Konzepte zur interaktiven Entwurfsraum-Exploration im Tragwerksentwurf

Dissertation

Zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

an der Fakultät Bauingenieurwesen

der Bauhaus-Universität Weimar

vorgelegt von

Fabian Gerold, M.Eng.

geboren am 11. Mai 1978 in Kressbronn am Bodensee

Mentor: Prof. Dr.-Ing. Karl Beucke

Gutachter:

Prof. Dr. Guido Morgenthal, Bauhaus-Universität Weimar

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

Tag der Disputation: 21. August 2013

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter an der Professur Informatik im Bauwesen an der Bauhaus-Universität Weimar. Dort war ich in der Lehre und Forschung tätig und konnte so viele unterschiedliche Aspekte der Bauinformatik kennenlernen.

Ganz herzlich möchte ich mich bei meinem Mentor Prof. Karl Beucke bedanken für die Betreuung der Arbeit und viele gute Gespräche während der letzten Jahre. Er gewährte mir die Freiheit, verschiedene Ansätze und Konzepte auszuprobieren, die im Wesentlichen in dieser Arbeit dargestellt sind. Durch seine vielen guten Kontakte, u. A. nach San Diego, wurde mir so ein interessanter Forschungsaufenthalt an der University of California San Diego ermöglicht. An dieser Stelle möchte ich auch dem DAAD für das Stipendium zu diesem Forschungsaufenthalt danken.

Für die freundliche Arbeitsatmosphäre, Unterstützung und ein stets offenes Ohr möchte ich mich bei allen Kolleginnen und Kollegen des Lehrstuhls bedanken.

Weimar im Februar 2013

Fabian Gerold

Kurzfassung

Numerische (computergestützte) Simulationen auf Basis der Finite-Elemente-Methode sind heute ein zentraler Bestandteil der Planung von Tragwerken aller Art.

Als in den 1970er und 1980er Jahren numerische Simulationen erstmals eine weite Verbreitung fanden, war es üblich, ein Berechnungsmodell in einem sequentiellen Arbeitsablauf zu erstellen. Die Eingabe, Berechnung und Ergebnisauswertung erfolgte dabei üblicherweise mit unterschiedlichen Programmen. Außerdem waren die Rechenzeiten erheblich länger als heute, und so wurden meist nur eine oder wenige Berechnungen des Modells vorgenommen.

Mehrere Faktoren führen dazu, dass heute oft viele Revisionen eines Simulationsmodells durchlaufen werden: Zum einen ergeben sich oft Planungsänderungen, zum anderen ist die Untersuchung von Planungsalternativen und die Suche nach einem Optimum wünschenswert.

In der vorliegenden Arbeit werden Ansätze beschrieben, die einen interaktiven Revisionszyklus einer numerischen Simulation ermöglichen. Erreicht wird dies durch eine bidirektionale Schnittstelle zwischen einer Anwendung zur Tragwerksmodellierung und einem Rechenserver. Die Schnittstelle umfasst Modelleingabe und Berechnungsergebnisse, sodass keine zusätzlichen Schnittstellen zu Dateien oder Datenbank-Systemen nötig sind. Änderungen im Modell können inkrementell auf das persistente Analysemodell des Rechenservers angewendet werden, was die Re-Analyse im Vergleich zu einer traditionellen Eingabedatei beschleunigt. So ist es möglich, den Entwurfsraum eines Tragwerks, der aus allen freien Parametern eines Modells aufgespannt wird, interaktiv zu durchsuchen, um zu einer möglichst guten Lösung zu gelangen.

Ein weiterer zentraler Aspekt in diesem Kontext der Entwurfsraum-Exploration ist die Möglichkeit, möglichst reibungslos und früh im Planungsprozess mit anderen Fachdisziplinen, insbesondere mit Architekten, kommunizieren zu können. Wichtigste Kommunikationsgrundlage im Bauwesen sind technische Dokumente (Pläne) eines Gebäudes. Diese haben sich in den letzten Jahren von der 2D-Zeichnung hin zu einem digitalen Modell entwickelt, das alle relevanten Informationen eines Gebäudes umfasst. Die wichtigste Kategorie von Informationen ist nach wie vor die Geometrie der Bauteile. Jedoch gibt es noch viele weitere Informationen, wie Material, Kopplungsbedingungen, räumliche Zuordnungen, Kosten etc.

In den letzten Jahren findet ein offener Standard für ein Bauwerks-Informations-Modell unter der Bezeichnung *Industry Foundation Classes* (IFC) zunehmend Verbreitung und Akzeptanz.

Die Kombination aus interaktiver numerischer Simulation und Interoperabilität durch die Anwendung von Konzepten zur Bauwerks-Informations-Modellierung im Tragwerksentwurf ist Ziel dieser Dissertation. Die Beschreibung der Konzeption und prototypischen Umsetzung ist Gegenstand der vorliegenden schriftlichen Arbeit.

Abstract

Today, numerical (computer-aided) simulations based on the finite element method are essential for the planning of all types of structures.

When in the 1970s and 1980s, numerical simulations for the first time came into widespread use, an analysis model was usually set up and calculated in a sequential workflow. The input, calculation and evaluation of results was done often with different programs. Furthermore, the computing times were much longer than today, and so only one or a few model analysis cycles were performed.

Several factors lead to a situation, where many revisions of a simulation model are analyzed today: First, there are often changes to the design by other parties. Second, the study of design alternatives and the search for an optimal solution is desirable.

In the dissertation at hand, an approach is described which allows an interactive revision cycle using numerical simulation. This is achieved with a bidirectional interface between an application for model input, and a computing server. The interface handles model input and calculation results, no separate database queries are needed. Design changes can be applied incrementally to the persistent analysis model of the computing server, which accelerates the re-analysis in comparison to a traditional input file. So it is possible to interactively explore the design space of a structure, that is spanned by all its free parameters.

Another important aspect in this context of design space exploration is the ability to smoothly communicate with other disciplines, especially with architects. The most important communication basis in the design process is the building plan, which has evolved in recent years from 2D drawings to a digital model that includes all the relevant information of a building. The most important category of information is still the geometry of building parts. However, there are many more details, such as materials, coupling conditions, spatial assignments, costs, etc.

In recent years, an open standard for such a building information model under the name *Industry Foundation Classes* (IFC) is increasingly used in research an industry.

The combination of interactive numerical simulation and interoperability through applying concepts of building information modeling in structural design is the goal of this thesis. The description of the design and prototype implementation is the subject of this written work.

Inhaltsverzeichnis

Abbildungsverzeichnis:

1 Einleitung

> "Scientists discover the world that exists; engineers create the world that never was." Theodore von Karmann

1.1 Themenbereich

Der Kern dieser Arbeit behandelt die Entwurfsraumsuche durch eine rechnergestützte Simulation von Tragwerken. Eine Definition des Wortes *Simulation* lässt erahnen, welches Potential in der Methode steckt:

> „Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.“ VDI-Richtlinie [VDI 3633 1992]

Es muss sich demnach nicht zwangsläufig um ein Computermodell handeln. In dieser Arbeit wird jedoch ausschließlich auf diesen Teilaspekt eingegangen, da er vermutlich das größte Entwicklungspotential besitzt.

Der Ausgangspunkt jedes Planungsprozesses ist ein mentales Modell, eine vage oder konkrete Vorstellung eines Bauwerkes und seiner Eigenschaften. Dieses wird meistens auf Papier oder auch direkt in ein Computermodell übertragen, wobei die heutigen Schnittstellen zwischen Mensch und Maschine vermutlich noch eine erhebliche Entwicklung vor sich haben.

Der Entwurf und die Simulation von Tragwerken finden bereits heute hauptsächlich rechnergestützt statt und ab einer gewissen Komplexität kann dies auch gar nicht anders bewältigt werden. Computergestützte Modelle haben u. a. die faszinierende Eigenschaft, dass enorm aufwendige Untersuchungen und Operationen auf diesem Modell in einer Programmiersprache einmal definiert, und dann ohne weitere Mühe beliebig oft ausgeführt werden können. Selbst wenn Fehler festgestellt werden, können diese praktisch immer durch punktuelle Änderungen in der Software dauerhaft behoben werden, während Berechnungen durch Menschen immer wieder von denselben Fehlern betroffen sind.

Eine Modellierung in der Denkweise der Objektorientierung ist sehr intuitiv, durch die Analogie von Eigenschaften und Verhalten realer Objekte und Modell-Objekte (Instanzen aus dem Datenmodell, Abbildung 1).

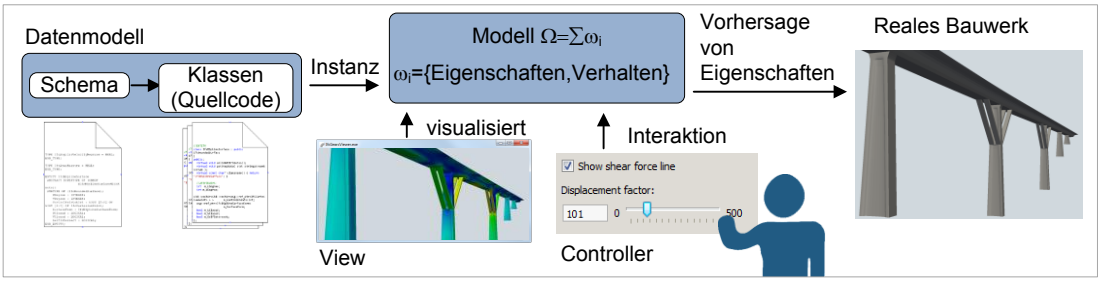


Abbildung 1: Definition eines Computermodells durch ein Datenmodell

Zwischen den Objekten ω_i des Modells Ω bestehen Referenzen, das Modell ist also ein *System*, und die Menge Ω ist eine strukturierte Menge. Eigenschaften (Attribute) und Verhalten

(Methoden) realer Objekte, wie auch Referenzen zwischen Objekten, lassen sich in einem objektorientierten Datenmodell abbilden.

Experimentierfähig muss ein Simulationsmodell laut o. g. Definition sein. Das Wort *Experiment* kommt aus dem Lateinischen und bedeutet „Versuch, Beweis, Prüfung“. Genau das soll mit Tragwerksmodellen möglichst umfassend getan werden. Der Begriff „Experiment am numerischen Modell“ wird in dieser Arbeit so interpretiert, dass die Sicherheit eines Tragwerks durch numerische Simulation verifiziert werden soll, aber auch, dass durch *Interaktion* mit dem Modell unterschiedliche Varianten und Alternativen generiert und ausgewertet werden. Durch Interaktion (Controller in Abbildung 1) können die Objekte und deren Eigenschaften verändert werden. Der Definitionsbereich der Attribute aller Objekte ω_i bildet dafür den Entwurfsraum.

Erkenntnisse aus Simulationen sollen „auf die Wirklichkeit übertragbar“ sein. Im Bereich der Strukturanalyse muss demnach die Modellierung der Geometrie, des Materials, der Lasten und der Lagerungsbedingungen zu möglichst realistischen und zuverlässigen Ergebnissen führen. Dies kann mit Hilfe einer Validierung der Ergebnisse einer numerischen Simulation sichergestellt werden.

Um eine sinnvolle Interaktion mit dem Modell zu erreichen, müssen die Auswirkungen der Veränderungen möglichst unmittelbar sichtbar gemacht werden. Dies erscheint offensichtlich, und ist auch in der grafischen Modellierung selbstverständlich. So waren bereits die ersten CAD-Systeme (*Computer Aided Design*) mit einer Visualisierung ausgestattet.

In der numerischen Simulation ist es jedoch bis heute selbstverständlich, dass keine unmittelbare Visualisierung der Eingabe und Interaktion stattfindet. Gründe dafür werden in Kapitel 2 genannt. Jedoch gibt es Forschungsansätze unter der Bezeichnung „Computational Steering“, die diese Problemstellung zu lösen versuchen. Eine Definition und Übersicht dazu findet sich in [Mulder 1999].

Da eine unmittelbare Rückkopplung auch der numerischen Simulation eigentlich eine Selbstverständlichkeit sein sollte (und damit in der Zukunft wohl auch keiner gesonderten Bezeichnung bedarf), wird auf diesen Aspekt als ein zentraler Bestandteil der Entwurfsraumsuche in den folgenden Abschnitten noch näher eingegangen.

