grammatik der Modellierungssprache wieder und bedarf demnach keines expliziten Systemkommandos.

Verzweigung: Aufgrund einer Bedingung kann der Programmablauf verzweigen. Die Bedingung wird als boolscher Ausdruck (`BoolExpr`, s. 4.3.1 auf Seite 89) formuliert. Ist die Bedingung erfüllt (`<TRUE>`), dann wird die darauffolgende Sequenz von Anweisungen (`StmtList`) ausgeführt. Andernfalls gilt der alternative Anweisungsblock `StmtList`) nach dem Schlüsselwort `else`. Das Systemkommando `if` leitet eine Verzweigung ein, das Systemkommando `endif` schließt diese ab. Da eine Verzweigung (`Choice`) selbst eine Anweisung (`Stmt`) ist, können sie ineinander geschachtelt werden. Es folgt die Syntax für eine Verzweigung (`Choice`).

```
Choice -> <IF> ";" BoolExpr ";" StmtList ";"
          <ELSE> ";" StmtList <ENDIF>
```

Wiederholung: Bei einer Wiederholung oder einer Schleife[19] wird eine Sequenz von Anweisungen (`StmtList`) wiederholt, solange eine Bedingung erfüllt ist. Die Bedingung wird als boolscher Ausdruck (`BoolExpr`, s. 4.3.1 auf Seite 89) formuliert. Das Systemkommando `while` leitet eine Wiederholung ein, das Systemkommando `endwhile` schließt diese ab. Wiederholungen können ineinander geschachtelt werden. Es folgt die Syntax für Wiederholungen (`Loop`).

```
Loop -> <WHILE> ";" BoolExpr ";" StmtList <ENDWHILE>
```

[19] engl. loop

4.3.5 Grafische Komponenten

Die operative Modellierungssprache sieht Systemkommandos vor, um einfache grafische Komponenten zu erzeugen. Grafische Komponenten sind Bestandteil von einfachen Dialogfenstern und dienen als grafische Benutzerschnittstelle.

Definition: Die Definition einer grafischen Komponente erfolgt mit einem entsprechenden Systemkommando. Die Tabelle 4.3 zeigt die verfügbaren Systemkommandos, die zugehörigen grafischen Komponenten sowie die Nichtterminalsymbole in der Grammatik.

Systemkommando	Grafische Komponente	Nichtterminalsymbol
`label`		`Label`
`text`		`Text`
`button`		`Button`
`checkbox`		`Checkbox`
`listbox`		`Listbox`
`dialog`		`Dialog`

Tabelle 4.3: Grafische Komponenten der Modellierungssprache

Label: Ein Label ist eine Beschriftung und wird durch einen Bezeichner (`<ID>`) und eine Aufschrift als Ausdruck (`Expression`) definiert.

Text: Zur Erzeugung eines Textfelds werden ein Bezeichner (`<ID>`), eine Breite (`<INTEGER>`) und ein optionaler Text als Ausdruck (`Expression`) angegeben. In ein Textfeld kann beliebiger einzeiliger Text eingegeben werden.

Button: Ein Button ist ein Schaltknopf und wird durch einen Bezeichner (`<ID>`) und eine Aufschrift in Form eines Ausdrucks (`Expression`) definiert.

Checkbox: Eine Checkbox ist ein Auswahlkasten. Zu dessen Erzeugung werden ein Bezeichner (`<ID>`) und eine Beschriftung als Ausdruck (`Expression`) definiert. Eine Checkbox kann die Zustände *markiert* und *nicht markiert* annehmen.

Listbox: Eine Listbox ist eine Auswahlliste und wird durch einen Bezeichner (`<ID>`) und eine Liste (`List`) der Auswahlelemente definiert. In einer Listbox kann genau ein

Element[20] der definierten Liste markiert sein.

Dialog: Ein Dialog ist ein grafisches Fenster mit einer Titelleiste und dient als Container für die zuvor genannten grafischen Komponenten. Ein Bezeichner (`<ID>`) und ein Titel als Ausdruck (`Expression`) werden in der Definition eines Dialogs angegeben.

Es folgt die Syntax der Systemkommandos zur Definition von grafischen Komponenten.

```
Label    -> <LABEL> <ID> "," Expression
Text     -> <TEXT> <ID> "," <INTEGER> ( "," Expression )?
Button   -> <BUTTON> <ID> "," Expression
Checkbox -> <CHECKBOX> <ID> "," Expression
Listbox  ->  <LISTBOX> <ID> "," List
Dialog   -> <DIALOG> <ID> "," Expression
```

Verwendung: Grafische Komponenten müssen definiert sein, um sie zu verwenden. Ein Dialog (`dialog`) stellt ein Fenster dar, dem andere grafische Komponenten mit dem Systemkommando `dialogadd` hinzugefügt werden. Das Systemkommando `dialogshow` dient zur Anzeige eines definierten Dialogs. Zur Positionierung grafischer Komponenten in einem Dialog wird ein einfaches Layout bereitgestellt. Das Bild 4.9 veranschaulicht das Layout und die Bedeutung der Konstanten (`<TOP>`, `<BOTTOM>`, `<CENTER>`, `<LEFT>`, `<RIGHT>`).

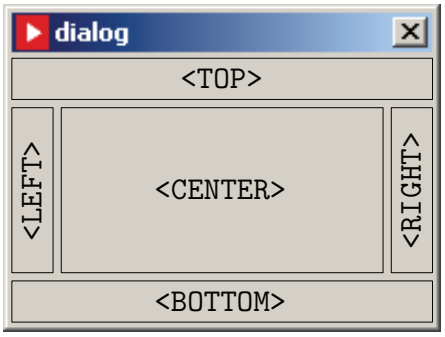


Bild 4.9: Layout in Dialogfenstern

DialogAdd: Das Systemkommando `dialogadd` fügt einem vorhandenen Dialog (`value`) eine grafische Komponente (`value`) hinzu und platziert diese entsprechend der angegebenen Layoutkonstante.

DialogShow: Das Systemkommando `dialogshow` zeigt einen zu spezifizierenden Dialog (`value`) an.

TextGet: Für den Zugriff auf den Text eines Textfelds (`<TEXT>`) mit einem Bezeichner (`value`) wird das Systemkommando `textget` bereitgestellt.

[20] hier engl. item

ButtonAction: Das Systemkommando `buttonaction` ordnet einem Schaltknopf (`<BUTTON>`) mit einem Bezeichner (`value`) einen Ausdruck (`Expression`) zu, der Anweisungen definiert, welche bei Aktivierung des Schaltknopfs ausgeführt werden sollen.

CheckGet: Zur Abfrage, ob eine Auswahlbox (`<CHECKBOX>`) mit einem Bezeichner (`value`) markiert ist, wird das Systemkommando `checkget` bereitgestellt.

ListGet: Der Zugriff auf das markierte Auswahlelement einer Auswahlliste (`<LISTBOX>`) mit einem Bezeichner (`value`) erfolgt mit dem Systemkommando `listget`.

Für das Hinzufügen grafischer Komponenten in ein Dialogfenster und den Zugriff auf definierte grafische Komponenten gilt nachfolgende Syntax.

```
DialogAdd    -> <DIALOGADD> Value <ARGSEP> Value <ARGSEP>
                ( <TOP> | <BOTTOM> | <CENTER> | <LEFT> | <RIGHT> )
DialogShow   -> <DIALOGSHOW> Value
TextGet      -> <TEXTGET> Value
ButtonAction -> <BUTTONACTION> Value <ARGSEP> Expression
CheckGet     -> <CHECKGET> Value
ListGet      -> <LISTGET> Value
```

4.4 Modellieroperationen

Zur Verarbeitung einer Modellinstanz mit einer Fachapplikation werden Modellieroperationen der OML-Sprachebene definiert. Zur Laufzeit der Fachapplikation werden die definierten Modellieroperationen vom Fachplaner in der Benutzerschnittstelle instanziiert und von der Verarbeitungskomponente auf die Modellinstanz angewendet. Die Anwendungskonzepte für Modellieroperationen werden in den Abschnitten 5.2 und 5.3 dieser Arbeit behandelt.

4.4.1 Definition

Eine Modellieroperation (`ModelOperation`) ist eine Anweisung und wird durch einen Bezeichner (`<ID>`) und eine Argumentenliste (`ArgList`) definiert. Nachfolgende Syntax gilt für die Definition von Modellieroperationen.

```
ModelOperation -> <ID> ( ArgList )?
ArgList        -> Expression ( "," Expression )*
```

Modellieroperationen der OML dienen der Beschreibung der Interaktion des Planers mit einer konkreten Fachapplikation in der Benutzerschnittstelle. Die Fachapplikation stellt die grafische Unterstützung des Fachplaners, beispielsweise bei der Selektion von Modellobjekten mit der Maus, bereit. Es werden Gruppen von Modellieroperationen[21] mit nachfolgender Funktionalität unterschieden.

- Hinzufügen von Modellobjekten – ADD
- Selektieren und Deselektieren von Modellobjekten – SELECT/UNSELECT
- Modifizieren von selektierten Modellobjekten – MODIFY
- Löschen von selektierten Modellobjekten – REMOVE
- Konfigurieren des Verarbeitungskontextes – SET
- Abfragen über die Modellinstanz oder selektierte Modellobjekte – GET
- Rückgängigmachen (undo), Wiederherstellen (redo) und Wiederholen (again) von Modellieroperationen – MISC

Beispiel 4.8: Sprachbasierte Definition von Modellieroperationen

Fachapplikation: Eine Fachapplikation dient dem Entwurf und der Berechnung der Stützmomente von Dreifeldträgern. Sie verarbeitet eine Modellinstanz, die mehrere Dreifeldträger mit jeweils einer ständigen Gleichlast g, feldweisen Verkehrslasten p und konstantem Trägheitsmoment I beschreibt. Jedem Spannfeld i sind eine Länge l_i und eine Verkehrslast p_i zugeordnet.

[21] vgl. Abschnitt 3.4.2 auf Seite 70

Fachliches Modell: Der Zustand und das Verhalten des fachlichen Modells für einen solchen Dreifeldträger wird durch einen Auszug aus dem Gleichungsapparat nach [Schneider 2002, S. 4.19 ff.] und das Bild 4.10 beschrieben[22].

$$K = 4(l_1 + l_2) \cdot (l_2 + l_3) - {l_2}^2 \qquad q_i = p_i + g$$

$$M_b^1 = -\frac{q_1 \cdot l_1^3}{2K}(l_2 + l_3) \qquad M_c^1 = +\frac{q_1 \cdot l_1^2}{4K}\, l_1 \cdot l_2$$

$$M_b^2 = -\frac{q_2 \cdot l_2^3}{2K}(l_2 + 2l_3) \qquad M_c^2 = +\frac{q_2 \cdot l_2^3}{4K}\,(l_2 + 2l_1)$$

$$M_b^3 = +\frac{q_3 \cdot l_3^2}{4K}\, l_3 \cdot l_2 \qquad M_c^3 = +\frac{q_3 \cdot l_3^3}{2K}\,(l_1 + l_2)$$

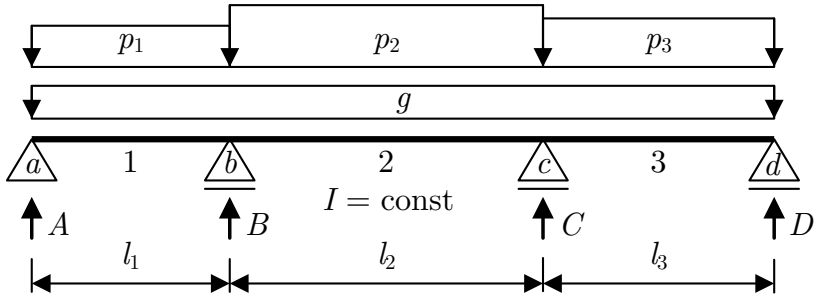


Bild 4.10: Fachliches Modell eines Dreifeldträgers

Objektorientiertes Modell: Der Fachapplikation liegt das im Bild 4.11 als UML-Klassendiagramm dargestellte objektorientierte Modell zugrunde. Das Modell (Model) verwaltet mehrere Dreifeldträger (ContBeam) und führt die Berechnung (Analysis) durch. Einem Träger sind eine ständige Last (DeadLoad) und drei Spannfelder (Span) zugeordnet. Jedes Spannfeld verwaltet eine Verkehrslast (LiveLoad).

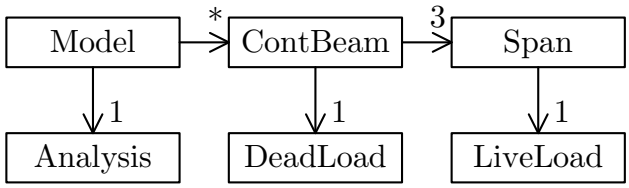


Bild 4.11: Objektorientiertes Modell für Dreifeldträgerberechnung

Modellieroperationen: Zur Verarbeitung von Instanzen dieses Modells werden nachfolgend aufgeführte Modellieroperationen definiert. Es wird davon ausgegangen, dass nach Erzeugung eines Dreifeldträgers sämtliche Lasten den Wert 0.0 haben und die Länge jedes Spannfeldes mit dem Wert 1.0 belegt ist.

[22] Das Moment M_b^1 ist dabei der Anteil des Stützmoments M aus dem Spannfeld 1 am Auflager b. Es gilt das Superpositionsprinzip: $M_b = M_b^1 + M_b^2 + M_b^3$

- Träger hinzufügen und positionieren (x,y)
  `AddBeam -> "addbeam" Number "," Number`

- Träger selektien (grafisch oder mittels POID)
  `SelectBeam -> "selectbeam" ( <POID> )?`

- Träger deselektieren (grafisch oder mittels POID)
  `UnselectBeam -> "unselectbeam ( <POID> )?`

- Den Wert `Number` der ständigen Last g für aktuell selektierte oder spezifizierte (POID) Träger ändern
  `ModDeadLoad -> "moddeadload" Number ( ",[" <POID> ( "," <POID>)* "]")?`

- Spannfeld selektieren (grafisch oder mittels POID und Nr.)
  `SelectSpan -> "selectspan ( "[" <POID> "," <INTEGER> "]" )?`

- Spannfeld deselektieren (grafisch oder mittels POID und Nr.)
  `UnselectSpan -> "unselectspan ( "[" <POID> "," <INTEGER> "]" )?`

- Den Wert `Number` der Länge l der aktuell selektierten oder spezifizierten (POID und Nr.) Spannfelder ändern
  `ModLength -> "modlength" Number ( ",[[" <POID> "," <INTEGER> "]" ( ",[" <POID> "," <INTEGER> "]" )* "]" )?`

- Den Wert `Number` der Verkehrslast p der aktuell selektierten oder spezifizierten (POID und Nr.) Spannfelder ändern
  `ModLiveLoad -> "modliveload" Number (",[[" <POID> "," <INTEGER> "]" ( ",[" <POID> "," <INTEGER> "]" )* "]" )?`

- Aktuell selektierte oder spezifizierte (POID) Träger löschen
  `Remove -> "remove" ( ",[" <POID> ( "," <POID> )* "]" )?`

- Berechnung und Ausgabe der Stützmomente der aktuell selektierten oder spezifizierten (POID) Träger
  `GetSupportMoments -> "getsupportmoments ( ",[" <POID> ( "," <POID>)* "]")?`

- Abfrage der gegenwärtig selektierten Träger
  `GetSelectedBeams -> "getselectedbeams"`

- Rückgängigmachen der letzten, das Modell verändernden Operationsinstanz
  `Undo -> "Undo"`

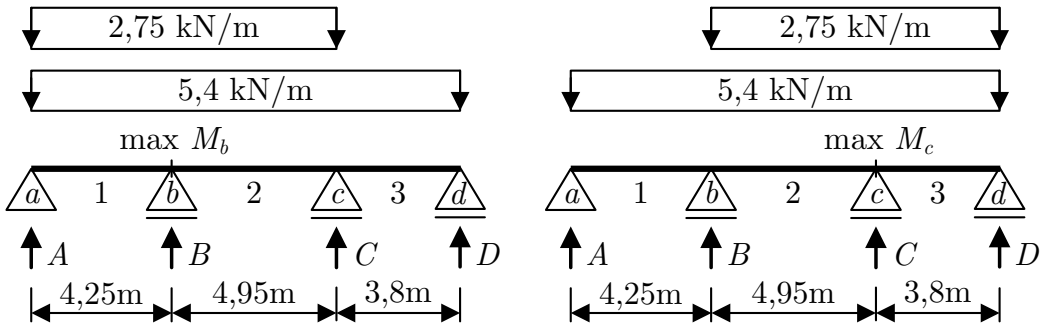
- Wiederherstellen der letzten rückgangig gemachten Operationsinstanz
  `Redo -> "Redo"`

4.4.2 Instanziierung

Definierte Modellieroperationen werden zur Verwendung vom Fachplaner in der Benutzerschnittstelle sprachbasiert instanziiert, indem die definierten Argumente mit Werten belegt werden. In einer Fachapplikation werden instanziierte Modellieroperationen interpretiert, um anschließend auf der objektorientierten Modellinstanz zu operieren.

Beispiel 4.9: Sprachbasierte Instanziierung von Modellieroperationen

Es werden die im vorangegangenen Beispiel 4.8 definierten Modellieroperationen und die beschriebene Fachapplikation zum Entwurf und zur Berechnung von Dreifeldträgern herangezogen. Die Aufgabe besteht darin, einen Dreifeldträger mit Spannfeldlängen von 4,25 m, 4,95 m und 3,8 m, einer ständigen Last von 5,4 kN/m und einer Verkehrslast von 2,75 kN/m zu entwerfen und die maximalen Stützmomente zu bestimmen. Dazu werden nach der Erzeugung des Trägers die im Bild 4.12 dargestellten Lastfälle definiert (vgl. hierzu [Schneider 2002, S. 4.20]).



(a) LF 1: max M_b (b) LF 2: max M_c

Bild 4.12: Lästfalle für maximale Stützmomente

Das Listing 4.1 verdeutlicht die sprachbasierte Instanziierung der definierten OML-Modellieroperationen. Die OML-Anweisungen (ohne Kommentare) werden zur Modellierung des Dreifeldträgers vom Fachplaner in der Benutzerschnittstelle der beschriebenen Fachapplikation eingegeben. Bei der Selektion von Modellobjekten wird der Fachplaner grafisch unterstützt.

```
// Träger an Position (0,0) erzeugen
addbeam 0.0, 0.0;

// Ständige Last setzen
selectbeam; // Träger mit der Maus grafisch selektieren
moddeadload 5.4; // Ständige Last ändern
unselectbeam; // Träger mit der Maus grafisch deselektieren

// Längen der Spannfelder setzen
selectspan;  // Spannfeld 1 mit der Maus grafisch selektieren
modlength 4.25; // Länge ändern
```

```
unselectspan; // Spannfeld 1 mit der Maus grafisch
              // deselektiere
selectspan; // Spannfeld 2 mit der Maus grafisch selektieren
modlength 4.95; Länge ändern
unselectspan; // Spannfeld 2 mit der Maus grafisch
              // deselektieren
selectspan; // Spannfeld 3 mit der Maus grafisch selektieren
modlength 3.8; // Länge ändern
unselectspan; // Spannfeld 3 mit der Maus grafisch
              // deselektieren

// Verkehrslast LF 1 setzen
selectspan; // Spannfeld 1 mit der Maus grafisch selektieren
selectspan; // Spannfeld 2 mit der Maus grafisch selektieren
modliveload 2.75; // Verkehrslast ändern
unselectspan; // Spannfeld 1 mit der Maus grafisch
              // deselektieren
unselectspan; // Spannfeld 2 mit der Maus grafisch
              // deselektieren

// Stützmomente ausgeben
selectbeam; // Träger mit der Maus grafisch selektieren
getsupportmoments; // Stützmomente berechnen
unselectbeam; // Träger mit der Maus grafisch deselektieren

// Verkehrslast LF 2 setzen
selectspan; // Spannfeld 1 mit der Maus grafisch selektieren
selectspan; // Spannfeld 2 mit der Maus grafisch selektieren
undo; // Selektion Spannfeld 2 rückgängig machen
modliveload 0.0; // Verkehrslast ändern
unselectspan; // Spannfeld 1 mit der Maus grafisch
              // deselektieren
selectspan; // Spannfeld 3 mit der Maus grafisch selektieren
modliveload 2.75; // Verkehrslast ändern
unselectspan; // Spannfeld 3 mit der Maus grafisch
              // deselektieren

// Stützmomente ausgeben
selectbeam; // Träger mit der Maus grafisch selektieren
getsupportmoments; // Stützmomente berechnen
unselectbeam; // Träger mit der Maus grafisch deselektieren
```

Listing 4.1: Sprachbasierte Instanziierung von Modellieroperationen in OML

Die Ausführung der im Listing 4.1 instanziierten Modellieroperationen führt zur Ausgabe der Stützmomente (LF 1: $M_b = -18.4$ kNm, LF 2: $M_c = -16.4$ kNm). Das Bild 4.13 zeigt die Benutzeroberfläche einer Fachapplikation zum Entwurf und zur Berechnung von Dreifeldträgern.

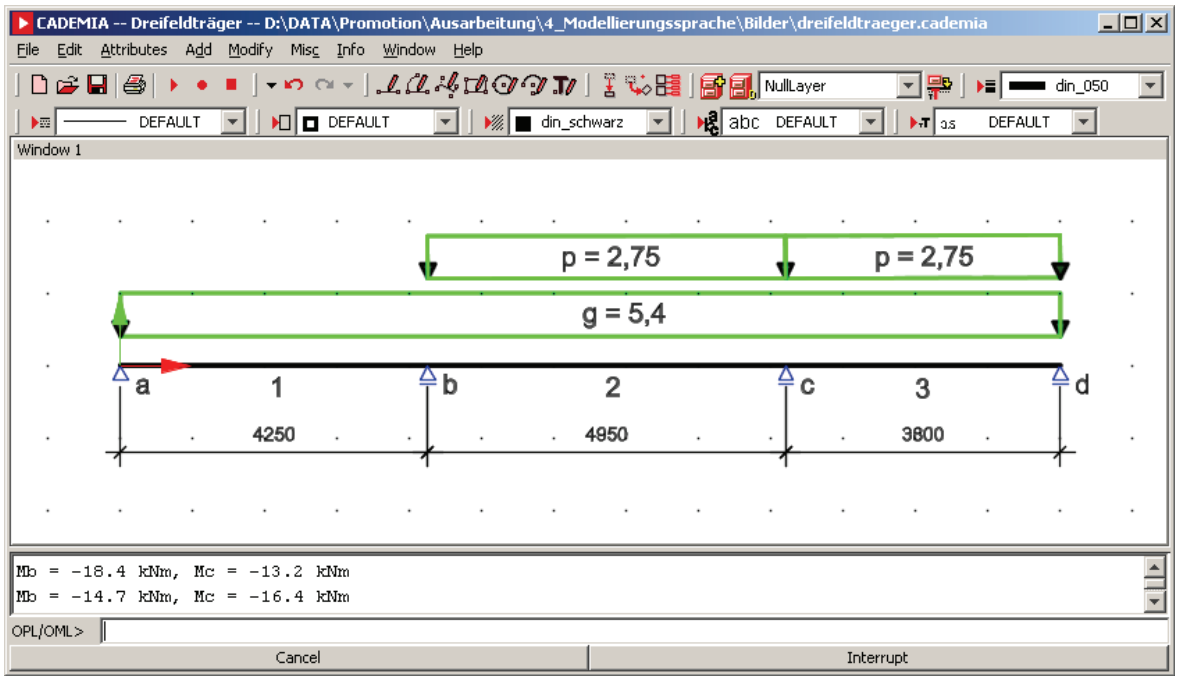


Bild 4.13: Benutzerschnittstelle einer Fachapplikation zum Entwurf und zur Berechnung von Dreifeldträgern

4.5 Persistente Änderungen

Zur Beschreibung von persistenten Änderungen einer Modellinstanz werden persistente Modellieroperationen der POML-Sprachebene definiert. Während instanziierte Modellieroperationen in OML in der Benutzerschnittstelle von Fachapplikationen verwendet werden, dienen Instanzen persistenter Modellieroperationen der Speicherung von Änderungen. Diese Änderungen können auf die Modellinstanz angewendet werden. Die Anwendungskonzepte für persistente Modellieroperationen der POML-Sprachebene werden in den Abschnitten 5.4, 5.5, 5.6 und 5.7 dieser Arbeit behandelt.

4.5.1 Definition

Persistente Modellieroperationen genügen der Syntax von Modellieroperationen (vgl. Abschnitt 4.4.1 auf Seite 96). Sie stellen demnach Anweisungen dar, die jeweils durch einen Bezeichner (`<ID>`) und eine Argumentenliste (`ArgList`) definiert werden. Persistente Modellieroperationen der POML unterscheiden sich von OML-Modellieroperationen im Hinblick auf die Semantik der Operationen. Im Gegensatz zu OML-Operationen werden in POML-Operationen Modellobjekte beim Hinzufügen, beim Modifizieren und beim Löschen direkt durch persistente Objektidentifikatoren (POIDs) als Operanden gekennzeichnet. Hier wird bewusst auf die Verwendung einer impliziten Selektionsmenge als Grundlage der Modifikations- und Löschoperationen verzichtet. Im Hinblick auf die Anwendung der POML beim Vergleich und beim Zusammenführen von Modellinstanzversionen[23] hat sich die Verwendung einer expliziten Operandenliste als zweckmäßig herausgestellt. Folgende Grundsätze gelten bei der Definition von persistenten Modellieroperationen.

- Beim Hinzufügen (ADD), beim Modifizieren (MODIFY) und beim Löschen (REMOVE) von Modellobjekten sind persistente Objektidentifikatoren (`<POID>`) als Operanden anzugeben.
- Funktionalität zum Selektieren und Deselektieren ist nicht Bestandteil der persistenten Modellieroperationen – SELECT/UNSELECT.
- Abfragen über Modellobjekte sind keine modellverändernden Operationen und stellen somit keine persistenten Modellieroperationen dar – GET.
- Das Rückgängigmachen (undo) und das Wiederherstellen (redo) von Modellieroperationen sind nicht Gegenstand der persistenten Modellieroperationen – MISC.

Beispiel 4.10: Sprachbasierte Definition von persistenten Modellieroperationen

Es wird die im Beispiel 4.8 auf Seite 96 beschriebene Fachapplikation zum Entwurf und zur Berechnung von Dreifeldträgern betrachtet. Zur Beschreibung von Änderungen einer Modellinstanz werden die nachfolgend aufgeführten persistenten Modellieroperationen definiert.

[23] s. Abschnitte 5.6 S. 129 und 5.7 S. 139

- Träger mit `POID` hinzufügen und positionieren (x,y)
  `AddBeam -> "addbeam" Number "," Number "," <POID>`
- Den Wert `Number` der ständigen Last g für Liste von Träger mit `POID`s ändern
  `ModDeadLoad -> "moddeadload" Number ",[" <POID> ("," <POID> )* "]"`
- Den Wert `Number` der Länge l des jeweils i-ten (`<INTEGER>`) Spannfeldes für Liste von Trägern mit `POID`s ändern
  `ModLength -> "modlength" Number ",[[" <POID> "," <INTEGER> "]" ( ",[" <POID> , <INTEGER> "]" )* "]"`
- Den Wert `Number` der Verkehrslast p des jeweils i-ten (`<INTEGER>`) Spannfeldes für Liste von Trägern mit `POID`s ändern
  `ModLiveLoad -> "modliveload" Number ",[[" <POID> "," <INTEGER> "]" ( ",[" <POID> "," <INTEGER> "]" )* "]"`
- Liste von Trägern mit `POID`s löschen
  `Remove -> "remove" "[" <POID> ( "," <POID> )* "]"`

4.5.2 Instanziierung

Persistente Modellieroperationen werden zur Beschreibung von Änderungen sprachbasiert instanziiert, indem die definierten Argumente mit Werten belegt werden. Während der Verarbeitung der Modellinstanz in einer Fachapplikation zeichnet der *Journaling*[24]-Mechanismus die persistenten Änderungen in POML auf.

Beispiel 4.11: Sprachbasierte Instanziierung von persistenten Modellieroperationen

Es wird der im Listing 4.1 auf Seite 99 betrachtete Entwurf eines Dreifeldträgers durch instanziierte OML-Modellieroperationen herangezogen. Während im Listing 4.1 die Modellinstanz durch OML-Anweisungen in der Benutzerschnittstelle verarbeitet wird, werden in diesem Beispiel die Änderung der Modellinstanz auf Basis der persistenten Modellieroperationen beschrieben. Vorlage dafür sind die im Beispiel 4.10 definierten POML-Modellieroperationen. Das Listing 4.2 zeigt die sprachliche Instanziierung der persistenten Modellieroperationen zur Formulierung der Änderung des Dreifeldträgers in POML.

```
// Träger erzeugt
addbeam 0.0, 0.0, <beam01@xyz>;

// Ständige Last modifiziert
moddeadload 5.4, [<beam01@xyz>];

// Längen der Spannfelder modifiziert
modlength 4.25, [[<beam01@xyz>,1]];
modlength 4.95, [[<beam01@xyz>,2]];
```

[24] s. Abschnitt 3.4.3 auf Seite 72

```
10 modlength 3.8, [[<beam01@xyz>,3]];

12 // Verkehrslast gemäß LF 1 modifiziert
   modliveload 2.75, [[<beam01@xyz>,1], [<beam01@xyz>,2]];
14
   // Verkehrslast gemäß LF 2 modifiziert
16 modliveload 0.0, [[<beam01@xyz>,1]];
   modliveload 2.75, [[<beam01@xyz>,3]];
```

Listing 4.2: Sprachbasierte Instanziierung von persistenten Modellieroperationen in POML

Die im Listing 4.2 beschriebenen persistenten Änderungen zeigen im Unterschied zum Listing 4.1 die Verwendung von persistenten Objektidentifikatoren (`POID`s) und Operandenlisten. Darüber hinaus ist ersichtlich, dass Abfrageoperationen wie `getsupportmoments` nicht Bestandteil der persistenten Änderung sind, weil sie die Modellinstanz nicht verändern. Bis auf die Angabe von Operanden bzw. Operandenlisten unterscheidet sich die POML-Syntax nicht von der OML-Syntax der vorgestellten operativen Modellierungssprache.

5 Anwendungskonzepte

„Grau, teurer Freund, ist alle Theorie, und grün des Lebens goldner Baum.“
Johann Wolfgang von Goethe (1749–1832), aus [Goethe 1808]

In diesem Kapitel werden die Anwendungskonzepte der verarbeitungsorientierten Modellierung und der operativen Modellierungssprache im kooperativen Bauplanungsprozess vorgestellt. Dazu erfolgen Untersuchungen zur Standardisierung operativer Modelle in Fachdomänen und zur Umsetzung operativer Modelle in Fachapplikationen. Ferner wird die Anwendung der Sprache in den Benutzer- und Programmierschnittstellen von Fachapplikationen sowie zu neuen Kooperationskonzepten beim Austausch, bei der Archivierung, beim Vergleich und bei der Zusammenführung von versionierten Bauwerksinformationen betrachtet.

5.1 Operative Modelle

Die Basis für die in den folgenden Abschnitten beschriebenen Anwendungskonzepte bilden operative Modelle. In diesem Abschnitt werden die Standardisierung operativer Modelle innerhalb von Fachdomänen und die Umsetzung operativer Modelle in Fachapplikationen beschrieben.

Operative Modelle: Operative Modelle sind die abstrakte Beschreibung der Verarbeitung von objektorientierten Modellinstanzen in Fachapplikationen durch eine definierte Menge von Modellieroperationen.

5.1.1 Standardisierung in Fachdomänen

Fachdomäne: Bei der Bauwerksplanung wird innerhalb einer Fachdomäne ein bestimmter Planungsaspekt des Bauwerks aus einer bestimmten fachlichen Sicht betrachtet. Die Fachplaner verwenden dabei untereinander ein einheitliches Fachvokabular, das in ihrer Fachsprache verankert ist. Das bedeutet, die Semantik von Begriffen ist eindeutig innerhalb einer Fachdomäne festgelegt. Jedoch unterscheiden sich die Begrifflichkeiten zwischen unterschiedlichen Domänen. Beispielsweise kann eine Wand in der Architekturdomäne aus Sicht eines Tragwerksplaners als Scheibe bezeichnet werden.

Fachapplikationen: Zur computergestützten Bauwerksplanung werden innerhalb einer Fachdomäne Fachapplikationen verschiedener Hersteller eingesetzt. Obwohl diese Fachapplikationen demselben Planungsaspekt zugeordnet sind, besitzen sie unterschiedlich strukturierte, native objektorientierte Bauwerksmodelle.

Standardisiertes operatives Modell: Auf der Grundlage der einheitlichen Semantik innerhalb einer Fachdomäne wird die Standardisierung von Operationen zum Modellieren mit Fachapplikationen vorgeschlagen. Ziel dieser Standardisierung ist die Einigung auf eine Menge festgelegter Modellieroperationen zur Verarbeitung von unterschiedlich strukturierten Objektmodellen. Die Menge von Standardoperationen bildet als operatives Modell die Basis zur Definition einer Ordnungsstruktur innerhalb der Vielzahl von unterschiedlichen Bauwerksmodellen und Fachapplikationen für ein und denselben Planungsaspekt. Diese Ordnungsstruktur gilt für genau eine Fachdomäne, ist aber von einzelnen Fachapplikationen und somit auch von einzelnen nativen Datenmodellen unabhängig. Das Bild 5.1 verdeutlicht die Funktion des standardisierten operativen Modells als Schnittstelle zwischen den Fachapplikationen einer Fachdomäne.

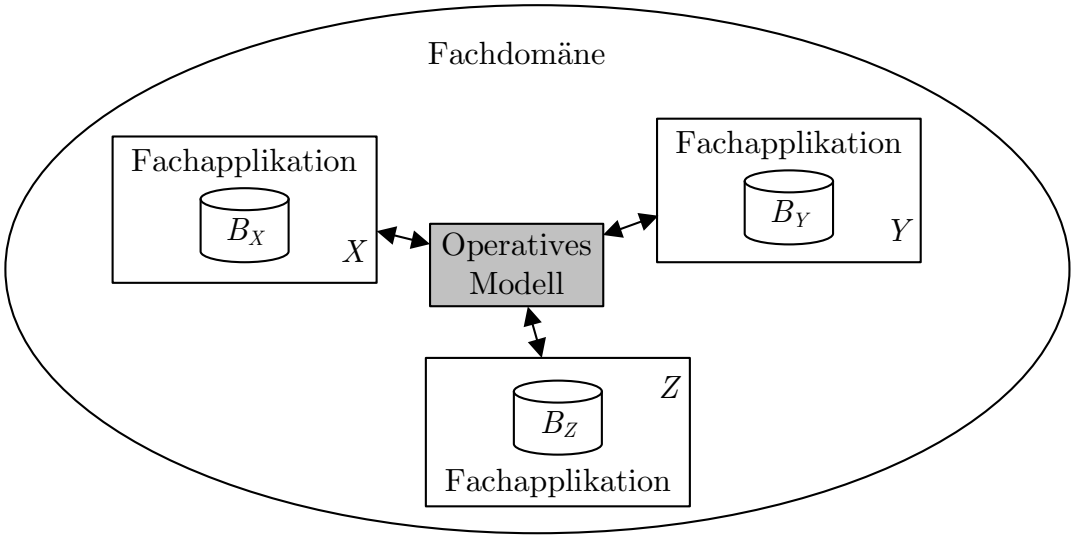


Bild 5.1: Standardisiertes operatives Modell in einer Fachdomäne

Formalisierung standardisierter Operationen: Es wird vorgeschlagen, die operative Modellierungssprache als Formalismus zur Definition von standardisierten Modellieroperationen zu verwenden. Neben der Festlegung der Syntax einzelner Operationen kommt der Definition der Semantik von Argumenten eine besondere Bedeutung zu.

Beispiel 5.1: Definition einer standardisierten Operation des operativen Modells

In den Beispielen 4.8 auf Seite 96 und 4.10 auf Seite 102 werden Modellieroperationen in OML bzw. POML zum Entwurf und zur Berechnung von Dreifeldträgern definiert. Beispielhaft wird angenommen, dass die Verarbeitung von Dreifeldträgern den Planungsaspekt einer Fachdomäne darstellt. Die Menge der in diesen Beispielen definierten Operationen entspricht dem standardisierten operativen Modell dieser Fachdomäne. Die Tabelle 5.1 zeigt eine formale Definition der Operation `ModLiveLoad`. Es werden sowohl die Syntax als auch die Semantik der Argumente angegeben.

Name	`ModLiveLoad`
Fachdomäne	Verarbeitung von Dreifeldträgern
Syntax (OML/ POML)	`ModLiveLoad -> "modliveload" Number ( ",[[" <POID>, "[" <POID>, <INTEGER> "]" ( ",[" <POID>, <INTEGER> "]" )* "]" )?`
Semantik	Modifikation der Verkehrslast auf neuen Wert `Number` in [kN/m] für selektierte oder spezifizierte Spannfelder; Spezifikation: i-te (`<INTEGER>`) Spannfelder der Dreifeldträger mit Identifikatoren (`<POID>`s)

Tabelle 5.1: Formale Definition einer standardisierten Operation

5.1.2 Umsetzung in Fachapplikationen

Sprachbasiertes operatives Modell: Jeder Modellieroperation des operativen Modells ist eine OML-Operation zugeordnet. Jede modellverändernde Modellieroperation wird durch eine entsprechende POML-Operation formuliert. Die sprachbasierte Beschreibung des operativen Modells entspricht demnach einer Menge von OML- und POML-Operationen. Die Definition und die Instanziierung der OML-Operationen zur Interaktion in der Benutzerschnittstelle sind im Abschnitt 4.4 auf Seite 96 beschrieben. Die Definition und die Instanziierung von persistenten POML-Operationen zur Formulierung von Änderungen sind im Abschnitt 4.5 auf Seite 102 erläutert.

Objektorientiertes operatives Modell: Jeder Modellieroperation des operativen Modells ist in einer objektorientierten Fachapplikation eine Operationsklasse zugeordnet. Die objektorientierte Beschreibung des operativen Modells entspricht demnach einer Menge von Operationsklassen. Zur Abbildung der Argumente einer Modellieroperation werden in den Operationsklassen entsprechende Attribute vorgesehen. Während der Laufzeit einer Fachapplikation werden Operationsklassen instanziiert und verarbeiten als Operationsobjekte die native Modellinstanz.

Operative Modelle in Fachapplikationen: Die Menge von semantischen Operationen zum Modellieren mit einer Fachapplikation wird das operative Modell der Fachapplikation genannt. In der Ein-/Ausgabekomponente von Fachapplikationen wird das operative Modell auf Basis der operativen Modellierungssprache beschrieben. Eine Operation wird dabei als OML-Operation definiert und zur Laufzeit instanziiert (OML-Operationsinstanz). In der Verarbeitungskomponente von Fachapplikationen wird das operative Modell auf Basis der Objektorientierung beschrieben. Eine Operation wird als Operationsklasse definiert und zur Laufzeit zum Operationsobjekt instanziiert. Operationsobjekte verarbeiten die Modellobjekte der nativen Modellinstanz einer Fachapplikation. Zur Formulierung von Änderungen wird das operative Modell auf Basis der operativen Modellierungssprache beschrieben. Eine persistente Operation wird dabei als POML-Operation definiert und zur Laufzeit instanziiert (POML-Operationsinstanz).