1.2 Ziel der Arbeit

Das Konzept des **interaktiven Tragwerksentwurfs** soll durch ein möglichst direktes, schnelles und intuitives Feedback das Verständnis für das Tragverhalten von Strukturen verbessern. Somit besteht die Chance, nicht nur Tragwerke in ihrer Qualität (Robustheit, Ästhetik, Wirtschaftlichkeit) zu verbessern, sondern auch den Entwurfsprozess zu beschleunigen.

Am Planungsprozess eines Bauwerkes sind viele verschiedene Disziplinen beteiligt (Abbildung 2). Gegenwärtig werden in den einzelnen Disziplinen jeweils eigene Modelle genutzt, die größtenteils inkompatibel sind, und somit eine manuelle Konvertierung und Neumodellierung erfordern. Insbesondere zwischen Architekten und Tragwerksplanern ist die Zusammenarbeit intensiv. Die wichtigste Kommunikationsgrundlage ist das geometrische Modell, inkompatible Modelle behindern hier eine gute Zusammenarbeit.

Die **Interoperabilität**[1] der Modelle und der Software verbessert unmittelbar auch die Zusammenarbeit zwischen den Disziplinen des architektonischen Entwurfs und der

[1] Definition und weitere Informationen dazu in Abschnitt 2.7.3

Tragwerksplanung, sie führt also nicht nur zu einer Zeit- und Kostenersparnis, sondern mit hoher Wahrscheinlichkeit auch zu einem besseren Resultat.

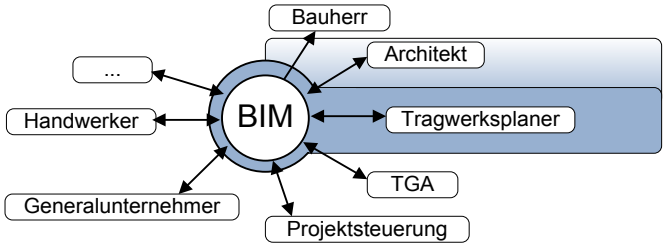


Abbildung 2: Fokus der Arbeit (dunkel hinterlegt)

Interoperabilität umfasst verschiedene Ebenen, vom physischen Format der Daten, über die Geometrie, Topologie[2], zu vielfältigen Relationen und anderen Informationen. Aus verschiedenen Gründen, die im Folgenden noch eine Rolle spielen, sind offene Standards in der Modellierung vorteilhaft, und somit soll auch diese Arbeit auf offenen Standards basieren.

In dieser Arbeit soll ein Konzept zur Anbindung eines Simulationsframeworks an eine Entwurfsanwendung entwickelt werden. Dieses Konzept soll auf offenen Standards basieren, plattformunabhängig, und möglichst auch anwendungsunabhängig sein, sodass es nicht nur für genau eine Anwendung nutzbar ist.

1.3 Aufbau der Arbeit

Kapitel 1: Einleitung

Hier erfolgt eine Darstellung der Themenbereiche der interaktiven numerischen Simulation und Interoperabilität.

Kapitel 2: Stand der Forschung und Technik

Ausgehend von einem geschichtlichen Überblick werden vorhandene Forschungsansätze und Anknüpfungspunkte für die eigene Arbeit dargestellt.

Kapitel 3: Problemstellung und Lösungsansatz

Ausgehend von den Abschnitten des vorhergehenden Kapitels wird ein Konzept entwickelt, wie eine ideale Entwurfsumgebung für den Tragwerksentwurf aussehen könnte. Der hier vorgestellte Ansatz soll nicht traditionelle Konzepte punktuell weiterentwickeln. Stattdessen wird unter Berücksichtigung der Interoperabilität, der Plattformunabhängigkeit und offener Standards ein mit heutigen technischen Möglichkeiten umsetzbares, neues Konzept einer Entwurfsumgebung definiert.

Kapitel 4: Umsetzung

Hier werden technische Details der Umsetzung der im vorigen Kapitel vorgestellten Konzepte dargelegt.

[2] Im Falle eines Gebäudemodells bezeichnet die Topologie die räumliche Lage von geometrischen Elementen und deren Nachbarschaftsbeziehungen zu anderen Elementen

Kapitel 5: Anwendungsbeispiele

An einem einfachen, textbasierten, und einem zweiten, etwas komplexeren, grafischen Beispiel wird die Verifizierung der Konzepte durchgeführt.

Kapitel 6: Interaktiver Entwurf in einer VR-Umgebung

In einer abgewandelten Form werden die in den Kapiteln 4 und 5 entwickelten Konzepte in einer VR-Umgebung (Virtuelle Realität) angewendet.

Kapitel 7: Diskussion der Ergebnisse

Hier werden die wesentlichen Ergebnisse der Arbeit zusammengefasst und eingeordnet.

2 Stand der Forschung und Technik

2.1 Computergestützte Planung und Berechnung von Tragwerken

Die Geschichte numerischer, computergestützter Berechnungsmethoden fand ihren Anfang im Jahre 1935, als der Bauingenieur Konrad Zuse sich das Ziel setzte, mühevolle und zeitaufwendige Rechenschritte statischer Berechnungen zu automatisieren. Er konstruierte zunächst eine mechanische und später eine auf Relais, also elektrischen Schaltungen, basierende Rechenmaschine, mit der die vier Grundrechenarten vorgenommen werden konnten.

1965 entwickelte Daniel Roos [Roos 1965] am Civil Engineering Systems Laboratory des M.I.T. ein integriertes Computersystem namens ICES (Integrated Computer System for Engineering Problem Solving). Dieses System verwendete bereits die Sprache FORTRAN, und konnte für relativ einfache Berechnungen wie z.B. Schnittpunkte eines Kreisbogens mit einer Linie programmiert werden. Zur Betreuung des Systems war ein „ICES Supervisor“ notwendig, um die Operationen der Nutzer zu koordinieren. Zu dieser Zeit kamen computergestützte Methoden in der Tragwerksplanung also ausschließlich im akademischen Umfeld zur Anwendung.

Im Flugzeugbau wurde bereits Mitte der 1950er Jahre die Finite-Elemente-Methode entwickelt; angewendet wurde diese aber bis in die siebziger Jahre ausschließlich auf teuren Großrechnern und hauptsächlich im Flugzeug- und Automobilbau.

Die Entwicklung grafischer Konstruktionssoftware (CAD) begann in den 1950er und 1960er Jahren, und fand ebenfalls zunächst nur auf Großrechnern an Universitäten oder im Flugzeugbau statt. Ab den 1980er Jahren, als die ersten PCs für eine breitere Masse erschwinglich wurden, kamen CAD-Programme wie CATIA oder AutoCAD auf den Markt, die dort auch heute noch vertreten sind.

Die ersten PCs wie der Xerox Alto aus dem Jahre 1973 kamen noch so gut wie nie in der Tragwerksplanung zur Anwendung. Computer hielten in der Praxis der Tragwerksplanung erst mit den für Ingenieurbüros erschwinglichen Commodore und IBM-PCs in den 1980er Jahren Einzug. Es wurden jedoch üblicherweise nur einzelne Bauteile, wie Stützen oder Wandscheiben, jeweils als Einzelposition in speziellen Programmen berechnet. So gab es (und gibt es bis heute) einzelne Programme für Durchlaufträger, Stützen, 2D-Rahmen, Scheiben und alle erdenklichen statischen Systeme. Darin spiegelt sich die vorher von Hand aufgestellte Statik aus Einzelpositionen wider. Die Verfahren wurden also nicht angepasst, sondern nur von Papier auf den Computer verlagert.

Die Finite-Elemente-Methode ist ein ausschließlich für die Anwendung auf Computern geeignetes Verfahren. Praktisch alle Arten von statischen Systemen, als Einzelposition oder als Gesamtmodell, können damit berechnet werden. Aufgrund der langen Rechenzeiten und des hohen Eingabeaufwandes kam dieses Verfahren aber ursprünglich im Bauwesen nur in speziellen Fällen zur Anwendung.

Mit der zunehmenden Komplexität der Bauwerke und Modelle, und durch das semi-probabilistische Sicherheitskonzept, das durch die Normung vorgegeben ist, werden heute Probleme der Tragwerksplanung mehr und mehr mit numerischen Methoden bewältigt. Zunehmend werden dabei auch Konzepte wie nichtlineare Materialmodelle angewendet. Effekte wie Kriechen und Schwinden des Betons, Wechselwirkungen zwischen Baugrund und Gebäuden, oder Einflüsse durch Bauphasen werden zunehmend berücksichtigt. Dadurch nähern sich die

Ergebnisse numerischer Simulationen immer weiter an die Realität an, sofern die Qualität der Modellierung ausreichend ist.

Einen technologischen Entwicklungssprung im Bereich der computergestützten Planung von Bauwerken stellen Produktmodelle dar [Eastman et al. 2011], [Eastman, Augenbroe 1998]. Nähere Ausführungen hierzu werden in Abschnitt 2.7 vorgenommen.

2.2 Numerische Simulation im konzeptuellen Entwurfsstadium

Bis heute werden numerische Berechnungsmethoden meist nur in späten Planungsphasen angewendet, da der manuelle Aufwand zur Aufstellung eines Simulationssystems hoch ist. Mangelhafte Interoperabilität[3] zwischen Software für den architektonischen Entwurf und die numerische Simulation erhöht den Aufwand und erschwert den Einsatz numerischer Simulationen im frühen Entwurfsstadium.

In einigen Forschungsarbeiten wird der Vorteil eines frühen Einsatzes numerischer Methoden untersucht. So wird in [Bazjanac 2004] beschrieben, wie numerische Simulationsmethoden (in diesem Fall thermische Gebäudesimulation) auch im frühen Entwurfsstadium zum Einsatz kommen können. Durch den Einsatz von neuen Konzepten zur Interoperabilität (in diesem Fall auf Basis des Informationsmodells IFC) konnte hier der Aufwand und die Dauer zur Erstellung des Simulationsmodells reduziert werden, sodass bereits während der frühen Entwurfsphase verschiedene Varianten untersucht und verglichen werden konnten. Es konnte so eine Entwurfsverbesserung (Verringerung des Kühlenergiebedarfs eines Gebäudes) erzielt werden.

Der möglichst frühe Einsatz von numerischen Analysen im Tragwerksentwurf, bzw. auch im architektonischen Entwurf, wurde ebenfalls in mehreren Forschungsansätzen untersucht [Fenves et al. 2000], [Mora et al. 2004], [Mora et al. 2008]. Aus diesen Arbeiten wird ein grundsätzliches Problem deutlich: Anwendungen zur geometrischen Modellierung und zur numerischen Simulation stammen historisch bedingt aus unterschiedlichen Disziplinen und können bis heute nur mit relativ hohem Aufwand kombiniert werden (mehr dazu in Abschnitt 2.7.2).

Als Lösungsansatz für dieses Problem bieten standardisierte Produktmodelle (mehr dazu in Abschnitt 2.7) die Möglichkeit, im frühen (architektonischen) Entwurfsstadium mit vertretbarem Aufwand ein numerisches Simulationsmodell zu generieren und eine Aussage über das Tragverhalten zu treffen.

2.3 Isogeometrische Analyse

Die Idee der Isogeometrischen Analyse [Cottrell et al. 2009] besteht darin, eine einheitliche[4] geometrische Repräsentation, etwa NURBS (Non-Uniform Rational B-Splines), für geometrische Modelle (Architektur) und numerische Modelle (Strukturanalyse) zu verwenden. Durch die einheitliche geometrische Repräsentation soll die mangelhafte Kopplung zwischen den Bereichen und Anwendungen der grafischen Modellierung (CAD) und der numerischen Analyse verbessert werden. Auf den ersten Blick ist die Isogeometrische Analyse somit ein vielversprechender Ansatz für die Thematik der Entwurfsraum-Exploration.

[3] Interoperabilität: von lateinisch opera ‚Arbeit' und inter ‚zwischen'

[4] Iso: von griechisch *isos*, ‚gleich'. Isogeometrisch bedeutet damit ‚gleich-geometrisch' oder ‚Geometrie-gleich'

Etablierte CAD-Softwaresysteme und andere grafische Modellierungsprogramme unterstützen zwar üblicherweise NURBS-Objekte, aber auch etliche andere, insbesondere unterschiedliche BRep[5]-Geometriemodelle. Dazu zählen beliebige Polyeder (dreidimensionaler Vielflach) mit Dreieck-, Viereck- oder allgemein polygonalen Oberflächen. Durch die weite Verbreitung der BRep-Geometriemodelle ist die Forderung nach ausschließlicher Verwendung von NURBS und damit der Verzicht auf alle anderen geometrischen Repräsentationen problematisch. Ob die Isogeometrische Analyse eine Bedeutung für die praktische Anwendung bekommen wird, ist somit ungewiss. Deswegen wird dieser Ansatz in der vorliegenden Arbeit nicht weiter verfolgt.

2.4 Entwurfsraumsuche durch mathematische Optimierung

Optimierungsalgorithmen können ein Modell hinsichtlich einer Zielfunktion optimieren. Eine Zielfunktion kann aus einer beliebigen Kombination (gewichtete Summe) von Größen wie dem Gesamtgewicht oder der Verzerrungsenergie aufgestellt werden. Weiterhin umfasst die Definition der Optimierungsaufgabe einen Vektor von Entwurfsvariablen und einen Vektor von Nebenbedingungen. Entwurfsvariablen können beliebige Parameter des Modells sein, etwa Knotenkoordinaten (Formoptimierung) oder Steuerungsparameter zur Verbindung der Knoten durch Elemente (Topologieoptimierung) [Kicinger et al. 2005].

Bei der Durchführung einer Optimierung übernimmt der Algorithmus die Kontrolle über die Entwurfsfindung, indem Instanzen des Modells mit verschiedenen Parametern berechnet werden. Nach jeder Berechnung wird der Wert der Zielfunktion ermittelt. So durchsucht der Algorithmus den Parameterraum systematisch (ableitungsgestützte Verfahren), nach dem Zufallsprinzip (Monte-Carlo-Verfahren) oder in teilweise zufälligen und teilweise systematischen Verfahren (Evolutionsstrategien, [Kost 2003]). Das Ziel aller Optimierungsverfahren besteht darin, die Zielfunktion zu minimieren[6].

In [Bletzinger et al. 2005] wird die Anwendung von Optimierungsverfahren zur Formfindung und Optimierung von Membranen und Schalentragwerken beschrieben. Die Optimierung dünnwandiger Schalentragwerke wie Kühltürme, Staumauern oder Fassaden des Hochbaus führt zu einer Minimierung der Biegebeanspruchung, da Lasten effizienter als axiale Druckkraft in einem Ring- oder kuppelförmigen Bauteil abgetragen werden können. Dadurch ergibt sich ein Zusammenhang mit Membrantragwerken, die durch reine Zugkraft gekennzeichnet sind. Dies ist ein gutes Beispiel für die Anwendung der Optimierung zur Entwurfsraumsuche.

Formoptimierung lässt sich gut kombinieren mit dem Konzept der Isogeometrischen Analyse wie in [Wall et al. 2008] beschrieben. Durch die einheitliche geometrische Repräsentation gibt es einen direkten Bezug zwischen Analyseergebnissen und geometrischen Parametern des Modells und damit auch den Entwurfsparametern der Optimierung.

Mathematische Optimierung wird in vielen Industriezweigen erfolgreich angewendet. Im Automobil- und Flugzeugbau werden damit Gewichtseinsparungen oder Erhöhungen der Stabilität von Bauteilen erzielt [Daoud 2005]. Im Bauwesen haben sich Verfahren der mathematischen Optimierung jedoch bisher in der Praxis nicht durchgesetzt. Das liegt wohl an den geringen Stückzahlen (Bauwerke sind meist Unikate), und daran, dass der erhebliche zusätzliche Aufwand

[5] BRep: “Boundary Representation“, Beschreibung eines beliebigen Volumens durch dessen Oberflächen

[6] Eine Maximierung kann durch min{ z(x) } = max { -z(x) } in eine Minimierung überführt werden, oder ein bestimmter Wert t durch min { $(z(x)-t)^2$ } erreicht werden. Daher stellt die Betrachtung der Minimierung keine Einschränkung der Allgemeinheit dar.

nicht wirtschaftlich darstellbar ist oder nicht honoriert wird. Ein weiteres Problem ist der Verlust an Flexibilität im Entwurf, der durch die formale und zwangsläufig umfassende Definition der Optimierungsaufgabe entsteht. Denn die meisten größeren Änderungen im Modell (also nicht nur Änderungen vorhandener Parameter), erfordern aufwendige Nacharbeiten an der Definition von Optimierungsvariablen, Zielfunktion und Nebenbedingungen. Daher werden in der vorliegenden Arbeit Verfahren der mathematischen Optimierung nicht angewendet, zugunsten einer manuell (durch den Ingenieur) gesteuerten Entwurfsraumsuche.

2.5 Grafisch-interaktive Entwurfsraumsuche

Heute am Markt verfügbare Software zur Tragwerksplanung erzwingt einen sequentiellen, nicht-interaktiven Arbeitsprozess [Mackie 2000]. Das liegt nicht daran, dass Ingenieure keinen interaktiven Tragwerksentwurf-und Simulation wollen, sondern daran, dass sich Softwaresysteme zur grafischen Modellierung und numerischen Berechnung über Jahrzehnte separat entwickelt haben. Numerische Berechnungen laufen praktisch immer im sog. Batch-Modus[7] („However, the basic philosophy behind the design of many programs is still essentially batch oriented“ [Mackie 2000]), mit einer Eingabedatei und einer oder mehreren Ausgabedateien- oder Datenbanken. Auf diese Weise lässt sich keine interaktive Simulation erreichen. Denn selbst eine minimale Änderung, beispielsweise einer Koordinate eines Knotens, erfordert eine vollständige neue Eingabedatei, die als Ganzes ausgewertet werden muss. Somit ist eine Änderung eine Neuberechnung des Systems, nicht eine interaktive Änderung eines bestehenden Analysemodells.

Die Kopplung zwischen CAD und Finite-Elemente-Analyse (FEA) kann somit als unzureichend bezeichnet werden, wie in [Mora et al. 2004] detailliert ausgeführt wird. Dort wird auch der Standpunkt vertreten, dass Interaktive Entwurfsraumsuche einem automatisierten Prozess vorzuziehen ist: „…user-model interactivity is devised as the most suitable computer methodology for driving the structural synthesis process“.

Weitere Forschungsansätze die sich mit interaktiver FE-Simulation beschäftigen sind u.a. in [Mackie 2000] und [Terdalkar, Rencis 2006] beschrieben.

In [Krafczyk et al. 2007] wird ein Ansatz zu einer interaktiven numerischen Simulation unter dem Namen *Computatinal Steering* beschrieben. Mit *Steuerung* ist die manuelle Suche im Entwurfsraum nach der optimalen Lösung gemeint. Die Anwendung dieses Konzepts in Kombination mit einer numerischen Simulation durch die FCM[8] wird in [Rank et al. 2012] beschrieben. In [Rank 2007] wird dargestellt, dass ein solches *Steering* auch auf komplexe, aber parallelisierbare Simulationen wie Raumluftströmungen angewendet werden kann.

Ein interessanter Forschungsansatz zur grafisch-interaktiven Entwurfsraumsuche ist *EasyStatics* [Anderheggen, Pedro 2005]. Dabei handelt es sich um eine experimentelle Software zur linear-elastischen Analyse von 2D-Rahmentragwerken. Knoten können interaktiv verschoben werden, M-N-V-Linien werden dabei sofort berechnet und grafisch dargestellt (Abbildung 3).

[7] Batch-Modus: Stapelweise, also sequentielle Abarbeitung von Berechnungsaufgaben

[8] FCM: “Finite Cell Method”. Methode zur Diskretisierung und Simulation dreidimensionaler Strukturen ohne Netzgenerierung und den damit häufig verbundenen Problemen.

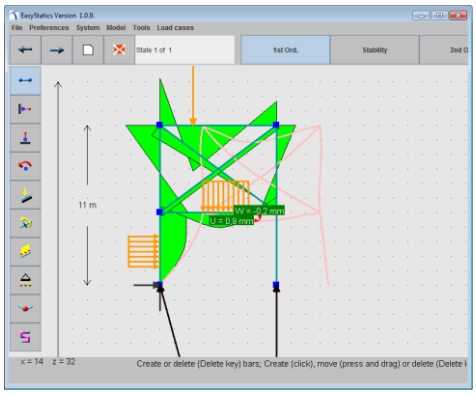


Abbildung 3: Programmfenster *easyStatics* mit Darstellung eines statischen Systems mit Momentenlinie

Entwickelt wurde das Programm an der ETH Zürich, mit der Zielrichtung die Lehre der Baustatik durch ein intuitives E-Learning-Simulationstool zu ergänzen. Die Software kann kostenlos heruntergeladen und installiert werden[9].

Ein in [Gerold 2010] genauer beschriebenes Projekt ist ebenfalls für die Lehre konzipiert und behandelt die interaktive Simulation und Visualisierung eines vorgespannten Stahlbetonträgers.

Diese Software wurde nicht im Rahmen der vorliegenden Arbeit erstellt, jedoch vom Autor programmiert. Da die Thematik gut zur Entwurfsraum-Exploration passt, soll hier eine kurze Darstellung des Programms erfolgen.

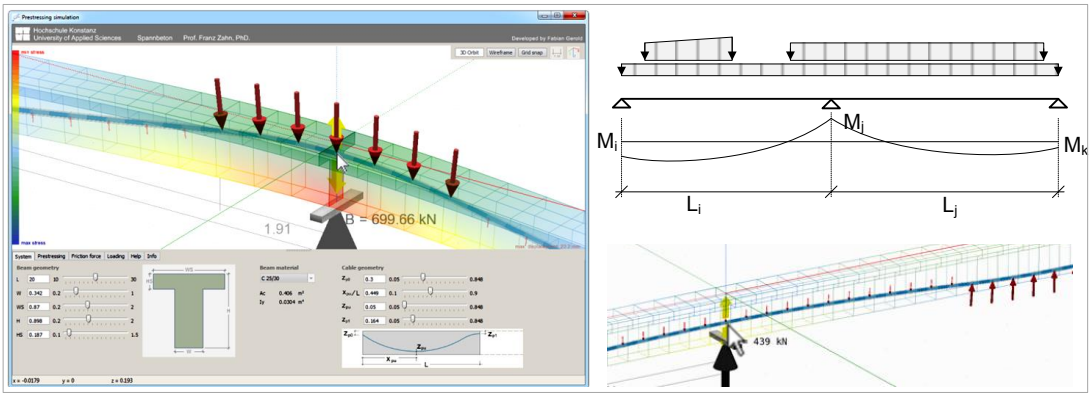


Abbildung 4: Programmfenster und statisches System des Vorspannungs-Tools

Verschiedene Parameter des Simulationsmodells, wie Spannkabelverlauf, Querschnittsabmessungen und Material, können interaktiv im 3D-Viewer bzw. in Steuerelementen der grafischen Nutzeroberfläche (Abbildung 4) verändert werden. Alle Berechnungsergebnisse wie Spannkraftverluste, Momentenlinie, Spannungen und Verformungen im System, werden sofort berechnet und visualisiert. So kann intuitiv und sehr schnell eine optimale Kombination von Querschnitt und Spannkabelverlauf gefunden werden. Im Vergleich mit Verfahren zur numerischen Optimierung lässt sich feststellen, dass der zeitliche Aufwand des interaktiven Findens des Optimums bei weitem geringer ist als allein das Aufstellen von Optimierungsvariablen, Zielfunktion und Randbedingungen eines Optimierungsmodells. Die Software kann kostenlos heruntergeladen und installiert werden[10]. Im Gegensatz zu allgemeinen

[9] http://easystatics.ethz.ch/, Abruf: 25.11.2012

[10] http://www.dictionary.bi.htwg-konstanz.de/PrestressedBeam/, Abruf: 25.11.2012

FE-Programmen kann durch o. g. Programm nur ein Zweifeldträger, also kein allgemeines statisches System berechnet werden. Somit sind die Rechenzeiten gering genug (wenige Millisekunden), um die Berechnungsergebnisse in Echtzeit zu visualisieren.