Das Bild 5.2 veranschaulicht die sprachbasierte und objektorientierte Umsetzung des operativen Modells in einer Fachapplikation.

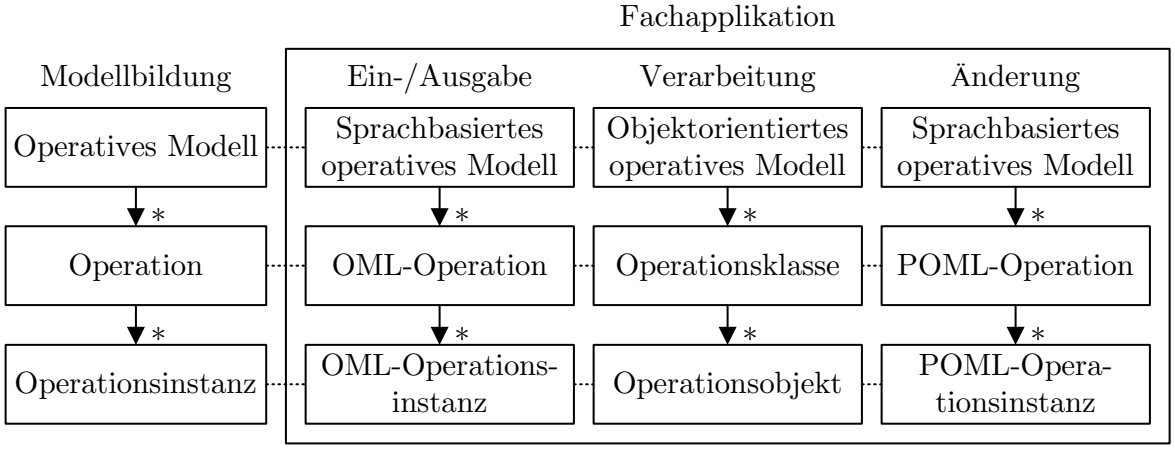


Bild 5.2: Operatives Modell in einer Fachapplikation: sprachbasiert und objektorientiert

5.2 Benutzerschnittstelle

In diesem Abschnitt wird ein Konzept zur Anwendung des verarbeitungsorientierten Modells und der operativen Modellierungssprache in der Benutzerschnittstelle von Fachapplikationen beschrieben. Es wird davon ausgegangen, dass verschiedene Fachapplikationen unterschiedliche objektorientierte Bauwerksmodelle verwenden. Die native Ein-/Ausgabekomponente und die native Verarbeitungskomponente einer bestehenden Fachapplikation werden dazu, wie im Folgenden beschrieben, erweitert.

5.2.1 Eingabe-/Ausgabe

Native Benutzerschnittstelle: In der nativen Benutzerschnittstelle einer Fachapplikation wird die Eingabe des Fachplaners entgegengenommen und ein (nativer) Befehl zur Verarbeitung der nativen Modellinstanz erzeugt. Zur Nutzerinteraktion zählen Mausereignisse, das Auswählen von Menüeinträgen, das Drücken von Knöpfen in Werkzeugleisten und im Allgemeinen auch das Eingeben von alphanumerischen Befehlen.

OML-Interpreter: Um die operative Modellierungssprache der OML-Sprachebene zur Formulierung von standardisierten Operationsinstanzen in der Benutzerschnittstelle verwenden zu können, muss die native Ein-/Ausgabekomponente um einen OML-Interpreter erweitert werden. Der OML-Interpreter wird auf Basis der OML-Grammatik der standardisierten Modellieroperationen entwickelt und ist in der Lage, die alphanumerischen OML-Operationsinstanzen zu interpretieren und entsprechende Operationsobjekte in der Verarbeitungskomponente zu erzeugen. Der in der nativen Eingabeschnittstelle spezifizierte alphanumerische Zeichenstrom wird dazu zum OML-Interpreter umgeleitet.

OML-Eingabestrom: Der vom OML-Interpreter auszuwertende alphanumerische Zeichenstrom wird OML-Eingabestrom genannt. Es werden zwei Quellen für den OML-Eingabestrom unterschieden.

- Direkte *Anwenderinteraktion* mit OML
- Vom Fachplaner erzeugte und in Dateien gespeicherte OML-*Modelliermakros*

Modelliermakros sind in Dateien zusammengefasste Folgen von OML-Operationsinstanzen.

Beispiel 5.2: Modelliermakros mit OML

Häufig wiederkehrende Modellierabläufe werden in Modelliermakros zusammengefasst. Für das häufige Erstellen von Zeichnungsrahmen mit einem Schriftfeld in CAD-Systemen können vorteilhaft Modelliermakros erstellt werden. In diesen Modelliermakros werden sequentiell OML-Operationsinstanzen zusammengefasst, die Linien und Texte in bestimmten Dicken und Höhen erstellen und positionieren.

Grafische Unterstützung: Bei der Ausführung von OML-Operationsinstanzen zur Selektion und Deselektion[1] wird der Fachplaner in der Ein-/Ausgabekomponente gra-

[1] Operationsgruppe SELECT/UNSELECT

fisch unterstützt. Die Ein-/Ausgabekomponente stellt Mechanismen bereit, die zur Objektidentifikation die Maus als Eingabegerät heranziehen, um grafisch interaktiv zu selektieren bzw. zu deselektieren. Diese Funktionalität ist im Allgemeinen in bestehenden Fachapplikationen verfügbar und kann verwendet werden.

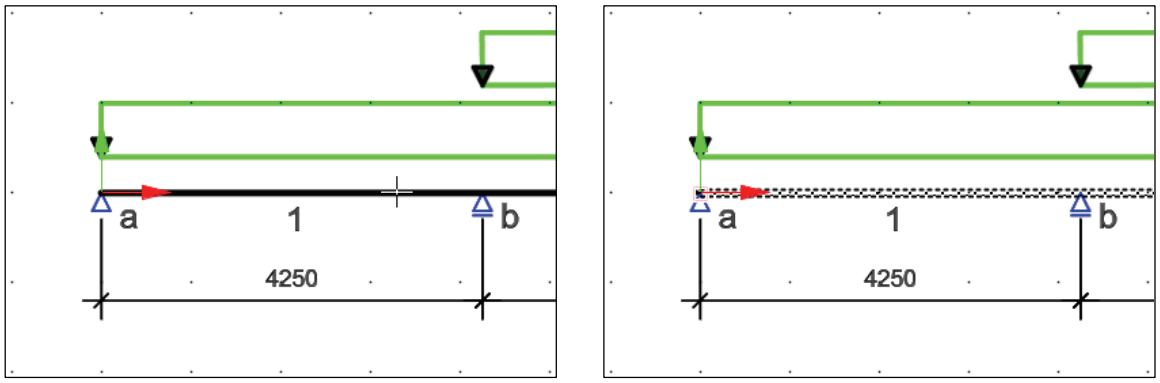
Alphanumerische Anzeige: Als Ergebnis von Abfrageoperationen[2] werden Informationen über die Modellinstanz oder über einzelne Modellobjekte als alphanumerischer Text in der Ein-/Ausgabekomponente der Fachapplikation angezeigt.

Beispiel 5.3: Anwenderinteraktion mit OML, grafische Unterstützung und alphanumerische Anzeige

In den Beispielen 4.8 und 4.9 auf den Seiten 96 bzw. 99 werden OML-Operationen zum Entwurf und zur Berechnung von Dreifeldträgern definiert und alphanumerisch instanziiert. Im Listing 4.1 des Beispiels 4.9 auf Seite 99 werden ab Zeile 20 nacheinander der Dreifeldträger selektiert (`selectbeam;`) und die Stützmomente berechnet und ausgegeben (`getsupportmoments;`). Der Anwender interagiert mit der Fachapplikation, indem er in der Benutzerschnittstelle die OML-Operationsinstanz zur Selektion ausführt.

```
OML> selectbeam
```

Die Fachapplikation reagiert und wartet auf die grafische Selektion des Balkens mit der Maus durch den Planer. Der Mauscursor wird beispielsweise als Fadenkreuz dargestellt (Bild 5.3a).



(a) Vor der Selektion (b) Nach der Selektion

Bild 5.3: Grafische Unterstützung bei der Selektion

Nachdem der Planer den Dreifeldträger grafisch identifiziert hat, wird dieser als selektiert angezeigt (Bild 5.3b). Der Fachplaner instanziiert alphanumerisch die Berechnung und Anzeige der Stützmomente für den selektierten Dreifeldträger, indem er die OML-Operationsinstanz zur Abfrage der Stützmomente ausführt. Die Fachapplikation berechnet die Stützmomente und zeigt die Ergebnisse der Abfrage alphanumerisch in der Benutzerschnittstelle an.

```
OML> getsupportmoments
Mb = -14.7 kNm, Mc = -16.4 kNm
```

[2]Operationsgruppe GET

5.2.2 Verarbeitung

Native Befehle: In der nativen Verarbeitungskomponente einer Fachapplikation stehen native Befehle bzw. Kommandos zur Verfügung. Die nativen Befehle verarbeiten die Modellinstanz, indem sie deren Zustand entsprechend der Applikations- und Modelllogik ändern.

Operationsklassen: Zur Verarbeitung der nativen Modellinstanz auf Basis des für die Fachdomäne standardisierten operativen Modells wird die Verarbeitungskomponente einer Fachapplikation um Operationsklassen erweitert. Die in der Ein-/Ausgabekomponente alphanumerisch formulierten OML-Operationsinstanzen werden in der Verarbeitungskomponente in Operationsobjekte überführt. Operationsobjekte werden anschließend auf die native Modellinstanz angewendet. Dazu rufen diese entweder entsprechende Methoden der Modellobjekte direkt oder verwenden native Befehle der Fachapplikation.

5.2.3 Applikationsunabhängigkeit

Standardisierte Modellierschnittstelle: Ein standardisiertes operatives Modell und die OML-Sprachebene der operativen Modellierungssprache bilden eine Modellierschnittstelle für Fachapplikationen innerhalb einer Fachdomäne. Die Verwendung der OML in der Benutzerschnittstelle macht das Modellieren unabhängig von einzelnen, spezifischen Fachapplikationen[3]. Wie im Bild 5.4 veranschaulicht, können Planer allein durch Kenntnis der im operativen Modell standardisierten OML-Modellieroperationen beliebige Fachapplikationen in einer Fachdomäne ad-hoc bedienen und somit ad-hoc modellieren.

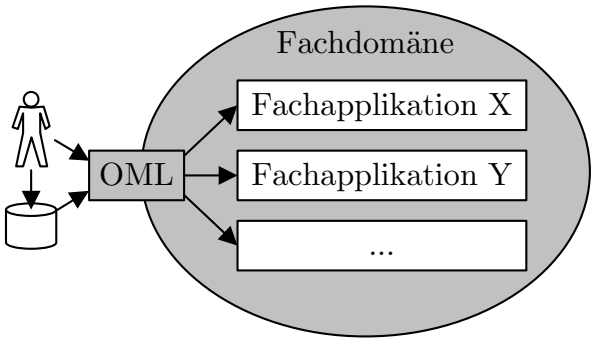


Bild 5.4: OML-Modellierung in beliebigen Fachapplikationen einer Fachdomäne

5.2.4 Zusammenfassung

Zur Anwendung der operativen Modellierungssprache in der Benutzerschnittstelle einer Fachapplikation wird die Ein-/Ausgabekomponente um einen OML-Interpreter erweitert. Der OML-Interpreter kapselt die native Eingabe, nimmt den alphanumerischen

[3] Als Analogie sei hier auf die Anwendung von SQL als unabhängige Definitions-, Abfrage- und Manipulationssprache bei relationalen Datenbanken verwiesen. Siehe [Date 2000, S. 83 ff.].

OML-Zeichenstrom entgegen und erzeugt Operationsobjekte in der Verarbeitungskomponente. Der OML-Eingabestrom wird entweder direkt vom Fachplaner eingegeben oder ist als Modelliermakro in einer Datei gespeichert. Die Verarbeitungskomponente wendet erzeugte Operationsobjekte auf die native Modellinstanz an, indem sie direkt oder über native Befehle auf der Modellinstanz operiert. Die Standardisierung von Modellieroperationen durch die operative Modellierungssprache der OML-Sprachebene bildet eine applikationsunabhängige Modellierschnittstelle für Fachapplikationen einer Fachdomäne. Planer können mit beliebigen Fachapplikationen direkt oder über Modelliermakros modellieren. Diesen Sachverhalt fasst das Bild 5.5 zusammen.

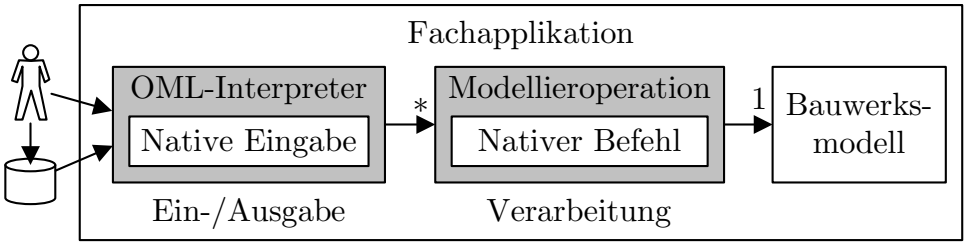


Bild 5.5: OML in der Benutzerschnittstelle einer Fachapplikation

5.3 Programmierschnittstelle

In diesem Abschnitt wird ein Konzept zur Anwendung des verarbeitungsorientierten Modells und der operativen Modellierungssprache in der Programmierschnittstelle von Fachapplikationen mit unterschiedlichen objektorientierten Modellen beschrieben. Mit Programmierung wird nicht die Erweiterung des nativen Bauwerksmodells verstanden, sondern das Automatisieren der Verarbeitung bzw. des Modellierens mit nativen Modellinstanzen in Fachapplikationen. Die native Verarbeitungskomponente einer bestehenden Fachapplikation wird wie im vorangegangenen Abschnitt 5.2 auf Seite 109 beschrieben, um Operationsklassen erweitert. Im Folgenden werden die Erweiterungen der Ein-/Ausgabekomponente dargelegt.

5.3.1 Eingabe-/Ausgabe

OPL-Interpreter: Um die operative Modellierungssprache der OPL-Sprachebene zur Formulierung von Modellierprogrammen in der Programmierschnittstelle einer Fachapplikation verwenden zu können, muss die native Ein-/Ausgabekomponente durch einen OPL-Interpreter erweitert werden. Der OPL-Interpreter wird auf Basis der applikationsunabhängigen OPL-Grammatik entwickelt und ist in der Lage, die alphanumerischen OPL-Anweisungen zu interpretieren. Der in der nativen Eingabeschnittstelle spezifizierte alphanumerische OPL-Zeichenstrom wird dazu zum OPL-Interpreter umgeleitet. Der OPL-Interpreter wertet das Modellierprogramm, bestehend aus Variablen, Prozeduren und Kontrollstrukturen, in eine Folge von OML-Operationsinstanzen aus. Diese Sequenz wird anschließend vom OML-Interpreter[4] weiterverarbeitet.

OPL-Eingabestrom: Der vom OPL-Interpreter zu verarbeitende alphanumerische Zeichenstrom heißt OPL-Eingabestrom. Es werden zwei Quellen für den OPL-Eingabestrom unterschieden.

- Direkte *Anwenderinteraktion* mit OPL
- Vom Fachplaner erzeugte und in Dateien gespeicherte OPL-*Modellierprogramme*

Direkte Anwenderinteraktion meint das Spezifizieren von parametrisierten OML-Operationsinstanzen auf Basis von Variablen, Prozeduren und Kontrollstrukturen durch den Planer zur Laufzeit einer Fachapplikation. OPL-Modellierprogramme hingegen sind Sammlungen von Prozedurdefinitionen, die in Prozedurdateien gespeichert werden. Nachdem diese Prozedurdateien zur Laufzeit der Fachapplikation geladen wurden, können die darin gespeicherten Prozeduren[5] ausgeführt werden.

Beispiel 5.4: Anwenderinteraktion und Modellierprogramme mit OPL und OML

Mit Bezug auf die Beispiele 4.8 und 4.9 auf den Seiten 96 bzw. 99 wird ein OPL-Modellierprogramm zur automatisierten Lastfallgenerierung für die maximalen Stützmomente bei Dreifeldträgern erstellt. Das OPL-Modellierprogramm wird in der im Listing 5.1 auf der nächsten Seite dargestellten Prozedurdatei `Traeger.pro` gespeichert.

[4] s. Abschnitt 5.2.1 auf Seite 109
[5] vgl. Abschnitt 4.3.3 auf Seite 91

```
// Prozedurdatei Traeger.pro
proc Run;
  selectlist = getselectedbeams;
  call Traeger.ShowDialog $selectlist[0];
endproc;

// Definieren und Anzeigen des Dialogs
proc ShowDialog beam;
  dialog d, "Maximale Stützmomente Mb und Mc";
  label l_beam, $beam;
  ...
  button b_ok, "OK";
  action = "call Traeger.EvalDialog $complist;";
  buttonaction $b_ok, $action;
  dialogadd $d, $b_ok, BOTTOM;
  listadd $complist, $b_ok;
  ...
  dialogshow $d;
endproc;

// Dialogkomponenten auswerten
proc EvalDialog complist;
  ...
  // Prozedur zur Lastfallgenerierung rufen
  Traeger.UseLoadCaseMaxMc $beam, $dl, $ll2, $ll3;
  ...
endproc;
...
// Lastfall max Mc erzeugen und Ergebnisse ausgeben
proc UseLoadCaseMaxMc beam, deadload, liveload2, liveload3;
  moddeadload $deadload, [$beam];
  modliveload 0.0, [[$beam, 1]];
  modliveload $liveload2, [[$beam, 2]];
  modliveload $liveload3, [[$beam, 3]];
  print "Lastfall max Mc:";
  getsupportmoments [$beam];
endproc;
```

Listing 5.1: Auszug aus der Prozedurdatei für ein OPL-Modellierprogramm zur automatisierten Lastfallgenerierung bei Dreifeldträgern

Nach der Erstellung des Dreifeldträgers mit Hilfe der standardisierten OML-Modelliervorgänge lädt der Planer zur Laufzeit der Fachapplikation die Prozedurdatei `Traeger.pro`. Anschließend selektiert er grafisch denjenigen Dreifeldträger, für den eine automatisierte Lastfallgenerierung durchgeführt werden soll (vgl. Beispiel 5.3 auf Seite 110). Zur Ausführung des OPL-Modellierprogramms ruft der Fachplaner die Prozedur `Traeger.Run` auf.

```
OPL> import Traeger;
OPL> selectbeam;
OPL> call Traeger.Run;
```

Das Bild 5.6 veranschaulicht die Benutzerschnittstelle der Fachapplikation mit Programmdialog, in dem der Planer entsprechende Eingaben macht.

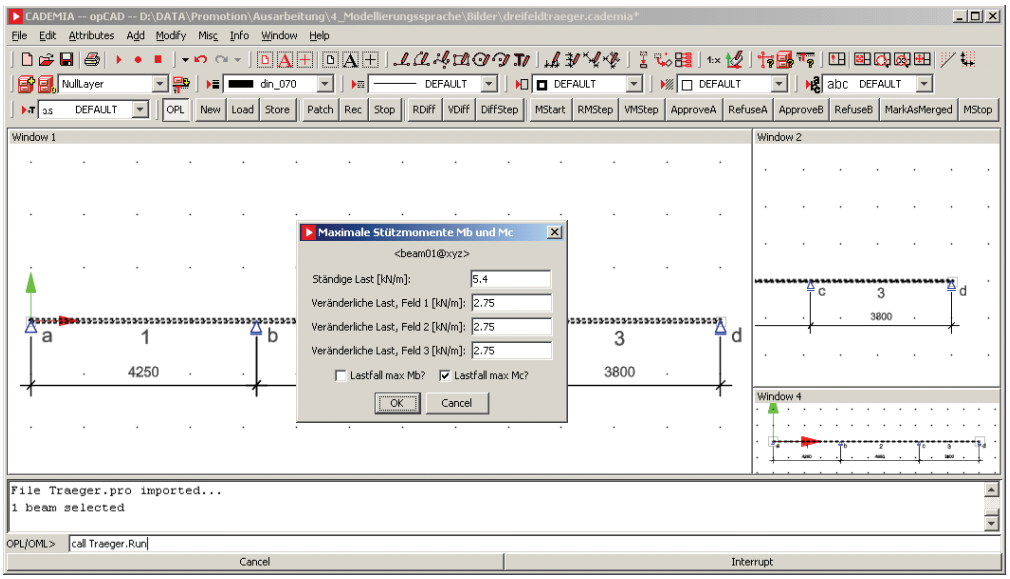


Bild 5.6: Dialog eines OPL-Modellierprogramms in einer Fachapplikation

Nach Bestätigung der Eingabe durch Klicken auf den OK-Knopf generiert das Modellierprogramm den Lastfall für das maximale Stützmoment am Auflager c und gibt die berechneten Stützmomente aus (s. Bild 5.7).

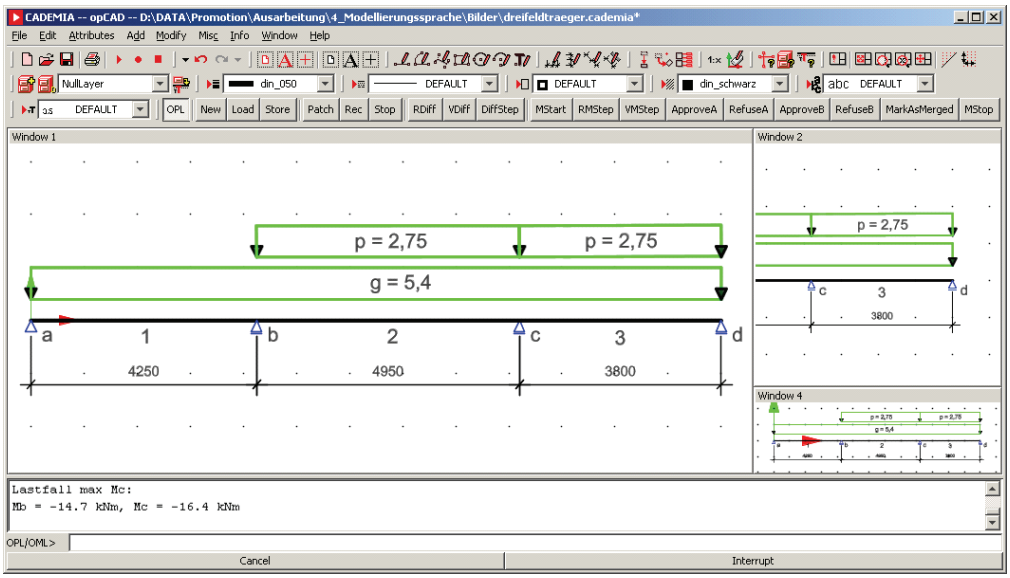


Bild 5.7: Fachapplikation nach Ausführung des OPL-Modellierprogramms

5.3.2 Verarbeitung

Operationsklassen: Die Verarbeitungskomponente der Fachapplikationen wird, wie bereits im Abschnitt 5.2.2 auf Seite 111 beschrieben, um das objektorientierte operative Modell in Form von Operationsklassen erweitert. Zur Laufzeit der Fachapplikation werden diese Klassen zu Operationsobjekten instanziiert und verarbeiten die native Modellinstanz.

5.3.3 Applikationsunabhängigkeit

Standardisierte Programmierschnittstelle: Standardisierte Modellieroperationen, die OPL- und die OML-Sprachebene der operativen Modellierungssprache bilden eine Programmierschnittstelle für die Modellierung mit Fachapplikationen innerhalb einer Fachdomäne. Die Verwendung der OPL in der Ein-/Ausgabekomponente macht das programmierte Modellieren unabhängig von einzelnen, spezifischen Fachapplikationen. Wie im Bild 5.8 veranschaulicht, können Fachplaner allein durch Kenntnis der OPL-Sprachkonstrukte und durch die standardisierten OML-Modellieroperationen beliebige Fachapplikationen in einer Fachdomäne ad-hoc programmieren und somit automatisiert modellieren. Einmal definierte Prozedurdateien werden als OPL-Modellierprogramme für verschiedene Fachapplikationen mit unterschiedlichen Objektmodellen eingesetzt und müssen nicht in applikationsspezifische Programme transformiert werden.

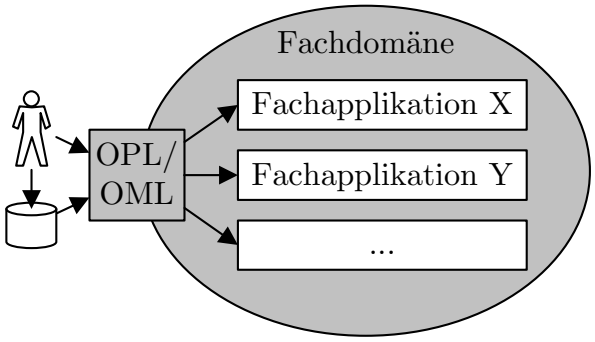


Bild 5.8: OPL-Programmierung von beliebigen Fachapplikationen

5.3.4 Zusammenfassung

Zur Anwendung der operativen Modellierungssprache in der Programmierschnittstelle einer Fachapplikation wird die Ein-/Ausgabekomponente um einen OPL-Interpreter erweitert. Der OPL-Interpreter kapselt die native Eingabe, nimmt den alphanumerischen OPL-Zeichenstrom entgegen und wertet die OPL-Anweisungen aus, indem eine Folge von OML-Operationsinstanzen erzeugt wird. Diese Sequenz wird vom OML-Interpreter weiterverarbeitet. Die Verarbeitungskomponente instanziiert entsprechende Operationsobjekte und operiert mit diesen auf der nativen Modellinstanz der Fachapplikation. Der alphanumerische OPL-Eingabestrom wird entweder direkt vom Fachplaner

eingegeben oder ist als Modellierprogramm in Prozedurdateien gespeichert. Die Standardisierung der programmierten Modellierung durch die operative Modellierungssprache der OPL- und OML-Sprachebene bildet eine applikationsunabhängige Programmierschnittstelle für Fachapplikationen einer Fachdomäne. Planer können beliebige Fachapplikationen direkt oder über in Prozedurdateien gespeicherte OPL-Modellierprogramme programmieren bzw. automatisiert modellieren. Diesen Sachverhalt fasst das Bild 5.9 zusammen.

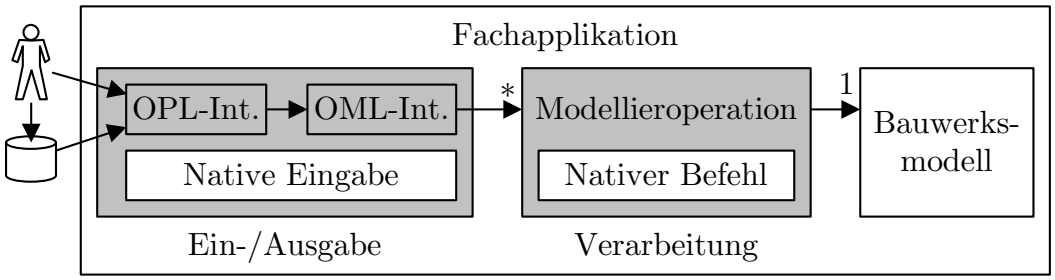


Bild 5.9: OPL in der Programmierschnittstelle einer Fachapplikation

5.4 Austausch von Bauwerksinformationen

In diesem Abschnitt wird ein Konzept zur Anwendung des verarbeitungsorientierten Modells und der operativen Modellierungssprache beim Austausch von Planungsständen zwischen Fachapplikationen mit unterschiedlichen objektorientierten Bauwerksmodellen beschrieben. Dem Informationsaustausch kommt im verteilten Bauplanungsprozess aufgrund der Probleme in der Praxis durch die große Applikations- und Modellvielfalt eine entscheidende Bedeutung zu. Aufbauend auf der Darstellung des grundsätzlichen Workflows innerhalb einer Fachdomäne wird die Erweiterung der Verarbeitungskomponente von Fachapplikationen um Funktionalität zum Aufzeichnen von Änderungen und zum Anwenden von gespeicherten Änderungen beschrieben. Die Vorteile des vorgeschlagenen Ansatzes gegenüber herkömmlichen Verfahren schließen diesen Abschnitt ab.

5.4.1 Workflow

Änderungsdatei: Um die verteilte Kooperation innerhalb einer Fachdomäne zu unterstützen, müssen Bauwerksinformationen zwischen den einzelnen Fachapplikationen ausgetauscht werden. Auf Grundlage des innerhalb einer Fachdomäne standardisierten operativen Modells werden persistente Änderungen der Modellinstanz als Änderungsdateien zwischen Fachapplikationen ausgetauscht. Änderungsdateien enthalten Sequenzen von alphanumerischen POML-Operationsinstanzen[6].

Verteilte Kooperation: Das Bild 5.10 veranschaulicht den Workflow zwischen den Fachapplikationen X und Y beim Informationsaustausch auf Basis von Änderungsdateien. Die verteilte Verarbeitung der Modellinstanz B und der Austausch erfolgen sequentiell. Während in X die Modellinstanzversion B_{aX} erstellt wird, werden die Änderungen in der Änderungsdatei $\delta_{\beta,a}$ gespeichert. Die Anwendung dieser Änderungen auf die virtuelle Version $B_{\beta Y}$ führt zur entsprechenden Version B_{aY} in Y. Anschließend erfolgt in Y die Weiterverarbeitung zur Version B_{bY}, die entsprechende Änderungsdatei $\delta_{a,b}$ wird gespeichert und auf die Version B_{aX} in X angewendet. Dieser Vorgang wiederholt sich analog für die Modellinstanzversion B_c.

5.4.2 Ein-/Ausgabe

Benutzerschnittstelle: Die Beschreibung des Informationsaustauschs zwischen Fachapplikationen auf Basis von Änderungsdateien ist von der Art der Benutzerschnittstelle unabhängig. Es spielt keine Rolle, ob native Eingabemöglichkeiten oder die operative Modellierungssprache der OPL- und OML-Sprachebenen zur Eingabe verwendet werden. Die Ein-/Ausgabekomponente von Fachapplikationen muss für Zwecke des Informationsaustauschs zwischen Fachapplikationen nicht erweitert werden.

[6] s. Abschnitt 4.5 auf Seite 102

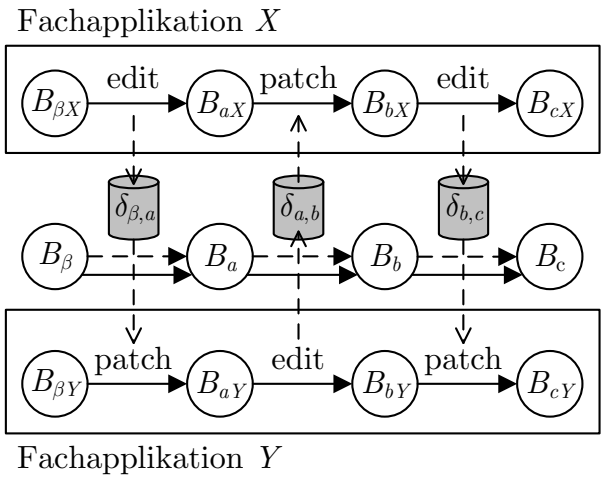


Bild 5.10: Sequentieller Workflow beim Austausch von Bauwerksinformationen auf Basis von Änderungsdateien

5.4.3 Verarbeitung

Operationsklassen: Die Verarbeitungskomponente der Fachapplikationen wird, wie bereits im Abschnitt 5.2.2 auf Seite 111 beschrieben, um das objektorientierte operative Modell in Form von Operationsklassen erweitert. Zur Laufzeit der Fachapplikation werden diese Klassen zu Operationsobjekten instanziiert und verarbeiten die native Modellinstanz.

Aufzeichnen von Änderungen

Journaling: Der Mechanismus zum Aufzeichnen von Änderungen der nativen Modellinstanz in Form von standardisierten POML-Operationsinstanzen wird Journaling[7] genannt.

Objektidentifikation: Dem vorgeschlagenen Modellierungsansatz folgend werden für die Beschreibung von Operanden persistente Objektidentifikatoren (POIDs[8]) verwendet. Nicht alle verfügbaren Fachapplikationen stellen POIDs zur persistenten Objektidentifikation bereit. Aus diesem Grund zählt die Generierung und die Verwaltung von POIDs zum Funktionsumfang des Journaling-Mechanismus. Jedem Objekt der nativen Modellinstanz wird genau eine POID zugeordnet.

POML-Rekorder: Um die Journaling-Funktionalität in einer Fachapplikation umzusetzen, muss die Verarbeitungskomponente um einen POML-Rekorder erweitert werden. Der POML-Rekorder zeichnet während der Verarbeitung der nativen Modellinstanz standardisierte POML-Operationsinstanzen in Änderungsdateien auf. Prinzipiell werden für diesen Mechanismus drei Verfahren unterschieden. Das Aufzeichnen von Änderungen kann

[7] vgl. Abschnitt 3.4.3 auf Seite 72
[8] s. Abschnitt 3.3.1 auf Seite 61

- auf der Basis standardisierter *OML-Modellieroperationen*,
- auf Grundlage *nativer Befehle* oder
- als Folge von Zustandsänderungen der *nativen Modellinstanz*

umgesetzt werden. Während das erste Verfahren davon ausgeht, dass die Modellinstanz durch standardisierte OML-Operationsinstanzen verarbeitet wird, stützen sich die anderen Verfahren auf die native Verarbeitungs- und die native Modellkomponente der Fachapplikation. Das zweite Verfahren zieht native Befehle der Fachapplikation zur Änderungsbeschreibung heran. Beim dritten Verfahren wird die Modellinstanz auf native Änderungen überwacht. In Abhängigkeit des Funktionsumfangs der Programmierschnittstellen von Fachapplikationen wird jeweils nur ein oder eine Kombination mehrerer dieser Verfahren zur Umsetzung des POML-Rekorders angewendet.

Modellieroperation: Erfolgt die Verarbeitung der Modellinstanz auf der Basis von standardisierten Modellieroperationen in Form von OML-Operationsinstanzen, dann werden die POML-Operationsinstanzen wie im Beispiel 4.11 auf Seite 103 beschrieben erzeugt und als Änderung aufgezeichnet.

Nativer Befehl: Werden in einer Fachapplikation native Befehle zur Verarbeitung der Modellinstanz verwendet, können diese vorteilhaft für das Journaling herangezogen werden. Die Semantik über die Zustandsänderung der Modellinstanz, das heißt, *was wie* geändert wurde, ist Gegenstand des nativen Befehls. Auf Basis dieser Informationen werden standardisierte POML-Operationen instanziiert und in Änderungsdateien gespeichert. Dieses Verfahren eignet sich besonders gut zum Aufzeichnen von Operationsinstanzen der Gruppe MODIFY[9].

Native Modellinstanz: Wird die native Modellinstanz in einer Fachapplikation direkt verarbeitet, ohne Befehle zu verwenden, dann muss der Zustand der Modellinstanz auf Änderungen überwacht werden. Hierfür eignet sich das Beobachter-Entwurfsmuster[10]. Der POML-Rekorder meldet sich als Beobachter an der Modellinstanz an und wird über einen geeigneten Notifikationsmechanismus über Zustandsänderungen der Modellinstanz informiert. Auf Grundlage dieser Informationen werden die standardisierten POML-Operationen instanziiert und in der Änderungsdatei aufgezeichnet. Dieses Verfahren eignet sich besonders gut zum Aufzeichnen von Operationsinstanzen der Gruppen ADD, SET und REMOVE. Im Bild 5.11 ist das Beobachter-Konzept zur Überwachung von Zustandsänderungen der Modellinstanz als UML-Klassendiagramm dargestellt.

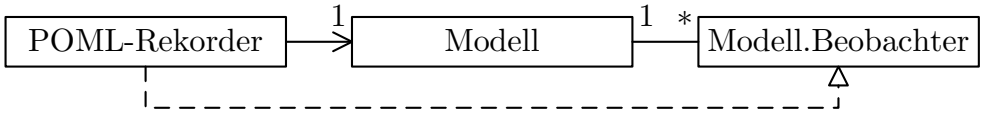


Bild 5.11: POML-Rekorder als Beobachter des nativen Modells

[9] vgl. Abschnitt 3.4.2 auf Seite 70
[10] engl.: Observer, s. [Gamma u. a. 2002]

Anwenden von Änderungen

Patching: Das Einspielen bzw. das Anwenden von gespeicherten Änderungen auf eine Version der nativen Modellinstanz wird Patching[11] genannt.

POML-Interpreter: Um die operative Modellierungssprache der POML-Sprachebene zur Formulierung von Änderungen in der Datenschnittstelle einer Fachapplikation verwenden zu können, muss die native Verarbeitungskomponente um einen POML-Interpreter erweitert werden. Der POML-Interpreter wird auf Basis der POML-Grammatik der standardisierten Modellieroperationen entwickelt. Er ist in der Lage, die alphanumerisch instanziierten POML-Operationen zu interpretieren und entsprechende Operationsobjekte in der Verarbeitungskomponente zu erzeugen. Der in der Änderungsdatei spezifizierte alphanumerische POML-Zeichenstrom wird dazu zum POML-Interpreter umgeleitet.

Zusammenfassung

Zur Anwendung des standardisierten operativen Modells und der operativen Modellierungssprache in der Datenschnittstelle einer Fachapplikation wird die Verarbeitungskomponente um einen POML-Rekorder und einen POML-Interpreter erweitert. Der POML-Rekorder implementiert einen Journaling-Mechanismus und zeichnet die während der Verarbeitung ausgeführten Änderungen der Modellinstanz in Form von POML-Operationsinstanzen in Änderungsdateien auf. Auf Basis dieser Änderungsdateien erfolgt der Informationsaustausch zwischen den Fachapplikationen innerhalb einer Fachdomäne. Der POML-Interpreter nimmt den alphanumerischen Zeichenstrom aus der Änderungsdatei entgegen, wertet die enthaltenen POML-Operationsinstanzen aus und erzeugt entsprechende Operationsobjekte zur Verarbeitung der nativen Modellinstanz. Die nativen objektorientierten Bauwerksmodelle der verschiedenen Fachapplikationen bleiben beim Informationsaustausch auf der Grundlage von Änderungsdateien unberührt. Diesen Sachverhalt fasst das Bild 5.12 auf der nächsten Seite zusammen.