In der Filmindustrie werden immer häufiger Filmsequenzen von Animationsfilmen nicht manuell durch Bewegungspfade erstellt, sondern mit numerischen Modellen simuliert [Lagler 2012]. Dabei liegt ein Fokus auf schneller Analyse, bei Computerspielen auf interaktiver Simulation in Echtzeit. In [Mezger et al. 2007] wird eine grafisch-interaktive physische Simulation von volumetrischen Objekten beschrieben.

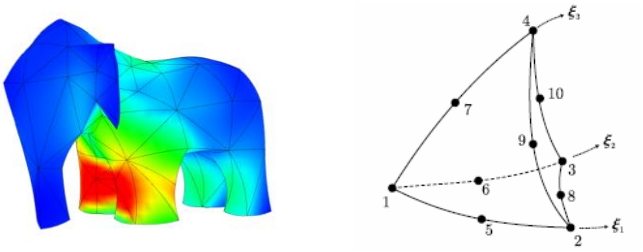


Abbildung 5: Volumetrische FE-Modellierung, 10-Knoten Tetraeder-Element. Quelle: [Mezger et al. 2007]

Zur Anwendung kommt dabei ein 10-Knoten Tetraeder-Element (Abbildung 5), mit quadratischer Ansatzfunktion. Es kommt ein nichtlineares Materialmodell zum Einsatz. Dynamische Effekte werden durch eine Zeitintegration berücksichtigt. Anwendungsziel ist der Spiele- und Grafik-Bereich, daher werden Approximationen vorgenommen, die auf die Einhaltung einer minimalen Frame-Rate[11] gerichtet sind, nicht auf die Genauigkeitsanforderungen wie sie im Tragwerksentwurf erforderlich sind.

2.6 Industrielösungen für interaktive numerische Simulation

Eine Recherche bzw. direkte Anfrage bei diversen Herstellen von Software für die Tragwerksplanung (Abaqus, ADINA, ANSYS, ATENA (Cervenka Consult), V8i (Bentley), SAP2000, Dlubal RFEM, Nastran, Nemetschek, LS-Dyna, RIB, SOFiSTiK) ergab, dass es so gut wie keine Lösungen gibt für interaktive numerische Simulation. Unter dem Stichwort „Interaktion“ finden sich zwar viele Softwarelösungen, jedoch ist damit so gut wie immer die Interaktion von Spannungen aus Normalkraft und Momenten gemeint, also ein völlig anderer Bereich.

Die einzige Ausnahme ist der Programmmodus „Model Alive“ von *SAP2000* (Hersteller: CSI[12]). Es handelt sich um einen grafischen 2D-Rahmen-Editor ähnlich *easyStatics*. Ein Video auf Youtube zeigt die Funktion [SAP2000 ModelAlive 2011] (Abbildung 6).

[11] Frame-Rate: Anzahl von Aktualisierungszyklen eines Grafik-Kontextes. Mit Frame-Raten von > 30 Frames/sec werden Animationen als flüssig wahrgenommen

[12] CSI: Computers and Structures Inc., http://www.csiberkeley.com

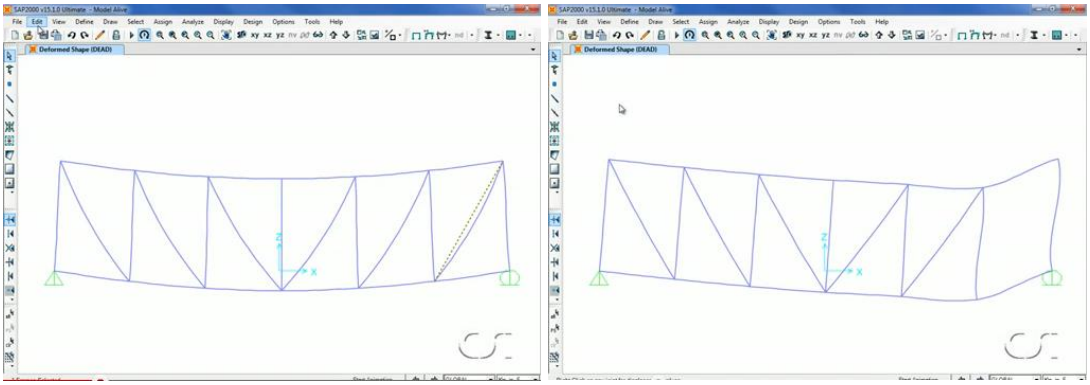


Abbildung 6: SAP2000 im „Model Alive“-Modus mit automatischer Re-Analyse eines 2D-Rahmentragwerks, Quelle: [SAP2000 ModelAlive 2011]

2.7 Produktmodellierung im Bauwesen

2.7.1 Datenaustausch

Wie bereits eingangs erwähnt, ist die Struktur der beteiligten Unternehmen an einem Bauprojekt üblicherweise stark fragmentiert [Campbell 2007]. Das Problem der ebenfalls fragmentierten Softwareprodukte und des Datenaustauschs wird in [Wassouf 2010] so beschrieben:

> „… existiert Software im Bauwesen heute nach wie vor in Form von Insellösungen für die verschiedenen Planungsphasen. Für einen Fortschritt bei der Entwicklung computerunterstützender Verfahren im Bauwesen ist es deshalb zunächst nötig, alle Objekte und Zustände eines Bauwerks eindeutig zu beschreiben, um sie dann übergreifend in verschiedenen Planungsphasen nutzen zu können.“

Auch in anderen Industriezweigen war die Notwendigkeit des Datenaustauschs zwischen verschiedenen Softwaresystemen schon früh offensichtlich, und so wurde in den 1980er Jahren das Datenformat DXF[13] als das bis heute bekannteste CAD-Datenaustauschformat entwickelt. Damit konnten jedoch nur geometrische Daten ausgetauscht werden, und offensichtlich kann eine Linie in einem reinen Zeichenprogramm etwas anderes bedeuten als in einem Finite-Elemente-Programm. Um das Problem der fehlenden Semantik[14] der Objekte zu lösen, wurde in vielen Forschungsarbeiten und Projekten wie [Gu, Chan 1995], [Sudarsan et al. 2005], [Zhao, Liu 2008] die semantische Modellierung (auch als Produktmodellierung bezeichnet) entwickelt.

2.7.2 Von der geometrischen zur semantischen Modellierung

Der heutige Standard für die Erstellung technischer Dokumente im Bauwesen ist CAD (Computer Aided Design). Dieser ursprünglich sehr allgemeine Begriff hat sich in den letzten Jahrzehnten beschränkt auf die computerbasierte, aber rein geometrische Modellierung von technischen Bauteilen, im Bauwesen wie in anderen Industriebereichen.

[13] Drawing Exchange Format

[14] Semantik: “Bedeutung” von Wörtern oder Symbolen. Ein semantisches Informationsmodell unterscheidet sich von einem rein geometrischen Modell z.B. eines Quaders dadurch, dass zusätzlich zur Geometrie des Quaders die Information abgelegt wird, dass dieser eine Wand darstellt

Das Problem rein geometrischer Modelle ohne semantische Bedeutung wird in einer Arbeit über Simulation von Bränden in Gebäuden [Spearpoint 2007] so ausgedrückt:

> „… the inability of CAD systems to adequately describe a building in a way that is meaningful to simulation software".

Und bezogen auf die Brandsimulation heißt es weiter:

> „… fire simulation software requires a much richer description of buildings than traditional CAD systems can provide".

Bezogen auf die Tragwerksplanung und viele andere Bereiche stellt sich die Problematik ähnlich dar. Als Lösung dieses Problems ist der Grundgedanke bei der Produktmodellierung, nicht nur die Geometrie zu modellieren, sondern allgemein *Informationen* über ein Gebäude („Building Information"). Die Geometrie und Topologie ist zwar weiterhin die wichtigste Kategorie von Informationen in einem solchen Modell, jedoch kommen u. a. eine Kostendimension, die Prozessmodellierung, also eine Zeitachse und vor allem die semantische Bedeutung jedes einzelnen Objektes hinzu.

2.7.3 Interoperabilität

Die Digitalisierung hat zwar in der AEC-Industrie[15] längst Einzug gehalten, hat aber häufig zu fachspezifischen Insellösungen geführt, und damit zu domänenspezifischen, nicht interoperablen[16] Computermodellen. So wird im architektonischen Entwurf mit rein geometrischen Modellen gearbeitet, im Tragwerksentwurf mit Strukturmodellen. Diese sind üblicherweise dimensionsreduziert, dafür aber mit Lasten, Materialdefinitionen, Lagerungs- und Kopplungsbedingungen angereichert. In anderen Bereichen oder in späteren Planungs-, Bau- und Nutzungsphasen entstehen wiederum andere, spezialisierte Modelle, die alle durch teilweise automatisierten Import, teilweise durch manuelle (Neu-)Eingabe entstehen. Das Problem ist folglich, dass die Sichtweisen auf ein Gebäude unterschiedlich sind, und damit auch die Computermodelle. Sie sind nicht nur unterschiedlich, sondern auch üblicherweise inkompatibel. Die vielfältigen und häufigen Datentransfers bzw. Neueingaben sind eine stete Fehlerquelle und verschlingen enorme (unnötige) Arbeitsleistungen. In [Sanguinetti et al. 2012] wird dieses Problem so ausgedrückt:

> „In traditional design where 2D drawings were manually generated, each domain expert or consultant participating in the project had their own analysis tools and manually generated datasets from the architect's design intent drawings […] Because this manual process was time consuming, the specialist consultants could not respond to the timeframe of design decision-making"

Die Probleme der unterschiedlichen Sichtweise und Modellierung sowie der (unter anderem daraus resultierenden) mangelhaften Interoperabilität sind eng miteinander verwandt. An einer Lösung wird schon lange gearbeitet, ein guter Überblick über die verschiedenen Forschungsansätze dazu wird in [Eastman et al. 2011] gegeben. Ebenfalls dort zu finden sind interessante generelle Überlegungen und Definitionen zu semantischer Modellierung.

[15] AEC: Architecture, Engineering, Construction

[16] Interoperabilität: von lateinisch opera ‚Arbeit' und inter ‚zwischen'

Mit GARM (General AEC Reference Model) wurde bereits Ende der 1980er Jahre ein STEP[17]-basiertes Produktdatenmodell vorgeschlagen, mit dem Ziel der Standardisierung des Datenaustauschs im Bauwesen und anderen Industriezweigen. In [Gielingh 1988] ist dieses Konzept genauer beschrieben.

Im Wesentlichen haben die verschiedenen Ansätze zu dem Konzept eines Produktdatenmodells geführt, oft bezeichnet als *Building Information Model(ing)* (BIM). Im Weiteren wird die Terminologie aus [van Treeck 2004] verwendet:

> „Das Datenschema eines solchen Modells wird als *Produktdatenmodell* bezeichnet; durch Instanziierung erhält man ein spezifisches *Produktmodell*".

Eine Möglichkeit, der mangelnden Interoperabilität verschiedener Software zu begegnen, ist ein quasi-Monopol eines einzigen proprietären Standards. Beispielsweise kann für ein Bauprojekt mit mehreren Beteiligten vom Bauherrn oder Generalplaner die Verwendung von Autodesk Revit® vorgeschrieben werden. Für die Forschung hat jedoch ein offener Standard den Vorteil, dass er ohne Einschränkungen genutzt und bei Bedarf weiterentwickelt werden kann.

Als offener Standard zur Bauwerks-Informations-Modellierung haben sich jedoch in den letzten Jahren die *Industry Foundation Classes* (IFC) etabliert, auf die in den folgenden Abschnitten näher eingegangen wird.

2.7.4 Die Datenmodellierungssprache EXPRESS

Die nach ISO 10303-11 standardisierte Modellierungssprache EXPRESS [ISO 10303 2004] wurde speziell für den Datenaustausch von Produktmodellen entwickelt. Die zugrunde liegende Denkweise ist die Objektorientierung. Ein EXPRESS-Schema kann in einer objektorientierten Programmiersprache (OOP) wie C++ oder Java implementiert werden oder zur Definition einer objektorientierten Datenschnittstelle wie SDAI[18] verwendet werden.

Wie in einer OOP-Sprache gibt es in EXPRESS verschiedene Datentypen, die im Folgenden kurz erläutert werden sollen.

Folgende **Einfache Datentypen** sind in EXPRESS vordefiniert, analog den elementaren Datentypen einer Programmiersprache:

STRING	Zeichenkette mit beliebiger Länge im Unicode Format
BINARY	Sequenz von Bits, sehr selten verwendet
LOGICAL	Wahrheitswert, kann die Werte TRUE, FALSE oder UNKNOWN annehmen
BOOLEAN	Wahrheitswert, kann die Werte TRUE oder FALSE annehmen
NUMBER	Verallgemeinerung der Datentypen REAL und INTEGER, kann also sowohl ganze als auch reelle Werte annehmen
INTEGER	Ganze Zahl
REAL	Reelle Zahl

Komplexe Datentypen werden in EXRPESS als ENTITY bezeichnet und entsprechen einer Klasse in der OOP. Als komplex wird ein Entity bezeichnet, weil es aus mehreren einfachen oder anderen komplexen Datentypen zusammengesetzt ist.

[17] STEP: „Standard for the Exchange of Product model data", nach ISO 1030

[18] Standard Data Access Interface

Enumerationen stellen einen Datentyp dar, der bestimmte Werte einer Aufzählung annehmen kann. Beispiel: Präfixe der SI-Einheiten:

```
TYPE IfcSIPrefix = ENUMERATION OF (
       …, MEGA, KILO, HECTO, DECA, DECI, CENTI, MILLI, … );
END_TYPE;
```

Hierbei ist „IfcSIPrefix“ der Name der Enumeration, die Aufzählung gibt den Wertebereich an.

Definierte Datentypen sind eine Spezialisierung anderer Datentypen. Beispiel: Längenmaß und positives Längenmaß:

```
TYPE IfcLengthMeasure = REAL;
END_TYPE;

TYPE IfcPositiveLengthMeasure = IfcLengthMeasure;
       WHERE
       WR1      :      SELF > 0.
END_TYPE;
```

SELECT-Datentypen können eine Auswahl von anderen Datentypen annehmen. Beispiel:

```
TYPE IfcUnit = SELECT (
IfcDerivedUnit, IfcNamedUnit, IfcMonetaryUnit);
END_TYPE;
```

Hierbei ist *IfcUnit* der neu definierte Datentyp, der als Attribut eines ENTITY verwendet werden kann. Anstelle des generalisierten IfcUnit-Datentyps wird aber ein konkretes Objekt eines Datentyps der Auswahl verwendet.

Aggregations-Datentypen definieren einen neuen Datentyp als eine Menge von Objekten anderen Datentyps. Dabei sind folgende Arten möglich:

SET	Ungeordnete Menge, beliebige Größe, Duplikate möglich
BAG	Ungeordnete Menge, beliebige Größe
LIST	Geordnete Menge, beliebige Größe
ARRAY	Geordnete Menge, feste Größe, kann leere Elemente enthalten

Elemente einer Aggregation können ebenfalls eine Aggregation sein, so können mehrdimensionale Felder definiert werden.

Attribute

Komplexe ENTITY-Datentypen können aus beliebig vielen anderen Datentypen zusammengesetzt sein, ein ENTITY ist damit nichts anderes als ein benannter Container für Daten. Die Daten werden damit zu Attributen (Eigenschaften) des Containers. Es gibt drei Arten von Attributen:

- Explizite Attribute, sie werden mit Name und Datentyp im Schema des Entity aufgeführt.
- Abgeleitete Attribute, beispielsweise die Anzahl der Elemente einer Liste oder andere Werte, die nach einer bestimmten Vorschrift aus den expliziten Attributen berechnet werden können.
- Inverse Attribute. Relationen zwischen Objekten sind meistens nur in eine Richtung explizit definiert. Um eine Navigation in umgekehrter Richtung zu ermöglichen, können die dazu notwendigen Referenzen durch inverse Attribute realisiert werden (ein Beispiel findet sich in Abbildung 7).

Vererbung

Sehr wichtig ist auch das Konzept der Vererbung. Ein ENTITY kann von einem anderen ENTITY abgeleitet sein und übernimmt damit automatisch dessen Attribute. Dies erlaubt eine effiziente Datenstruktur, die auch wiederum direkt in eine objektorientierte Programmiersprache übertragen werden kann. Durch folgende Notation kann eine Vererbung definiert werden:

```
ENTITY IfcCartesianPointList
  SUBTYPE OF IfcGeometricRepresentationItem;
  CoordList     :     LIST [1:?] OF LIST [3:3] OF IfcLengthMeasure;
END_ENTITY;
```

Das hier definierte Entity *IfcCartesianPointList* ist abgeleitet von *IfcGeometricRepresentationItem*, erbt damit dessen Attribute und erhält ein weiteres explizites Attribut *CoordList*. Dieses ist definiert als eine (mehrdimensionale) Aggregation.

Die Übersetzung eines solchen Schemas in eine Programmiersprache wird in Abschnitt 4.2.2 beschrieben.

EXPRESS-G

EXPRESS-G ist die grafische Darstellung der textbasierten Modellierungssprache EXPRESS. Darin können mehrere Datentypen und Referenzen in einem Diagramm übersichtlich dargestellt werden. EXPRESS-G ist dem gebräuchlicheren UML-Klassendiagramm relativ ähnlich. Letzteres wird im Folgenden ohnehin zur Beschreibung von Klassenhierarchien in verschiedenen Anwendungen verwendet, deswegen wird auf eine detaillierte Darstellung und Anwendung von EXPRESS-G zwecks besserer Übersichtlichkeit und Einheitlichkeit in dieser Arbeit verzichtet.

2.7.5 Das Produktdatenmodell IFC

Die *Industry Foundation Classes* (IFC) sind ein speziell für das Bauwesen entwickeltes Produktdatenmodell, definiert durch die buildingSMART International Ltd.[19]. Es handelt sich um einen offenen Standard, der in Forschung und Industrie frei verwendet werden kann.

Die IFC umfassen nicht nur die Geometrie und Topologie von Bauteilen, sondern zusätzlich Materialdefinitionen, vielfältige Relationen zwischen Objekten sowie Informationen bezüglich Tragwerksplanung, technischer Gebäudeausrüstung (TGA) und vieler weiterer Bereiche.

An dieser Stelle soll keine vollständige Beschreibung der IFC gegeben werden. Nur die Aspekte, die in der weiteren Arbeit verwendet werden, sollen kurz vorgestellt werden.

Das Produktmodell IFC ermöglicht prinzipiell den Datenaustausch im physischen Datenformat STEP oder XML[20], wobei in dieser Arbeit ausschließlich STEP verwendet wird. Das STEP-Format ist menschenlesbar und enthält ausschließlich ASCII-Zeichen, Sonderzeichen werden codiert.

Das Schema des Produktdatenmodells IFC wird in der Modellierungssprache EXPRESS definiert. Seit dem Beginn der Entwicklung im Jahr 1996 gab es mehrere Entwicklungsversionen des IFC-Modells. Heute gebräuchliche Versionen reichen von IFC 2x, veröffentlicht im Jahr 2000, bis zu IFC 2X3 TC1 (Technical Report 1). Momentan befindet sich die Version IFC4 (alte

[19] http://www.buildingsmart-tech.org/ Abruf: 10.11.2012

[20] Extended Markup Language, hierarchisch organisierte Auszeichnungssprache

Versionsbezeichnung: IFC 2X4) in Entwicklung, der aktuellste veröffentlichte Release Candidate trägt die Bezeichnung IFC4 RC4 [IFC4 Dokumentation 2012]. In der Software zu der vorliegenden Arbeit wird die Version IFC4 RC4 verwendet, und in den folgenden Ausführungen wird stets auf diese Version Bezug genommen.

Für eine ausführliche Beschreibung der Datenstruktur in IFC wird [Fliegner 2003] und [Liebich 2009] empfohlen.

Bauteil-Hierarchie in IFC

Ebenso wichtig wie die Modellierung von Bauteilen ist die Modellierung der Beziehungen zwischen diesen verschiedenen Objekten. Eine Wand als Informationseinheit ist nur sinnvoll, wenn sie einem Stockwerk zugeordnet werden kann und dieses Stockwerk einem Gebäude.

In IFC werden solche Beziehungen zwischen Objekten über indirekte Zuweisungen (objektivierte Relationen) realisiert (*IfcRelAggregates*, *IfcRelContainedinSpatialStructure* in Abbildung 7).

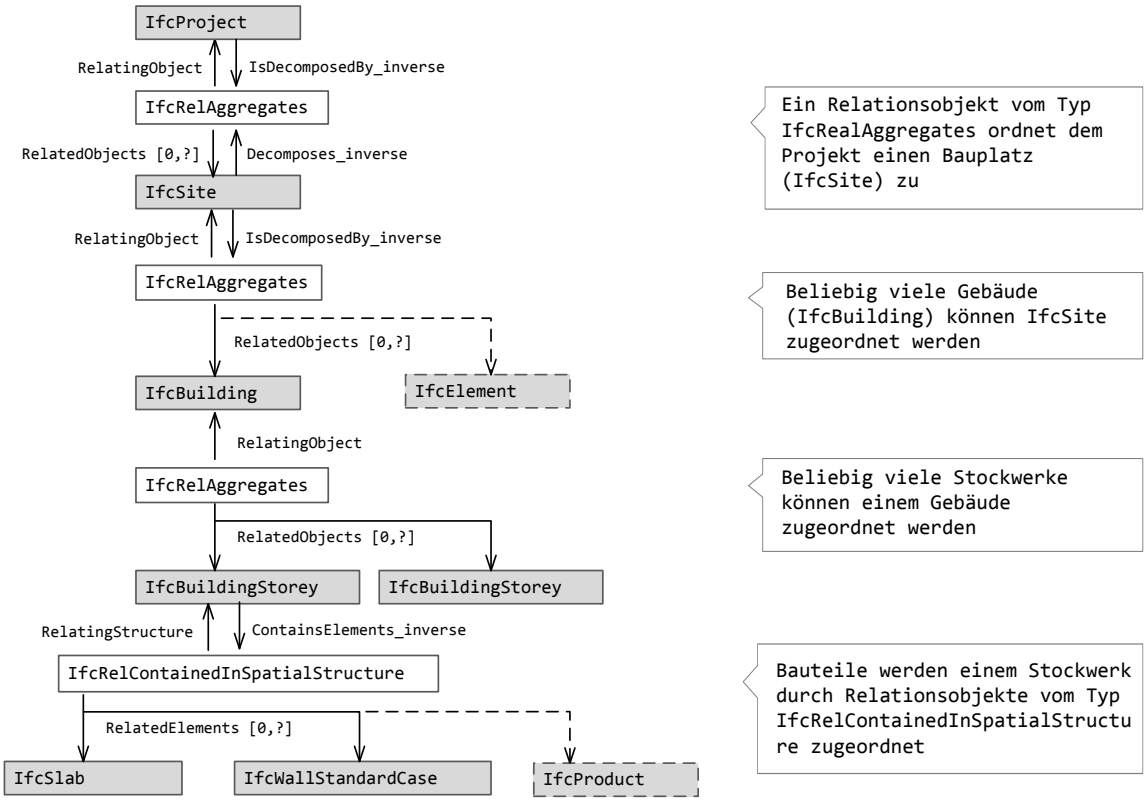


Abbildung 7: Aufbau der Bauwerks-und Bauteilstruktur eines Projektes in IFC

IfcSite und *IfcBuilding* sind nicht direkt miteinander verlinkt, nur von *IfcRelAggregates* aus gibt es Referenzen (*RelatingObject* und *RelatedObjects*). Über inverse Attribute können diese Referenzen jedoch auch in umgekehrter Richtung navigiert werden, wodurch indirekte Referenzen zwischen *IfcProject* und allen Objekten entstehen, die sich in der räumlichen Struktur („spatial structure“) des Projektes befinden.