5.4.4 Applikationsunabhängigkeit

Standardisierte Datenschnittstelle: Das standardisierten operative Modell und die POML-Sprachebene der operativen Modellierungssprache bilden eine Datenschnittstelle für Fachapplikationen innerhalb einer Fachdomäne. Die Verwendung der POML in der Datenschnittstelle macht den Informationsaustausch unabhängig von einzelnen, spezifischen Fachapplikationen und ihren unterschiedlichen Bauwerksmodellen. Wie im Bild 5.13 auf der nächsten Seite veranschaulicht, können Bauwerksinformationen auf Basis der im operativen Modell standardisierten POML-Operationen zwischen beliebigen Fachapplikationen innerhalb einer Fachdomäne ausgetauscht werden.

[11] vgl. Abschnitt 3.4.3 auf Seite 72

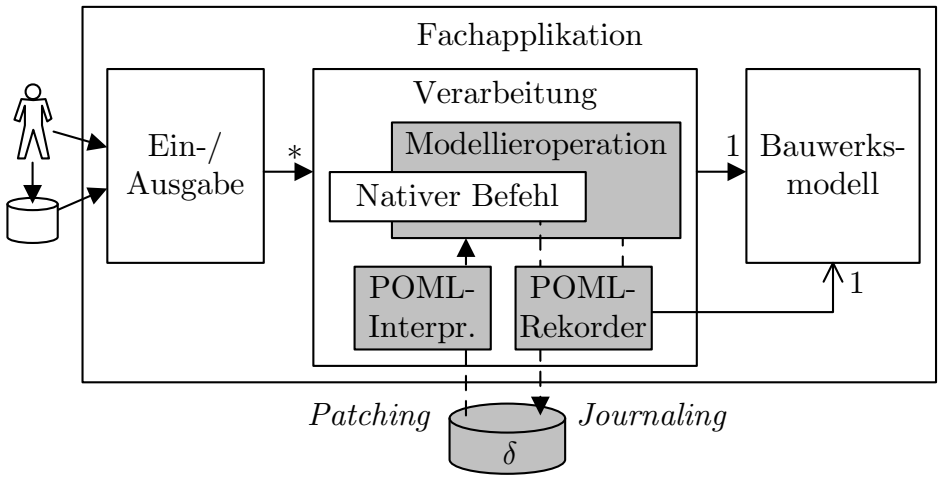


Bild 5.12: POML in der Datenschnittstelle einer Fachapplikation

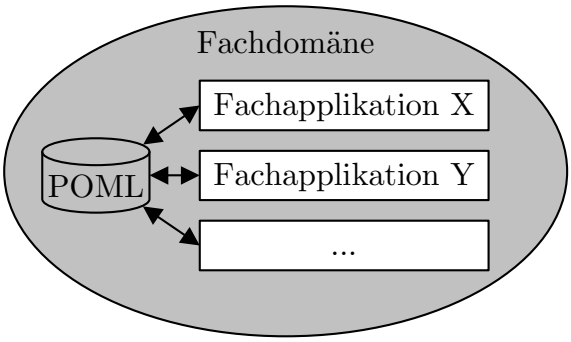


Bild 5.13: POML-Informationsaustausch zwischen beliebigen Fachapplikationen

5.4.5 Semantik

Operationen: Wie bereits im Abschnitt 3.3.2 auf Seite 62 in dieser Arbeit behandelt, besitzen Operation im Verarbeitungskontext von Fachapplikationen eine höhere Aussagekraft als die nativen Modelle der Applikationen selbst. Beim Austausch von Bauwerksinformationen auf der Basis von Operationen können im Gegensatz zum Austausch von ausgewerteten Modellinstanzen zusätzliche, semantische Änderungsinformationen zwischen einzelnen Fachapplikationen übertragen werden.

Entstehungsgeschichte und Entwurfsabsicht: In Analogie zum einleitenden Schachbeispiel des Abschnitts 1.2.1 auf Seite 3 speichern Änderungsdateien die Entstehungs- bzw. Verarbeitungsgeschichte der Modellinstanz. In gewisser Hinsicht wird auf diese Weise auch die Entwurfsstrategie und die Entwurfsabsicht[12] der jeweiligen Fachplaner beschrieben und gespeichert. Diese zusätzlichen Informationen können beim Vergleich[13] und bei der Zusammenführung[14] von Planungsständen berücksichtigt werden.

[12] engl.: design intent
[13] s. Abschnitt 5.6 auf Seite 129
[14] s. Abschnitt 5.7 auf Seite 139

5.4.6 Informationsverluste

Datentransformation: Bei herkömmlichen Verfahren werden innerhalb einer Fachdomäne Bauwerksinformationen in ausgewerteter Form, als Instanzen standardisierter zustandsorientierter Bauwerksmodelle ausgetauscht. Wie im Abschnitt 2.3.2 auf Seite 24 dieser Arbeit beschrieben, werden hierfür Datentransformationen von nativen Formaten in ein Standardformat und zurück vorgenommen. Diese Datentransformationen sind zwangsläufig mit kumulierenden Informationsverlusten verbunden, da einmal im Planungsprozess verlorengegangene Informationen nicht wiederhergestellt werden können.

Dateninterpretation: Beim vorgeschlagenen Anwendungskonzept werden Bauwerksinformationen in nicht-ausgewerteter Form, als Instanzen standardisierter operativer Modelle ausgetauscht. In diesem Verfahren werden die Daten nicht transformiert, sondern lediglich interpretiert und bleiben als Änderungsinformationen dauerhaft unverändert erhalten.

Informationsverluste: Beim Interpretieren und Anwenden der gespeicherten Änderungen können Informationsverluste auftreten, wenn das Ergebnis auszuführender Operationsinstanzen nicht in der nativen Modellinstanz abgebildet werden kann. Jedoch kumulieren die Informationsverluste beim sequentiellen Austausch von Änderungen nicht, weil die einmal gespeicherten Informationen nicht verändert werden und in anderen Fachapplikationen durchaus umgesetzt werden können. Die Informationsverluste treten demnach nur lokal in einzelnen Fachapplikationen auf. Das Bild 5.14 stellt diesen Sachverhalt am Beispiel des Informationsaustauschs zwischen drei Fachapplikationen schematisch dar. Die entstehenden Informationsverluste ∇_i werden als Differenzen von Informationsmengen nach [Firmenich 2004] beschrieben.

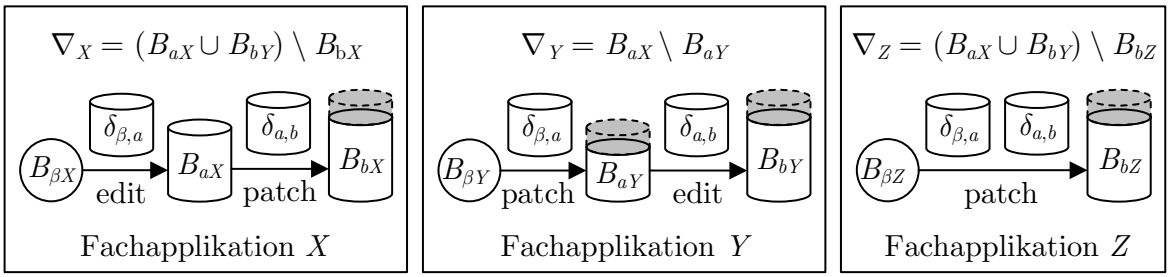


Bild 5.14: Nicht-kumulierender, lokaler Informationsverlust beim Austausch von Änderungen

In der Fachapplikation X wird die Modellinstanzversion B_{aX} erzeugt und die ausgeführten Änderungen $\delta_{\beta,a}$ werden gespeichert. Beim Anwenden dieser Änderungen auf die native Modellinstanzversion $B_{\beta Y}$ können lokal Informationen verlorengehen, wenn das native Modell von B die Operationsergebnisse nicht speichern kann. Der Informationsverlust in Y wird mit $\nabla_Y = B_{aX} \setminus B_{aY}$ beschrieben. Während der Verarbeitung der Modellinstanz in Y zur Version B_{bY} werden die Änderungen $\delta_{a,b}$ aufgezeichnet. Beim Patching dieser Änderungen $\delta_{a,b}$ in X wird die native Version B_{bX} mit möglichem

lokalem Informationsverlust ∇_X erzeugt. Die Originalinformation der derzeitigen Modellinstanzversion B_b wird durch die Vereinigung der Informationen aus B_{aX} und B_{bY} gebildet. Demnach ergibt sich der Informationsverlust ∇_X zu $(B_{aX} \cup B_{bY}) \setminus B_{bX}$. In der Fachapplikation Z soll der Modellinstanzzustand B_b erzeugt werden. Dazu werden die Änderungen $\delta_{\beta,a}$ und $\delta_{a,b}$ auf die Version $B_{\beta Z}$ angewendet. Es entsteht die native Modellinstanzversion B_{bZ} mit möglichem lokalem Informationsverlust ∇_Z. Mit der Originalinformation $(B_{aX} \cup B_{bY})$ ergibt sich ∇_Z zu $(B_{aX} \cup B_{bY}) \setminus B_{bZ}$. Die zu erkennenden Informationsverluste treten immer lokal in den jeweiligen Fachapplikationen auf, kumulieren aber nicht.

Bezieht man die Informationsverluste, unabhängig von einem bestimmten nativen Modell, allein auf die änderungsorientierten Informationsmengen im Standardformat δ (POML), dann werden die im Bild 5.15 dargestellten Informationsverluste bei lokaler Auswertung der Änderungen $\langle \delta_{\beta,a}, \delta_{a,b}, \delta_{a,c} \rangle$ zum Bauwerkszustand B_c ersichtlich. Ein Aufsummieren von Informationsverlusten ist beim sequentiellen Austausch von Zustandsänderungen nicht erkennbar (vgl. Bild 2.8 auf Seite 28).

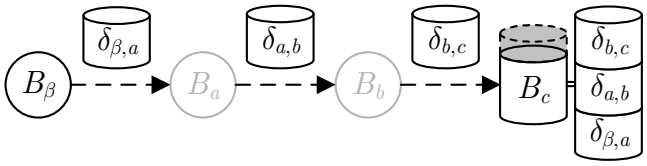


Bild 5.15: Informationsverluste in Bezug auf änderungsorientierte Informationsmengen im Standardformat δ (POML)

5.4.7 Datenmenge

Entwurfserstellung: Zu Beginn des Planungsprozesses ist die eine fachliche Instanz beschreibende Informationsmenge leer. Die erste Phase im Entwurfsprozess ist durch die vorwiegende Ausführung von Operationsinstanzen zur Konfiguration des Verarbeitungskontexts und zum Hinzufügen und Modifizieren von Modellobjekten gekennzeichnet. Vergleicht man die Datenmenge einer ausgewerteten Modellinstanz mit der Datenmenge der Änderungsdateien, dann ist die Datenmenge der gespeicherten Änderungen unter Umständen größer[15]. Dies steht jedoch dem Mehrnutzen im Hinblick auf die Speicherung der Entwurfsabsicht entgegen.

Entwurfsveränderung: Die folgenden Phasen der Entwurfsplanung sind durch eine iterative Entwurfsveränderung bzw. -anpassung gekennzeichnet. In diesen Phasen werden vorwiegend Modifikationsoperationen instanziiert und zur Verarbeitung verwendet. Im Gegensatz zur ersten Entwurfsphase sind die Datenmengen der Änderungsdateien viel kleiner als die Dateien zur Speicherung der vollständigen, ausgewerteten Modellinstanz. Hier bietet der vorgeschlagene Ansatz einen großen Vorteil.

[15] nicht zwangsläufig, vgl. Abschnitt 1.2.2 auf Seite 4

5.4.8 Zusammenfassung

Zur Anwendung eines standardisierten operativen Modells und der operativen Modellierungssprache der POML-Sprachebene beim Informationsaustausch wird die Verarbeitungskomponente von Fachapplikationen um einen POML-Rekorder und einen POML-Interpreter erweitert. Der POML-Rekorder implementiert einen Journaling-Mechanismus zum Aufzeichnen der bei der Verarbeitung der nativen Modellinstanz ausgeführten Änderungen in Form von POML-Operationsinstanzen in Änderungsdateien. Auf Basis dieser Änderungsdateien wird der Austausch von Bauwerksinformationen zwischen Fachapplikation in einer Fachdomäne durchgeführt. Der POML-Interpreter nimmt den alphanumerischen POML-Zeichenstrom aus der Änderungsdatei entgegen, wertet die POML-Operationsinstanzen aus und erzeugt entsprechende Operationsobjekte, welche auf die native Modellinstanz der Fachapplikation angewendet werden (Patching). Die Standardisierung von Modellieroperationen durch die operative Modellierungssprache der POML-Sprachebene bildet eine applikationsunabhängige Datenschnittstelle zwischen Fachapplikationen einer Fachdomäne. Die Verwendung der standardisierten POML-Operationen in Änderungsdateien erlaubt den Austausch von Bauwerksinformationen zwischen beliebigen Fachapplikationen. Dieser auf Operationen basierende Austausch ermöglicht im Gegensatz zu herkömmlichen Verfahren das Übertragen zusätzlicher Semantik zur Beschreibung der Entstehungsgeschichte einer Modellinstanzversion und der Entwurfsabsicht des Fachplaners. Darüber hinaus ist der vorgeschlagene Ansatz durch den großen Vorteil *nicht*-kumulierender Informationsverluste gekennzeichnet.

5.5 Archivierung von Bauwerksinformationen

In diesem Abschnitt wird ein Konzept zur Anwendung des verarbeitungsorientierten Modells und der operativen Modellierungssprache bei der Archivierung von Bauwerksinformationen beschrieben. Die zu archivierenden Informationen werden auf der Basis von Fachapplikationen mit unterschiedlichen objektorientierten Bauwerksmodellen erzeugt. Es wird dargelegt, dass das vorgeschlagene Konzept sehr gut zur Langzeitarchivierung geeignet ist.

5.5.1 Workflow

Änderungsdateien: Auf Grundlage des innerhalb einer Fachdomäne standardisierten operativen Modells werden persistente Änderungen der Modellinstanz in Änderungsdateien gespeichert. Änderungsdateien enthalten Sequenzen von alphanumerischen POML-Operationsinstanzen[16]. Soll eine bestimmte Modellinstanzversion in einer Fachapplikation wiederhergestellt werden, müssen entsprechend des änderungsorientierten Ansatzes die Änderungsdateien sequentiell auf die virtuelle Version angewendet werden. Somit bilden die Änderungsdateien und die Änderungsrelation den Archivierungsgegenstand. Es wird davon ausgegangen, dass die formale Beschreibung des standardisierten operativen Modells für eine Fachdomäne verfügbar ist[17]. Das Bild 5.16 veranschaulicht die zu archivierenden Änderungdateien mit ihren Beziehungen bei der verteilten Verarbeitung der Modellinstanz B.

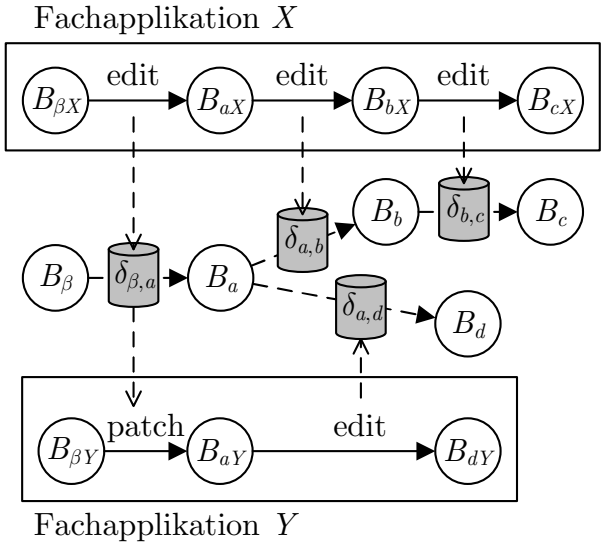


Bild 5.16: Workflow bei der Archivierung von Änderungsdateien

Vollständigkeit: Ein Nachteil bei der Archivierung von Änderungsinformation besteht in der Forderung der vollständigen Verfügbarkeit aller eine Version beschreibenden

[16] s. Abschnitt 4.5 auf Seite 102
[17] s. Abschnitt 5.1.1 auf Seite 105

Änderungsdateien. Fehlt eine Änderungsdatei, können betroffene Modellinstanzversionen nicht mehr erzeugt werden. Deshalb wird vorgeschlagen, Meilensteinversionen und Freigabestände der Modellinstanz zusätzlich in ausgewerteter Form zu archivieren[18].

5.5.2 Verarbeitung

Zur Anwendung eines standardisierten operativen Modells und der operativen Modellierungssprache der POML-Sprachebene bei der Archivierung werden die Verarbeitungskomponenten von Fachapplikationen entsprechend der Konzeption beim Informationsaustausch[19] um Operationsklassen, einen POML-Rekorder und einen POML-Interpreter erweitert werden.

5.5.3 Interpretierbarkeit

Dem Nachteil nach zwingender Vollständigkeit der Änderungsinformationen steht der Vorteil der einfachen Interpretierbarkeit von Änderungsdateien gegenüber. Nicht nur Applikationen, sondern auch die Fachplaner selbst können die Sequenzen von alphanumerischen POML-Operationsinstanzen interpretieren und von Hand in beliebigen Fachapplikationen ausführen.

5.5.4 Applikationsunabhängigkeit

Die Standardisierung des operativen Modells macht die archivierten Änderungen unabhängig von speziellen Fachapplikationen in einer Fachdomäne. Folglich hängen die Archivdaten auch nicht von den nativen Bauwerksmodellen der Fachapplikationen ab. Ziel der Archivierung ist die Gewährleistung, Informationen auch nach langer Zeit noch wiederherstellen zu können. Im Laufe dieser Zeit ändern sich jedoch die nativen Modelle[20] im Zuge der Weiterentwicklung von Fachapplikationen. Die Unabhängigkeit der archivierten Änderungsdateien stellt einen großen Vorteil dar. Die Verwendung einer Sprache zur Beschreibung der zu archivierenden Änderungen ist mit einem weiteren Plus verbunden. Bekanntermaßen ändert sich die Syntax und die Semantik einer Sprache nicht so schnell, wie sich die Struktur von Datenmodellen ändert[21]. Aufgrund dieser Eigenschaft ist das vorgeschlagene Archivierungskonzept zur Langzeitarchivierung gut geeignet.

5.5.5 Zusammenfassung

Auf der Basis eines standardisierten operativen Modells und der operativen Modellierungssprache der POML-Sprachebene werden Änderungsdateien für die Speicherung von Bauwerksinformationen innerhalb einer Fachdomäne eingesetzt. Die in den

[18] s. Abschnitt 2.3.3 auf Seite 31
[19] s. Abschnitt 5.4.3 auf Seite 119
[20] dieser Vorgang wird auch Schemaevolution genannt
[21] Analogie: SQL als unabhängige Sprache für unterschiedliche Datenbanken; s. [Date 2000, S. 83 ff.]

Änderungsdateien gespeicherten Sequenzen von alphanumerischen POML-Operationsinstanzen sind unabhängig von Fachapplikationen und ihren nativen Bauwerksmodellen, können vom Planer selbst und nicht nur von Software interpretiert werden und ändern bekanntermaßen nicht so häufig ihre Syntax und Semantik wie native Modellstrukturen. Diese Eigenschaften lassen den Schluss zu, dass der vorgeschlagene verarbeitungsorientierte Ansatz zusammen mit der operativen Modellierungssprache der POML-Sprachebene gut zur Langzeitarchivierung von Bauwerksinformationen geeignet ist.

5.6 Vergleich von Versionen

In diesem Abschnitt wird ein Konzept zur Anwendung des verarbeitungsorientierten Modells und der operativen Modellierungssprache beim Vergleich von unterschiedlichen Versionen des Planungsmaterials innerhalb einer Fachdomäne vorgestellt. Es wird davon ausgegangen, dass die Versionen auf Basis verschiedener Fachapplikationen mit unterschiedlichen objektorientierten Bauwerksmodellen erzeugt werden. Aufbauend auf der Darstellung des grundsätzlichen Workflows wird ein applikationsunabhängiger Diff-Algorithmus vorgestellt, der die Unterschiede zwischen Versionen auf der Basis der im Verarbeitungsgraphen gespeicherten Änderungsinformationen ermittelt. Abschließend wird ein Konzept für die Visualisierung von Versionsunterschieden in Fachapplikationen vorgestellt.

5.6.1 Workflow

Änderungsdateien: Im iterativen und verteilten Bauplanungsprozess entstehen verschiedene Zustände des Planungsmaterials – die Modellinstanzversionen. Es ist erforderlich, diese Versionen zu vergleichen, um Unterschiede herauszufinden und Entwürfe gegenüberzustellen. Auf Grundlage des innerhalb einer Fachdomäne standardisierten operativen Modells werden die gespeicherten Änderungen der Modellinstanz für einen solchen Vergleich herangezogen. Die Unterschiede zwischen einzelnen Versionen sind explizit in den Änderungsdateien enthalten.

Verteilte Kooperation: Im Bild 5.17 wird der Workflow beim Vergleichen von Modellinstanzversionen veranschaulicht. Während in X die Modellinstanzversion B_{cX} erzeugt wird, werden die entsprechenden Änderungen $\delta_{\beta,a}, \delta_{a,b}$ und $\delta_{b,c}$ gespeichert.

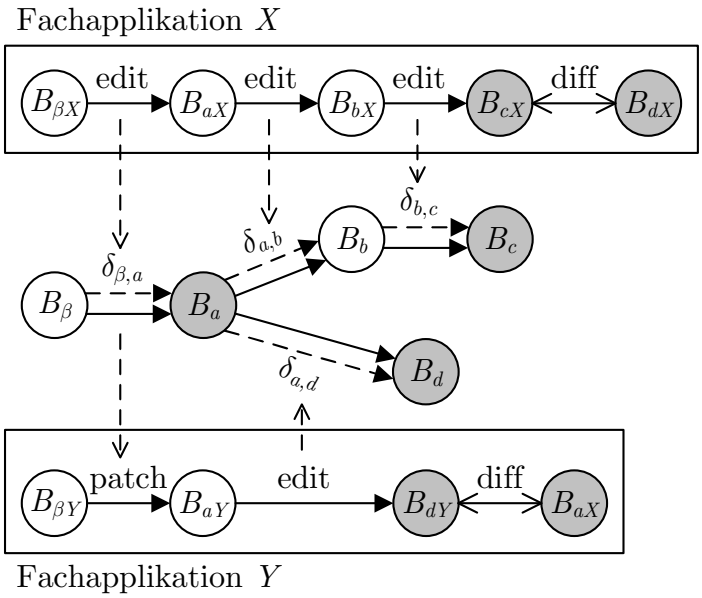


Bild 5.17: Workflow beim Vergleichen von Modellinstanzversionen

Der Planer mit der Fachapplikation Y entscheidet sich, die native Version B_{aY} auf Basis der Änderung $\delta_{\beta,a}$ zu generieren und anschließend eine Variantenplanung durchzuführen. Er erstellt parallel die Version B_{dY} und speichert die Änderung $\delta_{a,d}$. In X sollen anschließend die Varianten B_c und B_d verglichen werden, um die Entwürfe gegenüberzustellen. Weiterhin wird dargestellt, dass innerhalb von Y ein Revisionenvergleich zwischen der Version B_a und ihrer Revision B_d erfolgt. Wie die Vergleiche auf Basis der gespeicherten Änderungen durchgeführt werden, wird im folgenden Abschnitt beschrieben.

5.6.2 Diff-Algorithmus

Verarbeitungsgraph: Der vorgestellte Algorithmus zum Vergleichen von Modellinstanzversionen ist unabhängig von den unterschiedlichen, nativen Objektmodellen der Fachapplikationen. Er basiert lediglich auf der im Verarbeitungsgraphen[22] gespeicherten Änderungsrelation und den Änderungsdateien. Die Änderungsrelation bildet die Änderungsbeziehungen durch Kanten zwischen den Versionsknoten ab. Den Kanten sind die Änderungsdateien zugeordnet.

Algorithmus: Es besteht die Aufgabe, zwei Modellinstanzversionen $x_l, y_m \in V$ zu vergleichen. Dafür sind auf Basis der im Verarbeitungsgraphen gespeicherten Informationen zunächst die unterschiedlichen, den Versionen zugeordneten Änderungssequenzen wie folgt zu bestimmen (vgl. auf Seite 57 im Abschnitt 3.2.3):

1. Bestimmung der Wurzelpfade $r(x_l)$ und $r(y_m)$ der Modellinstanzversionen x_l bzw. y_m.
2. Bestimmung der Ursprungsversion $\varepsilon \in V$ als Endknoten des Durchschnittspfads $r(\varepsilon) = r(x_l) \cap r(y_m)$.
3. Bestimmung der unterschiedlichen Teilpfade $p_x = \langle(\varepsilon, x_i), \ldots, (x_{l-1}, x_l)\rangle$ und $p_y = \langle(\varepsilon, y_j), \ldots, (y_{m-1}, y_m)\rangle$ von der Ursprungsversion ε zur jeweiligen Version x_l und y_m.
4. Bestimmung der den Pfaden p_x und p_y zugeordneten Sequenzen von Änderungen $D_x = \langle\delta_{\varepsilon,i}, \ldots, \delta_{l-1,l}\rangle$ und $D_y = \langle\delta_{\varepsilon,j}, \ldots, \delta_{m-1,m}\rangle$.

Beispiel 5.5: Diff-Algorithmus für Revisionen und Varianten

Revisionen: Die im Verarbeitungsgraphen des Bildes 5.17 auf Seite 129 dargestellte Version B_a wird mit ihrer Revision B_d in der Fachapplikation Y verglichen. Die Wurzelpfade ergeben sich zu $r(B_a) = \langle(B_\beta, B_a)\rangle$ und $r(B_d) = \langle(B_\beta, B_a), (B_a, B_d)\rangle$. Als Ursprungsversion wird die Version B_a bestimmt. Aufgrund der Revisionsbeziehung zwischen den Versionen ergibt sich genau ein Teilpfad $p_d = \langle(B_a, B_d)\rangle$ von B_a zu B_d. Der Unterschied zwischen der Version B_a und ihrer Revision B_d ist in der diesem Pfad p_d zugeordneten Änderungssequenz $D_d = \langle\delta_{a,d}\rangle$ gespeichert.

[22] s. Abschnitt 3.2.4 auf Seite 59

Varianten: Es werden die im Verarbeitungsgraphen des Bildes 5.17 auf Seite 129 dargestellten Varianten B_c und B_d in der Fachapplikation X verglichen. Die Wurzelpfade der Versionen ergeben sich zu $r(B_c) = \langle (B_\beta, B_a), (B_a, B_b), (B_b, B_c) \rangle$ und $r(B_d) = \langle (B_\beta, B_a), (B_a, B_d) \rangle$. Als Ursprungsversion wird die Version B_a ermittelt. Folglich ergeben sich die unterschiedlichen Teilpfade zu $p_c = \langle (B_a, B_b), (B_b, B_c) \rangle$ und $p_d = \langle (B_a, B_d) \rangle$. Die die Unterschiede zwischen B_c und B_d speichernden Änderungssequenzen werden demnach als $D_c = \langle \delta_{a,b}, \delta_{b,c} \rangle$ und $D_d = \langle \delta_{a,d} \rangle$ ermittelt.

Modellinstanzkontext: Die Anwendung des vorgestellten Algorithmus liefert bei Revisionenvergleich eine (D_x), beim Variantenvergleich zwei Sequenzen (D_x, D_y) von Änderungen. Um diese auszuwerten, werden die in den Änderungen gespeicherten POML-Operationsinstanzen betrachtet. Die Zuordnung der Operationsinstanzen zu den Operationsgruppen ADD, MODIFY und REMOVE[23] und die persistente Identifikation der Operanden[24] erlaubt die Bestimmung derjenigen Modellobjekte, die in der jeweiligen Version x_l oder y_m hinzugefügt (A), geändert (C), gelöscht (R) oder nicht modifiziert (U) wurden. Die Modellobjekte $b \in B$ der Modellinstanz B werden dazu in die Objektmengen $A_x, A_y, C_x, C_y, R_x, R_y, U_x, U_y$ eingetragen.

$$A_x := \{b \mid b \text{ ist Operand einer Operationsinstanz } \omega \text{ in } D_x \wedge \omega \text{ ist der Operationsgruppe ADD zugeordnet}\} \subseteq B \quad (5.1)$$

$$A_y := \{b \mid b \text{ ist Operand einer Operationsinstanz } \omega \text{ in } D_y \wedge \omega \text{ ist der Operationsgruppe ADD zugeordnet}\} \subseteq B \quad (5.2)$$

$$C_x := \{b \mid b \text{ ist Operand einer Operationsinstanz } \omega \text{ in } D_x \wedge \omega \text{ ist der Operationsgruppe MODIFY zugeordnet}\} \subseteq B \quad (5.3)$$

$$C_y := \{b \mid b \text{ ist Operand einer Operationsinstanz } \omega \text{ in } D_y \wedge \omega \text{ ist der Operationsgruppe MODIFY zugeordnet}\} \subseteq B \quad (5.4)$$

$$R_x := \{b \mid b \text{ ist Operand einer Operationsinstanz } \omega \text{ in } D_x \wedge \omega \text{ ist der Operationsgruppe REMOVE zugeordnet}\} \subseteq B \quad (5.5)$$

$$R_y := \{b \mid b \text{ ist Operand einer Operationsinstanz } \omega \text{ in } D_y \wedge \omega \text{ ist der Operationsgruppe REMOVE zugeordnet}\} \subseteq B \quad (5.6)$$

$$U_x := B \setminus (A_x \cup C_x \cup R_x) \quad (5.7)$$

$$U_y := B \setminus (A_y \cup C_y \cup R_y) \quad (5.8)$$

Diese Objektmengen beschreiben die jeweiligen Unterschiede der betrachteten Version x_l bzw. y_m zu ihrer gemeinsamen Ursprungsversion ε. Ein semantischer Vergleich von Varianten ist direkt nicht möglich, sondern nur mit Bezug zur gemeinsamen Ursprungsversion sinnvoll.

Vergleichsmatrix der Modellinstanzvarianten: Im Bild 5.18 auf der nächsten Seite ist die Vergleichsmatrix der Modellinstanzversionen x_l und y_m dargestellt. Sie sagt aus, wie sich ein Modellobjekt der Modellinstanz in den Varianten bezüglich der Ursprungsversion entwickelt hat. Die jeweilige Entwicklung ist in den Änderungssequenzen

[23] Operationsgruppen: s. Abschnitt 3.4.2 auf Seite 70
[24] Objektidentifikation: s. Abschnitt 3.3.1 auf Seite 3.3.1

D_x und D_y gespeichert. Es ist ersichtlich, dass ein in einer Variante hinzugefügtes Modellobjekt in der anderen Variante weder hinzugefügt, geändert noch gelöscht werden kann, sondern unverändert sein muss. Dies liegt darin begründet, dass ein und dasselbe Modellobjekt nur in genau einer Fachapplikation zu genau einem Zeitpunkt erzeugt werden kann und einen eindeutigen persistenten Objektidentifikator (POID) erhält. Die potentiellen Konfliktsituationen für ein Modellobjekt sind hervorgehoben dargestellt.

Objekt $b \in$	A_y	C_y	R_y	U_y
A_x				in D_x hinzugefügt
C_x		in D_x und D_y geändert	in D_x geändert, in D_y gelöscht	in D_x geändert, in D_y unverändert
R_x		in D_x gelöscht, in D_y geändert	in D_x und D_y gelöscht	in D_x gelöscht, in D_y unverändert
U_x	in D_y hinzugefügt	in D_x unverändert, in D_y geändert	in D_x unverändert, in D_y gelöscht	in D_x und D_y unverändert

Bild 5.18: Vergleichsmatrix der Modellinstanzvarianten x_l und y_m für ein Modellobjekt

Konflikte: Zwischen zwei Varianten $x_l, y_m \in V$ besteht ein Entwurfskonflikt, wenn

- ein Modellobjekt in beiden Änderungssequenzen D_x und D_y modifiziert wurde oder
- ein Modellobjekt in einer Änderungssequenz gelöscht und in der anderen Änderungssequenz modifiziert wurde.

Die Menge derjenigen Modellobjekte $b \in B$, für die ein Konflikt bezüglich der Version $x_l, y_m \in V$ vorliegt, wird als Konfliktmenge $K(x_l, y_m)$ definiert.

$$K(x_l, y_m) := \{b \mid (b \in C_x \wedge b \in C_y) \vee (b \in C_x \wedge b \in R_y) \vee (b \in C_y \wedge b \in R_x)\} \tag{5.9}$$

Objektkontext: Im Gegensatz zum direkten Objektvergleich auf Basis der Attributwerte[25] werden im vorgeschlagenen Ansatz die Modellobjekte auf Basis der ihnen zugeordneten Operationsinstanzen verglichen. Diejenigen Operationsinstanzen, in denen das Modellobjekt als Operand spezifiziert ist, beschreiben die Änderungen des Objektzustandes von der gemeinsamen Ursprungsversion zu der jeweiligen Variante. Der Unterschied zwischen den Objektversionen ist somit explizit als Zustandsänderung verfügbar. Objekte, die ihren Zustand während der Verarbeitung zwar ändern, aber

[25] vgl. Abschnitt 2.3.4 auf Seite 33

nicht als Operanden in Operationsinstanzen auftreten, sind abhängige Objekte. Der Zustand von abhängigen Objekten wird nicht von Operationsinstanzen, sondern von anderen Modellobjekten entsprechend der Logik des nativen Modells verändert. Abhängige Objekte sind nicht im operativen Modell als Operanden berücksichtigt und werden deshalb für den Vergleich von Modellinstanzversionen nicht betrachtet.

Beispiel 5.6: Vergleich im Objektkontext

Gegeben sei ein einfaches Modell zur Beschreibung einer Zeichnung bestehend aus geometrischen Grundelementen und assoziativer Bemaßung und Schraffur. Im operativen Modell werden unter anderem Operationen für das geometrische Manipulieren der Grundelemente (Verschieben, Rotieren, Spiegeln usw.) definiert. Die Bemaßungs- und Schraffurobjekte sind abhängige Objekte, weil sie nicht direkt von Operationsinstanzen verarbeitet werden, sondern ihren Zustand genau dann ändern, wenn sich das bemaßte bzw. schraffierte Grundelement in seiner Geometrie ändert. Wie im Bild 5.19 veranschaulicht, ändert ein Rechteckobjekt x seine geometrischen Eigenschaften von der Zeichnungsversion B_a zur Version B_b durch die Änderungen $\delta_{a,b}$ und von Version B_a zur Version B_c durch die Änderung $\delta_{a,c}$. In diesen Änderungen sind jeweils eine Operationsinstanz zum Verschieben und eine andere zum Rotieren des Rechteckobjekts gespeichert.

$\delta_{a,b} = \langle$ `move 1.0, 1.0, [<x>];` $\rangle$
$\delta_{a,c} = \langle$ `rotate 1.0, 1.0, 40.0, [<x>];` $\rangle$

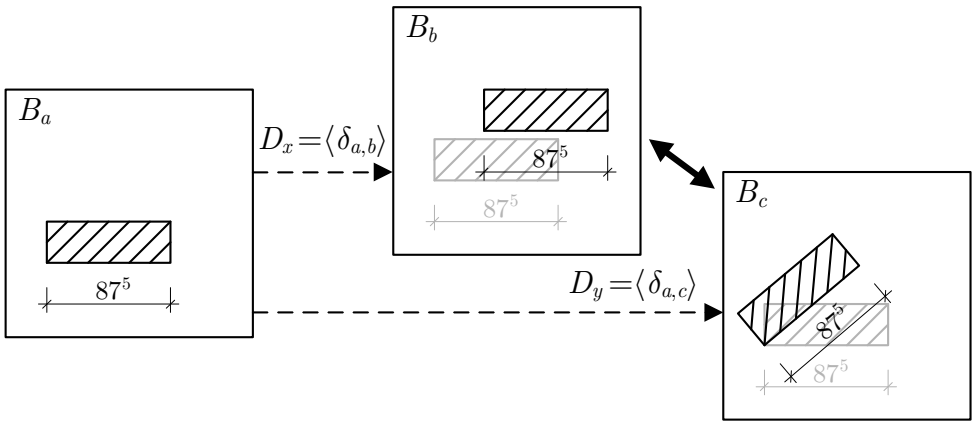


Bild 5.19: Vergleich im Objektkontext

Der Vergleich der Modellinstanzversion B_a mit der Revision B_c auf Basis von Attributwerten ergibt einen geometrischen Unterschied beim Rechteckobjekt, der nicht weiter spezifiziert werden kann. Im verfolgten Ansatz ist dieser Unterschied explizit in der Operationsinstanz als Rotationsvorschrift gespeichert. Der Unterschied zwischen Varianten B_b und B_c wird in Bezug auf die Ursprungsversion B_a bestimmt. Ein Vergleich auf Basis der Attributwerte kann zwar eine jeweils unterschiedliche geometrische Manipulation bezüglich der Ursprungsversion feststellen, doch diese nicht weiter spezifizieren. Die Betrachtung der in den Änderungen enthaltenen Operationsinstanzen ergibt: Die Versionen des Rechteckobjekts der Modellinstanzversionen B_b und B_c unterscheiden

sich semantisch dadurch, dass das Rechteck in B_b verschoben und in B_c gedreht wurde. Hier kommt der Vorteil des Objektvergleichs auf Basis semantischer Operationen zum Tragen.

Operationskontext: Der Vergleich auf Basis von gespeicherten Änderungen bietet einen zusätzlichen Aspekt beim Vergleichen von Modellinstanzversionen – den Vergleich im Operationskontext. Die gespeicherten Sequenzen von Operationsinstanzen geben explizit Aufschluss darüber, welche Teilmenge von Modellobjekten denselben Verarbeitungsschritt erfahren hat. Das heißt, welche Objektmenge wurde gemeinsam mit derselben Operationsinstanz manipuliert. Der Unterschied zwischen Modellinstanzversionen wird nicht allein als Summe der Einzelunterschiede im Objektkontext verstanden. Der vorgestellte Ansatz erlaubt die Bestimmung von aussagekräfigen Unterschieden zwischen Versionen auf Basis von Objektmengen.

Beispiel 5.7: Vergleich im Operationskontext

Gegeben sei ein einfaches Modell für den Entwurf und die Bemessung eines Fachwerkträgers. Im operativen Modell werden unter anderem Operationen für die Manipulation der Querschnittseigenschaften der Fachwerkstäbe definiert. Wie im Bild 5.20 dargestellt, ändern sich die Querschnitte der Obergurte v, w und der Untergurte x, y, z. Es wird angenommen, dass – aufgrund einer bestimmten Entwurfsintention – während der Verarbeitung die Ober- und Untergurte zusammen selektiert und anschließend durch *eine* Operationsinstanz manipuliert werden.