Koordinatensysteme

Bauteile (bzw. allgemein Produkte, d. h. alle von *IfcProduct* abgeleiteten Entities) haben jeweils ein Attribut zur (optionalen) Definition eines lokalen Koordinatensystems (*IfcObjectPlacement,* Abbildung 8).

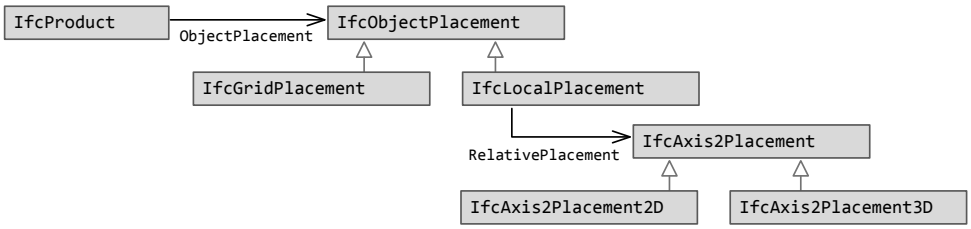


Abbildung 8: Koordinatensysteme in IFC

Alle Koordinatensysteme sind jeweils relativ zur nächsthöheren Ebene der Gebäudestruktur. Dadurch kann eine Kaskade aus mehreren relativen Koordinatensystemen entstehen.

Geometrische Repräsentationen

Produkte haben ein optionales Attribut mit dem Datentyp *IfcProductRepresentation*. Darüber lassen sich verschiedene grafische oder topologische Repräsentationen einem Produkt zuordnen.

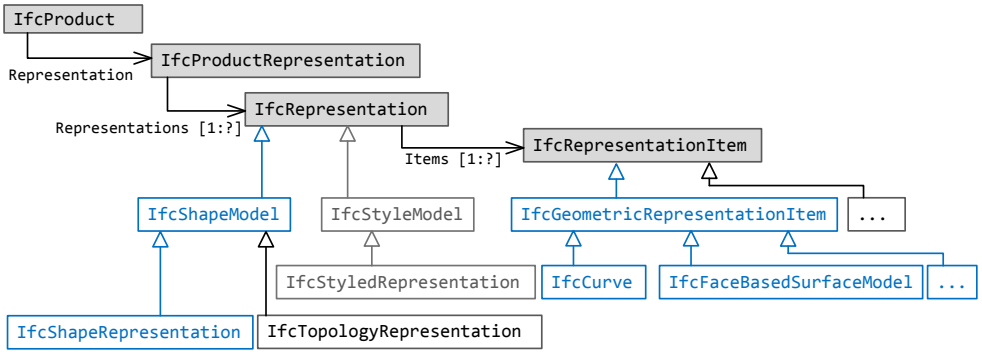


Abbildung 9: Grafische und andere Repräsentationen in IFC

Wie in Abbildung 9 ersichtlich, kann eine Menge von beliebig vielen grafischen Elementen (*Items*) einem Objekt des Typs *IfcRepresentation* (bzw. *IfcShapeRepresentation* im Falle einer grafischen Repräsentation) zugeordnet werden.

2.7.6 IFC-Erweiterungen

Eine Einschränkung der aktuellen IFC-Releases ist die Fokussierung auf den Hochbau. So wird in der allgemeinen Datenstruktur ein Projekt (*IfcProject*) in Standorte (*IfcSite*), und diese wiederum in Gebäude (*IfcBuilding*) untergliedert. Gebäude bestehen aus Stockwerken (*IfcBuildingStorey*), und dort erst werden die Bauteile wie Deckenplatten, Wände usw. zugeordnet (Abbildung 7). Auf Tiefbauprojekte wie Tunnel oder Brücken lässt sich diese Datenstruktur schlecht anwenden. Darüber hinaus fehlen passende Entities für Bauteile des Tiefbaus. Eine mögliche Lösung besteht darin, das allgemeine Entity *IfcBuildingElementProxy* für Bauteile zu verwenden, und fehlende Attribute durch generische Mengen von Eigenschaften (*IfcPropertySet)* zu ergänzen. Der Nachteil dabei ist jedoch die Anwendungsabhängigkeit der generischen Attribute. Denn diese Daten sind

nicht Bestandteil des offiziellen Standards und bedürfen daher einer gesonderten Absprache zwischen den Softwareherstellern.

Ein Ansatz zur Lösung dieser Problematik sind die durch die buildingSMART koordinierten „IFC Extension Projects“. In einem solchen Projekt können Anwender, Hersteller oder Forscher Teilmodelle mit dem Ziel entwickeln, diese später in eine offizielle Version einfließen zu lassen.

Ein Beispiel für ein solches Projekt ist „CI-2 Bridge“[21]. Dessen Zielrichtung ist die Modellierung des Entwurfs, der Konstruktion und der Unterhaltung von Brücken. Eine detaillierte Beschreibung findet sich in [Lebegue et al. 2007]. Das Hauptmerkmal der darin definierten Modellierung ist, dass Elemente entlang einer Referenzachse (z.B. Straßenachse) angeordnet werden können. Weiterhin wird analog zur räumlichen Anordnung der Elemente in *IfcBuilding* und *IfcBuildingStorey*, eine Anordnung in *IfcBridge* und *IfcBridgePart* definiert.

2.7.7 Bisherige Anwendungsbereiche der IFC

Die IFC bilden als offener Standard für semantische Modellierung eine ideale Plattform für verschiedene Forschungsprojekte und Innovationen im Bauwesen. Einige seien hier exemplarisch genannt:

- In [Rönneblad 2003] wird die Anwendung von IFC im Bereich Stahlbeton hinsichtlich ihrer Praxistauglichkeit untersucht.
- In [van Treeck 2004] werden Raumluftströmungen simuliert. Das Simulationsmodell wird direkt aus den geometrischen Einheiten des IFC-Modells generiert.
- Die Implementierung einer IFC-basierten Plattform zur Kommunikation über das Internet wird in [Chen et al. 2005] beschrieben.
- In [Spearpoint 2007] wird beschrieben, wie auf Basis eines IFC-Modells eine Brand-Simulation angesteuert wird.
- Die Verwendung von IFC-Modellen für eine virtuelle Trainings-Umgebung zur Brandbekämpfung wird in [Rüppel, Schatz 2010] beschrieben.
- In [Tulke 2010] wird ein Verfahren zur Erstellung von Terminplänen auf Basis von IFC beschrieben.
- In [Tauscher 2011] wird ein Modell vorgestellt, mit dem aus einem Bauwerksinformationsmodell automatisiert ein Bauablaufplan gewonnen werden kann.
- Yang und Xu stellen in [Yang, Xu 2004] ein System zur automatisierten Überprüfung von verschiedenen Normen im Bauwesen vor. Die durch die Normen vorgegebenen Regeln können auf IFC-Modelle angewendet und ihre Einhaltung überprüft werden.

2.7.8 IFC in der Tragwerksplanung

Die IFC beinhalten die sog. *Structural Elements Domain* und *Structural Analysis Domain* [Wan et al. 2004], die eine Modellierung von Tragwerken ermöglichen. Es handelt sich dabei um ein hierarchisches Modell ähnlich dem architektonischen IFC-Modell. Momentan ist das

[21] http://buildingsmart-tech.org/future-extensions/ifc_extension_projects/current/ci2, Abruf: 05.01.2013

Strukturmodell in IFC ein dimensionsreduziertes[22] Modell (Abbildung 10), das zusätzlich zum architektonischen Modell erstellt und gespeichert wird.

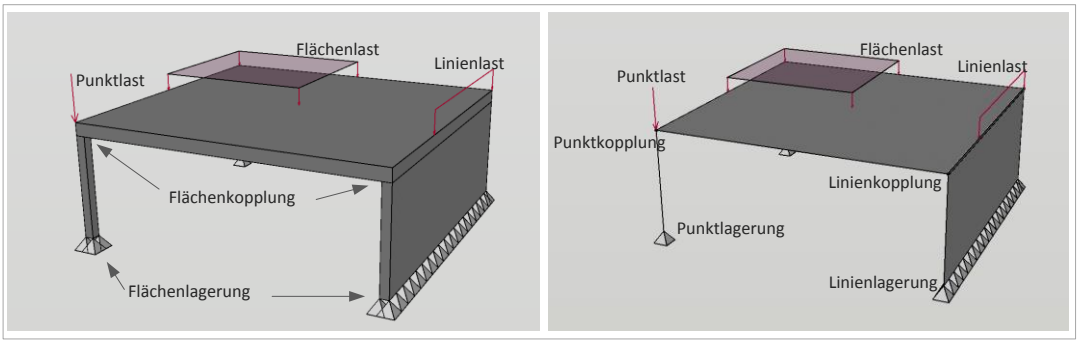


Abbildung 10: Strukturmodell, volumetrisch und dimensionsreduziert

Ein vom Architekt aufgestelltes Produktmodell enthält jedoch nicht automatisch ein solches Strukturmodell. Ein Strukturmodell kann weitgehend automatisch aus einem architektonischen Modell heraus aufgestellt werden, wie in [Romberg et al. 2004] beschrieben. In der Praxis wird diese Arbeit jedoch üblicherweise von Hand erledigt.

Die IFC eignen sich in der aktuellen Version als Datenaustauschmodell für die Tragwerksplanung. So leisten sie bereits heute einen Beitrag zur Verbesserung der Interoperabilität im Tragwerksentwurf.

2.8 Beschleunigung der numerischen Simulation

2.8.1 Notwendigkeit der Beschleunigung

Die in Abschnitt 1.2 definierte Zielsetzung einer unmittelbaren Rückkopplung der Berechnungsergebnisse ergibt die Notwendigkeit einer möglichst verzögerungsfreien Simulation.

Bis zu einer gewissen Modellgröße können Tragwerke in herkömmlicher Weise auf einem lokalen Arbeitsplatz-Rechner schnell genug berechnet werden für eine Nachführung der Berechnungsergebnisse in Echtzeit.

Das Lösen von großen linearen Gleichungssystemen, wie sie bei der FEA auftreten, führt jedoch zu einem mit der Anzahl der Freiheitsgrade[23] stark ansteigenden Rechenaufwand. Somit stellt sich bei Tragwerken ab einer Größe von einigen zehntausend Freiheitsgraden die Frage nach einer Beschleunigung der Simulation. Eine Beschleunigung kann auf mehrere Arten erreicht werden, die im Folgenden erläutert werden sollen.

2.8.2 Serverbasierte Simulation

Eine naheliegende Lösung zur Beschleunigung der Simulation ist die Nutzung eines dedizierten und auf numerische Simulationen spezialisierten Servers. Unter anderem in [Yang 2004] werden die Vorteile einer serverseitigen Infrastruktur zur numerischen Simulation hervorgehoben und es wird ein sehr großes Potential für einen generellen Trend in diese Richtung ausgemacht.

[22] Dimensionsreduzierung: Volumetrische Bauteile werden im Falle von Stützen oder Balken als Linie abstrahiert, im Falle von Wänden oder Deckenplatten dagegen als Fläche.

[23] Ein Freiheitsgrad ist eine Bewegungsmöglichkeit eines Objektes. Ein Punkt im dreidimensionalen Raum hat 6 Freiheitsgrade, 3 Translationen und 3 Rotationen

Aaron Weiss formuliert das Konzept folgendermaßen:

> "Why not just move all processing power to the cloud, and walk around with an ultralight input device with a screen?" [Weiss 2007]

Der Begriff *Cloud* steht in diesem Kontext für serverbasierte Ressourcen wie Rechenleistung oder Speicherplatz.