$\delta_{a,b} = \langle$ `modsection ... [<v>, <w>, <x>, <y>, <z>];` $\rangle$

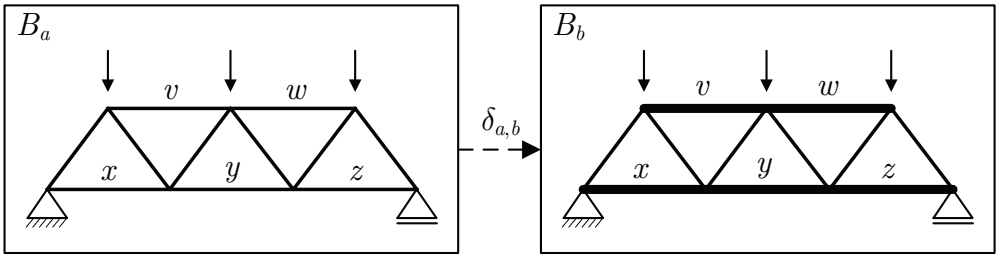
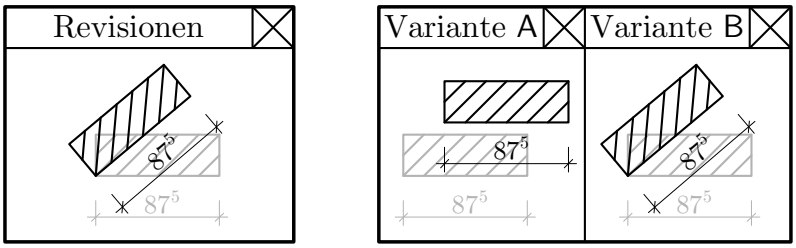


Bild 5.20: Vergleich im Operationskontext

Ein Vergleich im Objektkontext ergibt einen Unterschied zwischen den Trägerversionen B_a und B_b, der durch die fünf Einzelunterschiede der separaten Objektzustände charakterisiert ist. Im Gegensatz dazu wird beim Vergleich im Operationskontext *ein* Unterschied auf Basis der modifizierten Objektmenge $\{v, w, x, y, z\}$ ermittelt. Die gemeinsame Querschnittsmodifikation aller Ober- und Untergurtobjekte unterscheidet die beiden Fachwerkträgerversionen voneinander. Dieser aussagekräftige Unterschied zwischen zwei Versionen kommt sowohl beim Revisionen- als auch beim Variantenvergleich zum Tragen.

5.6.3 Visualisierungskonzept

Grafische Fenster: Die Berechnung der Unterschiede zweier Versionen ist applikationsunabhängig. Zur Präsentation der Vergleichsergebnisse wird jedoch die Funktionalität von Fachapplikationen zur grafischen Darstellung der Modellinstanzzustände herangezogen. Entsprechend der herkömmlichen Unterschiedvisualisierung beim Dokumentenvergleich[26] werden hinzugefügte, geänderte, gelöschte und unveränderte Modellobjekte durch unterschiedlich farbliche Darstellung hervorgehoben. Beim Vergleich einer Version mit ihrer Revision wird in einem grafischen Fenster der Zustand der Revision in Bezug auf die andere Version, die Ursprungsversion dargestellt (Bild 5.21a). Beim Variantenvergleich hingegen, werden zwei grafische Fenster verwendet, die jeweils den Zustand der Variante in Bezug auf die gemeinsame Ursprungsversion visualisieren (Bild 5.21b).



(a) Revisionenvergleich (b) Variantenvergleich

Bild 5.21: Grafische Visualisierung des Vergleichs von Modellinstanzversionen

Ordnungs- und Navigationsstruktur: Neben der grafischen Darstellung der Modellinstanzzustände werden die ermittelten Unterschiede zwischen Versionen je nach Kontext aufbereitet und in einer geordneten Navigationsstruktur abgelegt.

Modellinstanzkontext: Auf Modellinstanzebene wird eine hierarchische Navigationsstruktur nach Operationsgruppen ADD, MODIFY, REMOVE und SET vorgeschlagen. Beliebige übergeordnete modellspezifische Hierarchien[27] sind denkbar. Die Modellobjekte werden den Operationsgruppen ADD ($A_x \cup A_y$), MODIFY ($C_x \cup C_y$) und REMOVE ($R_x \cup R_y$) entsprechend der in den Gleichungen 5.1 bis 5.6 auf Seite 131 definierten Objektmengen zugeordnet.

Objektkontext: Die Navigationsstruktur auf Modellinstanzebene wird im Hinblick auf die Vergleichsergebnisse auf Objektebene erweitert. Jedem Modellobjekt werden die das Objekt verarbeitenden Operationsinstanzen sequentiell zugeordnet. Wie bereits erwähnt, werden abhängige Modellobjekte im Vergleich und somit auch bei der Visualisierung der Vergleichsergebnisse nicht berücksichtigt. Dies reduziert die Komplexität für den Fachplaner erheblich, denn nur direkt verarbeitete Modellobjekte werden in der Navigationsstruktur angezeigt. Operationsinstanzen der Gruppe SET konfigurieren den

[26] vgl. Abschnitt 2.3.4 auf Seite 32
[27] z. B. Gruppierungen nach Etagen, Bauteilen, Layern usw.

Verarbeitungskontext, sind also keinem Objekt zugeordnet und werden demnach direkt als Elemente in die Ebene der Operationsgruppe SET eingetragen.

Beispiel 5.8: Objektbezogene Navigation bei Visualisierung von Unterschieden

Das Bild 5.22 veranschaulicht schematisch die vorgeschlagene objektbezogene Navigationsstruktur bei der Präsentation der Vergleichsergebnisse als Auswertung der Änderungssequenzen D_x und D_y. In D_x werden das Modellobjekt `<u>` hinzugefügt und modifiziert, die Objekte `<w>` und `<x>` verändert und das Objekt `<z>` gelöscht. Die Änderungssequenz D_y speichert das Hinzufügen des Modellobjekts `<v>`, das Modifizieren des Objekts `<w>` und das Löschen der Objekte `<x>` und `<y>`. Die Modellobjekte `<w>` und `<x>` sind hervorgehoben, weil für sie Konflikte festgestellt wurden. Während beide Änderungssequenzen die Modifikation des Objekts `<w>` speichern, wird das Objekt `<x>` in D_x geändert und parallel dazu in D_y gelöscht.

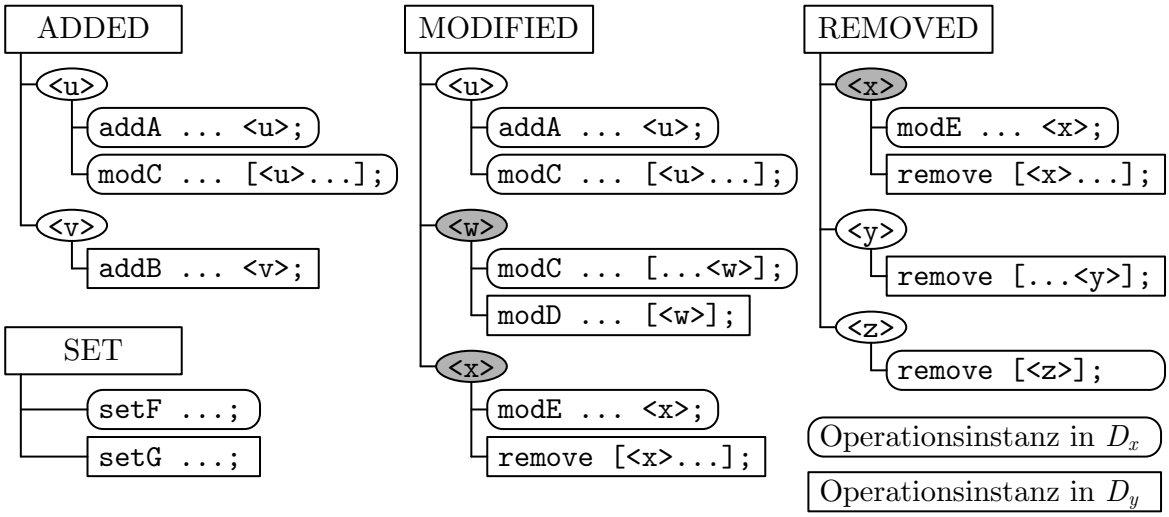


Bild 5.22: Objektbezogene Navigationsstruktur

Operationskontext: Für die Aufbereitung der Vergleichsergebnisse im Operationskontext wird ebenfalls eine hierarchische Ordnungs- bzw. Navigationsstruktur vorgeschlagen. Die oberste Ebene entspricht wiederum den Operationsgruppen. In die darunterliegende Ebene werden die zugehörigen Operationsinstanzen der Ausführungsreihenfolge entsprechend sequentiell eingetragen. Die dritte und unterste Ebene repräsentiert die Modellobjekte, die mit ein und derselben Operationsinstanz verarbeitet wurden. Auf Basis dieser Navigationsstruktur erkennt der Fachplaner, welche Modellobjekte gemeinsam als Objektmenge manipuliert wurden.

Beispiel 5.9: Operationsbezogene Navigation bei Visualisierung von Unterschieden

Das Bild 5.23 veranschaulicht schematisch die vorgeschlagene operationsbezogene Navigationsstruktur bei der Präsentation der Vergleichsergebnisse in Bezug auf das vorangegangene Beispiel 5.8. Es ist ersichtlich, dass die Modellobjekte `<u>` und `<w>` mit derselben Operationsinstanz `modC` modifiziert und die Objekte `<x>` und `<y>` gleichzeitig gelöscht wurden.

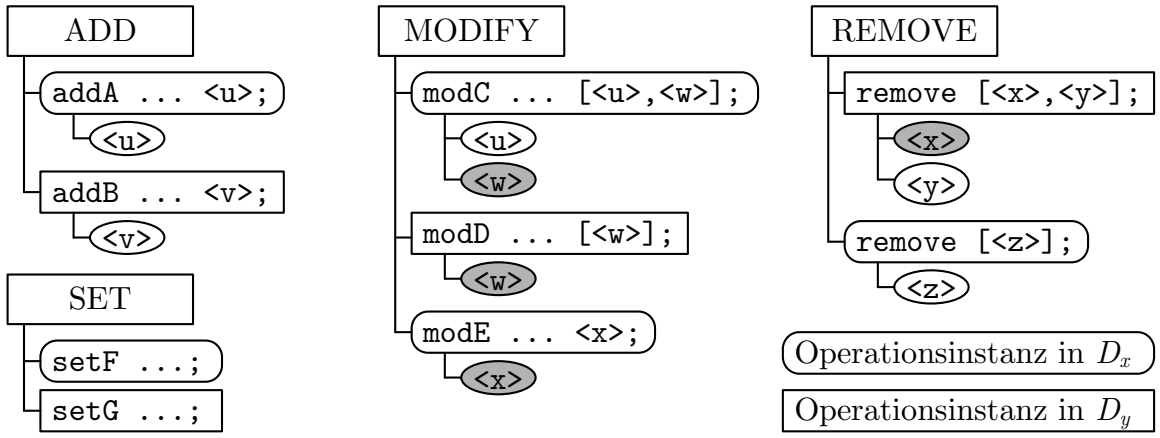


Bild 5.23: Operationsbezogene Navigationsstruktur

Schritt-für-Schritt-Vergleich: Eine weitere sinnvolle Möglichkeit der Visualisierung von Vergleichsergebnissen auf der Basis von gespeicherten Änderungen ist das schrittweise Ausführen und Rückgängigmachen von Operationsinstanzen. Auf Grundlage eines langsamen „Vor- und Zurückspielens“ wird der Fachplaner in die Lage versetzt, die ursprünglichen Entwurfsgedanken nachzuvollziehen und Änderungen besser zu verstehen. Das Ausführen von Operationsinstanzen kann dabei objektbezogen oder operationsbezogen erfolgen. In den vorgestellten Navigationsstrukturen werden entweder Modellobjekte oder Operationsinstanzen identifiziert. Während beim objektbezogenen Abspielen nur die die ausgewählten Modellobjekte betreffenden Operationsinstanzen angewendet werden, werden im Operationskontext alle Operationsinstanzen nacheinander ausgeführt. Dem Fachplaner werden so die Unterschiede zwischen den Versionen anhand ausgewählter Modellobjekte bzw. der gesamten Modellinstanz Schritt für Schritt präsentiert. Diese Art der Visualisierung von Unterschieden ist auf Basis des direkten Vergleichs der Modell- und Objektzustände nicht möglich.

5.6.4 Zusammenfassung

Auf der Basis eines standardisierten operativen Modells und der operativen Modellierungssprache der POML-Sprachebene beschreiben Änderungsdateien den Verarbeitungsprozess von Modellinstanzen innerhalb einer Fachdomäne. Die im Verarbeitungsgraphen der Modellinstanz gespeicherten Änderungsinformationen bilden die Grundlage für einen neuartigen, applikations- und datenmodellunabhängigen Diff-Algorithmus zum Vergleich von Modellinstanzversionen. Die Unterschiede zwischen einzelnen Versionen sind explizit als separate Sequenzen von POML-Änderungen gespeichert. Auf Grundlage dieser Änderungssequenzen werden Modellobjekte ermittelt, für die ein Entwurfskonflikt vorliegt. Im Gegensatz zu herkömmlichen Verfahren bietet der vorgestellte Ansatz die Möglichkeit, Unterschiede zwischen Versionen eines Modellobjekts nicht nur zu erkennen, sondern genau zu spezifizieren. Darüber hinaus erlaubt der Versionenvergleich im Operationskontext, Modifikationen von Objektmengen als Einheit zu betrachten und so Entwurfsintentionen bei der Unterschiedbestimmung zu

berücksichtigen. Die dem Fachplaner präsentierten objektbezogenen und operationsbezogenen Ordnungsstrukturen sowie ein Schritt-für-Schritt-Vergleich bilden die visuelle Basis für die Navigation durch die ermittelten Unterschiede zwischen Modellinstanzversionen.

5.7 Zusammenführung von Versionen

In diesem Abschnitt wird ein Konzept zur Anwendung des verarbeitungsorientierten Modells und der operativen Modellierungssprache beim Zusammenführen von unterschiedlichen Versionen des Planungsmaterials innerhalb einer Fachdomäne vorgestellt. Es wird davon ausgegangen, dass die Versionen auf Basis verschiedener Fachapplikationen mit unterschiedlichen objektorientierten Modellen erzeugt wurden. Aufbauend auf der Darstellung des grundsätzlichen Workflows wird ein applikationsunabhängiger Merge-Algorithmus vorgestellt, der die unterschiedlichen Versionen auf der Basis der im Verarbeitungsgraphen gespeicherten Änderungsinformationen und den Konzepten zum Journaling und Patching zusammenführt. Abschließend wird ein Konzept für die visuelle Unterstützung bei der Zusammenführung von Versionen in Fachapplikationen beschrieben.

5.7.1 Workflow

Änderungsdateien: Im iterativen und verteilten Bauplanungsprozess entstehen verschiedene Zustände des Planungsmaterials – die Modellinstanzversionen. Bilden diese Versionen Teillösungen innerhalb einer größeren Planungsaufgabe ab oder bestehen Planungskonflikte zwischen ihnen, ist es erforderlich, diese Versionen nach dem Vergleichen zu einer Gesamtlösung zusammenzuführen. Auf Grundlage des innerhalb einer Fachdomäne standardisierten operativen Modells werden die gespeicherten Änderungen der Modellinstanz für eine solche Zusammenführung herangezogen.

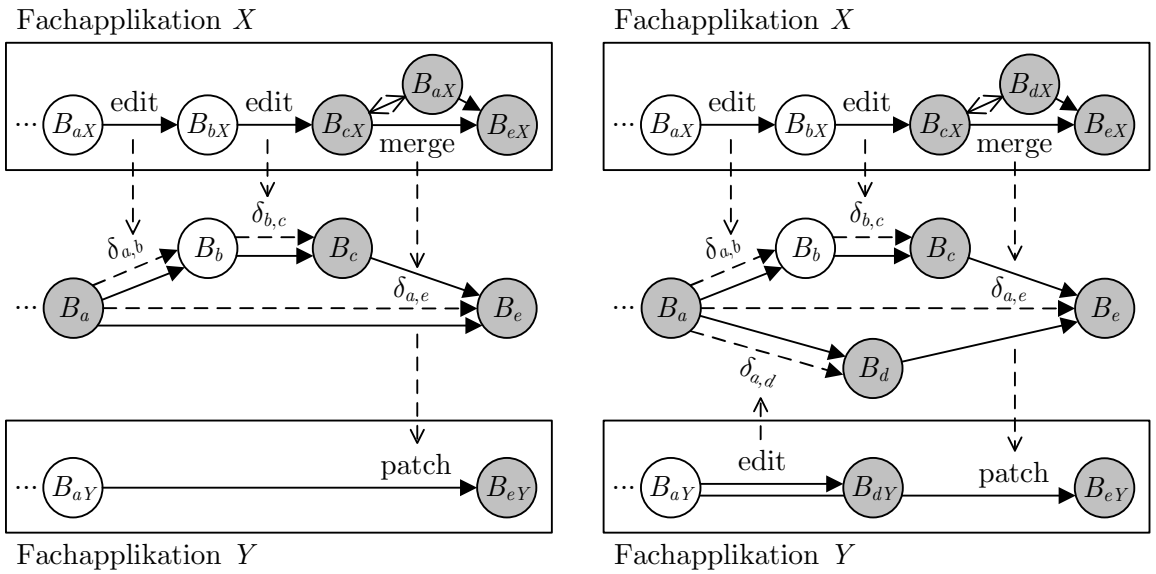
Verteilte Kooperation: Im Bild 5.24 auf der nächsten Seite wird der Workflow beim Zusammenführen von Modellinstanzversionen veranschaulicht. Während im Bild 5.24a die Version B_a und ihre Revision B_c zusammengeführt werden, ist im Bild 5.24b die Zusammenführung der Varianten B_c und B_d dargestellt. Ergebnis beider Zusammenführungen ist jeweils die Version B_e. Während der Zusammenführung wird die native Modellinstanz in der Fachapplikation X verarbeitet und die Änderung $\delta_{a,e}$ auf Basis des standardisierten operativen Modells mittels Journaling aufgezeichnet. Auf Grundlage dieser Änderung kann in der Fachapplikation Y der zusammengeführte Modellinstanzzustand B_{eY} mittels Patching erzeugt werden.

5.7.2 Merge-Algorithmus

Verarbeitung: Der Algorithmus zum Zusammenführen von Modellinstanzversionen ist unabhängig von den unterschiedlichen, nativen Objektmodellen der Fachapplikationen. Er basiert zum einen auf den im Verarbeitungsgraphen[28] gespeicherten Änderungsinformationen und zum anderen auf den vorgestellten Konzepten zum Journaling und Patching[29] von standardisierten Änderungen.

Vergleich: Bevor unterschiedliche Modellinstanzversionen x_l und y_m zusammengeführt werden, müssen sie miteinander verglichen werden. Der Vergleich erfolgt auf Basis des

[28] s. Abschnitt 3.2.4 auf Seite 59
[29] s. Abschnitt 5.4.3 auf Seite 119



(a) Zusammenführung B_a mit Revision B_c (b) Zusammenführung von Varianten B_c und B_d

Bild 5.24: Workflow beim Zusammenführen von Modellinstanzversionen

im Abschnitt 5.6.2 auf Seite 130 vorgestellten Diff-Algorithmus. Ergebnis dieses Verfahrens sind beim Revisionenvergleich eine Sequenz (D_x) und beim Variantenvergleich zwei Sequenzen (D_x, D_y) von Änderungen sowie die Konfliktmenge $K(x_l, y_m)$ im Fall eines Variantenvergleichs.

Algorithmus: Es besteht die Aufgabe, zwei Modellinstanzversionen $x_l, y_m \in V$ auf der Basis der die Unterschiede speichernden Sequenzen von Änderungen D_x und D_y zusammenzuführen. In den Änderungen sind Folgen von alphanumerischen POML-Operationsinstanzen gespeichert. Innerhalb einer Fachapplikation wird der Zusammenführungsprozess wie folgt durchgeführt:

1. Herstellen des Zustands der Modellinstanz entsprechend der Ursprungsversion $\varepsilon \in V$

2. Sequentielle Verarbeitung der Modellinstanz entsprechend der Zusammenführungsintention durch:

   a) Anwenden einer gespeicherten POML-Operationsinstanz aus D_x bzw. D_y (Patching),

   b) Überspringen der gespeicherten POML-Operationsinstanz aus D_x bzw. D_y oder

   c) Erzeugen eines neuen Verarbeitungsschrittes auf der Basis nativer Befehle oder standardisierter Modellieroperationen (OML).

3. Sequentielles Aufzeichnen von standardisierten POML-Operationsinstanzen als Zusammenführungsänderung (Journaling)

Beispiel 5.10: Merge-Algorithmus für Revisionen und Varianten

Revisionen: Die im Verarbeitungsgraphen des Bildes 5.24a auf Seite 140 dargestellte Version B_a wird zunächst mit ihrer Revision B_c verglichen. Das Vergleichsergebnis des vorgestellten Diff-Algorithmus ist die Änderungssequenz $D_x = \langle \delta_{a,b}, \delta_{b,c} \rangle$. Anschließend wird als Grundlage für die Zusammenführung der native Modellinstanzzustand B_{aX} der gemeinsamen Ursprungsversion B_a in der Fachapplikation X hergestellt. Im Prozess der Zusammenführung wird auf dieser Modellinstanzversion operiert, indem je nach Zusammenführungsziel (a) POML-Operationsinstanzen aus D_x angewendet, (b) übersprungen oder (c) neue Operationen instanziiert werden. Während der Zusammenführung zur Version B_e wird die Änderung $\delta_{a,e}$ aufgezeichnet.

Varianten: Die im Verarbeitungsgraphen des Bildes 5.24b auf Seite 140 dargestellten Varianten B_c und B_d werden verglichen. Das Ergebnis des Diff-Algorithmus sind die in den Änderungssequenzen $D_x = \langle \delta_{a,b}, \delta_{b,c} \rangle$ und $D_y = \langle \delta_{a,d} \rangle$ gespeicherten Unterschiede. Als nächstes wird in der Fachapplikation X der Modellinstanzzustand B_{aX} der gemeinsamen Ursprungsversion B_a hergestellt. Anschließend folgt die Zusammenführung, bei der (a) gespeicherte POML-Operationsinstanzen entweder aus D_x oder aus D_y auf die Modellinstanz angewendet, (b) übersprungen oder (c) neue Operationen instanziiert werden. Die beim Vergleich ermittelten Konfliktsituationen zwischen Modellobjektversionen werden vom Fachplaner im Sinne seiner Entwurfsintention manuell behandelt. Analog zur Revision wird die Zusammenführung zur Version B_e als Änderung $\delta_{a,e}$ aufgezeichnet.

Objektkontext: Die Zusammenführung von Modellinstanzversionen kann objektbezogen erfolgen. Ausgehend von den präsentierten Vergleichsergebnissen (vgl. Abschnitt 5.6.2 auf Seite 130) werden geänderte (hinzugefügte, modifizierte oder gelöschte) Modellobjekte selektiert. Es werden die sie verarbeitenden Operationsinstanzen aus den gespeicherten Änderungssequenzen ermittelt und die Operandenlisten so angepasst, dass diese nur selektierte Modellobjekte enthalten. Anschließend können die angepassten Operationsinstanzen sequentiell auf die Ursprungsversion angewendet werden. Der Fachplaner steuert demnach die Zusammenführung auf der Basis der von ihm ausgewählten Modellobjekte.

Beispiel 5.11: Zusammenführung im Objektkontext

Mit Bezug auf das Beispiel 5.7 auf Seite 134 wird die Zusammenführung der Revisionen B_a und B_b zur Version B_c im Objektkontext beschrieben. Die Änderung $\delta_{a,b}$ der Version B_a beschreibt die gemeinsame Manipulation der Ober- und Untergurtobjekte v, w, x, y und z des Fachwerkträgers. Es entsteht die Revision B_b.

$$\delta_{a,b} = \langle \texttt{ modsection ..., [<v>, <w>, <x>, <y>, <z>]; } \rangle$$

Die Zusammenführung basiert auf der Auswahl der als geändert markierten Modellobjekte. Im Bild 5.25 auf der nächsten Seite sind der objektbezogene Navigationsbaum zur Visualisierung des Vergleichsergebnisses und der Fachwerkträger schematisch dargestellt. Die objektbezogene Zusammenführung erfolgt nacheinander für jedes einzelne Modellobjekt. Beispielhaft sollen alle Modifikationen aus der Änderung $\delta_{a,b}$ in die neue Version B_c übernommen werden. Das Bild zeigt, dass das Modellobjekt v bereits zu-

sammengeführt wurde, das Objekt w für eine Zusammenführung selektiert ist und die anderen Objekte noch nicht berücksichtigt wurden.

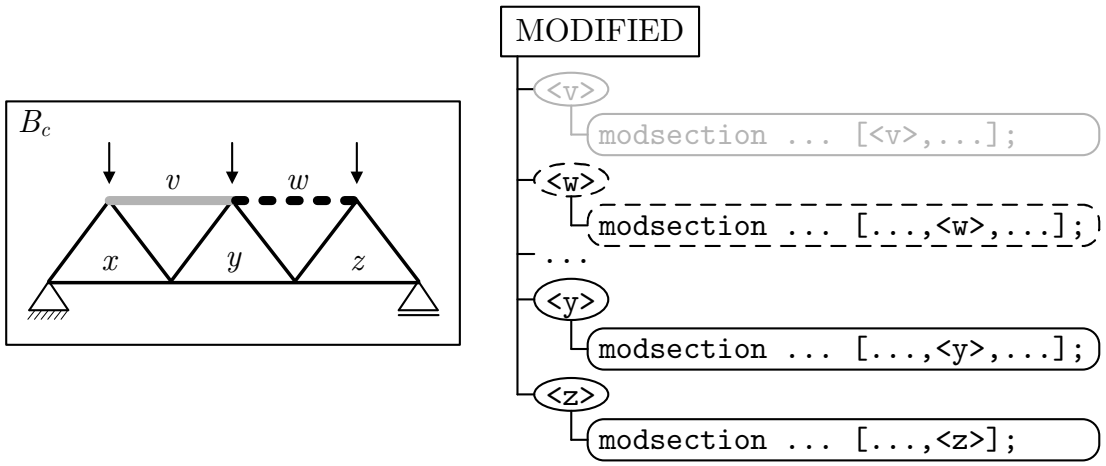


Bild 5.25: Zusammenführung im Objektkontext

Die Zusammenführung im Objektkontext ergibt die Änderung $\delta_{a,c}$ als Sequenz fünf einzelner Operationsinstanzen, die jeweils für ein Modellobjekt der Gurte die Querschnittseigenschaft ändert.

$$\delta_{a,c} = \langle\ \texttt{modsection ... [<v>]; modsection ... [<w>]; modsection ... [<x>]; modsection ... [<y>]; modsection ... [<z>];}\ \rangle$$

Die ursprüngliche gemeinsame Querschnittsmodifikation aller Ober- und Untergurtobjekte wird bei der Zusammenführung im Objektkontext in fünf einzelne Modifikationen aufgelöst. Das erwünschte Zusammenführungsergebnis wird zwar erreicht, die ursprüngliche Entwurfsabsicht bleibt jedoch nicht vollständig erhalten. Im Unterschied zur Zusammenführung von Objektversionen auf Attributbasis (vgl. Abschnitt 2.3.5 auf Seite 33) wird aber die Änderungssemantik in Form der konkreten Operationsinstanz `modsection` für einzelne Modellobjekte berücksichtigt.

Operationskontext: Analog zum Vergleich kann das Zusammenführen von Modellinstanzversionen auch im Operationskontext erfolgen. Die gespeicherten Sequenzen von Operationsinstanzen geben explizit Aufschluss darüber, welche Teilmenge von Modellobjekten mit derselben Operationsinstanz verarbeitet wurde. Im Gegensatz zum herkömmlichen Verfahren werden die Modellobjekte bei der Zusammenführung nicht einzeln verarbeitet, sondern können auf der Basis ihrer gemeinsamen Operationsinstanz in einem Verarbeitungsschritt als Objektmenge manipuliert werden (vgl. Beispiel 5.7 auf Seite 134). Diese Art der Zusammenführung wird somit auf der Basis von Operationsinstanzen gesteuert.

Beispiel 5.12: Zusammenführung im Operationskontext

Mit Bezug auf das vorangegangene Beispiel 5.11 auf Seite 141 wird die Zusammenführung der Revisionen B_a und B_b zur Version B_c im Operationskontext betrachtet. Im

Unterschied zur objektbezogenen Zusammenführung wird die präsentierte Operationsinstanz `modsection..., [<v>, <w>, <x>, <y>, <z>];` ausgewählt. Die in der Operandenliste spezifizierten Modellobjekte werden dabei markiert. Zur Übername aller Modifikationen aus der Änderung $\delta_{a,b}$ wird die ausgewählte Operationsinstanz ausgeführt. Das Bild 5.26 veranschaulicht die ausgewählte Operationsinstanz im operationsbezogenen Navigationsbaum der Vergleichsvisualisierung und die selektierten Modellobjekte.

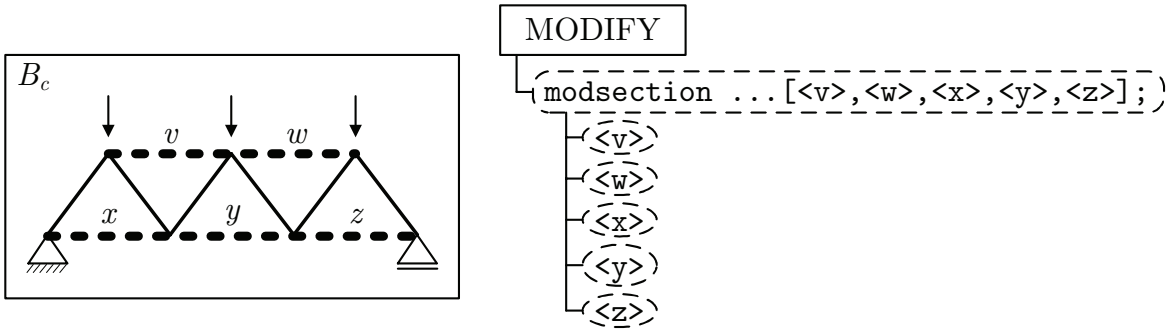


Bild 5.26: Zusammenführung im Operationskontext

Die Zusammenführung im Operationskontext ergibt die Änderung $\delta_{a,c}$ als eine einzige Operationsinstanz, die für alle Gurtobjekte gemeinsam die Querschnittseigenschaften ändert.

$$\delta_{a,c} = \langle \texttt{ modsection ... [<v>, <w>, <x>, <y>, <z>]; } \rangle$$

Die ursprüngliche gemeinsame Querschnittsmodifikation aller Ober- und Untergurtobjekte wird bei der Zusammenführung im Operationskontext berücksichtigt und bleibt vollständig erhalten. Hier kommt der Vorteil einer Zusammenführung von Modellinstanzversionen auf Grundlage gespeicherter Entwurfsabsichten zum Tragen.

Abhängige Änderungen: Das Entwurfsergebnis hängt oft von der Ausführungsreihenfolge der in den Änderungen gespeicherten Operationsinstanzen ab. Ist dies der Fall, handelt es sich um inhaltlich voneinander abhängige Änderungen. Beispielsweise sind die geometrischen Transformationen Translation und Rotation nicht kommutativ, das heißt, die Transformationsreihenfolge spielt eine entscheidende Rolle. Ein weiteres Beispiel stellt das Hinzufügen von Modellobjekten dar, das von der aktuellen Konfiguration des Verarbeitungskontextes[30] abhängt. Die Operationsinstanzen der Gruppe ADD sind demnach abhängig von denen der Gruppe SET. Auch das Löschen von Modellobjekten ist in Bezug auf abhängige Änderungen problematisch. Werden Objekte in einer Änderung gelöscht, ist ein anschließendes Modifizieren entsprechend einer anderen Änderung unmöglich. Ein inkonsistenter Zustand der Modellinstanz kann jedoch bei der vorgeschlagenen operationsbasierten Zusammenführung nicht entstehen. Operationinstanzen sind lediglich nicht ausführbar. Die Zusammenführung voneinander abhängiger Änderungen basiert auf gespeicherten Operationsinstanzen, wird aber vom

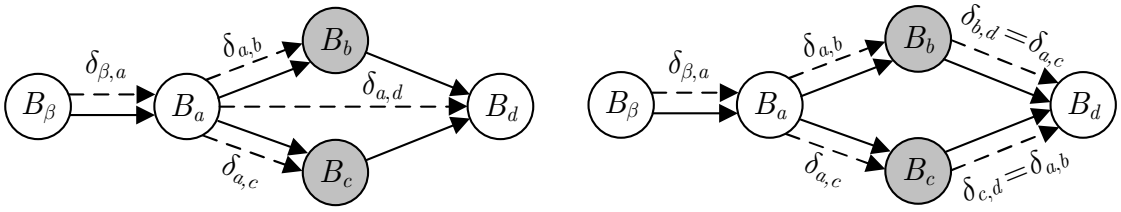
[30] s. Abschnitt 3.4.2 auf Seite 70

Fachplaner entsprechend seiner Entwurfsintention als neue Änderung erzeugt und in einer neuen Änderungsdatei gespeichert.

Unabhängige Änderungen: Führt das Anwenden von Änderungen unabhängig von der Anwendungsreihenfolge zu ein und demselben Entwurfsergebnis, sind diese Änderungen inhaltlich voneinander unabhängig. Wird beispielsweise in einer Änderung das Material eines Modellobjekts geändert und in der anderen Änderung eine geometrische Transformation desselben Objekts durchgeführt, liegt zwar ein potentieller Konflikt vor, die Änderungen sind aber inhaltlich voneinander unabhängig. Bei der Zusammenführung voneinander unabhängiger Änderungen wird keine neue Änderungsdatei erzeugt, sondern die gespeicherten Änderungen werden direkt und unabhängig von der Ausführungsreihenfolge verwendet. Welche Operationen voneinander abhängig und welche voneinander unabhängig sind, kann bei der Spezifikation des operativen Modells formal festgelegt und bei der Zusammenführung angezeigt werden.

Beispiel 5.13: Zusammenführung voneinander abhängiger und unabhängiger Änderungen

Im Bild 5.27 werden die Zusammenführungen von abhängigen und unabhängigen Änderungen veranschaulicht. Aufgrund der inhaltlichen Abhängigkeit zwischen den Änderungen $\delta_{a,b}$ und $\delta_{a,c}$ wird im Bild 5.27a die neue Änderung $\delta_{a,d}$ im Zusammenführungsprozess erzeugt. Im Gegensatz dazu stellt das Bild 5.27b eine Zusammenführung der inhaltlich voneinander unabhängigen Änderungen $\delta_{a,b}$ und $\delta_{a,c}$ dar. Hier kann die Änderung $\delta_{a,c}$ auf die Version B_b oder die Änderung $\delta_{a,b}$ auf die Version B_c anwendet werden, um die Version B_d zu erzeugen.



(a) Voneinander abhängige Änderungen (b) Voneinander unabhängige Änderungen

Bild 5.27: Zusammenführung von Änderungen

Semantik und Konsistenz

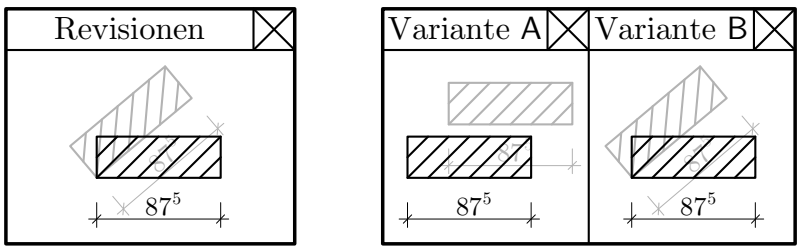
Semantik: Die gespeicherten Änderungen repräsentieren, wie bereits im Abschnitt 5.4.5 auf Seite 122 dargelegt, Entwurfsabsichten der Fachplaner. Durch die Wiederverwendung dieser Änderungen im Zusammenführungsprozess von Versionen werden Planungsintentionen vorteilhaft berücksichtigt. Diese zusätzliche Aussagekraft der Modellieroperationen bleibt somit auch nach dem Prozess der Zusammenführung erhalten.

Konsistenz: Die Verarbeitung der Modellinstanz auf Basis von Operationen sichert, wie im Abschnitt 3.3.3 auf Seite 65 dargelegt, die Modellinstanzkonsistenz. Bei herkömmlichen Verfahren werden Modellobjekte auf Attributbasis verglichen und anschlie-

ßend zusammengeführt, ohne die Konsistenz auf Modellinstanzebene zu gewährleisten[31]. Hier bietet der vorgeschlagene, auf Operationen basierende Merge-Algorithmus einen signifikanten Vorteil.

5.7.3 Visualisierungskonzept

Grafische Fenster und Navigationsstrukturen: Die Zusammenführung von Versionen erfolgt nach dem Versionenvergleich. Das im Abschnitt 5.6.3 auf Seite 135 vorgestellte Visualisierungskonzept für den Vergleich bildet die Grundlage für die Zusammenführung. Bei der Zusammenführung einer Version mit ihrer Revision wird in einem grafischen Fenster der Zustand der Revision in Bezug auf die andere Version, die Ursprungsversion dargestellt (Bild 5.28a). Im Unterschied zum Revisionenvergleich erscheint der Ursprungszustand als Ausgangspunkt der Zusammenführung im Original, der Zustand der Revision wird farblich hervorgehoben. Bei einer Variantenzusammenführung hingegen werden zwei grafische Fenster verwendet, die jeweils den Zustand der Variante in Bezug auf die gemeinsame Ursprungsversion visualisieren (Bild 5.28b). Im Gegensatz zum Variantenvergleich wird der jeweilige Variantenzustand farblich markiert und der Ursprungszustand als Ausgangspunkt normal dargestellt. Der Zusammenführungsvorgang selbst kann in beiden Fenstern vorgenommen werden, um die Möglichkeit zu bieten, sich beispielsweise bei Konstruktionsvorgängen auf die jeweiligen alternativen Endzustände zu beziehen. Der aktuelle Modellinstanzzustand der neuen Version wird während dieses Prozesses in beiden Fenstern dargestellt.



(a) Revisionenzusammenführung

(b) Variantenzusammenführung

Bild 5.28: Grafische Visualisierung der Zusammenführung von Modellinstanzversionen

Navigationsstrukturen: Die zur Vergleichsvisualisierung verwendeten Navigationsstrukturen (objektbezogen und operationsbezogen) präsentieren dem Fachplaner die Unterschiede und die Konflikte der betrachteten Modellinstanzversionen während des Zusammenführungsvorgangs (vgl. Beispiele 5.11 und 5.12 auf Seite 141 bzw. 142). Bereits zusammengeführte Modellobjekte und dafür verwendete Operationsinstanzen werden aus den Navigationsbäumen entsprechend entfernt. So behält der Fachplaner den Überblick über noch zuzammenzuführende Modellobjekte.