Zur serverbasierten Simulation gibt es bereits viele Forschungsansätze. So wird in [Blachowicz, Wieja 2006] ein System zur serverbasierten numerischen Analyse von Magnetfeld-Simulationen beschrieben. Zur Kommunikation wird dabei XML-RPC[24] verwendet. Dabei wird XML-RPC zweistufig angewendet: 1. Zur Client-Server-Kommunikation, und 2. Zur Parallelisierung und Zusammenführung (Fork-Join) der Simulationsprozesse des Servers. Durch die erforderlichen Methodensignaturen[25] beim Aufruf einer Methode in XML-RPC entsteht eine unvermeidliche, sehr spezifische Abhängigkeit zwischen Client und serverseitiger Implementierung. Der Ansatz einer serverseitigen Simulation mit Anbindung von Clients durch ein auf XML basierendes Kommunikationsprotokoll soll in der vorliegenden Arbeit aufgegriffen werden. Die hohe Abhängigkeit von der serverseitigen Implementierung soll jedoch vermieden werden, wie in späteren Abschnitten erläutert wird.

In einer Arbeit von [Chen et al. 2005] wird ein serverbasiertes System zur Strukturanalyse vorgestellt. Dabei wird zunächst vom Architekt eine IFC-Datei per Web-Browser an den Server übermittelt. Der Server extrahiert Stabelemente aus der IFC-Datei, erstellt daraus ein Strukturmodell und schickt dem Anwender ein Java-Applet zur Visualisierung des Systems. Dieses Konzept eignet sich nicht für interaktive Simulation, entsprechend der Zielrichtung nach Abschnitt 1.2. Denn jede Änderung im Modell erfordert eine explizite Anforderung einer Neuberechnung und das Öffnen und Schließen mehrerer Programmfenster. So geht der intuitive Zusammenhang zwischen Modelländerung und Auswirkungen auf das Tragverhalten verloren. Jedoch wird die Idee einer auf einem Produktmodell basierenden Kommunikation zwischen Client und Server in Abschnitt 3.3.4 aufgegriffen und entsprechend der Zielstellung angepasst und weiterentwickelt.

Die Schnittstellen einer serverbasierten Computing-Infrastruktur sollten auf offenen Standards wie TCP oder XML basieren, damit möglichst viele Client-Anwendungen die Dienste nutzen können. Über die konkrete Ausgestaltung einer solchen Schnittstelle verfolgen jedoch alle o. g. Forschungsansätze ein anderes Konzept.

Als Plattformen für Client-Anwendungen kommen aufgrund der technischen Entwicklung auch aktuelle Webbrowser infrage. Mit neuen Technologien wie 3D-OpenGL-Grafik[26] direkt im Browser (WebGL[27]). Die üblichere Methode ist jedoch nach wie vor eine herkömmliche Desktop-Anwendung mit einem Betriebssystem wie Windows oder Linux als Plattform. Dort können OpenGL- oder TCP- Schnittstellen implementiert, und so Rechenserver zur numerischen Analyse genutzt werden. Beide Varianten sind unter o. g. Forschungsarbeiten zu finden.

Die o. g. Ansätze zur serverbasierten Finite-Elemente-Analyse behandeln nicht die interaktive Arbeitsweise, die heute besser zur typischen Arbeitsweise im Tragwerksentwurf passt (mehrere

[24] XML-RPC: Extensible Markup Language – Remote Procedure Calling

[25] Signatur: Eindeutige Kombination aus Name und Übergabeparametern

[26] OpenGL: Open Graphics Library, Industriestandard zur Darstellung dreidimensionaler grafischer Objekte.

[27] http://www.khronos.org/webgl/, Abruf: 05.11.2012

Iterationszyklen anstelle einer einzigen Berechnung). Diese Fähigkeit, ein serverbasiertes Analysemodell inkrementell zu aktualisieren, wird in Abschnitt 3.3.5 behandelt.

2.8.3 Parallelisierte Gleichungslöser

Eine serverseitige Simulation eröffnet prinzipiell die Möglichkeit der gleichzeitigen Nutzung vieler Computer bzw. CPUs zur Beschleunigung einer Simulation, da die Hardware dann bekannt ist und von einer größeren Anzahl CPUs ausgegangen werden kann. Bei einer rechenintensiven Aufgabe ist es besser, eine große Anzahl CPUs über eine kurze Zeitspanne zu nutzen, anstelle weniger CPUs über eine lange Zeitspanne. Eine Voraussetzung dafür ist die Parallelisierbarkeit der Rechenaufgabe, d. h. die Möglichkeit der Aufteilung in Teilaufgaben, die unabhängig voneinander gelöst werden können. Dies ist im Falle einer Finite-Elemente-Simulation nicht ohne weiteres möglich, da herkömmliche, direkte Gleichungslöser in jedem Rechenschritt die Ergebnisse des vorigen Schrittes benötigen. Iterative Gleichungslöser können zwar parallel (gleichzeitig auf verschiedenen Recheneinheiten) eingesetzt werden, jedoch ist der Aufwand dafür erheblich.

Da ein substanzieller Fortschritt auf diesem Gebiet nur durch eine Fokussierung auf diesen Aspekt erreicht werden kann, werden Techniken der Parallelisierung in der vorliegenden Arbeit nicht aufgegriffen, sondern auf Arbeiten wie [Chen, Lin 2008] verwiesen. Darin wird eine Systemarchitektur zur webbasierten Finite-Elemente-Analyse vorgestellt. Client-Anwendungen sind dabei in Java geschrieben, das serverseitige FEA-Framework dagegen in C++. Mittels MPI[28] wird das Lösen der Gleichungssysteme auf mehrere Maschinen verteilt. Die Eingabedaten werden über eine Socket-Verbindung (Java Socket-Klasse) als Eingabedatei an den Server gesendet. Ergebnisse werden im nativen Format des FEA-Frameworks über den Socket zurück an den Client gesendet und dort visualisiert.

2.8.4 GPU als Co-Prozessor einer numerischen Simulation

Die Grafik-Hardware (GPU[29]) von PCs hat sich in den letzten Jahren sowohl hinsichtlich der Geschwindigkeit, als auch in der Programmierbarkeit stark weiterentwickelt. Ursprünglich wurden die Daten (Vertices, Farben pro Vertex, Normalenvektoren pro Vertex, Vertex-Indizes für Dreiecke, Texturen) in einer sog. *Fixed-Function-Pipeline* verarbeitet, also in einer festen Reihenfolge mit festgelegten Algorithmen. 2004 wurde dann GLSL[30] als Programmiersprache für GPUs eingeführt, die Fixed-Function-Pipeline wurde damit zu einer wesentlich flexibleren Render-Pipeline, in der verschiedene Stufen frei programmiert werden können.

Heute können GPUs durch Programmierschnittstellen wie CUDA[31] oder OpenCL[32] nicht nur zur Berechnung von Grafiken verwendet werden, sondern für allgemeine Rechenaufgaben. GPUs, die diese Schnittstellen unterstützen, werden daher auch als *General Purpose GPU* (GPGPU) bezeichnet.

Solche GPGPUs eignen sich zur Beschleunigung einer Finite-Elemente-Analyse. In [Göddecke et al. 2005] wird die Nutzung von CPU und GPU in Kombination beschrieben, um FE-Gleichungslöser zu

[28] MPI: Message Passing Interface. Standard für die Kommunikation bei parallelen Berechnungen auf verteilten Computersystemen.

[29] GPU: Graphics Processing Unit

[30] GLSL: OpenGL Shading Language

[31] CUDA: Compute Unified Device Architecture, von NVidia entwickelt

[32] OpenCL: Open Cumputing Language, spezifiziert von der Khronos Group

beschleunigen. Es wurden insbesondere die Auswirkungen auf die Datengenauigkeit untersucht und der Einfluss der Datengenauigkeit auf die Skalierung[33]. Eine höhere Datengenauigkeit wird auf einer GPU (natürlich auch auf einer CPU) mit einer geringeren Geschwindigkeit erkauft. Da GPUs im Allgemeinen auf Berechnungen mit einfacher Genauigkeit (Datentyp *float*) ausgelegt sind, für numerische Simulationen aber meist doppelte Genauigkeit (Datentyp *double*) erforderlich ist, wird die theoretische Anzahl der FLOPS[34] einer GPU ungefähr halbiert. Dennoch sind aktuelle GPUs in der Geschwindigkeit auch modernen Mehrkern-CPUs gegenüber im Vorteil, der Aufwand zum Einsatz von GPUs für numerische Simulationen ist jedoch erheblich.

2.8.5 Hybride Parallelisierung

Die Nutzung von GPUs zur numerischen Simulation steht nicht im Wiederspruch zu o. g. Konzept der serverbasierten Simulation.

Eine GPGPU kann clientseitig wie auch serverseitig für numerische Berechnungen genutzt werden. Clientseitig besteht jedoch der Nachteil, dass die GPU normalerweise bereits für die herkömmliche Grafik-Berechnung teilweise oder ganz ausgelastet ist.

Bei einem serverseitigen Einsatz der GPU besteht der Vorteil darin, dass die Anzahl der GPUs und der genaue Typ bekannt sind, was bei der GPU-Programmierung nahezu zwingend erforderlich ist.

Eine Kombination verschiedener Methoden der Parallelisierung wird als hybride Parallelisierung bezeichnet:

- Aufteilung auf mehrere Rechenkerne einer CPU (z.B. mittels OpenMP)
- Aufteilung auf mehrere Rechenkerne einer GPU als Co-Prozessor (z.B. mittels Cuda oder OpenCL)
- Aufteilung auf mehrere Computer (z.B. mittels MPI)

Für Details zur hybriden Parallelisierung wird auf [Mahinthakumar 2002] verwiesen.

[33] Die Skalierung ist ein Maß für die Beschleunigung einer parallel ausgeführten Berechnung. Die Skalierung ist immer kleiner als die Anzahl der parallelen Recheneinheiten, da ein gewisser Overhead der Verwaltung, sowie für die Aufteilung der Prozesse (Fork) und die Zusammenführung der Ergebnisse (Join) entsteht.

[34] FLOPS: Floating Point Operation per Second

3 Problemstellung und Lösungsansatz

> "Wenn ich die Menschen gefragt hätte, was sie wollen, hätten sie gesagt: schnellere Pferde." Henry Ford

3.1 Forschungsbedarf

Wie aus den Abschnitten 2.1 und 2.2 hervorgeht, sind CAD und (numerische) Analyse, architektonischer Entwurf und Tragwerksentwurf unzureichend miteinander gekoppelt.

Ein wesentlicher Grund ist, dass beide Teilbereiche ursprünglich und lange Zeit unabhängig voneinander entwickelt wurden. Ein weiterer Grund ist, dass beide Bereiche jeweils hohe Anforderungen an die Hardware-Ressourcen haben, und damit schwierig zu kombinieren sind.

Bei der geometrischen Modellierung von Bauwerken (CAD) ist es selbstverständlich, die Auswirkungen von Operationen (Eingaben, Änderungen) oder die Resultate boolescher Operationen[35] *sofort* zu sehen, um die Konsequenzen abschätzen zu können. Bei der numerischen Simulation ist es bis heute ebenso selbstverständlich, Ergebnisse erst nach einem mehr oder weniger langwierigen Prozess mit verschiedenen Programmfenstern und Nutzerabfragen zu erhalten. Änderungen erfordern jeweils das erneute Durchlaufen dieses sequentiellen Arbeitsprozesses, wodurch der intuitive Zusammenhang zwischen Änderung und Auswirkung weitgehend verloren geht.

Dem Anwender wird somit ein sequentieller Arbeitsablauf aufgezwungen, weil die Software so konzeptioniert ist bzw. historisch so gewachsen ist. Vorhandene Schnittstellen zu FEA-Frameworks sind immer noch üblicherweise Batch-orientiert, also mit Ein- und Ausgabedateien und vorgeschriebenem Ablauf (Abbildung 11).