[31] vgl. Abschnitt 2.3.5 auf Seite 33

Operationslisten: Als weitere Navigationsstruktur zur visuellen Unterstützung der Zusammenführung werden Operationslisten vorgeschlagen. In einem grafischen Fenster werden die auf Basis der Navigationsbäume selektierten Operationsinstanzen als Listen dargestellt. Bei der Betrachtung von Revisionen präsentiert eine Liste die Operationsinstanzen aus der beim Vergleich ermittelten Änderungssequenz D_x und eine andere Liste stellt die bereits während der Zusammenführung ausgeführten (übernommenen oder neuen) Operationsinstanzen dar. Im Fall einer Variantenzusammenführung wird neben der aktuellen Zusammenführungsliste (Version C) für jede der zwei Varianten (A und B) eine separate Operationsliste entsprechend der ermittelten Unterschiede (D_x und D_y) präsentiert. Das Bild 5.29 verschaulicht schematisch die Operationslisten für eine Variantenzusammenführung. Der Fachplaner kann auf Basis von Schaltknöpfen Operationsinstanzen aus den oberen beiden Listen überspringen (skip), übernehmen (grau hervorgehoben) und auf die Version C anwenden (appr) oder eine neue Anweisung (new), je nach Zusammenführungsintention, ausführen. Diejenigen Operationsinstanzen, die aufgrund von Abhängigkeiten (vgl. Seite 143) generell nicht ausführbar sind, werden gesondert (gestrichelt) hervorgehoben.

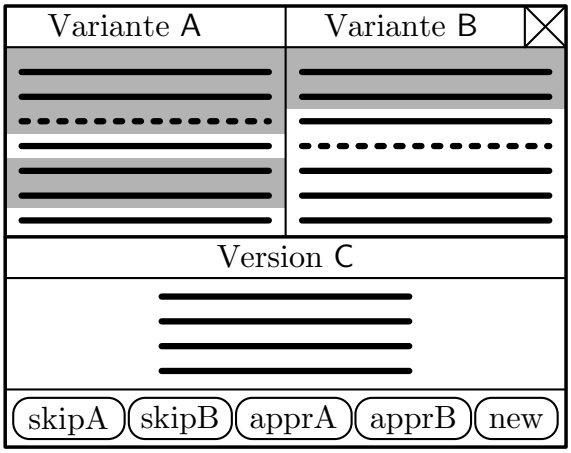


Bild 5.29: Zusammenführung auf Grundlage von Operationslisten

Schritt-für-Schritt-Zusammenführung: Der Fachplaner navigiert, wie auch beim Schritt-für-Schritt-Vergleich, vorwärts und rückwärts durch die Gesamtheit der ihm präsentierten Unterschiede in Form von Änderungssequenzen. Während des „Nachspielens“ entscheidet der Fachplaner, welche Operationsinstanzen ausgeführt, übersprungen oder neu erzeugt werden sollen, um betrachtete Modellinstanzversionen Schritt-für-Schritt zusammenzuführen. Das Zusammenführen auf Grundlage von Operationen erlaubt dem Fachplaner auch das Rückgangigmachen (undo) und das Wiederherstellen (redo) einzelner Schritte, um so verschiedene Möglichkeiten „durchzuspielen“.

5.7.4 Zusammenfassung

Auf der Basis eines standardisierten operativen Modells und der operativen Modellierungssprache der POML-Sprachebene beschreiben Änderungsdateien den Verarbei-

tungsprozess von Modellinstanzen innerhalb einer Fachdomäne. Die im Verarbeitungsgraphen der Modellinstanz gespeicherten Änderungsinformationen bilden die Grundlage für einen neuartigen Merge-Algorithmus für das Zusammenführen von Versionen. Der vorgestellte Diff-Algorithmus liefert zunächst die Unterschiede zwischen einzelnen Planungsständen als separate Folgen von POML-Änderungen. Auf Basis dieser Änderungssequenzen werden die unterschiedlichen Modellinstanzversionen ausgehend von ihrer Ursprungsversion zusammengeführt. Während des Zusammenführungsvorgangs entscheidet der Fachplaner, ob die gespeicherten Operationsinstanzen angewendet oder übersprungen werden oder grundsätzlich neue Verarbeitungsanweisungen die Modellinstanz manipulieren. Alle in diesem Prozess ausgeführten Verarbeitungsschritte werden konsequenterweise als Änderung aufgezeichnet und im Verarbeitungsgraphen hinterlegt. Dieses Vorgehen stellt zum einen die Modellinstanzkonsistenz während des Zusammenführens sicher und berücksichtigt zum anderen ursprüngliche Entwurfsintentionen im Prozess der Zusammenführung. Die dem Fachplaner präsentierten objektbezogenen und operationsbezogenen Ordnungsstrukturen und Operationslisten sowie eine Schritt-für-Schritt-Zusammenführung bilden die visuelle Basis für die Navigation durch die ermittelten Unterschiede bei der Zusammenführung von Modellinstanzversionen.

6 Pilotimplementierung

„Der Beweis aber muß in keinem Hirngespinst, sondern in Tatsachen bestehen. Hierdurch unterscheidet sich unsere jetzige Zeit vorzüglich von den vorigen Jahrhunderten.“

Johann Christian Wiegleb (1732–1800), aus [Wiegleb 1777]

In diesem Kapitel wird eine prototypische Umsetzung der Anwendung des verarbeitungsorientierten Modells und der operativen Modellierungssprache auf Basis des Open-Source-Systems CADEMIA[1] vorgestellt. Aufbauend auf der Darstellung der Systemarchitektur werden die Erweiterungsschnittstellen des Systems beschrieben. Anschließend werden die sprachbasierte und die objektorientierte Umsetzung eines operativen Zeichnungsmodells gezeigt. Zur Verifikation des vorgeschlagenen Modells und der operativen Modellierungssprache werden die Anwendung des operativen Zeichnungsmodells in der Benutzer- und Programmierschnittstelle sowie beim Vergleich und bei der Zusammenführung von Zeichnungsversionen vorgestellt.

6.1 CADEMIA

CADEMIA stellt eine Open-Source[2]-Entwicklungsplattform für geometrisch orientierte Fachapplikationen im Bauwesen dar. Sie wurde in der objektorientierten Programmiersprache Java[3] an der Bauhaus-Universität Weimar entwickelt und ist unter der GPL[4]-Lizenz frei verfügbar.

Systemarchitektur: Das Grundsystem von CADEMIA setzt sich aus den Komponenten Ein-/Ausgabe, Befehl, View und Modell zusammen (s. Bild 6.1 auf der nächsten Seite). Die alphanumerische oder grafische Anwendereingabe wird interpretiert und in native Befehle umgesetzt. Die Befehle verarbeiten eine Instanz des Modells. Die Ausgabe präsentiert dem Fachplaner auf der Basis von Views grafisch den Zustand der Modellinstanz.

[1] s. [Firmenich u. a. 2008], [Firmenich 2006] und `http://www.cademia.org`, Version 1.4.1. CAPE TOWN

[2] deutsch: quelloffen, s. Open Source Initiative: `http://www.opensource.org`

[3] s. [Flanagan 2003], [Horstmann u. Cornell 2003, 2002], [Hardy 2000] und `http://java.sun.com`

[4] General Public License, s. `http://www.gnu.org/copyleft/gpl.html`

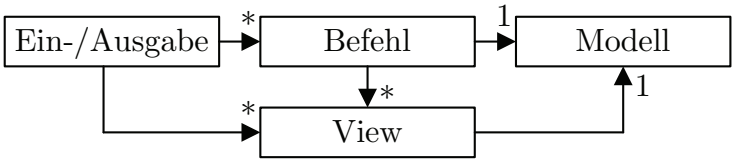


Bild 6.1: Systemarchitektur von CADEMIA

Im Folgenden werden die einzelnen Komponenten des CADEMIA-Grundsystems beschrieben. Dabei wird auf die Erweiterungsmöglichkeiten zur Umsetzung des sprachbasierten und des objektorientierten operativen Modells eingegangen. Neue Komponenten bzw. Klassen sind grau dargestellt.

6.1.1 Ein-/Ausgabe

Eingabe: Die Basis für die Beschreibung von Befehlen stellt eine einfache, native Eingabesprache dar. Sämtliche Eingabeinteraktion (TextInput, MouseInput) in der Benutzerschnittstelle (UserInterface) wird durch alphanumerischen Text repräsentiert und von nativen Interpretern (ArgTokenizer, NumInterpreter) ausgewertet. Zur Integration des sprachbasierten operativen Modells implementiert ein eigener Interpreter[5] (MyInterpreter), wie im Bild 6.2 dargestellt, die vorhandenen Schnittstellen.

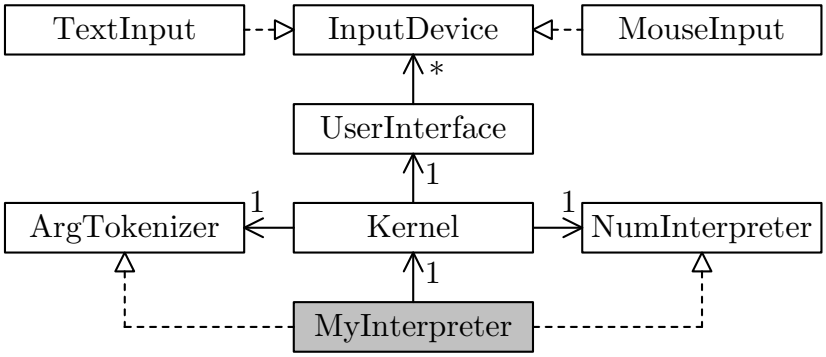


Bild 6.2: Integration des sprachbasierten operativen Modells durch einen Interpreter

6.1.2 Verarbeitung

Befehle: Die Verarbeitung der Modellinstanz basiert auf nativen Befehlsobjekten. Ein Befehlsmanager (CmdMgr) verwaltet die Befehlshistorie und sorgt für das Undo-/Redo-Handling. Das Bild 6.3 veranschaulicht die Integration des objektorientierten operativen Modells durch die Erweiterung des nativen Befehlsmanagers sowie die Implementierung der nativen Schnittstelle für Befehlsobjekte (Cmd) und die Erweiterung nativer Befehle (NativeCmd) zur Umsetzung von Operationsklassen (MyOpImpl, MyOpExt).

[5] OPL-/ OML- und POML-Interpreter, s. Abschnitte 5.2.1 S. 109, 5.3.1 S. 113 bzw. 5.4.3 S. 119

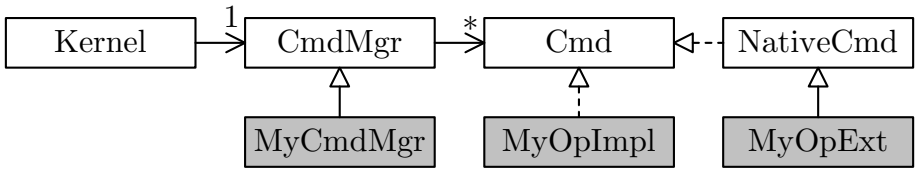


Bild 6.3: Integration des objektorientierten operativen Modells durch Operationsklassen

6.1.3 Bauwerksmodell

Zeichnungsmodell: Gegenstand der Betrachtung ist ein in CADEMIA umgesetztes Zeichnungsmodell. Das Zeichnungsmodell ist ein natives objektorientiertes Modell zur Beschreibung des fachlichen Modells einer technischen Zeichnung nach DIN 1356-1[6] mit geometrischen Komponenten wie Linien, Polylinien, Kreisen, Bögen, Bemaßung usw., sowie Layern und bauspezifischen Attributen wie Linienarten und Linienbreiten. Zur Implementierung eines operativen Zeichnungsmodells muss das native Modell (Database) nicht verändert werden. Das Bild 6.4 zeigt die Umsetzung der Modellbeobachtung (MyListener) für den Journaling-Mechanismus[7]. Der POIDManager verwaltet persistente Objektidentifikatoren und verschiedene Objektversionen bzw. -zustände für den Vergleich und das Zusammenführen von Zeichnungsversionen. Dazu verwendet er den nativen Namensraum (NameSpace).

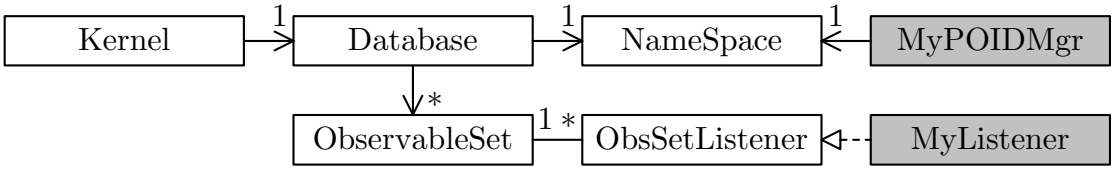


Bild 6.4: Integration der Modellbeobachtung und der persistenten Objektidentifikation

6.1.4 Visualisierung

View: Die grafische Visualisierung der Modellinstanz in Fenstern (GeometryPanel) basiert auf nativen ViewControllern und Views. Diese werden zur Umsetzung der Visualisierungskonzepte für den Vergleich[8] und die Zusammenführung[9] von Zeichnungsversionen, wie im Bild 6.5 auf der nächsten Seite dargestellt, erweitert (MyViewCtrl).

[6] s. [DIN 1356-1 1995]
[7] s. 5.4.3 auf Seite 119
[8] s. 5.6.3 auf Seite 135
[9] s. 5.7.3 auf Seite 145

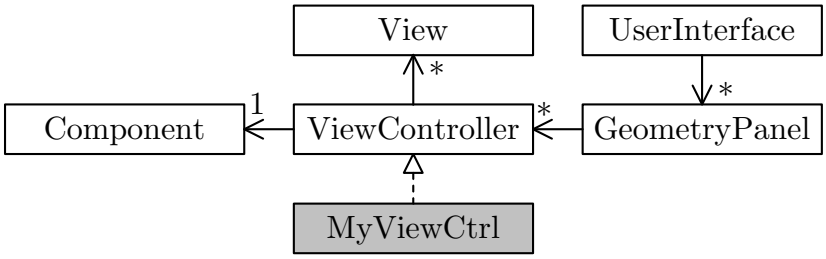


Bild 6.5: Integration der Visualisierung von Vergleichsergebnissen

6.2 Operatives Zeichnungsmodell

Das operative Zeichnungsmodell beschreibt die Verarbeitung einer Zeichnung durch eine Menge von standardisierten Zeichnungsoperationen. Aufbauend auf der Definition dieser Zeichnungsoperationen werden die sprachbasierte und die objektorientierte Umsetzung des operativen Zeichnungsmodells in CADEMIA verdeutlicht.

6.2.1 Zeichnungsoperationen

Sprachbasierte Definition: Das operative Zeichnungsmodell wird durch OML- und POML-Operationen sprachbasiert definiert. In den Anhängen A.2 auf Seite 196 und A.3 auf Seite 203 sind auszugsweise die OML- bzw. die POML-Grammatiken zur Definition von Operationen zur Verarbeitung einer Zeichnung dargestellt. Im Folgenden wird für jede Operationsgruppe stellvertretend eine Zeichnungs- bzw. Modellieroperation beschrieben.

Hinzufügen: Zeichnungsoperationen der Gruppe ADD fügen der Zeichnung geometrische Komponenten, Layer und Attribute hinzu. Zur persistenten Beschreibung erhalten Komponenten und Layer persistente Objektidentifikatoren (POIDs).

Beispiel 6.1: Zeichnungsoperation `AddLine`

Für das Erzeugen und Hinzufügen einer Linie werden die Koordinaten des Start- und des Endpunktes `P1=(x1,y1)` bzw. `P2=(x2,y2)`) angegeben. In der Definition der entsprechenden persistenten Modellieroperation wird zusätzlich die POID der Linie als Operand `p` festgelegt.

```
// OML
AddLine ::= "addline" x1=Number <ARGSEP> y1=Number
            <ARGSEP> x2=Number <ARGSEP> y2=Number
// POML
AddLine ::= "addline" x1=Number <ARGSEP> y1=Number <ARGSEP>
            x2=Number <ARGSEP> y2=Number <ARGSEP> p=<POID>
```

Selektieren/ Deselektieren: Zeichnungsoperationen der Gruppe SELECT/UNSELECT werden in der Benutzer- und Programmierschnittstelle der Zeichnungsapplikation verwendet, nicht aber zur Speicherung von persistenten Änderungen eingesetzt.

Die Selektion bzw. Deselektion von geometrischen Komponenten kann beispielsweise grafisch durch Objektidentifikation mit der Maus erfolgen.

Beispiel 6.2: Zeichnungsoperation `Select`

Für das Selektieren von Zeichnungskomponenten muss lediglich der Operationsname angegeben werden. Der Objektidentifikator p als POID ist optional.

```
// OML
Select ::= "select" ( p=<POID> )?
```

Modifizieren: Zeichnungsoperationen der Gruppe MODIFY dienen der Modifikation geometrischer Eigenschaften von Komponenten und fachlicher Eigenschaften von Komponenten, Layern und Attributen. Während in der Benutzer- und Programmierschnittstelle die aktuell selektierten oder spezifizierten Komponenten bzw. Komponententeile oder auszuwählende Layer bzw. Attribute modifiziert werden, wird bei der Definition von persistenten Modifikationsoperationen eine POID-Liste als Operandenliste angegeben.

Beispiel 6.3: Zeichnungsoperation `ModRotate`

Für die Rotationstransformation geometrischer Komponenten werden die Koordinaten des Ankerpunktes `P=(x,y)` und der Rotationswinkel `w` spezifiziert. Optional ist die Operandenliste `p`. In der Definition der entsprechenden persistenten Modellieroperation muss die Operandenliste `p` angegeben werden.

```
// OML
ModRotate ::= "modrotate" x=Number <ARGSEP> y=Number <ARGSEP>
              w=Number ( p=PoidList )?
// POML
ModRotate ::= "modrotate" x=Number <ARGSEP> y=Number
              <ARGSEP> w=Number <ARGSEP> p=PoidList
```

Löschen: Zeichnungsoperationen der Gruppe REMOVE löschen geometrische Komponenten, Layer oder Attribute. OML-Operationen beschreiben das Löschen von aktuell selektierten oder spezifizierter Komponenten und auszuwählenden Layern und Attributen. Bei der persistenten Definition wird eine POID-Liste als Operandenliste angegeben.

Beispiel 6.4: Zeichnungsoperation `Remove`

Das Entfernen von Zeichnungskomponenten basiert in der Benutzer- und Programmierschnittstelle auf der aktuellen Selektionsmenge oder auf der optional angegebenen Operandenliste p. In der Definition der entsprechenden persistenten Modellieroperation muss die Operandenliste `p` spezifiziert werden.

```
// OML                                  // POML
Remove ::= "remove" (p=PoidList)?       Remove ::= "remove" p=PoidList
```

Konfigurieren: Zeichnungsoperationen der Gruppe SET konfigurieren den Verarbeitungskontext. Dazu zählen beispielsweise die Einstellung für den aktuellen Layer, die aktuellen Attribute, den Zeichnungsmaßstab, das verwendete Nutzerkoordinatensystem und die natürlichen Zeichnungseinheiten.

Beispiel 6.5: Zeichnungsoperation `SetDefLayer`

Das Einstellen des aktuellen Layers basiert in der Benutzer- und Programmierschnittstelle auf der Auswahl von vorhandenen Layern. In der Definition der entsprechenden persistenten Modellieroperation muss die POID des Layers als Operand `p` angegeben werden.

```
// OML
SetDefLayer ::= "setdeflayer" ( p=<POID> )?

// POML
SetDefLayer ::= "setdeflayer" p=<POID>
```

Abfragen: Zeichnungsoperationen der Gruppe GET ändern die Zeichnung nicht, sondern dienen der Abfrage von Eigenschaften der Zeichnung, von selektierten Komponenten, von auszuwählenden Layern oder Attributen sowie zur Eingabeaufforderung von Argumenten bei Modellierprogrammen.

Beispiel 6.6: Zeichnungsoperation `GetSelected`

Für die Abfrage der aktuell selektierten Zeichnungskomponenten wird lediglich der Operationsname angegeben. Diese Abfrage liefert eine Liste von POIDs.

```
// OML
GetSelected ::= "getselected"
```

6.2.2 Sprachbasierte Umsetzung

JavaCC: Für die Programmiersprache Java ist der Parsergenerator JavaCC[10] frei verfügbar. Wie im Bild 6.6 veranschaulicht, wird mittels JavaCC auf der Basis einer BNF[11]-Grammatik und in Java-Methoden definierten Aktionen (`*.jj`) Quellcode für Java-Klassen generiert und anschließend zum lauffähigen Parser bzw. Interpreter compiliert (`*.class`).

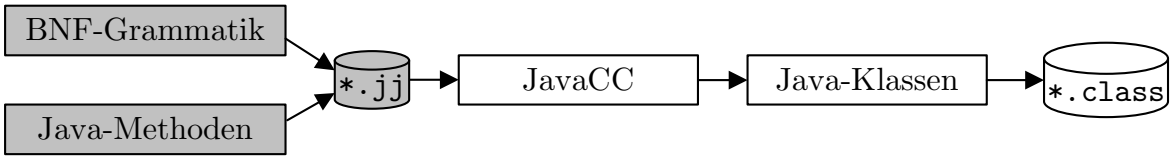


Bild 6.6: Funktionsweise des Java Compiler Compilers (JavaCC)

[10] Java Compiler Compiler: s. [JavaCC]
[11] s. Abschnitt 4.1.2 auf Seite 78

OPL-Interpreter: Mit JavaCC und der im Abschnitt 4.3 auf Seite 86 beschriebenen Grammatik für OPL-Modellierprogramme wird ein OPLInterpreter[12] umgesetzt und wie im Bild 6.7 dargestellt in CADEMIA integriert. Die Klasse Scope verwaltet auf Basis einer Symboltabelle (SymbolTable) die Gültigkeitsbereiche[13] sowie die Bezeichner und Werte für Variablen und Prozeduren. Der Befehl SetOPLInterpreter ersetzt den nativen Interpreter durch den OPL-Interpreter und leitet die Texteingabe (TextInput) zu diesem um.

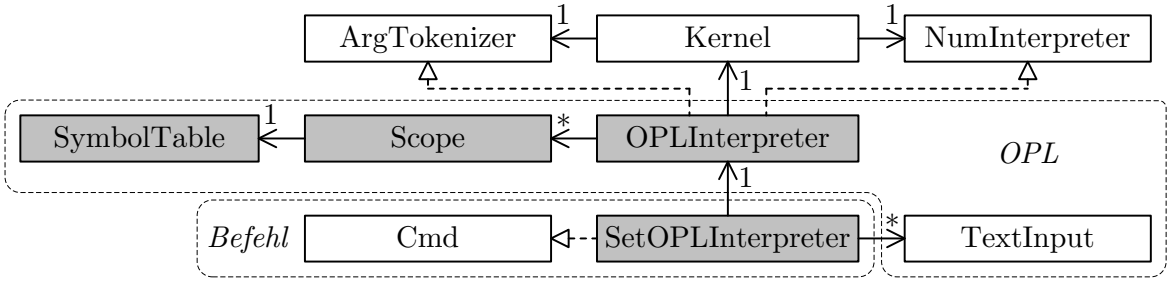


Bild 6.7: Integration des OPL-Interpreters

Das Listing 6.1 zeigt einen Auszug aus der JavaCC-Datei `OPLInterpreter.jj` zur Umsetzung des OPL-Interpreters. Beispielsweise sind die Terminalsymbole `<EOS>`, `<COMMENT>`, `<NULL>` und `<IF>` ab Zeile 14, das Startsymbol `Start` in Zeile 21 und die Nichtterminalsymbole `StmtList`, `Statement`, `Command`, `ModelOperation` aus der OPL-Grammatik[14] in den Zeilen 25, 30, 37 bzw. 44 definiert.

```
1  options { ... }
   PARSER_BEGIN(OPLInterpreter)
3  ...
   public class OPLInterpreter implements NumberInterpreter,
      ArgumentTokenizer {
5    Scope m_rootScope = new Scope(null);
     Scope m_curScope = m_rootScope;
7    ...
   }
9  PARSER_END(OPLInterpreter)
   ...
11 // Terminalsymbole (Anweisungsende, Kommentar,
      Schlüsselwörter, Bezeichner,
   // Zahlen, Zeichenketten, POIDs, Operatoren usw.)
13 TOKEN : {
      < EOS : (";")+ >
15  | < COMMENT : "//" (~["\n","\r"])* ("\n"|"\r"|"\r\n")>
    | < NULL: "null">
17  | < IF: "if" >
```

[12] vgl. Abschnitt 5.3.1 auf Seite 113
[13] engl.: scope, s. [Aho u. a. 1988, S. 429ff]
[14] s. Abschnitt 4.3 auf Seite 86

```
...
}
// Startsymbol
Object Start(List ret) : { Object value = null; } {
  value = StmtList(ret) { return value; }
}
// Weitere Nichtterminalsymbole
Object StmtList(List ret) : { Object value = null; } {
  ( value = Statement(ret) <EOS> )* {
    return value;
  }
}
Object Statement(List ret) : { Object value = null; } {
  // Zuweisung
  Assignment() { return null;   }
  |
  // Kommando
  value = Command(ret) { return value; }
}
Object Command(List ret) : { Object value = null; } {
  // Systemkommando
  value = SysCommand(ret) { return value;       }
  |
  // Modellieroperation
  ModelOperation(ret) { ... }
}
void ModelOperation(List ret) : {
  Token opName;
  List args = new ArrayList();
}
{
  opName = <ID> (ArgList(args))? {
    // Name 'opName' und Argumente 'args' weitergeben
    kernel.doCmd(opName.image, args); ...
  }
}
...
```

Listing 6.1: Umsetzung des OPL-Interpreters mit JavaCC (Auszug)

Eingabe: Aufgrund der Tatsache, dass OML-Modellieroperationen auch der OPL-Syntax genügen[15], kann der OPL-Interpreter sowohl OPL- als auch OML-Eingabeströme interpretieren. Ist der OPL-Interpreter in CADEMIA gesetzt, können OPL-Zeichnungsprogramme und OML-Zeichnungsoperationen in der Eingabe instanziiert werden. Der OPL-Interpreter wertet Variablen, Prozeduren und Kontrollstrukturen in Folgen von Zeichnungsoperationsinstanzen (`ModelOperation`) aus und gibt die Operations-

[15] vgl. Gleichung 4.11 auf Seite 84

namen sowie Operationsargumente zur Instanziierung von Operationsklassen an den Kernel weiter.

Patching[16]**:** Bevor die gespeicherten Änderungen auf die aktuelle Zeichnungsversion angewendet werden, erfolgt im Befehl POMLPatch eine Schemaüberprüfung hinsichtlich der definierten persistenten POML-Zeichnungsoperationen. Die Klasse POMLValidator wird mit JavaCC auf Basis der POML-Grammatik für das operative Zeichnungsmodell entwickelt und validiert die in den Änderungsdateien enthaltenen Sequenzen von POML-Operationsinstanzen in Bezug auf Namen, Anzahl und Typen der Argumente. Darüber hinaus wird im POMLValidator eine Plausibilitätskontrolle durchgeführt, die prüft, ob zu modifizierende oder zu löschende Zeichnungskomponenten überhaupt vorhanden sind und ob die Eindeutigkeit von POIDs gewährleistet ist (MyPOIDMgr). Neben der Funktion eines OML-Interpreters übernimmt der OPLInterpreter auch die Aufgabe des POML-Interpreters, da POML-Operationsinstanzen in Änderungen auch der OPL-Syntax entsprechen[17]. In dem Befehl POMLPatch wird der POML-Eingabestrom aus der zu spezifizierenden Änderungsdatei (POMLFile) zum OPL-Interpreter umgeleitet. Die Verwendung des OPL-Interpreters für das Patching verdeutlicht das Bild 6.8.

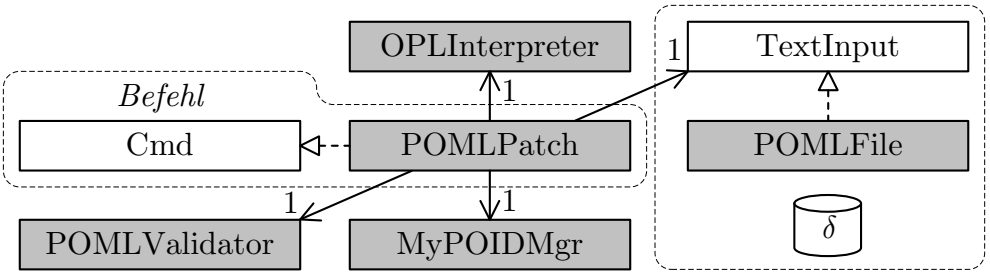


Bild 6.8: Umsetzung und Integration des Patchings

6.2.3 Objektorientierte Umsetzung

Operationsklassen: Nach Interpretation und Auswertung der Operationsnamen und -argumente werden zur Verarbeitung der Zeichnung Operationsklassen instanziiert. Operationsklassen erweitern dazu die native Befehlsschnittstelle Cmd und fragen zur Ausführung Befehlsargumente entsprechend der Definition der OML- bzw. POML-Zeichnungsoperationen ab. Das Listing 6.2 zeigt die für die Umsetzung von Operationsklassen definierte Schnittstelle OMLOperation.

```
public interface OMLOperation extends Cmd {
  // POML-Beschreibung der in diesem Befehl
  // durchgeführten Zeichnungsänderung
  public String getPOMLString();
```

[16] vgl. Abschnitt 5.4.3 auf Seite 119
[17] vgl. Gleichung 4.11 auf Seite 84

```
6   // OPL-Beschreibung des Befehlsergebnisses
    public String getOPLResult();
8 }
```

Listing 6.2: Schnittstelle für Operationsklassen

Eine Operation liefert nach Ausführung des Befehls die Zeichnungsänderung in Form einer sprachbasierten POML-Operationsinstanz als Zeichenkette (getPOMLString()). Die Methode getOPLResult() wird definiert, um beispielsweise bei der Umsetzung von Operationsklassen der Gruppe GET und der Gruppe ADD das Abfrageergebnis bzw. die POID des hinzugefügten Modellobjekts als Zeichenkette gemäß OPL-Syntax zurückzugeben.

Zeichnungsoperationen: Zeichnungsoperationen implementieren die definierte Schnittstelle OMLOperation für Operationsklassen. Ist für eine bestimmte Zeichnungsoperation ein entsprechender nativer Befehl verfügbar, dann wird dieser gekapselt[18] (Bild 6.9 links). Die Umsetzung und Integration der Operationsklasse ModRotate ist beispielhaft im Bild 6.9 rechts dargestellt.

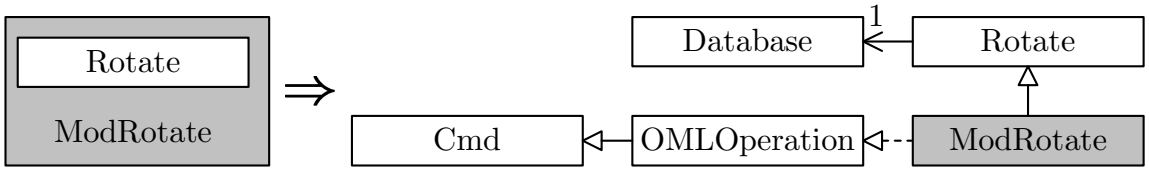


Bild 6.9: Integration der Operationsklasse `ModRotate`

Das folgende Listing 6.3 zeigt auszugsweise die Umsetzung der Operationsklasse `ModRotate`. Diese erweitert den nativen Befehl `Rotate` und implementiert die Schnittstelle `OMLOperation` (Zeile 1). Die Methode `doCmd` (Zeile 7) zur einmaligen Ausführung einer konkreten Operationsinstanz wird überschrieben. Im Fall der nativen Eingabe führt der native Befehl die Rotationsoperation durch (Zeile 11), die Operationsargumente werden ermittelt und vorgehalten. Beim Patching werden die Operationsargumente der POML-Grammatik entsprechend abgefragt (ab Zeile 17) und für die native Argumentenverarbeitung vorbereitet. Nach dem Anpassen der nativen Selektionsmenge in Bezug auf die POML-Operandenliste wird der native Befehl zur Rotationsausführung aufgerufen (Zeile 23). Anschließend wird die Selektionsmenge auf ihren ursprünglichen Zustand zurückgesetzt. Erfolgt die Anwenderinteraktion auf der Basis von OML-Anweisungen, dann wird nach der Abfrage und Aufbereitung der OML-Operationsargumente (ab Zeile 28) die native Rotationsfunktionalität herangezogen (Zeile 36). Unabhängig von der Art der Eingabe wird in der Methode `getPOMLString` (Zeile 31) die Änderung der Zeichnung als POML-Operationsinstanz für das Journaling vorgehalten. Die Methode `getOPLResult` (Zeile 46) liefert `"true"` oder `"false"`, je nachdem ob die Ausführung der Operation erfolgreich war oder abgebrochen wurde.

[18] vgl. Bild 5.5 auf Seite 112

```
class ModRotate extends Rotate implements OMLOperation {
  private double m_x, m_y, m_w;
  private String m_list;
  private boolean m_valid;
  ...
  // Befehl einmalig ausführen
  public void doCmd(Object context) throws ... {
    ...
    if (NativeMode) { // Nativer Eingabestrom
      // Native Superklasse für Rotation rufen
      super.doCmd(context);
      // Rotation (x,y,w) und Operanden ermitteln
      // und vorhalten
      ...
    }
    else if (PatchMode) { // POML-Eingabestrom
      m_x = krnl.readDouble("Enter x");
      m_y = krnl.readDouble("Enter y");
      m_w = krnl.readDouble("Enter angle");
      m_list = readPoidList("Enter operand list");
      // Argumente aufbereiten, Selektion anpassen
      ...
      super.doCmd(context);
      // Selektion zurücksetzen
      ...
    }
    else { // OML-Eingabestrom
      m_x = krnl.readDouble("Enter x");
      m_y = krnl.readDouble("Enter y");
      m_w = krnl.readDouble("Enter angle");
      if (knrl.hasMoreArguments()) { // Operandenliste
                                     //vorhanden?
         m_list = readPoidList(); // Operandenliste lesen
      }
      // Argumente aufbereiten, Operanden vorhalten
      ...
      super.doCmd(context);
    }
  }
  ...
  // POML-Operationsinstanz
  public String getPOMLString() {
    return "modrotate" + m_x + "," + m_y + "," + m_w
                       + "," + m_list + ";";
  }
  // OPL-Ergebnis: Befehl erfolgreich?
  public String getOPLResult() {
```

```
48      if (m_valid) return "true";
        else return "false";
50    }
  }
```

Listing 6.3: Umsetzung der Operationsklasse `ModRotate` (Auszug)

Existiert kein nativer Befehl für eine standardisierte Zeichnungsoperation, dann wird ein neuer Befehl als Operationsklasse umgesetzt und, wie am Beispiel der Zeichnungsoperation ModTransform im Bild 6.10 dargestellt, in CADEMIA integriert. Die Verarbeitungsfunktionalität muss folglich vollständig neu entwickelt und direkt auf dem Zeichnungsmodell (Database) implementiert werden.

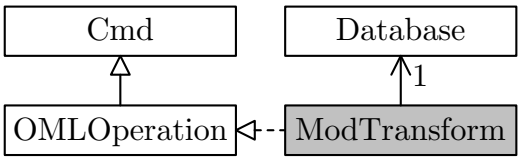


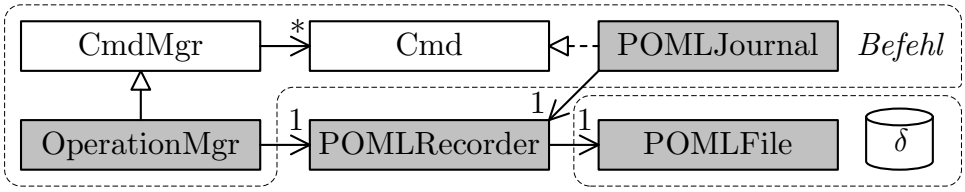
Bild 6.10: Integration der Operationsklasse `ModTransform`

Verarbeitung: Die generische Instanziierung der Operationsklassen für Zeichnungsoperationen basiert auf registrierten Operationsnamen. Der OPL-Interpreter liefert den Operationsnamen und der Kernel erzeugt entsprechende Operationsobjekte, die durch Aufruf der Methode doCmd(Object context) die Zeichnung verarbeiten. Zur Umsetzung der Zeichnungsoperationen der Gruppe GET und für den anschließend beschriebenen Journaling-Mechanismus wird der Operationsmanager (OperationMgr) eingeführt. Dieser erweitert den nativen Befehlsmanager (CmdMgr, s. Bild 6.11 und Listing 6.4). Nach Ausführung einer Zeichnungsoperation wird beispielsweise das Ergebnis einer Abfrageoperation oder die POID einer hinzugefügten Zeichnungskomponente durch den Operationsmanager vorgehalten und vom OPL-Interpreter zur Weiterverarbeitung abgerufen[19].

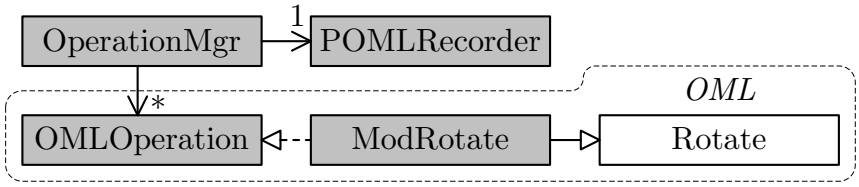
Journaling[20]: Zum Aufzeichnen von Änderungen wird der Befehl POMLJournal umgesetzt. Der Operationsmanager (OperationMgr) verwaltet auf Grundlage nativer Funktionalität (CmdMgr) die Befehlshistorie und verwendet den POML-Rekorder (POMLRecorder) zum Speichern der Änderungen in der Änderungsdatei (POMLFile) (s. Bild 6.11a). Für das Journaling von Zeichnungsoperationen der Gruppe MODIFY werden die nativen Befehle (Rotate) von Operationsklassen (OMLOperation, ModRotate) überschrieben (s. Bild 6.11b). Das Bild 6.11c verdeutlicht die Umsetzung der Beobachtung der nativen Modellinstanz zum Aufzeichnen von Operationen der Gruppen ADD, REMOVE und SET. Dazu meldet der POML-Rekorder (POMLRecorder) mehrere Beobachter an der Datenbasis (Database) an und wird so über Zustandsänderungen der Zeichnung informiert.

[19] beispielsweise, wenn die POID einer hinzugefügten Linienkomponente zurückgegeben und in einer Variablen gespeichert werden soll: `a = addline 1, 2, 3, 4;`

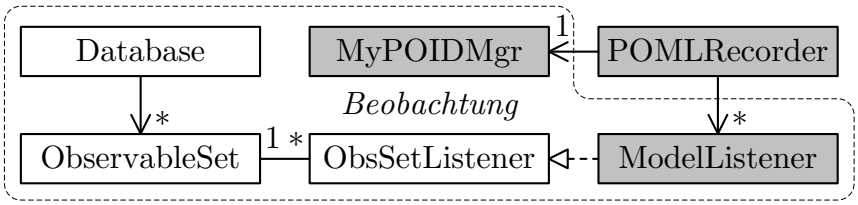
[20] vgl. Abschnitt 5.4.3 auf Seite 119



(a) Journaling-Befehl, Operationsmanager und Änderungsdatei



(b) Verwendung von Operationsklassen (OMLOperation)



(c) Beobachtung der nativen Modellinstanz (Database)

Bild 6.11: Umsetzung und Integration des Journalings

Das folgende Listing 6.4 zeigt auszugsweise die Umsetzung des Operationsmanagers `OperationMgr`. Dieser erweitert den nativen Befehlsmanager `CmdMgr` (Zeile 1) und verwendet den POML-Rekorder `m_rec` (Zeile 2). Ein Befehl wird in der Methode `doCmd` (Zeile 5) ausgeführt. Ist das Journaling aktiviert (Zeile 7), dann werden im Fall einer instanziierten Operationsklasse (Zeile 8) die Modellbeobachtung angehalten (Zeile 9), native Funktionalität zum Ausführen der Operation gerufen (Zeile 11), die Änderung vom POML-Rekorder aufgezeichnet (Zeile 12), anschließend die Modellbeobachtung fortgesetzt (Zeile 14) und das Operationsergebnis vorhalten (Zeile 15). Andernfalls wird die Funktionalität des nativen Befehlsmanager verwendet (Zeile 18, Zeile 21). Auf das gespeicherte Operationsergebnis `m_result` kann mittels der Methode `getOPLResult` (Zeile 28) zugegriffen werden.

```
1 public class OperationMgr extends CmdMgr {
    private POMLRecorder m_rec;
3   private String m_result = null;
    ...
5   public int doCmd(Cmd cmd, Object context) {
      ...
7     if (JournalMode) { // POML-Rekorder eingeschaltet
        if (cmd instanceof OMLOperation) { // Operationsklasse
9         m_rec.suspendListening(); // Modellbeobachtung
                                    // anhalten
```

```
11          // Operation ausführen und Änderung aufzeichnen
            super.doCmd(cmd, context);
13          m_rec.record(((OMLOperation)cmd).getPOMLString());
            ...
15          m_rec.resumeListening(); // Modellbeobachtung
                                     // fortsetzen
17          m_result = ((OMLOperation)cmd).getOPLResult();
          }
19        // Nativer Befehl: Ausführung durch Superklasse
          else { super.doCmd(cmd, context); }
21      }
        else {
23        super.doCmd(cmd, context); // Native Ausführung
          // Vorhalten des Operationsergebnisses
25        if (cmd instanceof OMLOperation) {
            m_result = ((OMLOperation)cmd).getOPLResult();
27      } }
      } ...
29    // Rückgabe des Operationsergebnisses
      public String getOPLResult() { return m_result; }
31  }
```

Listing 6.4: Umsetzung des Operationsmanagers `OperationMgr` (Auszug)

6.3 Zeichnungsversionen

Der Vergleich und die Zusammenführung von Zeichnungsversionen macht die Verwaltung und Visualisierung von Zeichnungskomponenten in mehreren Zuständen — den Objektversionen — notwendig. Fachapplikationen wie CADEMIA verarbeiten aber in der Regel unversionierte Modellinstanzen und müssen deshalb erweitert werden. Im Folgenden werden die Umsetzung und die Integration von Funktionalität zum Verwalten und Visualisieren mehrerer Versionen von Zeichnungskomponenten in CADEMIA beschrieben.

6.3.1 Verwaltung

Komponentenversionen: Jeder Zustand einer Zeichnungskomponente entspricht einem Objekt vom Typ Component. Die jeweils aktuell betrachtete bzw. zu verarbeitende Komponentenversion wird in die Komponentenmenge (ComponentSet) der Datenbasis eingetragen. Die Verwaltung der Komponentenversionen übernimmt ein Diff-Merge-Manager (DiffMergeMgr) auf Basis der persistenten Objektidentifikatoren (MyPOIDMgr). Diesen Sachverhalt veranschaulicht das Bild 6.12.

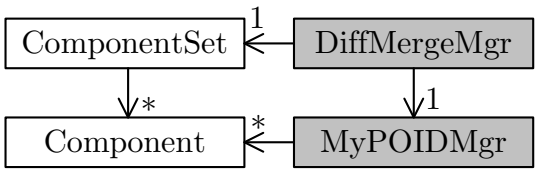


Bild 6.12: Verwaltung der Komponentenversionen

Vergleich und Zusammenführung: Die Befehle (DiffCmd, MergeCmd) starten den Vergleich bzw. das Zusammenführen von Zeichnungsversionen. Der Diff-Merge-Manager (DiffMergeMgr) implementiert den Diff- und den MergeAlgorithmus[21] auf der Basis der gespeicherten Änderungen (POMLFile) (s. Bild 6.13). Die berechneten Unterschiede werden in entsprechenden Objektmengen[22] (DiffMergeList) für Zeichnungskomponenten verwaltet und vorgehalten.

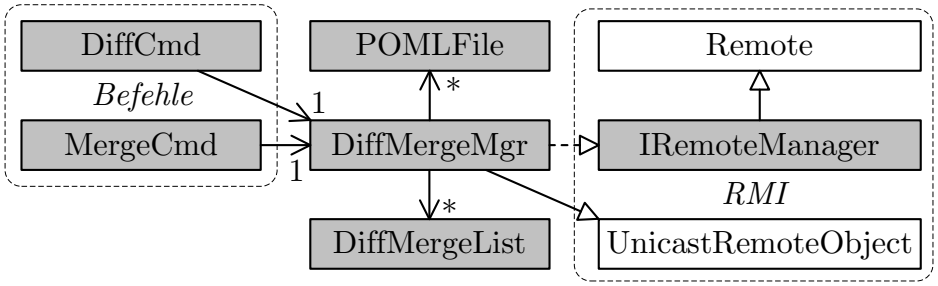


Bild 6.13: Umsetzung und Integration des Vergleichs und der Zusammenführung von Zeichnungsversionen

Im Gegensatz zu Zeichnungsrevisionen wird für den Vergleich und die Zusammenführung von Zeichnungsvarianten zusätzlich eine zweite CADEMIA-Instanz gestartet und auf der Basis von *RMI*[23] mit dem Diff-Merge-Manager verbunden. Die Kommunikation zur Visualisierungssteuerung basiert auf übertragenen alphanumerischen Befehlen, die interpretiert und anschließend ausgeführt werden (Makrofähigkeit).

6.3.2 Visualisierung

Zur grafischen Darstellung zweier Zustände einer Zeichnungskomponente in einem grafischen Fenster wird ein DiffView-Controller (DiffViewCtrl) implementiert, der die vorhandene Basisfunktionalität (BasicViewCtrl) erweitert (s. Bild 6.14). Entsprechend der vom Diff-Merge-Manager verwalteten Komponentenversionen erzeugt der DiffView-Controller Ansichten einer Zeichnungskomponente (Views) im Ursprungszustand und im Revisions- bzw. Variantenzustand. Beide Zustände werden bei selektierter Komponente gleichzeitig angezeigt. Hinzugefügte Komponenten werden grün, modifizierte

[21] s. Abschnitt 5.6.2 auf Seite 130 bzw. Abschnitt 5.7.2 auf Seite 139
[22] vgl. Diff-Algorithmus im Abschnitt 5.6.2 auf Seite 130
[23] Remote Method Invocation, deutsch: entfernter Methodenaufruf, s. [Horstmann u. Cornell 2002, S. 374 ff.]

Objekte gelb, gelöschte Zeichnungskomponenten rot und unveränderte Objekte grau dargestellt.

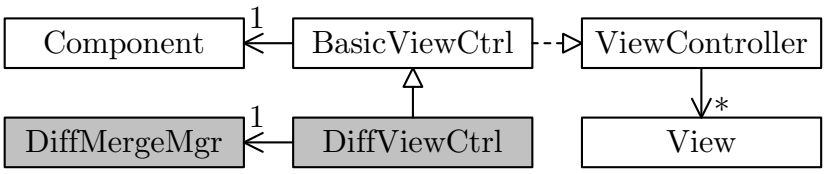
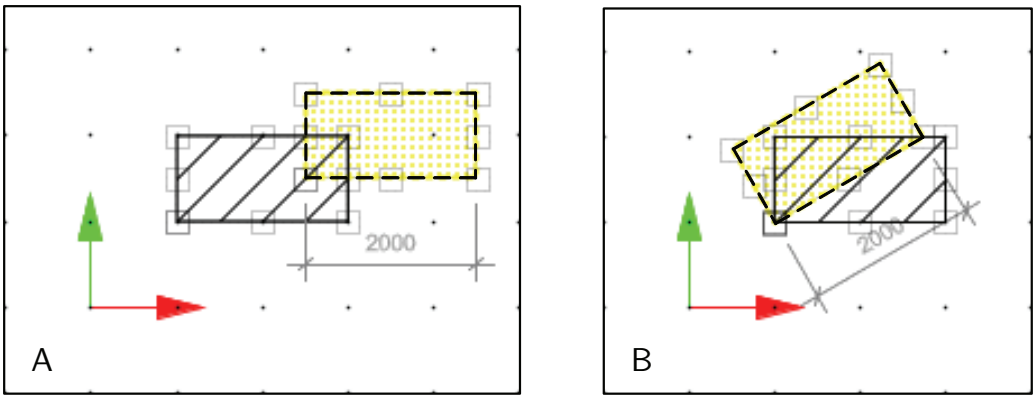


Bild 6.14: Integration des DiffView-Controllers

Beispiel 6.7: Visualisierung beim Variantenvergleich

Das Bild 6.15 stellt die Visualisierung einer Rechteckkomponente beim Variantenvergleich dar. In Anlehnung an das Bild 5.21 auf Seite 135 zeigen die Bilder 6.15a bzw. 6.15b die selektierte Darstellung der Rechteckvarianten jeweils in den grafischen Fenstern von unterschiedlichen CADEMIA-Instanzen.



(a) Rechteckversion in Variante A (b) Rechteckversion in Variante B

Bild 6.15: Grafische Visualisierung einer Rechteckkomponente beim Variantenvergleich

6.3.3 Navigation

Die Umsetzung der hierarchischen Ordnungs- und Navigationsstruktur zur Unterschiedpräsentation erfolgt, wie im Bild 6.16 dargestellt, durch einen Navigationsdialog (DiffNaviDialog). Auf der Basis des die Unterschiede vorhaltenden Diff-Merge-Managers (DiffMergeMgr) werden entsprechend der vorgeschlagenen objekt- und operationsbezogenen Visualisierungskonzepte[24] Baum- und Listenstrukturen (ComponentTree, OperationTree, JList) erzeugt und im Navigationsdialog angezeigt.

[24] s. Abschnitt 5.6.3 auf Seite 135

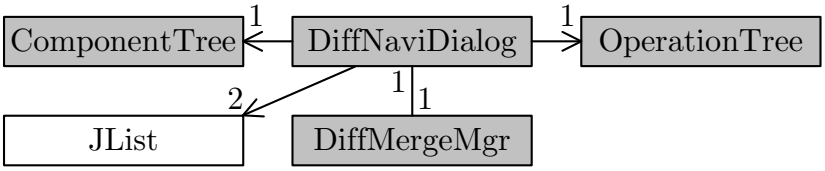


Bild 6.16: Integration des Navigationsdialogs zur Unterschiedpräsentation

Beispiel 6.8: Navigationsdialog beim Variantenvergleich

Mit Bezug auf das vorangegangene Beispiel 6.7 zeigt das Bild 6.17 den Navigationsdialog zur Unterschiedpräsentation für einen Variantenvergleich. Das Objekt mit der POID obj0@40f... stellt die Rechteckkomponente dar. Es ist der Konflikt zu erkennen, dass dieses Rechteck in der einen Variante (A) verschoben und in der anderen Variante (B) gedreht wird.

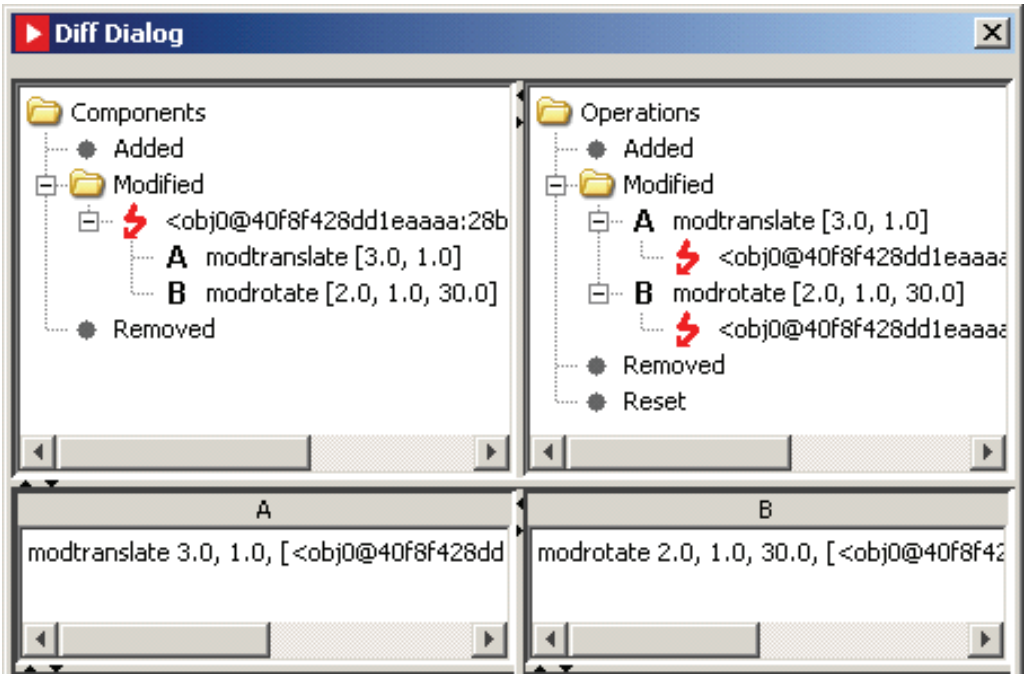


Bild 6.17: Navigationsdialog zur Unterschiedpräsentation von Zeichnungsvarianten

6.4 Beispielszenarios

In diesem Abschnitt werden Beispielszenarios für das umgesetzte operative Zeichnungsmodell in CADEMIA beschrieben. Planungsbeispiele verdeutlichen anhand von Screenshots die Anwendung von operativen Modellierprogrammen in der Benutzer- und Programmierschnittstelle sowie die Anwendung des operativen Zeichnungsmodells beim Vergleich und bei der Zusammenführung von Zeichnungsversionen.

6.4.1 Modellierprogramme

Beispiel 6.9: Modellierprogramm zur automatisierten Lastfallgenerierung

Ein erstes Beispiel stellt das bereits vorgestellte Modellierprogramm zu automatisierten Lastfallgenerierung innerhalb von CADEMIA für den Entwurf und die Stützmomentenberechnung von Dreifeldträgern dar. Die Umsetzung und die Anwendung dieses Modellierprogramms werden im Beispiel 5.4 auf Seite 113 des Abschnitts 5.3.1 beschrieben.

Beispiel 6.10: Modellierprogramm zur automatisierten Stahlbauprofilgenerierung

Als ein weiteres Beispiel wird ein OPL-Modellierprogramm auf Basis des vorgestellten operativen Zeichnungsmodells entwickelt. Es dient der Generierung von Stahlbauprofilen (IPE, HEA, HEB, U-Stahl) nach DIN 1025/ 1026[25] auf der Basis von gefüllten, krummlinig umrandeten Flächen. Das Programm stellt demnach eine Profil-Bibliothek für die Verwendung bei der Erstellung technischer Konstruktionszeichnungen des Stahlbaus dar.

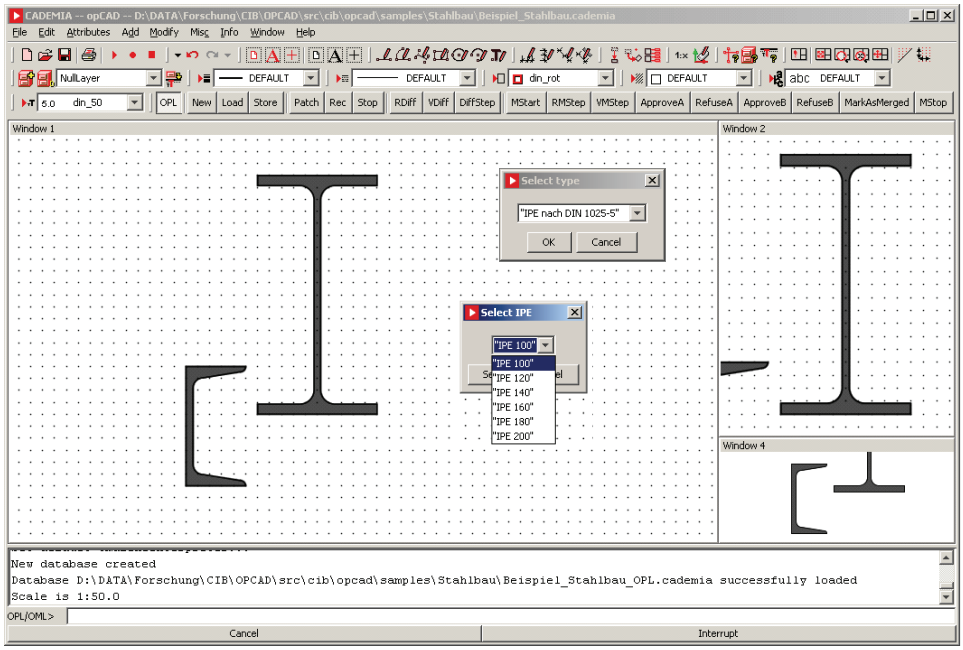


Bild 6.18: Screenshot: CADEMIA-Umgebung für ein Modellierprogramm zur Profilgenerierung im Stahlbau

[25] s. [Schneider 2002, S. 8.169 ff.]

Im Folgenden wird das Vorgehen für die Umsetzung und die Anwendung des Modellierprogramms in CADEMIA beschrieben:

1. Erzeugen der OPL-Prozedurdatei[26] mit Profildaten und einfachen grafischen Eingabekomponenten[27]
2. OPL-Interpreter starten, Prozedurdatei laden und Modellierprogramm starten
3. Profilarten auswählen, Platzierungspunkte spezifizieren und Profile generieren (s. Bild 6.18)
4. Modellierprogramm beenden und zur nativen Umgebung zurückkehren (nativen Interpreter setzen)
5. Native CAD-Funktionalität (Hinzufügen, Modifizieren von Geometriekomponenten, Bemaßung, Texten usw.) zur Fertigstellung der Konstruktionszeichnung nutzen (s. Bild 6.19)

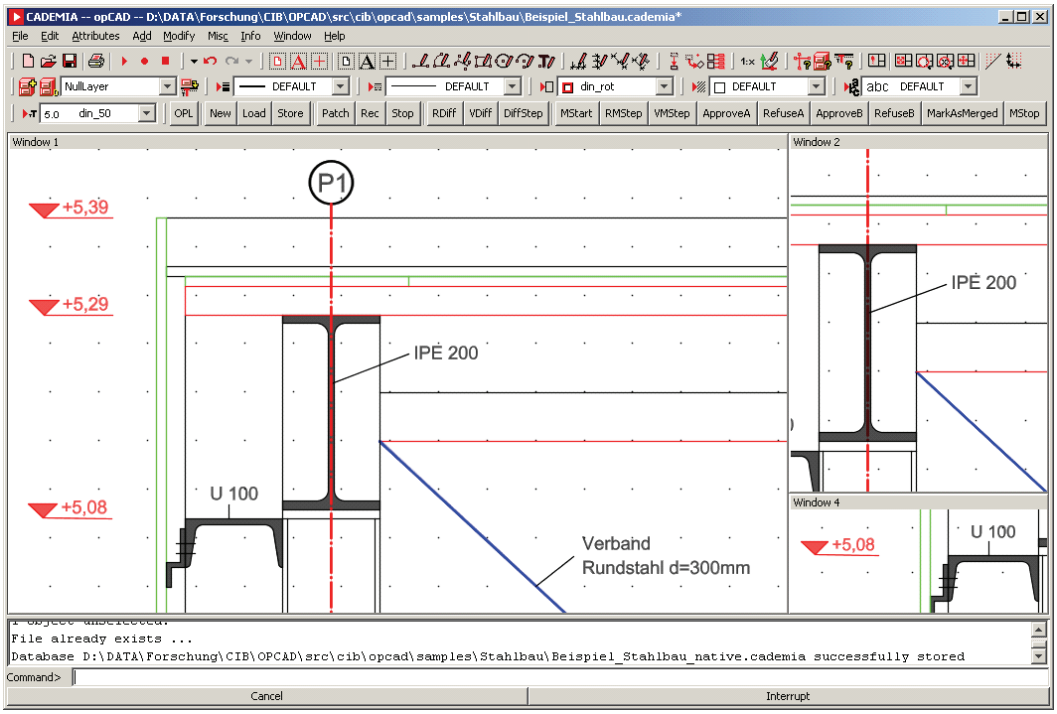


Bild 6.19: Screenshot: Native CADEMIA-Umgebung zur Weiterverarbeitung der Konstruktionszeichnung des Stahlbaus

Hinweis: Ist in einer anderen Fachapplikation innerhalb der Domäne *2D-Konstruktionszeichnung* das vorgestellte operative Zeichnungsmodell sprachbasiert und objektorientiert umgesetzt, dann kann das OPL-Modellierprogramm zur Stahlbauprofilgenerierung ohne Einschränkungen und ohne Modifikationen eingesetzt bzw. wiederverwendet werden.

[26] s. Abschnitt 4.3.3 auf Seite 91
[27] s. Abschnitt 4.3.5 auf Seite 93

6.4.2 Variantenplanung für eine Kranbahn

Ausgangssituation: In einem Gebäude wird eine Kranbahn eingeplant. In der Fachapplikation CADEMIA wird auf Basis der nativen Zeichnungsfunktionalität die Kranbahn unter anderem in der Zeichnung des Gebäudeschnitts (Schnitt A-A) konstruiert. Ausgehend von der virtuellen Zeichnungsversion B_β ist der aktuelle Zeichnungsstand des Gebäudeschnitts als Version B_i im nativen CADEMIA-Format in der Datei `Schnitt_A-A_Bi.cademia` gespeichert. Die bis dahin ausgeführten Änderungen $\langle \delta_{\beta,\ldots}, \ldots, \delta_{\ldots,i} \rangle$ werden in entsprechenden Änderungsdateien `Schnitt_A-A_delta_*.pom l` vorgehalten. Das Bild 6.20 veranschaulicht den aktuellen Verarbeitungsgraphen für die CADEMIA-Zeichnung des Gebäudeschnitts.

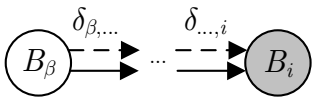


Bild 6.20: Verarbeitungsgraph der CADEMIA-Zeichung im Ausgangszustand

Änderungsanforderung: Aufgrund neuer Nutzungsanforderungen muss die lichte Höhe vom Geschossboden bis zum Kranhaken der Kranbrücke, ursprünglich bei 6,30 m, auf 6,50 m verändert werden. Das Bild 6.21 zeigt die Zeichnungsversion B_i des Gebäudeschnittes mit einer Änderungsanforderung an die eingeplante Kranbahn.

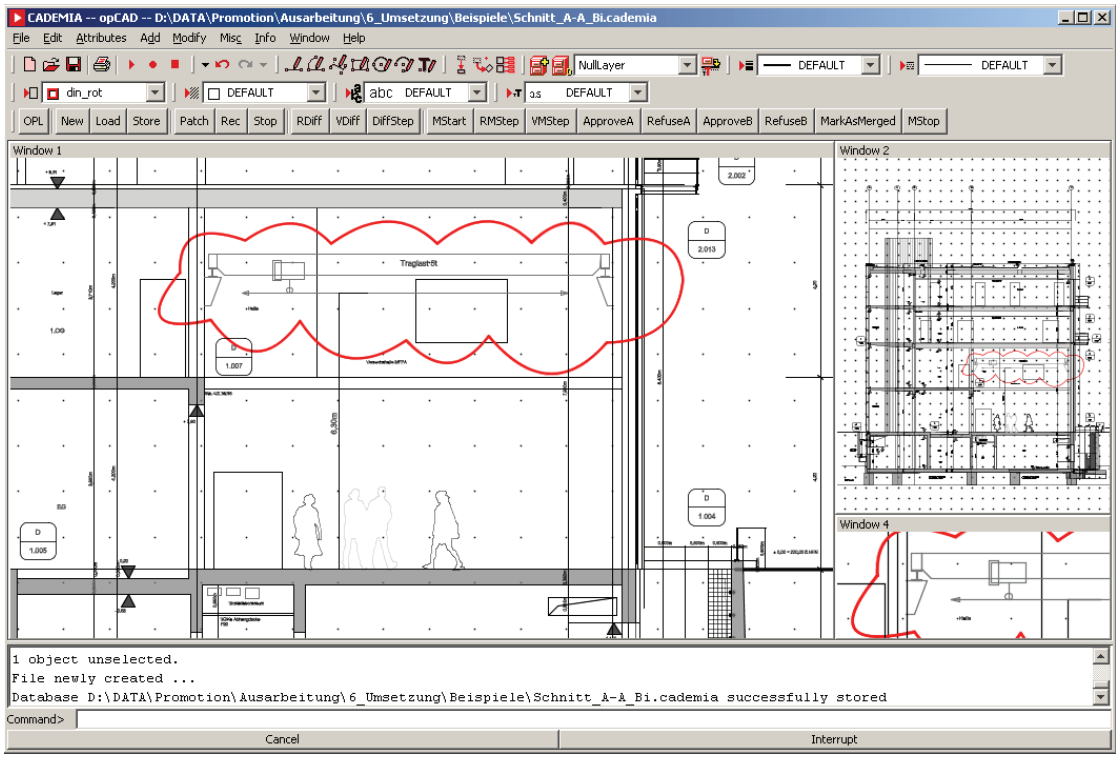


Bild 6.21: Screenshot: CADEMIA-Zeichnungsversion B_i mit Änderungsanforderung an die eingeplante Kranbahn

Variantenplanung: Für die Umsetzng dieser Planungsänderung werden folgende zwei Vorschläge zur Modifikation der Kranbahn gemacht:

A: Die gesamte Kranbahn mit Konsolen, Kranbrücke, Kranbahnträger und Kranfahrwerk um 0,20 m nach oben verschieben.

B: Einen anderen Typ Kranbrücke mit Höhengewinn einbauen und Konsolen, Kranbahnträger und Kranfahrwerk entsprechend anpassen.

Die Varianten A und B werden auf Grundlage der Ursprungsversion B_i in separaten Zeichnungsversionen B_a bzw. B_b konstruiert. Dazu wird die Version B_i in Form der Datei `Schnitt _A-A_Bi.cademia` geladen, entsprechend der jeweiligen Entwurfsintention modifiziert und jeweils als native CADEMIA-Zeichnungsdatei `Schnitt_A-A_Ba.cademia` bzw. `Schnitt_A-A _Bb.cademia` gespeichert. Die dabei ausgeführten Änderungen $\delta_{i,a}$ bzw. $\delta_{i,b}$ werden jeweils in eine POML-Änderungsdatei `Schnitt_A-A_delta_ia.poml` bzw. `Schnitt_A-A_delta_ib .poml` geschrieben. Während das Bild 6.22 den aktuellen Verarbeitungsgraphen der Gebäudeschnittzeichnung nach der Variantenplanung darstellt, veranschaulichen die Bilder 6.23a und 6.23b auf der nächsten Seite die unterschiedlichen Planungsstände und die entspechenden Änderungsdateien.

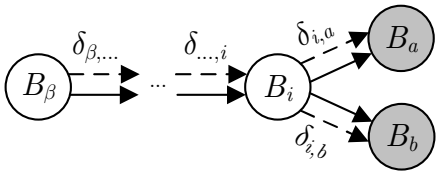
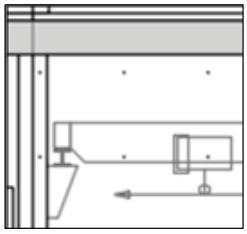


Bild 6.22: Verarbeitungsgraph der CADEMIA-Zeichung nach der Variantenplanung

Variantenvergleich: In diesem Beispiel wird davon ausgegangen, dass weder Planungsvariante A noch Variante B einzeln für eine endgültige Lösung infrage kommen. Die Grundlage für eine beabsichtigte Zusammenführung der Kranbahnvarianten bildet der Vergleich der Zeichnungsversionen B_a und B_b. Dazu lädt der Fachplaner die Ursprungsversion B_i der Varianten aus der Datei `Schnitt_A-A_Bi.cademia` in die Fachapplikation und starten den Vergleich (DiffCmd, vgl. 6.3 auf Seite 162). Auf der Basis der in den Änderungsdateien `Schnitt_A-A_delta_ia.poml` und `Schnitt_A-A_delta_ib.poml` gespeicherten Änderungen werden die Unterschiede bestimmt und dem Fachplaner präsentiert. Das Bild 6.24 auf Seite 171 zeigt die CADEMIA-Fenster zur Unterschiedvisualisierung mit Navigationsdialog. Während im oberen Applikationsfenster die Zeichnungsvariante A dargestellt ist, wird im unteren Fenster die Zeichnungsvariante B betrachtet.

Unterschiede: Neben erkannten Konflikten für alle modifizierten Kranbahnkomponenten zeigt der Operationsbaum im Navigationsdialog (Bild 6.24 auf Seite 171 rechts oben), dass in Variante A mit der Operationsinstanz `modtranslate [0.0, 0.2]` alle Kranbahnkomponenten gleichzeitig verschoben werden. Weiterhin ist im Komponentenbaum des Dialogs (links oben) zu erkennen, dass die selektierte Kranbrücke in Variante A einmal geändert wird, in Variante B hingegen drei Modifikationen erfährt. Wie

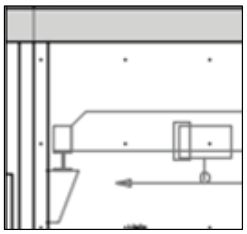


Schnitt_A-A_Ba.cademia

```
#poml "1.0";
#Schnitt_A-A_Bi->Schnitt_A-A_Ba;
modtranslate 0.0, 0.2,
        [<obj68@538f5e80e331cb3b:-7ffb>,
         <obj63@538f5e80e331cb3b:-7ffb>,
         <obj74@538f5e80e331cb3b:-7ffb>,
         <obj67@538f5e80e331cb3b:-7ffb>,
         ...
         <obj62@538f5e80e331cb3b:-7ffb>,
         <obj72@538f5e80e331cb3b:-7ffb>,
         <obj65@538f5e80e331cb3b:-7ffb>];
modtranslate 0.0, 0.2, [[<LINE@2E5D@*D255>, 0],
        <LINE@2E5C@*D255>];
modfeature "TEXT", ["6,50m"], [<MTEXT@2E60@*D255>];
```

Schnitt_A-A_delta_ia.poml

(a) Version B_a mit Änderung $\delta_{i,a}$



Schnitt_A-A_Bb.cademia

```
#poml "1.0";
#Schnitt_A-A_Bi->Schnitt_A-A_Bb;
modtransform 0.999999933539, -3.6458494E-4, ...
        0.999999933539, -0.009410481324, ...,
        [<obj73@538f5e80e331cb3b:-7ffb>];
modtranslate 1.80443749E-4, -0.298547092825,
        [<obj73@538f5e80e331cb3b:-7ffb>];
...
modtranslate 0.0, 0.03,
        [<obj68@538f5e80e331cb3b:-7ffb>,
         <obj74@538f5e80e331cb3b:-7ffb>,
         ...
         <obj63@538f5e80e331cb3b:-7ffb>];
modfeature "TEXT", ["6,50m"], [<MTEXT@2E60@*D255>];
```

Schnitt_A-A_delta_ib.poml

(b) Version B_b mit Änderung $\delta_{i,b}$

Bild 6.23: CADEMIA-Zeichnungsvarianten mit Änderungen

sich einzelne Modifikationen auswirken, kann dem Fachplaner im Schritt-für-Schritt-Vergleichsmodus im Hinblick auf die beabsichtigte Zusammenführung präsentiert werden.

Variantenzusammenführung: Folgende Randbedingungen und Anforderungen charakterisieren das favorisierte Entwurfsergebis nach einer Zusammenführung der Planungsvarianten A und B:

a) Die Konsolen müssen aufgrund anderer Abhängigkeiten unverändert bleiben.

b) Der Typ und die Position der Kranbrücke sind gemäß Planungsvariante B zu wählen.

c) Das Kranfahrwerk ist entsprechend Planungsvariante A zu konstruieren.

d) Der Kranbahnträger mit Seitenführung ist im Hinblick auf die Höhe neu zu entwerfen.

Nachdem der Fachplaner die Varianten B_a und B_b der Gebäudeschnittzeichung miteinander verglichen hat, führt er diese gemäß der beschriebenen Anforderungen zusammen. Dazu modifiziert der Fachplaner die bereits geladene Ursprungsversion B_i wie folgt. Er startet die Zusammenführung und aktiviert den Journaling-Mechanismus (MergeCmd, vgl. 6.3 auf Seite 162). Der Fachplaner wird nun in die Lage versetzt, ihm in Listen (s. Bild 6.24, Navigationsdialog unten) präsentierte POML-Operationsinstanzen aus den gespeicherten Änderungen abzulehnen (a), zu übernehmen und anzuwenden (b, c) oder auch neue native Befehle auszuführen (d). Anstelle eines kompletten Neuentwurfs stützt sich der Fachplaner zur Umsetzung der Anforderungen (b)

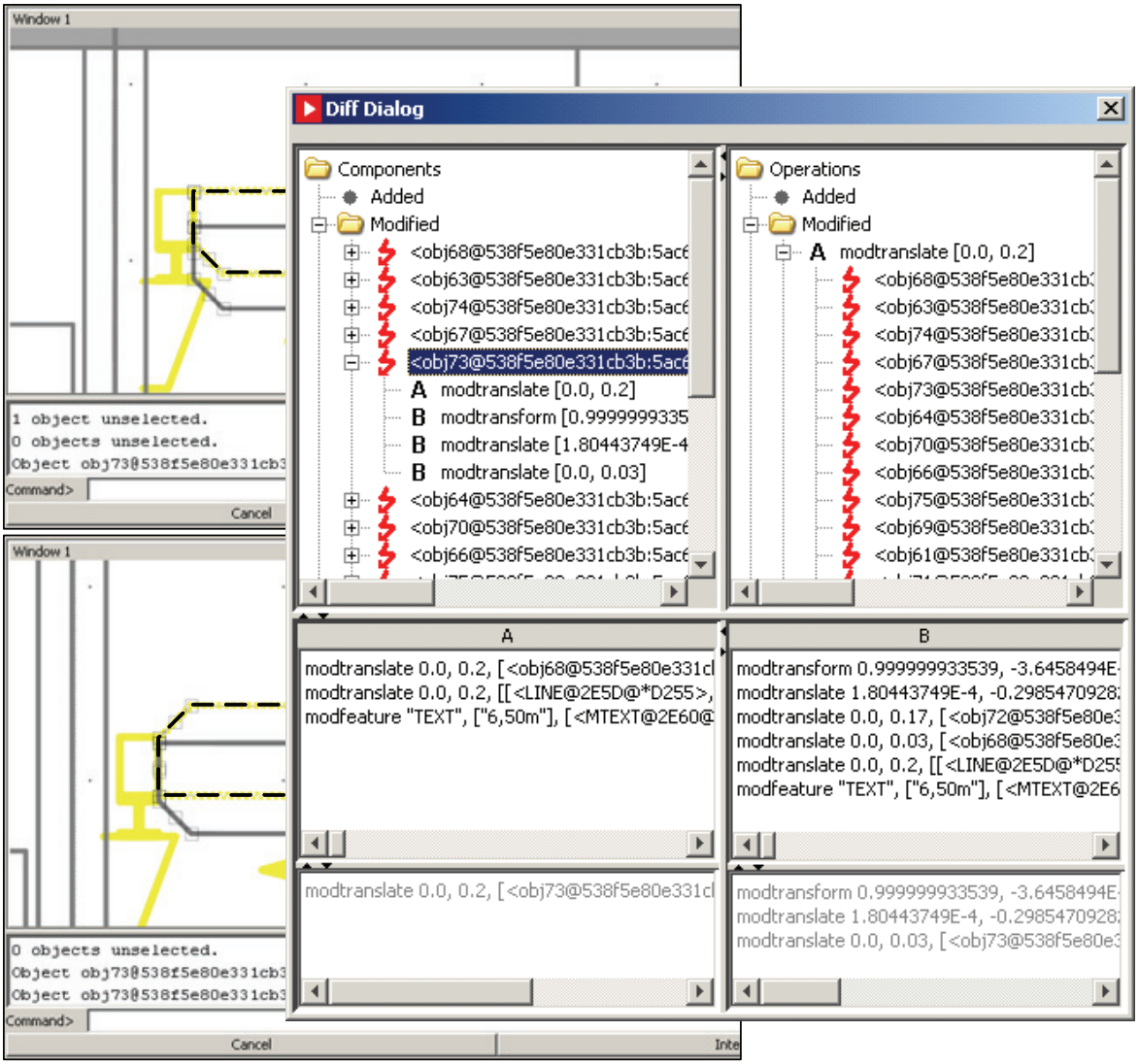


Bild 6.24: Screenshot: CADEMIA-Umgebung beim Vergleich und beim Zusammenführen von Zeichnungsvarianten A (oben) und B (unten)

und (c) auf die ursprünglichen Entwurfsabsichten. Er selektiert die Modellobjekte der Kranbrücke und wendet die objektbezogenen Operationsinstanzen aus der Änderung $\delta_{i,b}$ auf diese Objekte an. Anschließend wiederholt er den Vorgang analog für die Modellobjekte des Kranfahrwerks, der Maßlinien und des Maßtextes auf der Basis der Änderung $\delta_{i,a}$. Hier wird deutlich, dass ursprüngliche Planungsintentionen im Zusammenführungsvorgang vorteilhaft wiederverwendet werden können. Darüber hinaus bleibt die Konsistenz der Zeichnung gesichert, da entweder Operationsinstanzen oder native Befehle auf der Zeichnung operieren. Als Ergebnis entsteht die Zeichnungsversion B_c. Sämtliche Verarbeitungsschritte, die im Prozess der Zusammenführung auf die Ursprungsversion B_i angewendet werden, zeichnet der POML-Rekorder als neue Änderung $\delta_{i,c}$ auf. Während das Bild 6.25 den aktuellen Verarbeitungsgraphen der Gebäudeschnittzeichnung nach der Variantenzusammenführung darstellt, veranschau-

licht das Bild 6.26 die CADEMIA-Zeichnung in der Version B_c. Diese Version wird in der Datei `Schnitt_A-A_Bc.cademia` gespeichert und bildet die Grundlage für eventuell folgende Planungsschritte.

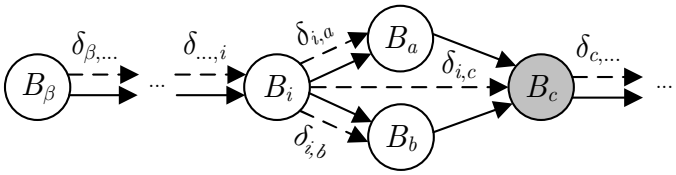


Bild 6.25: Verarbeitungsgraph der CADEMIA-Zeichung nach der Zusammenführung

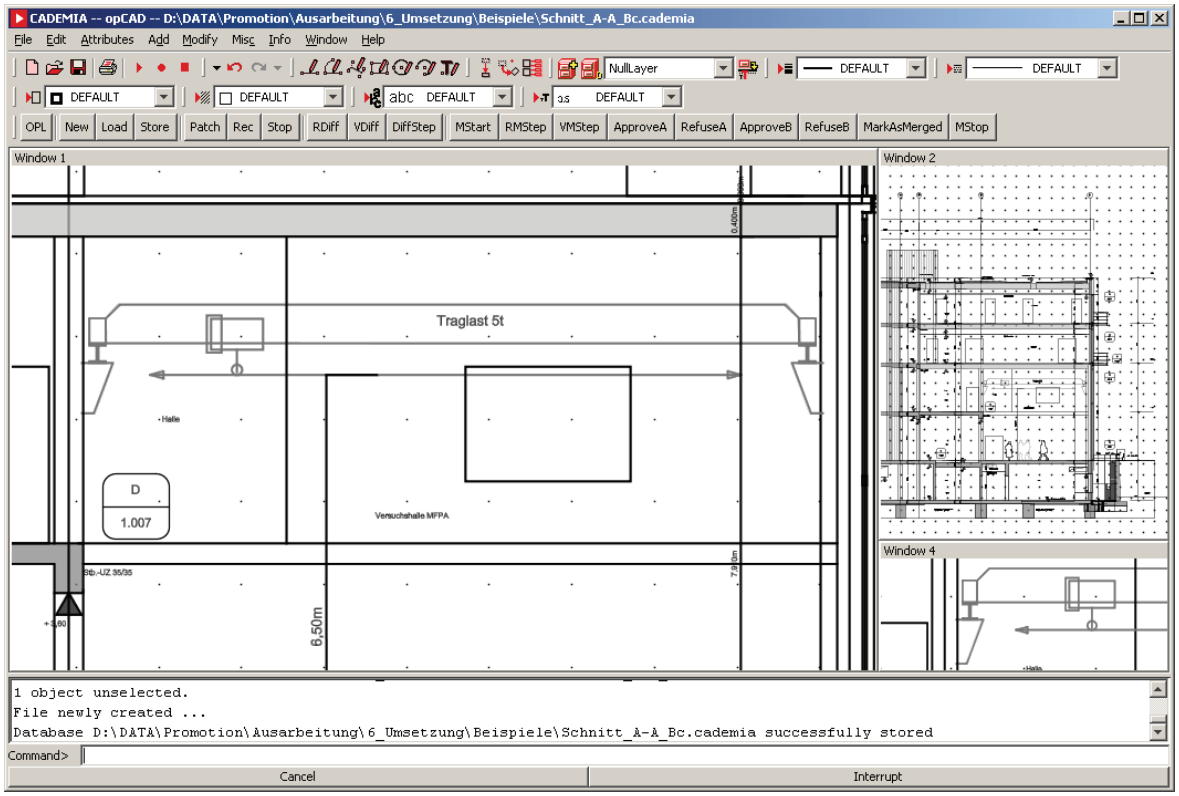


Bild 6.26: Screenshot: Zusammengeführte CADEMIA-Zeichnungsversion des Gebäudeschnitts

Hinweis: Eine Betrachtung der Dateigrößen der nativen Zeichnungsdateien `*.cademia` und der Größen der Änderungsdateien `*.poml` ergibt einen Speicheraufwand für eine ausgewertete Zeichnung von ca. 600 KB und für eine nicht-ausgewertete Änderung von ca. 2 KB. Das bedeutet, für kleine Änderungen kann der Speicheraufwand durch das Vorhalten von nicht-ausgewerten Informationen erheblich reduziert werden.

7 Zusammenfassung und Ausblick

"Good positions don't win games, good moves do."
Gerald Abrahams (1907–1980)

7.1 Zusammenfassung

Problemstellung: Im Bauplanungsprozess arbeiten verschiedene Fachplaner an einer gemeinsamen Aufgabe – der Planung eines Bauwerks – zusammen. Verfügbare Kommunikations- und Softwaretechnologien unterstützen diesen Prozess heutzutage vorwiegend durch digitale Medien.

Die Komplexität der spezialisierten Planungsaufgaben hat zu einer großen Anzahl verschiedener digitaler Bauwerksmodelle und Fachapplikationen geführt. Traditionelle Kooperationsansätze beschäftigen sich mit der Integration von Modellen und Applikationen durch Standardisierung der unterschiedlichen Datenstrukturen und widmen sich darauf basierenden Konzepten bei der Versionierung, beim Austausch und der Archivierung sowie beim Vergleich und bei der Zusammenführung von Bauwerksinformationen.

Dokumentenmanagementsysteme werden nach dem aktuellen Stand der Technik zur Unterstützung der vernetzt-kooperativen Bauwerksplanung eingesetzt. Bauwerksinformationen werden in versionierten Dokumenten gespeichert. Dokumentenmanagementsystemen ist weder die Struktur noch der Inhalt der verwalteten Dokumentversionen bekannt. Aus diesem Grund können solche Systeme nur eingeschränkt als Kooperationsplattform genutzt werden.

Verteilte Bauwerksmodelle, die auf Basis der Objektorientierung den virtuellen Bauwerkszustand beschreiben und versionieren, bilden den aktuellen Stand der Forschung. In diesen zustandsorientierten Modellen bleibt die Verarbeitungssemantik – änderungsorientierte Informationen, die den Planungsvorgang charakterisieren – unberücksichtigt. Dieser Mangel führt zu derzeitigen Problemen beim Austausch, beim Vergleich und bei der Zusammenführung von versionierten Bauwerksinformationen. Eine ausreichende Kooperationsunterstützung ist auf Grundlage versionierter Bauwerkszustände allein nicht gegeben.

Zielsetzung: Die grundlegende Idee der Verarbeitungsorientierung kann anschaulich in Analogie zum Schachspiel beschrieben werden: Am Ende einer Schachpartie stehen noch einzelne Figuren auf dem Schachbrett. Das Ergebnis des Spiels ist als Zustand sichtbar – wie es dazu kam jedoch nicht. Aus diesem Grund wird beispielsweise in Zeitungen zusätzlich zur Schlussstellung die Sequenz der einzelnen Züge von Spielbeginn

an in einer bestimmten Notation abgedruckt. Auf Basis dieser änderungsorientierten Informationen können Schachinteressierte alle Züge nachvollziehen und Spielstrategien erkennen. Zur Beschreibung einer Schachpartie ist das Spielergebnis als Zustand allein nicht ausreichend. Die Spielhistorie wird deshalb zusätzlich in Form von Änderungen angegeben.

Ziel der vorliegenden Arbeit ist die Entwicklung eines Modells zur ganzheitlichen Betrachtung der digitalen Bauwerksmodellierung. Das verarbeitungsorientierte Modell soll das virtuelle Bauwerk sowohl zustandsorientiert als auch änderungsorientiert beschreiben. Dabei sind verfügbare Methoden, Modelle und Applikationen der derzeitigen Bauwerksplanung zu berücksichtigen, wiederzuverwenden und zu erweitern.

Ein weiteres Ziel betrifft den Entwurf eines allgemeingültigen, übertragbaren und offenen Ansatzes für die Formalisierung des entwickelten verarbeitungsorientierten Modells.

Auf Grundlage des vorgeschlagenen Ansatzes sind neue Kooperationskonzepte zur Unterstützung des Austauschs, der Archivierung, des Vergleichs und der Zusammenführung von versionierten Bauwerksinformationen zu erarbeiten.

Vorgehensweise: Die *Einleitung* beschreibt die Problemstellung und die Zielsetzung der vorliegenden Arbeit. Einführende Beispiele dienen dazu, das Anliegen und die grundlegende Idee des vorgeschlagenen Ansatzes zu verdeutlichen. Das in dieser Arbeit entwickelte Systemkonzept der Verarbeitungsorientierung ist im Bild 7.1 dargestellt.

Den *Stand der Technik und Forschung* (Bild 7.1 (2)) bilden die objektorientierte Methode zur Bauwerks- und Fachapplikationsmodellierung und darauf basierende Kooperationskonzepte bei der Versionierung, beim Austausch, bei der Archivierung, beim Vergleich und bei der Zusammenführung von Bauwerksinformationen. Begleitend zur Beschreibung verfügbarer Ansätze zur Kooperationsunterstützung werden Probleme identifiziert sowie Zusammenhänge und Abgrenzungen zur vorliegenden Arbeit dargelegt.

Aufbauend auf den Ansätzen der Objekt-, der Versions- und der Änderungsorientierung wird der vorgeschlagene Modellierungsansatz als ganzheitliche Betrachtung entwickelt. Zusätzlich zur Zustandsbeschreibung in Form von Bauwerksversionen werden Zustandsänderungen des virtuellen Bauwerks als Sequenz von Modellieroperationen in der *Modellbildung* (Bild 7.1 (3)) betrachtet. Mit der Absicht, das vorgeschlagene Modell dauerhaft und formal zu beschreiben, wird eine mathematische Formulierung auf Basis der Mengenlehre und der Relationenalgebra gewählt. Auf Grundlage des entwickelten Modells wird die Bedeutung von Modellieroperationen im Kontext der objektorientierten Bauwerksmodellierung herausgestellt.

In Analogie zur Schachnotation für Spielzüge wird zur Formalisierung der Operationen eine *operative Modellierungssprache* (Bild 7.1 (4)) definiert. Ausgehend von den Anforderungen an eine solche Sprache werden drei aufeinander aufbauende Sprachebenen unterschieden. Es wird untersucht, wie die Sprache zur Formulierung von Modellierprogrammen, Modellieroperationen und Änderungen angewendet werden kann. Die Sprachsyntax wird für jede dieser Ebenen formal durch eine Grammatik definiert.

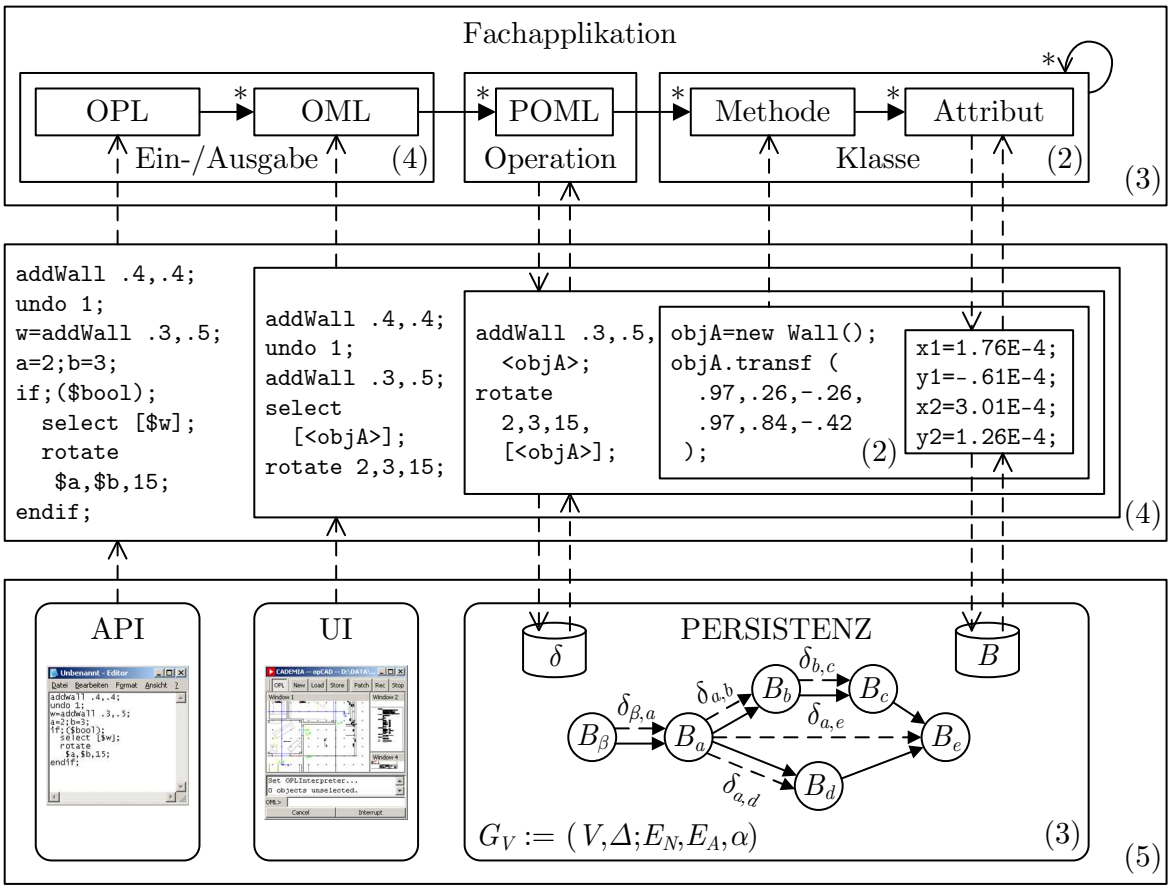


Bild 7.1: Systemkonzept der Verarbeitungsorientierung

Die *Anwendungskonzepte* (Bild 7.1 (5)) des verarbeitungsorientierten Modells und der operativen Modellierungssprache im kooperativen Bauplanungsprozess werden entwickelt. Dazu werden die Standardisierung operativer Modelle in Fachdomänen untersucht und die Umsetzung operativer Modelle in Fachapplikationen beschrieben. Darüber hinaus erfolgen Betrachtungen zur Anwendung der Sprache in den Benutzer- und Programmierschnittstellen von Fachapplikationen sowie Untersuchungen zu neuen Kooperationskonzepten beim Austausch, bei der Archivierung, beim Vergleich und bei der Zusammenführung von versionierten Bauwerksinformationen.

Durch die vorgestellte *Pilotimplementierung* wird die prinzipielle Anwendbarkeit des verarbeitungsorientierten Modells und der Modellierungssprache auf Basis eines verfügbaren Open-Source-Systems nachgewiesen. Es wird sowohl die sprachbasierte als auch die objektorientierte Umsetzung eines operativen Zeichnungsmodells beispielhaft gezeigt. Zur Verifikation der Lösungskonzepte wird die Anwendung des operativen Zeichnungsmodells in der Benutzer- und Programmierschnittstelle des Open-Source-Systems sowie beim Vergleich und bei der Zusammenführung von Zeichnungsversionen beschrieben.

Erkenntnisse und Bewertung: Die wesentlichen Erkenntnisse aus dieser Arbeit können wie folgt zusammengefasst und bewertet werden:

- Das verarbeitungsorientierte Modell stellt die ganzheitliche Betrachtung der digitalen Bauwerksmodellierung dar. Hiermit wird der vereinheitlichte Zusammenhang zwischen dem objektorientierten, dem versionsorientierten und dem änderungsorientierten Ansatz hergestellt. Die formale Beschreibung des Modells auf der Basis der Mengenlehre und der Relationenalgebra garantiert die Nachvollziehbarkeit, Übertragbarkeit und Offenheit des Lösungsansatzes.

- Das operatives Modell bildet als Menge von Modellieroperationen eine Verarbeitungsschnittstelle für Objektmodelle, reichert objektorientierte Bauwerksmodelle mit Änderungssemantik an und trägt zur Konsistenzsicherung in diesen Modellen bei. Zur eindeutigen Identifizierung der Modellobjekte in Bauwerksmodellen sind persistente Objektidentifikatoren unabdingbar.

- Die operative Modellierungssprache bildet eine Schnittstelle zwischen Fachplaner und Fachapplikation. Sie ist sowohl zur Interaktion in der Benutzerschnittstelle, zur Formulierung von Modellierprogrammen in der Programmierschnittstelle als auch zur Beschreibung von Änderungen geeignet. Die von aktuellen Umsetzungstechnologien unabhängige und damit dauerhafte Definition der Modellierungssprache wird durch eine Grammatik erreicht.

- Auf der Basis der Modellierungssprache können Modellieroperationen innerhalb einer Fachdomäne standardisiert werden. Ein standardisiertes operatives Modell bildet als definierte Menge von Modellieroperationen eine applikationsunabhängige Benutzer- und Programmierschnittstelle für Fachapplikationen einer Fachdomäne. Fachplaner können ad-hoc beliebige Fachapplikationen bedienen und automatisiert modellieren.

- Auf der Grundlage von Änderungen können Bauwerksinformationen versioniert werden. Operative Modelle sind sowohl in unversionierten als auch in versionierten Planungsumgebungen anwendbar. In versionierten Umgebungen unterstützt der vorgeschlagene Ansatz die asynchrone, die synchron parallele und die synchron wechselseitige Kooperation.

- Der auf Änderungen basierende Informationsaustausch ermöglicht das Übertragen von Entwurfsabsichten und ist durch *nicht*-kumulierende Informationsverluste gekennzeichnet. Langzeitarchivierte Änderungen sind applikationsneutral und können nicht nur von Software, sondern auch vom Fachplaner interpretiert werden.

- Zur Beschreibung von Unterschieden als Ergebnis eines Vergleichs von Planungsständen können Änderungen vorteilhaft herangezogen werden. Im Gegensatz zu zustandsorientierten Verfahren ist nicht nur bekannt, *dass* sich ein Modellobjekt geändert hat, sondern auch *wie* es im Modellkontext modifiziert wurde. Im Falle eines Vergleichs von Revisionen sind die Unterschiede explizit als Änderungen

verfügbar. Neben der objektbezogenen Visualisierung präsentieren operationsbezogene Unterschiede die ursprünglichen Entwurfsabsichten beim Vergleich von Planungsständen.

- Die Zusammenführung unterschiedlicher Zustände des virtuellen Bauwerks auf der Basis von gespeicherten Änderungen ist durch zwei wesentliche Vorteile gegenüber zustandsorientierten Ansätzen gekennzeichnet. Zum einen werden ursprüngliche Entwurfsintentionen im Prozess der Zusammenführung berücksichtigt, indem gespeicherte Modellieroperationen erneut ausgeführt werden können. Zum anderen bleibt die Konsistenz der Bauwerksinformationen während der Zusammenführung gesichert, da Modellieroperationen nicht auf Objektebene allein, sondern im gesamten Modellkontext wirken.

- Die Pilotimplementierung zeigt, dass operative Modelle unter Berücksichtigung der Wiederverwendung verfügbarer Methoden, Modelle und Applikationen umgesetzt werden können und in Benutzer- und Programmierschnittstellen sowie beim Vergleich und bei der Zusammenführung von Planungsständen prinzipiell anwendbar sind.

7.2 Ausblick

Prinzipielle praktische Anwendbarkeit: Im derzeit bearbeiteten DFG[1]-Transferprojekt zum Thema „Operatives Modellieren im Bauwesen“ wird in Zusammenarbeit der Bauhaus-Universität Weimar und dem Praxispartner HOCHTIEF Construction AG die prinzipielle praktische Anwendbarkeit des erforschten verarbeitungsorientierten Modellierungsansatzes untersucht. Die Anwendungsbereiche gehen dabei vom Informationsaustausch bis hin zum Versions- und Änderungsmanagement inklusive des Vergleichs von Planungsständen in der Schal- und Bewehrungsplanung mit der kommerziellen Software AutoCAD.

Fachdomänenübergreifende Kooperation: Im Graduiertenkolleg 1462 an der Bauhaus-Universität Weimar zum Thema *Modellqualitäten* beschäftigt sich ein Forschungsschwerpunkt mit der Entwicklung von Kooperationsplattformen für gekoppelte (Partial-) Modelle unterschiedlicher Fachdomänen. Hierzu soll ein Lösungsansatz auf Basis nicht-ausgewerteter, operativer Modelle untersucht werden. Ein typischer Anwendungsfall der operativen Modellierung besteht in der synchronen Manipulation einer gemeinsamen Modellinstanz im Kontext von Mehrfeldproblemen. Dazu propagieren die verteilt kooperierenden Ingenieure ihre Änderungsanforderungen zur zentralen Modellinstanz, wo sie sequentialisiert werden. Nach der Ausführung werden entsprechende Antworten an die Ingenieure gesendet, damit die Visualisierung stets aktuell ist. Durch den Einsatz operativer Modelle wird eine effiziente Wiederverwendung von Single-User-Anwendungen in einer verteilten Umgebung erwartet. Die bisherigen Ergebnisse und Erkenntnisse zum verarbeitungsorientierten Modellieren bilden hier die Grundlage für künftige Forschung.

[1] Deutsche Forschungsgemeinschaft

Simulationsmodellierung: Es zu untersuchen, wie auf Basis zusätzlicher nicht-ausgewerteter Informationen Simulationsmodelle beschrieben werden können, indem den Modellieroperationen eine Zeit t zugeordnet wird. Als Simulation wird hier die sequentielle und parallele Ausführung von Operationsinstanzen verstanden, die eine Modellinstanz zeitabhängig verändert. Beispielsweise können zeitabhängige operative Modelle zur Beschreibung von Simulationen des Bauablaufs (4D-CAD zur Planung des Ausführungsprozesses) und von Tragwerkssimulationen (im Entwurfsprozess) eingesetzt werden.

Anwendungen im Baubetrieb: In [König u. a. 2006] wird ein fachliches Modell zur Abbildung und Berechnung von Bauablaufplänen mit verschiedenen Ausführungsvarianten beschrieben. Eine Instanz dieses Modells wird parallel zur Bauausführung gepflegt und dazu verwendet, jederzeit noch ausführbare Varianten zu bestimmen, wenn Änderungen oder Störungen im geplanten Bauablauf eintreten. Entscheidungen über die Wahl von Ausführungsalternativen und die ausgeführte Variante selbst sollen nachvollziehbar gespeichert werden, um dieses Wissen in zukünftigen ähnlichen Projekten wiederzuverwenden. Neben der Simulation von Bauabläufen kann der verarbeitungsorientierte Modellierungsansatz dazu herangezogen werden, Modellieroperationen zur Manipulation der Instanz des Bauablaufmodells in Form von Änderungen zu beschreiben und nachvollziehbar zu speichern (z. B. Entscheidungsdokumentation).

Verzeichnis der Beispiele

Verzeichnis der Listings

Literaturverzeichnis

[Adachi u. a. 2003] Adachi, Yoshinobu ; Forester, James ; Hyvarinen, Juha ; Karstila, Kari ; Liebich, Thomas ; Wix, Jeffrey: *Industry Foundation Classes IFC 2x Edition 2*. International Alliance for Interoperability (IAI), 2003. http://www.iai-international.org/Model/documentation/R2x2_Final/Online_Documents/index.htm. – Abruf: 27.04.2005

[Aho u. a. 1988] Aho, Alfread V. ; Sethi, Ravi ; Ullman, Jeffrey D.: *Compilers: Principles, Techniques, and Tools*. Reading u. a. : Addison-Wesley, 1988

[Aish 2005] Aish, Robert: *Introduction to Generative Components*. White Paper. http://www.bentley.com/en-US/Markets/Building/White+Papers/White+Papers.htm. Version: 2005. – Abruf: 05.03.2007

[AutoDesk 2008] AutoDesk, Inc.: *DXF Reference*, Januar 2008. http://images.autodesk.com/adsk/files/acad_dxf.pdf. – Abruf: 16.06.2008

[Balzert 2005] Balzert, Heide: *Lehrbuch der Objektmodellierung*. 2. Auflage. Heidelberg u. a. : Elsevier GmbH, Spektrum Akademischer Verlag, 2005

[Beer 2006] Beer, Daniel G.: *Systementwurf für verteilte Applikationen und Modelle im Bauplanungsprozess*. Aachen : Shaker Verlag, 2006 (Berichte aus der Bauinformatik). – Dissertation

[Beucke u. a. 1990] Beucke, K. ; Caprano, P. ; Firmenich, B.:: Offene CAD-Konzepte für Lösungen im Bauwesen. In: *Bauingenieur* 65 (1990), S. 477–484

[Beucke 2002] Beucke, Karl: *CAE im Planungsprozess*, Bauhaus-Universität Weimar, Vorlesungsskript, 2002. http://gonzo.uni-weimar.de/~bauinf/lehre/CAEPlan/Vorlesung/CAE_Skript.pdf. – Stand: März 2002

[Beucke 2006] Beucke, Karl: Versioned Objects as a Basis for Engineering Cooperation. In: Smith, Ian F. C. (Hrsg.): *Intelligent Computing in Engineering and Architecture*. Berlin u. a. : Springer-Verlag, 2006, S. 74–82. – 13th EG-ICE Workshop 2006

[Bilchuk 2005] Bilchuk, Irina: *Generalisierte Informationsstrukturen für Anwendungen im Bauwesen*, Technische Universität Berlin, Diss., 2005

[Booch u. a. 2001] Booch, G. ; Rumbaugh, J. ; Jacobson, I.: *The Unified Modeling Language User Guide*. Boston u. a. : Addison-Wesley, 2001

[Booch 1994] BOOCH, Grady: *Object-oriented analysis and design*. 2. Edition. Redwood City u. a. : Benjamin/Cummings, 1994

[Bourne 1988] BOURNE, Stephen R.: *Das UNIX System V*. Bonn u. a. : Addison-Wesley, 1988. – Dt. Übersetzung von Hans Peter Huber und Brigitte Weisshaar-Huber

[Bretschneider 1998] BRETSCHNEIDER, Dirk: *Modellierung rechnergestützter, kooperativer Arbeit in der Tragwerksplanung*. Düsseldorf : VDI Verlag, 1998. – Fortschritt-Berichte VDI: Reihe 4 Nr. 151

[Broy 1998a] BROY, Manfred: *Informatik – Eine grundlegende Einführung*. Bd. 1: *Programmierung und Rechnerstrukturen*. 2. Auflage. Berlin u. a. : Springer-Verlag, 1998

[Broy 1998b] BROY, Manfred: *Informatik – Eine grundlegende Einführung*. Bd. 2: *Systemstrukturen und Theoretische Informatik*. 2. Auflage. Berlin u.a. : Springer-Verlag, 1998

[Butke 2006] BUTKE, Gerhard: Anwendung des PDF im Bauwesen – Teil 4 / bauingenieur24. de. Version: 2006. http://www.bauingenieur24.de/fachbeitraege/software/1777.htm. 2006. – Forschungsbericht. – Online-Fachbeitrag

[Choi u. a. 2002] CHOI, Guk-Heon ; MUN, Duhwan ; HAN, Soonhung: Exchange of CAD Part Models Based on the Macro-Parametric Approach. In: *International Journal of CAD/CAM* 2 (2002), S. 13–21

[Claus u. Schwill 2001] CLAUS, Volker ; SCHWILL, Andreas ; LEXIKONREDAKTION, Meyers (Hrsg.): *Duden Informatik*. 3. Auflage. Mannheim u. a. : Dudenverlag, 2001

[Cook u. a. 1989] COOK, Robert D. ; MALKUS, David S. ; PLESHA, Michael E.: *Concepts and Applications of Finite Element Analysis*. 3. Edition. New York : Wiley, 1989

[Corney u. Lim 2001] CORNEY, Jonathan ; LIM, Theodore: *3D Modeling with ACIS*. Kippen, Stirling : Saxe-Coburg Publications, 2001

[Crowley u. Watson 2000] CROWLEY, A. J. ; WATSON, A. S.: CIMsteel Integration Standards Release 2: Volume 1 – Overview / The Steel Contruction Institute, Berkshire, UK and School of Engineering, University of Leeds, UK. Version: 2000. http://www.cis2.org/documents/Vol1/SCI-P265on.pdf. 2000 (P265). – SCI Publication

[Dahlenburg 1996] DAHLENBURG, Anke: *Modellierung architektonischer Objekte in den frühen Entwurfsphasen mittels architekturspezifischer Fachsprache*, Hochschule für Architektur und Bauwesen Weimar, Diss., 1996

[Date 2000] DATE, C. J.: *An Introduction to Database Systems*. 7. Edition. Reading u. a. : Addison-Wesley, 2000

[DIN 1356-1 1995] *Bauzeichnungen, Teil 1: Arten, Inhalte und Grundregeln der Darstellung.* Deutsches Institut für Normung e. V., 1995

[DIN 6789-5 1995] *Dokumentationssystematik, Teil 5: Freigabe in der Technischen Produktdokumentation.* Deutsches Institut für Normung e. V., 1995

[Eastman 1999] EASTMAN, Charles M.: *Building product models.* Boca Raton u. a. : CRC Press, 1999

[Firmenich 2002] FIRMENICH, Berthold: *CAD im Bauplanungsprozess: Verteilte Bearbeitung einer strukturierten Menge von Objektversionen.* Aachen : Shaker Verlag, 2002 (Berichte aus dem Bauwesen). – Dissertation

[Firmenich 2004] FIRMENICH, Berthold: A Novel Modelling Approach for the Exchange of CAD Information in Civil Engineering. In: *Proceedings of the 5th European Conference on Product and Process Modelling in the Building and related Industries (ECPPM): eWork and eBusiness in Architecture, Engineering and Construction.* Leiden u. a. : A. A. Balkema Publishers, 2004

[Firmenich 2006] FIRMENICH, Berthold: *Grundlagenorientierter Systementwurf*, Bauhaus-Universität Weimar, Vorlesungsskript, 2006. http://gonzo.uni-weimar.de/~bauinf/cad/lehre/cib1/downloads/Skript/CIB1.pdf. – Stand: Mai 2006

[Firmenich u. a. 2005] FIRMENICH, Berthold ; KOCH, Christian ; RICHTER, Torsten ; BEER, Daniel G.: Versioning structured object sets using text based Version Control Systems. In: SCHERER, R. J. (Hrsg.) ; KATRANUSCHKOV, P. (Hrsg.) ; SCHAPKE, S.-E (Hrsg.): *Proceedings of the 22nd CIB-W78 Conference on Information Technology in Construction.* Dresden : Institute of Construction Informatics, 2005, S. 105–112

[Firmenich u. a. 2008] FIRMENICH, Berthold ; KOCH, Christian ; RICHTER, Torsten ; OLIVIER, Albertus H. ; BEER, Daniel G.: CADEMIA: A Platform for the Development of Civil Engineering Applications. In: *Proceedings of the 12th International Conference on Computing in Civil and Building Engineering (ICCCBE-XII)*, 2008. – angenommener Aufsatz

[Firmenich u. Rank 2007] FIRMENICH, Berthold ; RANK, Ernst: Überlick zum Themenbereich Verteilte Produktmodelle. In: RÜPPEL, Uwe (Hrsg.): *Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau.* Berlin, Heidelberg : Springer-Verlag, 2007, S. 121–132

[Flanagan 2003] FLANAGAN, David: *Java in a Nutshell.* Deutsche Ausgabe der 4. Auflage. Beijing u. a. : O'Reilly, 2003. – Dt. Übersetzung von Matthias Kalle Dalheimer und Harald Selke

[Flynt 1999] FLYNT, Clif: *Tcl/Tk for real programmers.* San Diego u. a. : Academic Press Professional, 1999

[Gamma u. a. 2002] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns: Elements of Reusable Objekt-Oriented Software*. 24. Printing. Boston u. a. : Addison-Wesley, 2002

[Gerold u. Koch 2007] GEROLD, Fabian ; KOCH, Christian: OpenSource im Bauwesen: Ingenieurgerechtes Pre- und Postprocessing in der Numerischen Simulation. In: MERKEL, A. (Hrsg.) ; SCHÜTZ, R. (Hrsg.) ; WIESSFLECKER, T. (Hrsg.): *Forum Bauinformatik 2007*. Graz : Verlag der Technische Universität Graz, 2007

[Goethe 1808] GOETHE, Johann Wolfgang v.: *Faust. Eine Tragödie. von Goethe.* Tübingen : J. G. Cotta'schen Buchhandlung, 1808

[Gulbins u. a. 1999] GULBINS, Jürgen ; SEYFRIED, Markus ; STRACK-ZIMMERMANN, Hans: *Dokumenten-Management: Vom Imaging zum Business-Dokument*. Berlin u. a. : Springer-Verlag, 1999

[Gumm u. Sommer 2002] GUMM, Heinz-Peter ; SOMMER, Manfred: *Einführung in die Informatik*. 5. Auflage. München, Wien : Oldenbourg Verlag, 2002

[Haas 1998] HAAS, Wolfgang: *What is STEP-CDS?* Haas+Partner GmbH, 1998. http://cic.vtt.fi/vera/Documents/1998.09.23_STEP-CDS.pdf. – Präsentation, Abruf:

[Hanff 2003] HANFF, Jochen: *Abhängigkeiten zwischen Objekten in ingenieurwissenschaftlichen Anwendungen*, Technische Universität Berlin, Diss., 2003

[Hardy 2000] HARDY, Vincent J.: *Java 2D API Graphics*. Upper Saddle River, NJ : Prentice Hall, 2000

[Harrison 1997] HARRISON, Mark: *TCL/ TK tools*. Cambridge u. a. : O'Reilly, 1997

[Hartmann 2000] HARTMANN, Dietrich (Hrsg.): *Objektorientierte Modellierung in Planung und Konstruktion: Forschungsbericht*. Weinheim : Wiley-VCH, 2000. – Deutsche Forschungsgemeinschaft

[Hartmann 2007] HARTMANN, Markus: *Wissensbasierte Modellierung vernetzt-kooperativer Gebäudeplanung unter Verwendung von Methoden der Fuzzy-Logik*, Universität Duisburg-Essen, Diss., 2007

[Horstmann u. Cornell 2002] HORSTMANN, Cay S. ; CORNELL, Gary: *Core Java 2*. Bd. 2: *Expertenwissen*. München : Markt+Technik Verlag, 2002

[Horstmann u. Cornell 2003] HORSTMANN, Cay S. ; CORNELL, Gary: *Core Java 2*. Bd. 1: *Grundlagen*. München : Markt+Technik Verlag, 2003. – Dt. Übersetzung von Frank Langenau

[Hunt u. McIlroy 1976] HUNT, J. W. ; MCILROY, M. D.: An Algorithm for Differential File Comparison / Bell Laboratories. Version: 1976. http://www.cs.dartmouth.edu/~doug/diff.ps. 1976 (41). – Forschungsbericht

[ISO 10303-111 2003] *Product data representation and exchange: Integrated application resource: Construction history features.* International Organisation for Standardization (ISO). http://www.tc184-sc4.org/SC4_Open/SC4_and_Working_Groups/WG12/N-DOCS/Files/WG12N2234_Construction_History_Features.doc. Version: 2003-09-08. – Abruf: 12.01.2007

[ISO 10303-112 2004] *Product data representation and exchange: Integrated application resource: Standard modelling commands for the procedural exchange of 2D CAD models.* International Organization for Standardization (ISO). http://www.tc184-sc4.org/SC4_Open/SC4_and_Working_Groups/WG12/N-DOCS/Files/WG12N2897_Part_112_CD.pdf. Version: 2004-10-29. – Abruf: 04.07.2006

[ISO 10303-55 2002] *Product data representation and exchange: Integrated generic resource: Procedural and hybrid representation.* International Organization for Standardization (ISO). http://www.tc184-sc4.org/SC4_Open/SC4_and_Working_Groups/WG12/N-DOCS/Files/wg12n1036.pdf. Version: 2002-11-30. – Abruf: 11.01.2007

[JavaCC] *The JavaCC Tutorial.* : *The JavaCC Tutorial*, http://www.engr.mun.ca/~theo/JavaCC-Tutorial/. – Abruf: 08.11.2004

[Katz 1990] KATZ, Randy H.: Toward a unified framework for version modeling in engineering databases. In: *ACM Computing Surveys* 22(4) (1990), S. 375–409

[Kiviniemi u. a. 2005] KIVINIEMI, A. ; FISCHER, M. ; BAZJANAC, V.: Integration of Multiple Product Models: IFC Model Servers as a Potential Solution. In: SCHERER, Raimar J. (Hrsg.) ; KATRANUSCHKOV, Peter (Hrsg.) ; SCHAPKE, Sven-Eric (Hrsg.): *Proceedings of the 22nd CIB-W78 Conference on Information Technology in Construction.* Dresden : Institute for Construction Informatics, TU Dresden, 2005 (CIB Publication 304), S. 37–40

[Kiviniemi u. a. 2008] KIVINIEMI, Arto ; TARANDI, Väino ; KARLSHØJ, Jan ; BELL, Håvard ; KARUD, Ole J.: Review of the Development and Implementation of IFC compatible BIM / Erabuild. Version: 2008. http://www.senternovem.nl/mmfiles/Erabuild%20BIM%20Final%20Report%20January%202008_tcm24-253611.pdf. 2008. – Forschungsbericht

[Klingelhöller 2001] KLINGELHÖLLER, Harald: *Dokumentenmanagementsysteme: Handbuch zur Einführung.* Berlin u. a. : Springer-Verlag, 2001

[König u. a. 2006] KÖNIG, Markus ; BEUCKE, Karl ; TAUSCHER, Eike: Management and evaluation of alternative construction tasks. In: *Proceedings of the 11th International Conference on Computing in Civil and Building Engineering (ICCCBE-XI).* Montreal : Université du Québec, 2006, S. 2247–2254

[Laabs 1998] LAABS, Andreas: *Methoden für die Modellierung mit Objekten im Bauingenieurwesen*, Technische Universität Berlin, Diss., 1998

[Liebich 2004] LIEBICH, Thomas: *IFC 2x Edition 2 – Model Implementation Guide*. International Alliance for Interoperability, 2004. http://www.iai-international.org/model/files/20040318_ifc2x_modelimplguide_v1-7.pdf. – Abruf: 27.04.2005

[Mazzoni u. a. 2006] MAZZONI, Silvia ; MCKENNA, Frank ; SCOTT, Michael H. ; AL., Gregory L. F.: *Open System for Earthquake Engineering Simulation: User Command-Language Manual*. Berkeley: Pacific Earthquake Engineering Research Center, University of California, 2006. http://opensees.berkeley.edu/OpenSees/manuals/usermanual/index.html. – Abruf: 17.07.2007

[Mun u. a. 2003] MUN, D. ; HAN, S. ; KIM, J. ; OH, Y.: A set of standard modeling commands for the history-based parametric approach. In: *Computer-Aided Design* 35 (2003), S. 1171–1179

[Norman 2008] *„Viele Leute brauchen gar keinen Computer“*. SPIEGEL ONLINE. http://www.spiegel.de/netzwelt/web/0,1518,536340,00.html. Version: 2008. – Interview mit dem IT-Visionär Don Norman, Abruf: 25.02.2008

[Nour u. a. 2006] NOUR, M. M. ; FIRMENICH, Berthold ; RICHTER, Torsten ; KOCH, Christian: A versioned IFC database for multi-disciplinary synchronous cooperation. In: *Digital Proceedings of the 11th International Conference on Computing in Civil and Building Engineering (ICCCBE-XI)*. Montreal : Université du Québec, 2006, S. 636–645

[OMG 2003] OMG: *OML Unified Modeling Language Specification*. http://www.omg.org, 2003. http://www.omg.org/docs/formal/03-03-01.pdf. – Abruf: 11.12.2007

[Ousterhout 1995] OUSTERHOUT, John K.: *Tcl und Tk*. 1. Edition. Bonn u. a. : Addison-Wesley, 1995

[Pahl u. Damrath 2000] PAHL, Peter J. ; DAMRATH, Rudolf: *Mathematische Grundlagen der Ingenieurinformatik*. Berlin u. a. : Springer-Verlag, 2000

[Pratt 2007] PRATT, M.: STEP approaches to LTKR. In: *Digital Proceedings of the Atlantic Workshop on Long Term Knowledge Retention (LTKR)*, 2007. – Presentation: http://www.ukoln.ac.uk/events/ltkr-2007/presentations/m-pratt.ppt

[Pratt 2003] PRATT, Michael J.: Procedural Modelling, Generative Geometry, and the International Standard ISO 10303 (STEP). In: *Proceedings of the 10th IMA International Conference*. Berlin, Heidelberg : Springer-Verlag, 2003

[Pratt u. a. 2005] PRATT, Michael J. ; ANDERSON, Bill D. ; RANGER, Tony: Towards the standardized exchange of parameterized feature-based CAD models. In: *Computer-Aided Design* 37 (2005), S. 1251–1265

[Pratt u. Kim 2006] PRATT, Michael J. ; KIM, Junhwan: Experience in the exchange of procedural shape models using ISO 10303 (STEP). In: *SPM '06: Proceedings of the 2006 ACM symposium on Solid and physical modeling*. New York : ACM, 2006, 229–238

[Ramunno 2005] RAMUNNO, Enia: *Vergleich und Zusammenführung unterschiedlicher Planungsstände*. http://gonzo.uni-weimar.de/~bauinf/Publikationen/BA/BARamunno.pdf. Version: 2005. – Bachelorarbeit an der Bauhaus-Universität Weimar

[Richter u. Beucke 2006] RICHTER, Torsten ; BEUCKE, Karl: Diff and Merge for Net-distributed Applications in Civil Engineering. In: *Proceedings of the 11th International Conference on Computing in Civil and Building Engineering (ICCCBE-XI)*. Montreal : Université du Québec, 2006, S. 3716–3725

[Rüppel 2007a] RÜPPEL, Uwe: Grundlegende Betrachtungen zur vernetzten Kooperation. In: RÜPPEL, Uwe (Hrsg.): *Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau*. Berlin, Heidelberg : Springer-Verlag, 2007, S. 3–17

[Rüppel 2007b] RÜPPEL, Uwe (Hrsg.): *Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau*. Berlin, Heidelberg : Springer-Verlag, 2007

[Rudolph u. a. 1993] RUDOLPH, Dietmar ; STÜRZNICKEL, Thomas ; WEISSENBERGER, Leo ; RUDOLPH, Dietmar (Hrsg.): *Der DXF-Standard*. München : Rossipaul Verlag, 1993. – Herausgegeben in Zusammenarbeit mit AutoDesk

[Rumbaugh u. a. 2001] RUMBAUGH, J. ; JACOBSON, I. ; BOOCH, G.: *The Unified Modeling Language Reference Manual*. Bosten u. a. : Addison-Wesley, 2001

[Sapir 1921] SAPIR, Edward: *Language: An Introduction to the Study of Speech*. New York: Harcourt, Brace and company, 1921

[Schäfer 2006] SCHÄFER, Sascha: *Vergleichen und Zusammenführen von objektorientierten Ingenieurmodellen*. http://gonzo.uni-weimar.de/~bauinf/Publikationen/DA/Schaefer/DASchaefer.pdf. Version: 2006. – Diplomarbeit an der Bauhaus-Universität Weimar

[Schneider 2002] SCHNEIDER, Klaus-Jürgen (Hrsg.): *Bautabellen für Ingenieure*. 15. Auflage. Düsseldorf : Werner Verlag, 2002

[Weise 2006] WEISE, Matthias: *Ein Ansatz zur Abbildung von Änderungen in der modell-basierten Objektplanung*. Institut für Bauinformatik der TU Dresden, 2006 (Berichte des Institutes für Bauinformatik Heft 4). – Dissertation

[Wiegleb 1777] WIEGLEB, Johann C.: *Historisch-kritische Untersuchung der Alchemie oder der eingebildeten Goldmacherkunst; von ihrem Ursprunge sowohl Fortgange, und was nun von ihr zu halten sey*. Weimar, 1777

[Willenbacher 2002] WILLENBACHER, Heiko: *Interaktive verknüupfungsbasierte Bauwerksmodellierung als Integrationsplattform für den Bauwerkslebenszyklus*, Bauhaus-Universität Weimar, Diss., 2002. http://e-pub.uni-weimar.de/volltexte/2004/32/pdf/Willenbacher.pdf

[Wissel 1989] WISSEL, Christian: *Theoretische Ökologie*. Heidelberg : Springer-Verlag, 1989

[Zeller 1997] ZELLER, Andreas: *Configuration Management with Version Sets*, Technische Universität Braunschweig, Diss., 1997

A Grammatiken in Backus-Naur-Form

A.1 OPL für operative Modellierprogramme

```
// Terminals
< EOS           : (";")+ >
< COMMENT       : "//" (~["\n","\r"])* ("\n"|"\r"|"\r\n") >
< ARGSEP        : "," >
< PLUS          : "+" >
< MINUS         : "-" >
< TIMES         : "*" >
< DIVIDEDBY     : "/" >
< EXP           : "**" >
< EQUAL         : "==" >
< NOTEQUAL      : "!=" >
< AND           : "&&" >
< OR            : "||" >
< LESS          : "<" >
< GREAT         : ">" >
< ASSIGN        : "=" >
< TRUE          : "true" >
< FALSE         : "false" >
< NULL          : "null">
< IF            : "if" >
< ELSE          : "else">
< ENDIF         : "endif">
< WHILE         : "while" >
< ENDWHILE      : "endwhile">
< IMPORT        : "import">
< PROC          : "proc" >
< ENDPROC       : "endproc">
< CALL          : "call" >
< PRINT         : "print" >
< CONST         : "const" >
< RETURN        : "return" >
< LISTADD       : "listadd" >
< LISTREMOVE    : "listremove" >
< LISTSIZE      : "listsize" >
< LISTCONTAINS : "listcontains" >
< CONCAT        : "concat" >
```

```
< DIALOG        : "dialog" >
< DIALOGADD     : "dialogadd" >
< DIALOGSHOW    : "dialogshow" >
< BUTTON        : "button" >
< BUTTONACTION : "buttonaction" >
< LABEL         : "label" >
< TEXT          : "text" >
< TEXTGET       : "textget" >
< CHECKBOX      : "checkbox" >
< CHECKGET      : "checkoget" >
< LISTBOX       : "listbox" >
< LISTGET       : "listget" >
< TOP           : "TOP" >
< BOTTOM        : "BOTTOM" >
< CENTER        : "CENTER" >
< LEFT          : "LEFT" >
< RIGHT         : "RIGHT" >

< INTEGER       : "0" | ( ["1"-"9"] ( ["0"-"9"] )* ) >

< EXPONENT      : ["e","E"] ( ["+","-"] )? ( ["0"-"9"] )+ >

< DOUBLE        : ( "0" | ["1"-"9"] ( ["0"-"9"] )* )?
                  "." ( ["0"-"9"] )* ( <EXPONENT> )? >

< STRING      : "\""
                ( ~["\n","\t","\b","\r","\f","\\","\'","\""]
                | ("\\"
                    ( ["n","t","b","r","f","\\","'","\""]
                      | ["0"-"7"] ( ["0"-"7"] )?
                      | ["0"-"3"] ["0"-"7"] ["0"-"7"]
                    )
                  ) )* "\"" >

< LETTER      : ["a"-"z","A"-"Z"] >

< ID          : ("_")* <LETTER>
                ( ~["\"","\\","\n"," ","\t","\r",",",";",
                   "(",")","{","}","[","]",".","<",">",
                   "$","-","=","+","*","/","#"] )* >

< BOOLEAN     : <TRUE> | <FALSE> >

< POID        : "<" <LETTER>
                ( ~["\"","\\","\n"," ","\t","\r",",
                    ",",";","(",")","{","}","[","]",
                    ".","<",">","$","="]
                | ("\\"
```

```
85                    ( ["n","t","b","r","f","\\","'","\"",":"]
                      | ["0"-"7"] ( ["0"-"7"] )?
87                    | ["0"-"3"] ["0"-"7"] ["0"-"7"]
                      )
89                  ) )* ">" >

91  // Non-terminals
    Start         ::= StmtList
93
    StmtList      ::= Statement ( <EOS> Statement )*
95
    Statement     ::= Assignment | Command
97
    Assignment    ::= ( <CONST> )? <ID> <ASSIGN>
99                    ( Command | Expression )

101 Command       ::= SysCommand | ModelOperation

103 SysCommand    ::= Print
                    | Concat
105                 | Return
                    | Choice
107                 | Loop
                    | ListAdd
109                 | ListRemove
                    | ListContains
111                 | ListSize
                    | Import
113                 | Proc
                    | ProcCall
115                 | Dialog
                    | DialogAdd
117                 | DialogShow
                    | Button
119                 | ButtonAction
                    | Label
121                 | Text
                    | TextGet
123                 | Checkbox
                    | CheckGet
125                 | Listbox
                    | ListGet
127
    Print         ::= <PRINT> Expression
129
    Concat        ::= <CONCAT> Expression <ARGSEP> Expression
131
    Return        ::= <RETURN> ( Expression )?
```

```
133
    Choice        ::= <IF> <EOS> BoolExpr <EOS> StmtList <EOS>
135                     <ELSE> <EOS> StmtList <ENDIF>

137 Loop          ::= <WHILE> <EOS> BoolExpr <EOS>
                      StmtList <ENDWHILE>
139
    ListAdd       ::= <LISTADD> Value <ARGSEP> ArgList
141
    ListRemove    ::= <LISTREMOVE> Value <ARGSEP> ArgList
143
    ListContains  ::= <LISTCONTAINS> Value <ARGSEP> Expression
145
    ListSize      ::= <LISTSIZE> Value
147
    Import        ::= <IMPORT> <ID>
149
    Proc          ::= <PROC> <ID> IdList StmtList <ENDPROC>
151
    ProcCall      ::= <CALL> <ID> ( "." <ID> )? ( ArgList )?
153
    Dialog        ::= <DIALOG> <ID> <ARGSEP> Expression
155
    DialogAdd     ::= <DIALOGADD> Value <ARGSEP> Value <ARGSEP>
157                     ( <TOP> | <BOTTOM> | <CENTER> | <LEFT>
                        | <RIGHT> )
159
    DialogShow    ::= <DIALOGSHOW> Value
161
    Button        ::= <BUTTON> <ID> <ARGSEP> Expression <ARGSEP>
163                     Expression

165 ButtonAction  ::= <BUTTONACTION> Value <ARGSEP> Expression

167 Label         ::= <LABEL> <ID> <ARGSEP> Expression

169 Text          ::= <TEXT> <ID> <ARGSEP> <INTEGER>
                      ( <ARGSEP> Expression )?
171
    TextGet       ::= <TEXTGET> Value
173
    Checkbox      ::= <CHECKBOX> <ID> <ARGSEP> Expression
175
    CheckGet      ::= <CHECKGET> Value
177
    Listbox       ::= <LISTBOX> <ID> <ARGSEP> List
179
    ListGet       ::= <LISTGET> Value
```

```
181
   ModelOperation ::= <ID> ( ArgList )?
183
   ArgList       ::= Expression ( <ARGSEP> Expression )*
185
   IdList        ::= <ID> ( <ARGSEP> <ID> )*
187
   Boolean       ::= <TRUE> | <FALSE>
189
   Expression    ::= BoolExpr | NonBoolExpr
191
   BoolExpr      ::= Boolean
193                 |  "(" (<NOT>)? BoolExpr
                       ( (<AND> | <OR>) (<NOT>)? BoolExpr )? ")"
195                 |  "(" (<NOT>)? NonBoolExpr
                       ( (<EQUAL> | <NOTEQUAL> | <LESS> | <GREAT>)
197                    (<NOT>)? NonBoolExpr )? ")"

199 NonBoolExpr   ::= <ID>
                    |  <POID>
201                 |  <NULL>
                    |  <STRING>
203                 |  Value
                    |  List
205                 |  Sum

207 Value         ::= "$" <ID> ( "[" <INTEGER> "]" )?

209 List          ::= "[" Expression ( <ARGSEP> Expression )* "]"

211 Sum           ::= Prod ( (<PLUS> | <MINUS>) Prod )?

213 Prod          ::= Exp ( (<TIMES> | <DIVIDED_BY>) Exp )?

215 Exp           ::= Unary ( <EXP> Exp )*

217 Unary         ::= ( <PLUS> )? Number
                    |  <MINUS> Number
219
   Number        ::= <INTEGER>
221                 | <DOUBLE>
                    | Value
223                 | "(" Sum ")"
```

Listing A.1: Grammatik der operativen Modellierungssprache der OPL-Sprachebene

A.2 Auszug der OML-Zeichnungsoperationen

```
...
// Non-terminals
PoidPnt       ::= "[" <POID> <ARGSEP> <INTEGER> "]"

PoidList      ::= "[" <POID> ( <ARGSEP> <POID> )* "]"
                |  "[" PoidPnt ( <ARGSEP> PoidPnt )* "]"
...
ModelOperation ::= AddCommand
                 |  SelCommand
                 |  ModCommand
                 |  RemCommand
                 |  SetCommand
                 |  GetCommand
                 |  MiscCommand

AddCommand    ::= AddLine
                |  AddRect
                |  AddCircle
                |  AddArc
                |  AddEllipse
                |  AddEllipseArc
                |  AddPoly
                |  AddQuad
                |  AddCurve
                |  AddText
                |  AddDim

                |  AddCopyTranslate
                |  AddCopyRotate
                |  AddCopyMirror
                |  AddCopyScale

                |  AddLayer
                |  AddLineWidth
                |  AddLinePattern
                |  AddDrawPaint
                |  AddFillPaint
                |  AddFontFamily
                |  AddTextSize

                |  AddDimBinding
                ...

SelCommand    |  Select
                |  SelectPoints
```

```
46                  |  SelectAll
                    |  Unselect
48                  |  UnselectPoints
                    |  UnselectAll
50
ModCommand          |  ModTranslate
52                  |  ModRotate
                    |  ModMirror
54                  |  ModScale
                    |  ModTransform
56
                    |  ModFeature
58                  |  ModLayer
                    |  ModLineWidth
60                  |  ModLinePattern
                    |  ModDrawPaint
62                  |  ModFillPaint
                    |  ModFontFamily
64                  |  ModTextSize
                    ...
66
RemCommand          ::= Remove
68                  |  RemLayer
                    |  RemAttribute
70
SetCommand          ::= SetScale
72                  |  SetUserCoordSystem
                    |  SetNaturalUnits
74
                    |  SetDefLayer
76                  |  SetDefLineWidth
                    |  SetDefLinePattern
78                  |  SetDefDrawPaint
                    |  SetDefFillPaint
80                  |  SetDefFontFamily
                    |  SetDefTextSize
82                  ...

84 GetCommand       ::= GetPoint
                    |  GetInteger
86                  |  GetDouble
                    |  GetString
88                  |  GetSelected

90                  |  GetLayer
                    |  GetLayerAttributes
92                  |  GetLineWidth
                    |  GetLinePattern
```

```
94              |  GetDrawPaint
                |  GetFillPaint
96              |  GetFontFamily
                |  GetTextSize
98              |  GetFeature
                ...
100
MiscCommand  ::= Undo | Redo | Again
102
// *** ADD ***
104 // Hinzufügen einer Linie mit den Punkten P1=(x1,y1) und
// P2=(x2,y2)
106 AddLine ::= "addline" x1=Number <ARGSEP> y1=Number
              <ARGSEP> x2=Number <ARGSEP> y2=Number
108
// Hinzufügen eines Rechtecks mit den Punkten P1=(x1,y1)
110 // und P2=(x2,y2) auf einer Diagonalen
AddRect ::= "addrect" x1=Number <ARGSEP> y1=Number
112            <ARGSEP> x2=Number <ARGSEP> y2=Number

114 // Hinzufügen eines Kreises mit dem Mittelpunkt P=(x,y)
// und dem Radius r
116 AddCirle ::= "addcircle" x=Number <ARGSEP> y=Number
              <ARGSEP> r=Number
118 ...
// Hinzufügen eines Textes mit dem Einfügepunkt P=(x,y),
120 // dem Drehwinkel w und der Zeichenkette s
AddText ::= "addtext" x=Number <ARGSEP> y=Number <ARGSEP>
122            w=Number <ARGSEP> s=<STRING>

124 // Hinzufügen einer linearen Bemaßung durch den Punkt
// P=(x,y), mit dem Bemaßungswinkel w und den Bemaßungs-
126 // punkten P1=(x1,y1) und P2=(x2,y2)
AddDim ::= "adddim" x=Number <ARGSEP>
128           y=Number <ARGSEP> w=Number
              ( <ARGSEP> x1=Number <ARGSEP> y1=Number )?
130           ( <ARGSEP> x2=Number <ARGSEP> y2=Number )?

132 // Hinzufügen von Kopien der selektierten Komponenten auf
// Basis des Verschiebungsvektors v=(dy,dy)
134 AddCopyTranslate ::= "addcopytranslate" dx=Number <ARGSEP>
                      dy=Number
136
// Hinzufügen eines Layers mit der Semantik s nach Auswahl
138 // von Attibuten
AddLayer ::= "addlayer" s=<STRING>
140
```

```
142 // Hinzufügen einer Linienbreite lw mit Liniendicke d in [mm]
    AddLineWidth ::= "addlinewidth" d=Number <ARGSEP> lw=<STRING>
144
    // Hinzufügen eines Linienmusters lp mit
146 // Strich-Lücke-Liste l
    AddLinePattern ::= "addlinepattern" l=List <ARGSEP>
148                    lp=<STRING>

150 // Hinzufügen einer Linienfarbe dp mit RGB-Werten (r,g,b)
    AddDrawPaint ::= "adddrawpaint" r=<INTEGER> <ARGSEP>
152                  g=<INTEGER> <ARGSEP> b=<INTEGER>
                     <ARGSEP> dp=<STRING>
154
    // Hinzufügen einer Assoziativität zwischen selektierten
156 // Komponenten-Kontrollpunkten und selektierter Bemaßung
    AddDimBinding ::= "adddimbinding"
158 ...

160 // *** SELECT/UNSELECT ***
    // Selektieren von Komponenten
162 Select ::= "select" ( <POID> )?

164 // Selektieren von Komponenten-Kontrollpunkten mit der Maus
    SelectPoints ::= "selectpoints" ( PoidPnt )?
166
    // Alles selektieren
168 SelectAll ::= "selectall"

170 // Deselektieren von Komponenten
    Unselect ::= "unselect" ( <POID> )?
172
    // Deselektieren von Komponenten-Kontrollpunkten
174 UnselectPoints ::= "unselectpoints" ( PoidPnt )?

176 // Alles deselektieren
    UnselectAll ::= "unselectall"
178 ...

180 // *** MODIFY ***
    // Verschieben selektierter oder spezifizierter Komponenten
182 // oder -teile um Verschiebungsvektor v=(dx,dy)
    ModTranslate ::= "modtranslate" dx=Number <ARGSEP> dy=Number
184                  ( <ARGSEP> PoidList )?

186 // Rotieren selektierter oder spezifizierter Komponenten
    // oder -teile um Punkt P=(x,y) mit Winkel w
188 ModRotate ::= "modrotate" x=Number <ARGSEP> y=Number <ARGSEP>
                w=Number ( <ARGSEP> PoidList )?
```

```
190
// Spiegeln selektierter oder spezifizierter Komponenten oder
192 // -teile um Gerade mit Punkten P1=(x1,y1) und P2=(x2,y2)
ModMirror ::= "modmirror" x1=Number <ARGSEP> y1=Number
194              <ARGSEP> x2=Number <ARGSEP> y2=Number
                 ( <ARGSEP> PoidList )?
196
// Skalieren selektierter oder spezifizierter Komponenten
198 // oder -teile bzgl. Punkt P1=(x,y) um Faktoren sx und sy
ModScale ::= "modscale" x=Number <ARGSEP> y=Number
200             <ARGSEP> sx=Number <ARGSEP> sy=Number
                ( <ARGSEP> PoidList )?
202
// Transformieren selektierter oder spezifizierter
204 // Komponenten oder -teile auf Basis der Transformations-
// matrix T=(m00,m10,m01,m11,m02,m12)
206 ModTransform ::= "modtransform" m00=Number <ARGSEP>
                    m10=Number <ARGSEP> m01=Number <ARGSEP>
208                 m11=Number <ARGSEP> m02=Number <ARGSEP>
                    m12=Number ( <ARGSEP> PoidList )?
210
// Modifizieren der Eigenschaft f in den Wert v für
212 // selektierte oder spezifizierte Komponenten
ModFeature ::= "modfeature" f=<STRING> <ARGSEP> v=List
214               ( <ARGSEP> PoidList )?

216 // Modifizieren des Layers nach Auswahl für selektierte
// oder spezifizierte Komponenten
218 ModLayer ::= "modlayer" ( <ARGSEP> PoidList )?

220 // Modifizieren der Linienbreite in lw für selektierte
// oder spezifizierte Komponenten
222 ModLineWidth ::= "modlinewidth" lw=<STRING>
                    ( <ARGSEP> PoidList )?
224
// Modifizieren des Linienmusters in lp für selektierte
226 // oder spezifizierte Komponenten
ModLinePattern ::= "modlinepattern" lp=<STRING>
228                   ( <ARGSEP> PoidList )?
...
230
// *** REMOVE ***
232 // Löschen von selektierten oder spezifizierten Komponenten
Remove ::= "remove" ( PoidList )?
234
// Löschen von Layern
236 RemLayer ::= "remlayer" ( PoidList )?
...
```

```
238
// *** SET ***
240 // Setzen des aktuellen Zeichnungsmaßstabs s
SetScale ::= "setscale" s=Number
242
// Setzen des aktuellen Nutzerkoordinatensystems mit
244 // Rotationswinkel w und Verschiebungsvektor v=(dx,dy)
SetUserCoordSystem ::= "setucs" w=Number <ARGSEP>
246                        dx=Number <ARGSEP> dy=Number

248 // Setzen der natürlichen Zeichnungseinheit nu in [mm]
SetNaturalUnits ::= "setnu" nu=Number
250
// Setzen des aktuellen Layers nach Auswahl
252 SetDefLayer ::= "setdeflayer"

254 // Setzen der aktuellen Linienbreite lw
SetDefLineWidth ::= "setdeflinewidth" lw=<STRING>
256
// Setzen des aktuellen Linienmusters lp
258 SetDefLinePattern ::= "setdeflinepattern" lp=<STRING>
...
260


262 // *** GET ***
// Grafisches Abfragen eines Punktes
264 GetPoint ::= "getpoint"

266 // Abfragen einer Ganzzahl
GetInteger ::= "getint"
268
// Abfragen einer Gleitpunktzahl
270 GetDouble ::= "getdouble"

272 // Abfrage selektierter Komponenten
GetSelected ::= "getselected"
274
// Abfrage des Layers selektierter oder spezifizierter
   Komponenten
276 GetLayer ::= "getlayer" ( PoidList )?

278 // Abfrage von Attributen von Layern
GetLayerAttributes ::= "getlayerattributes" ( PoidList )?
280
// Abfrage nach Linienbreite selektierter oder spezifizierter
282 // Komponenten
GetLineWidth ::= "getlinewidth" ( PoidList )?
284
```

```
// Abfrage nach Wert der Eigenschaft f selektierter
// oder spezifizierter Komponenten
GetFeature ::= "getfeature" f=<STRING> ( <ARGSEP> PoidList )?
...

// *** MISC ***
// Rückgängigmachen von n Operationen
Undo ::= "undo" <INTEGER>

// Wiederherstellen von n rückgängig gemachten Operationen
Redo ::= "redo" <INTEGER>

// Wiederholen der letzten Operation
Again ::= "again"
```

Listing A.2: Grammatik für OML-Zeichnungsoperationen (Auszug)

A.3 Auszug der POML-Zeichnungsoperationen

```
...
// Non-terminals
PoidPnt       ::= "[" <POID> <ARGSEP> <INTEGER> "]"

PoidList      ::= "[" <POID> ( <ARGSEP> <POID> )* "]"
                |  "[" PoidPnt ( <ARGSEP> PoidPnt )* "]"
...
ModelOperation ::= AddCommand
                 |  SelCommand
                 |  ModCommand
                 |  RemCommand
                 |  SetCommand
                 |  GetCommand
                 |  MiscCommand

AddCommand    ::= AddLine
                |  AddRect
                |  AddCircle
                |  AddArc
                |  AddEllipse
                |  AddEllipseArc
                |  AddPoly
                |  AddQuad
                |  AddCurve
                |  AddText
                |  AddDim

                |  AddCopyTranslate
                |  AddCopyRotate
                |  AddCopyMirror
                |  AddCopyScale

                |  AddLayer
                |  AddLineWidth
                |  AddLinePattern
                |  AddDrawPaint
                |  AddFillPaint
                |  AddFontFamily
                |  AddTextSize

                |  AddDimBinding
                ...

ModCommand    |  ModTranslate
                |  ModRotate
```

```
46                |  ModMirror
                  |  ModScale
48                |  ModTransform

50                |  ModFeature
                  |  ModLayer
52                |  ModLineWidth
                  |  ModLinePattern
54                |  ModDrawPaint
                  |  ModFillPaint
56                |  ModFontFamily
                  |  ModTextSize
58                |  ...

60 RemCommand    ::= Remove
                  |  RemLayer
62                |  RemAttribute
                  |  ...
64
   SetCommand    ::= SetScale
66                |  SetUserCoordSystem
                  |  SetNaturalUnits
68
                  |  SetDefLayer
70                |  SetDefLineWidth
                  |  SetDefLinePattern
72                |  SetDefDrawPaint
                  |  SetDefFillPaint
74                |  SetDefFontFamily
                  |  SetDefTextSize
76                |  ...

78 // *** ADD ***
   // Hinzufügen einer Linie mit den Punkten P1=(x1,y1) und
80 // P2=(x2,y2) und der POID p
   AddLine ::= "addline" x1=Number <ARGSEP> y1=Number <ARGSEP>
82             x2=Number <ARGSEP> y2=Number <ARGSEP> p=<POID>

84 // Hinzufügen eines Rechtecks mit den Punkten P1=(x1,y1)
   // und P2=(x2,y2) auf einer Diagonalen und der POID p
86 AddRect ::= "addrect" x1=Number <ARGSEP> y1=Number <ARGSEP>
             x2=Number <ARGSEP> y2=Number <ARGSEP> p=<POID>
88
   // Hinzufügen eines Kreises mit dem Mittelpunkt P=(x,y),
90 // dem Radius r und der POID p
   AddCirle ::= "addcircle" x=Number <ARGSEP> y=Number
92              <ARGSEP> r=Number <ARGSEP> p=<POID>
```

```
94 // Hinzufügen eines Textes mit dem Einfügepunkt P=(x,y),
   // dem Drehwinkel w, der Zeichenkette s und der POID p
96 AddText ::= "addtext" x=Number <ARGSEP> y=Number <ARGSEP>
             w=Number <ARGSEP> s=<STRING> <ARGSEP> p=<POID>
98
   // Hinzufügen einer linearen Bemaßung durch den Punkt
100 // P=(x,y), mit dem Bemaßungswinkel w, den Bemaßungspunkten
   // P1=(x1,y1) und P2=(x2,y2) und der POID p
102 AddDim ::= "adddim" x=Number <ARGSEP> y=Number
            <ARGSEP> w=Number <ARGSEP> x1=Number
104         <ARGSEP> y1=Number <ARGSEP> x2=Number
            <ARGSEP> y2=Number <ARGSEP> p=<POID>
106
   // Hinzufügen von Kopien der Komponenten mit POIDs src auf
108 // Basis des Verschiebungsvektors v=(dy,dy) als neue
   // Komponenten mit POIDs dst
110 AddCopyTranslate ::= "addcopytranslate" dx=Number <ARGSEP>
                      dy=Number <ARGSEP> src=PoidList
112                   <ARGSEP> dst=PoidList

114 // Hinzufügen eines Layers mit der Semantik s,
   // der Linienbreite lw, des Linienmusters lp, der Linien-
116 // farbe dp, der Füllfarbe fp, der Fontfamilie ff, der
   // Texthöhe ts und der POID p
118 AddLayer ::= "addlayer" s=<STRING> <ARGSEP> lw=<STRING>
              <ARGSEP> lp=<STRING> <ARGSEP> dp=<STRING>
120           <ARGSEP> fp=<STRING> <ARGSEP> ff=<STRING>
              <ARGSEP> ts=<STRING> <ARGSEP> p=<POID>
122
   // Hinzufügen einer Linienbreite lw mit Liniendicke d in [mm]
124 AddLineWidth ::= "addlinewidth" d=Number <ARGSEP>
                  lw=<STRING>
126
   // Hinzufügen eines Linienmusters lp mit
128 // Strich-Lücke-Liste l
   AddLinePattern ::= "addlinepattern" l=List <ARGSEP>
130                  lp=<STRING>

132 // Hinzufügen einer Linienfarbe dp mit RGB-Werten (r,g,b)
   AddDrawPaint ::= "adddrawpaint" r=<INTEGER> <ARGSEP>
134                g=<INTEGER> <ARGSEP> b=<INTEGER> <ARGSEP>
                   dp=<STRING>
136
   // Hinzufügen einer Assoziativität mit POID a zwischen
138 // Komponenten-Kontrollpunkt-POIDs p1 und p2 und der
   // Bemaßung mit POID p
140 AddDimBinding ::= "adddimbinding" PoidPnt <ARGSEP> PoidPnt
                   <ARGSEP> p=<POID> <ARGSEP> a=<POID>
```

```
...

// *** MODIFY ***
// Verschieben von Komponenten mit POIDs p um
// Verschiebungsvektor v=(dx,dy)
ModTranslate ::= "modtranslate" dx=Number <ARGSEP>
                 dy=Number <ARGSEP> p=PoidList

// Rotieren von Komponenten mit POIDs p um Punkt P=(x,y)
// mit Winkel w
ModRotate ::= "modrotate" x=Number <ARGSEP> y=Number
              <ARGSEP> w=Number <ARGSEP> p=PoidList

// Spiegeln von Komponenten mit POIDs p um Gerade mit
// Punkten P1=(x1,y1) und P2=(x2,y2)
ModMirror ::= "modmirror" x1=Number <ARGSEP> y1=Number
              <ARGSEP> x2=Number <ARGSEP> y2=Number
              <ARGSEP> p=PoidList

// Skalieren von Komponenten mit POIDs p bzgl. Punkt
// P1=(x,y) um Faktoren sx und sy
ModScale ::= "modscale" x=Number <ARGSEP> y=Number
             <ARGSEP> sx=Number <ARGSEP> sy=Number
             <ARGSEP> p=PoidList

// Transformieren von Komponenten mit POIDs p auf Basis
// der Transformationsmatrix T=(m00,m10,m01,m11,m02,m12)
ModTransform ::= "modtransform" m00=Number <ARGSEP>
                 m10=Number <ARGSEP> m01=Number <ARGSEP>
                 m11=Number <ARGSEP> m02=Number <ARGSEP>
                 m12=Number <ARGSEP> p=PoidList

// Modifizieren der Eigenschaft f in den Wert v für
// die Komponenten mit POIDs p
ModFeature ::= "modfeature" f=<STRING> <ARGSEP> v=List
               <ARGSEP> p=PoidList

// Modifizieren des Layers zum Layer l von Komponenten
// mit POIDs p
ModLayer ::= "modlayer" l=<POID> <ARGSEP> p=PoidList

// Modifizieren der Linienbreite in lw von Komponenten
// mit POIDs p
ModLineWidth ::= "modlinewidth" lw=<STRING>
                 <ARGSEP> p=PoidList

// Modifizieren des Linienmusters in lp von Komponenten
// mit POIDs p
```

```
ModLinePattern ::= "modlinepattern" lp=<STRING>
                   <ARGSEP> p=PoidList
...

// *** REMOVE ***
// Löschen von Komponenten mit POIDs p
Remove ::= "remove" p=PoidList

// Löschen von Layern mit POIDs p
RemLayer ::= "remlayer" p=PoidList
...

// *** SET ***
// Setzen des aktuellen Zeichnungsmaßstabs s
SetScale ::= "setscale" s=Number

// Setzen des aktuellen Nutzerkoordinatensystems mit
// Rotationswinkel w und Verschiebungsvektor v=(dx,dy)
SetUserCoordSystem ::= "setucs" w=Number <ARGSEP> dx=Number
                       <ARGSEP> dy=Number

// Setzen der natürlichen Zeichnungseinheit nu in [mm]
SetNaturalUnits ::= "setnu" nu=Number

// Setzen des aktuellen Layers mit POID p
SetDefLayer ::= "setdeflayer" p=<POID>

// Setzen der aktuellen Linienbreite lw
SetDefLineWidth ::= "setdeflinewidth" lw=<STRING>

// Setzen des aktuellen Linienmusters lp
SetDefLinePattern ::= "setdeflinepattern" lp=<STRING>
...
```

Listing A.3: Grammatik für POML-Zeichnungsoperationen (Auszug)

B Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Bei der Auswahl und Auswertung von Material haben mir andere Personen weder entgeltlich noch unentgeltlich geholfen.

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Ich versichere ehrenwörtlich, dass ich nach bestem Wissen die reine Wahrheit gesagt und nichts verschwiegen habe.

Weimar, den 30. Juni 2008 Christian Koch

C Über den Autor

C.1 Lebenslauf

Persönliche Daten

Name:	Christian Koch
Geburtstag und -ort:	1. Mai 1979 in Anklam
Familienstand:	ledig

Ausbildung

09/1985 – 08/1991:	Polytechnische Oberschule Neverin
09/1991 – 06/1997:	Neues Friedländer Gymnasium; Abschluss: Allgemeine Hochschulreife
07/1997 – 05/1998:	Grundwehrdienst
10/1998 – 03/2004:	Studium des Bauingenieurwesens an der Bauhaus-Universität Weimar, Studienrichtung Bauinformatik; Abschluss: Diplom-Ingenieur (Dipl.-Ing.); Hochschulpreis für die Diplomarbeit „Abhängigkeiten im CAD am Beispiel der Baubemaßung“

Beruflicher Werdegang

11/2002 – 05/2003:	Studentische Hilfskraft an der Professur *Informations- und Wissensverarbeitung* (Prof. Hübler) der Bauhaus-Universität Weimar
05/2004 – 07/2004:	Wissenschaftliche Hilfskraft an der Juniorprofessur *CAD in der Bauinformatik* (Prof. Firmenich)
seit 08/2004:	Wissenschaftlicher Mitarbeiter an der Juniorprofessur *CAD in der Bauinformatik* (Prof. Firmenich) – angebunden an die Professur *Informatik im Bauwesen* (Prof. Beucke); Bearbeitung des DFG-Forschungsprojekts „Eine formale Sprache für operative CAD-Modelle im Bauwesen“ und des DFG-Transferprojekts „Operatives Modellieren im Bauwesen“

C.2 Publikationen

[Koch 2004] Koch, Christian: Grundlagen und Umsetzung strukturierter Objektmengen im CAD. In: Zimmermann, J. (Hrsg.) ; Geller, S. (Hrsg.): *Forum Bauinformatik 2004*. Aachen : Shaker-Verlag, 2004 (Reihe Bauinformatik 2004), S. 161–168

[Koch 2005] Koch, Christian: Standardisierung von CAD durch eine Sprache für operative CAD-Modelle. In: Schley, F. (Hrsg.) ; Weber, L. (Hrsg.): *Forum Bauinformatik 2005*. Cottbus : Brandenburgische Technische Universität Cottbus, 2005, S. 26–33

[Firmenich u. a. 2005] Firmenich, Berthold ; Koch, Christian ; Richter, Torsten ; Beer, Daniel G.: Versioning structured object sets using text based Version Control Systems. In: Scherer, R. J. (Hrsg.) ; Katranuschkov, P. (Hrsg.) ; Schapke, S.-E. (Hrsg.): *Proceedings of the 22nd CIB-W78 Conference on Information Technology in Construction*. Dresden : Institute of Construction Informatics, 2005, S. 105–112

[Koch u. Firmenich 2006a] Koch, Christian ; Firmenich, Berthold: A Novel Diff and Merge Approach on the Basis of Operative Models. In: *Digital Proceedings of the 11th International Conference on Computing in Civil and Building Engineering (ICCCBE-XI)*. Montreal : Université du Québec, 2006

[Nour u. a. 2006] Nour, M. M. ; Firmenich, Berthold ; Richter, Torsten ; Koch, Christian: A versioned IFC database for multi-disciplinary synchronous cooperation. In: *Digital Proceedings of the 11th International Conference on Computing in Civil and Building Engineering (ICCCBE-XI)*. Montreal : Université du Québec, 2006

[Koch u. Firmenich 2006b] Koch, Christian ; Firmenich, Berthold: Operative Models for the Introduction of Additional Semantics into the Cooperative Planning Process. In: Smith, Ian F. C. (Hrsg.): *Intelligent Computing in Engineering and Architecture*. Berlin u. a. : Springer-Verlag, 2006, S. 376–382. – 13th EG-ICE Workshop 2006 (Revised Selected Paper)

[Koch u. a. 2006] Koch, Christian (Hrsg.) ; Richter, Torsten (Hrsg.) ; Tauscher, Eike (Hrsg.): *Forum Bauinformatik 2006*. Verlag der Bauhaus-Universität Weimar, 2006

[Gerold u. Koch 2007] Gerold, Fabian ; Koch, Christian: OpenSource im Bauwesen: Ingenieurgerechtes Pre- und Postprocessing in der Numerischen Simulation. In: Merkel, A. (Hrsg.) ; Schütz, R. (Hrsg.) ; Wiessflecker, T. (Hrsg.): *Forum Bauinformatik 2007*. Graz : Verlag der Technische Universität Graz, 2007

[Koch u. Firmenich 2008] Koch, Christian ; Firmenich, Berthold: Processing-oriented modeling in computer-aided building design. In: *Proceedings of the 12th International Conference on Computing in Civil and Building Engineering (ICCCBE-XII).* Beijing : Tsinghua University, 2008

[Firmenich u. a. 2008] Firmenich, Berthold ; Koch, Christian ; Richter, Torsten ; Olivier, Albertus H. ; Beer, Daniel G.: CADEMIA: A Platform for the Development of Civil Engineering Applications. In: *Proceedings of the 12th International Conference on Computing in Civil and Building Engineering (ICCCBE-XII).* Beijing : Tsinghua University, 2008

Alles wird gut.