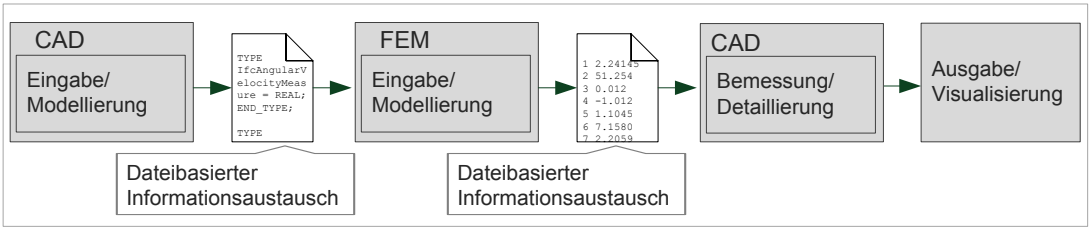


Abbildung 11: Traditioneller, sequentieller Berechnungsablauf

Die Notwendigkeit einer engeren funktionalen Kopplung zwischen den Softwarekomponenten zum Entwurf und zur Analyse ist somit offensichtlich, wenn der Arbeitsablauf interaktiv statt sequentiell sein soll. Die heutigen Möglichkeiten einer engeren Kopplung hinsichtlich einer interaktiven Arbeitsweise sollen im Folgenden diskutiert werden.

Ziel einer interaktiven Entwurfsraum-Exploration ist eine Simulation und Visualisierung der Ergebnisse mit möglichst geringen Verzögerungen und Brüchen im Arbeitsfluss, damit die Auswirkungen von Entwurfsveränderungen intuitiv abgeschätzt werden können. Dies führt mit hoher Wahrscheinlichkeit zu einem verbesserten Entwurf. Darüber hinaus ergibt sich ein intuitiver Lernprozess durch die Verdeutlichung der Konsequenzen verschiedener Entwurfsvarianten.

[35] Subtraktion, Addition oder Durchschnitt beliebiger dreidimensionaler Körper

3.2 Softwaretechnische Möglichkeiten zur Kopplung von Entwurf und Simulation

Die funktionale Kopplung der Entwurfs- und Simulationsanwendung muss möglichst eng sein, um eine direkte Rückkopplung der Ergebnisse der numerischen Simulation zu gewährleisten. Gleichzeitig soll aber die softwaretechnische Anbindung möglich lose sein, damit beide Komponenten (Entwurf und Simulation) unabhängig voneinander für jeweils am besten geeignete Plattformen entwickelt werden können.

Grundsätzlich gibt es unterschiedliche Konzepte zur Kopplung verschiedener Anwendungen:

1. Die komplette monolithische Verschmelzung des Quellcodes beider Anwendungen in einer neuen Anwendung.
2. Einbindung der FE-Analyse über eine API zur Laufzeit.
3. Separate Anwendungen, Kommunikation über Dateien.
4. Separate Anwendungen, Kommunikation über eine Netzwerkschnittstelle.

Die beiden ersten Möglichkeiten stellen die engste softwaretechnische Kopplung dar, mit den entsprechenden Vor-und Nachteilen. Geringe Antwortzeiten sind möglich, solange die Modellgrößen überschaubar und die Rechenzeiten kurz sind. Längere Rechenzeiten können die gesamte Anwendung blockieren, numerische Probleme in der Berechnung oder unsauberes Fehlermanagement[36] wirken sich auf die gesamte Anwendung aus.

Die 3. Variante ist die traditionelle und bis heute am häufigsten verwendete Schnittstelle in der numerischen Simulation. Die Kommunikation über Dateien erlaubt keine direkten Kommandos/Nachrichten[37] und die Geschwindigkeit ist relativ gering. Somit ist dieses Verfahren für eine interaktive Arbeitsweise nicht gut geeignet.

Die 4. Variante erfüllt alle Anforderungen an eine Softwareumgebung für eine interaktive Entwurfsraum-Exploration von Strukturmodellen. Nachrichten oder Kommandos können ebenso wie große Datenmengen direkt übermittelt werden, bei sehr geringen Verzögerungen. Den grundsätzlichen Aufbau einer Softwareumgebung mit einer solchen Schnittstelle zeigt Abbildung 12.

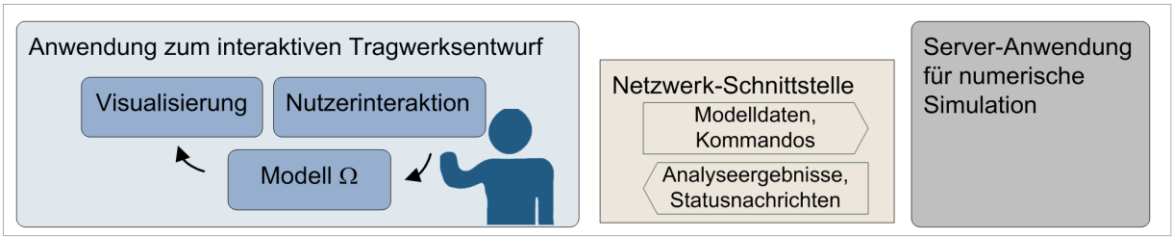


Abbildung 12: Entwurfsumgebung mit Schnittstellen zu einem Server für numerische Simulation

Zur Funktion der Netzwerk-Schnittstelle (Abbildung 12) ist die Definition eines Protokolls erforderlich. Die Entwicklung einer solchen Definition wird im Abschnitt 3.3.3 vorgenommen.

[36] Leider nicht unüblich bei Bibliotheken zur numerischen Simulation

[37] Etwa die Nachricht der Simulationsanwendung: “Analyse beendet. Ergebnisse stehen bereit“ kann zwar in eine Datei geschrieben werden, jedoch nicht direkt an die Entwurfsanwendung übermittelt werden.

3.3 Netzwerkschnittstelle zur numerischen Simulation

3.3.1 Anforderungen

Eine Schnittstelle zur numerischen Simulation muss verschiedene Informationen übertragen können. Hierzu gehören vor allem:

- Modelldaten und Kommandos vom Client zum Server.
- Berechnungsergebnisse und Statusmeldungen vom Server zum Client.

Das hier vorgeschlagene Konzept soll alle Datenübertragungen in einer einheitlichen Schnittstelle zusammenfassen, muss also bidirektional sein. Dadurch sollen Zugriffe auf Berechnungsergebnisse, Fehlermeldungen oder andere Informationen über zusätzliche, separate Schnittstellen wie SQL oder andere Datenbankschnittstellen vermieden werden.

Zur Spezifikation einer Schnittstelle ist ein Protokoll erforderlich, das den Informationsaustausch regelt. Ein solches Protokoll zur interaktiven numerischen Simulation muss folgende Eigenschaften aufweisen:

- **Inkrementell**: Fähigkeit, Entwurfsänderungen (hinzufügen/ändern/löschen) auch einzelner Elemente zu verarbeiten, ohne jeweils eine Eingabe des gesamten Modells vorzunehmen (wie es bei der klassischen Eingabedatei der Fall ist).
- **Einheitlich**: eine bidirektionale Schnittstelle für Modelleingabe, Kommandos, Berechnungsergebnisse und Statusmeldungen.
- **Interoperabel**: Datenformat für Modelleingabe, Kommandos, Berechnungsergebnisse und Statusmeldungen sollten auf einer offenen Spezifikation basieren.
- **Unabhängigkeit und Neutralität**: beliebige Client-Anwendungen, auf Basis üblicher Programmiersprachen, die offene Netzwerkstandards unterstützen, sollten in der Lage sein, die Schnittstelle zu nutzen.
- **Netzwerkfähigkeit**: Die Schnittstelle sollte lokal und über ein Netzwerk nutzbar sein.

Die Randbedingungen für eine Protokolldefinition sollen nun näher erläutert werden.

3.3.2 Netzwerkschichten

Netzwerkprotokolle sind üblicherweise in einer mehrschichtigen Architektur konzipiert. Ein weit verbreitetes Beispiel ist hier das OSI-Referenzmodell[38], Details siehe [Gumm, Sommer 2002] oder [DIN EN ISO/IEC 1994]. Die Aufteilung der Funktionalität in Schichten von der Bitübertragung über Sicherung, Vermittlung (z.B. IP[39]), Transport (z.B. TCP[40]), bis hin zu Protokollen der Anwendungsschicht (z.B. HTTP), hat den Vorteil, dass zur Definition eines neuen Protokolls sehr einfach auf eine beliebige vorhandene Schicht aufgesetzt werden kann.

Für das Protokoll zur interaktiven numerischen Simulation bietet sich hier TCP als Transportschicht an. TCP ist eine Netzwerkschicht zur zuverlässigen, bidirektionalen Übertragung beliebiger Daten zwischen zwei sog. Sockets (IP-Adresse in Kombination mit Port-Nummer).

[38] OSI: „Open Systems Interconnection Reference Model", spezifiziert durch die ISO. Unter anderen basiert das "Hypertext Transfer Protocol" (HTTP) auf dem OSI-Referenzmodell.

[39] Internet Protocol

[40] Transmission Control Protocol

Zur Übermittlung von Modelldaten, Kommandos und Statusnachrichten erscheint zunächst die Verwendung noch höherer Netzwerkschichten, etwa von bestehenden Middleware-Bibliotheken wie CORBA[41] sinnvoll.

Dadurch ergeben sich jedoch Abhängigkeiten von der serverseitigen Implementierung, deshalb ist dieses Konzept besser geeignet als Insellösung zwischen genau einem Client und einem Server, aber nicht für eine offene Protokolldefinition. Auch kann niemand voraussagen, ob eine bestimmte Bibliothek in Zukunft (etwa in neuen Programmiersprachen) weiter unterstützt und problemfrei lizenziert wird.

Für ein offenes und wirklich plattformunabhängiges Protokoll für alle gegenwärtigen und zukünftigen Betriebssysteme und Programmiersprachen wird hier XML[42] als beste Basis angesehen. XML bedarf nicht einmal einer speziellen Unterstützung durch eine Programmiersprache oder Betriebssystem, da es sich um menschen- und maschinenlesbaren Text handelt, dessen Verarbeitung elementarer Bestandteil praktisch jeder Programmiersprache ist.

Somit wird für die weiteren Ausführungen XML über TCP als technische Basis für das Protokoll zur interaktiven Simulation gewählt.

3.3.3 Protokolldefinition

Die reguläre XML-Syntax besteht zunächst aus dem eigentlichen Inhalt in Kombination mit Auszeichnungselementen:

```
<EinzelElement />
<AuszeichnungsElement attribut1="wert1" attribut2="wert2">Inhalt</ AuszeichnungsElement>
<TagName><![CDATA[Ungeparster Inhalt]]></TagName>
```

Auszeichnungselemente müssen entweder ein sog. Empty-Element-Tag sein (Zeile 1 in obiger Auflistung), oder durch einen End-Tag mit gleichlautendem Tag-Name geschlossen werden (Zeile 2).

Innerhalb von XML gibt es Abschnitte, die der XML-Syntax unterliegen (hierarchische Untergliederung in Auszeichnungs-Abschnitte, Codierung von Sonderzeichen), daher werden diese Abschnitte als PCDATA (parsed character data) bezeichnet. Die dritte Zeile in der obigen Auflistung enthält auch einen Abschnitt mit Daten, die eine beliebige Struktur haben können. Diese werden als (unparsed) character data, CDATA, bezeichnet.

Diese CDATA-Abschnitte eignen sich zur Übertragung von Modelldaten oder Berechnungsergebnissen in einem geeigneten Format. Im Folgenden sind einige Beispiele für Datenpakete der Simulations-Schnittstelle aufgelistet (vollständige Spezifizierung im Anhang A):

```
<Connect customerId="…" password="" />
<ModelAdd><![CDATA[…]]></ModelAdd>
<ModelChange><![CDATA[…]]></ModelChange>
<ModelRemove><![CDATA[…]]></ModelRemove>
<AnalysisRun />
<AnalysisResultGet nodeId="all" resultType="displacement" />
<AnalysisResultGet elementId="all" resultType="engineering strain" />
```

Hier wird ersichtlich, dass die Modelleingabe, Ergebnisabfrage und andere Kommandos an den Server in einer einheitlichen Datenstruktur abgebildet werden können.

[41] Common Object Request Broker Architecture

[42] XML: Extensible Markup Language, human-readable and machine-readable, versatile document format

Auf alle Kommandos des Clients erfolgt eine entsprechende Antwort des Servers:

```
<ConnectResponse sessionId="…" />
<ModelAddResponse />
<ModelChangeResponse />
<ModelRemoveResponse />
<AnalysisRunResponse />
<AnalysisResultGetResponse><![CDATA[…]]></AnalysisResultGetResponse>
<StatusMessage type="error">missing element<StatusMessage/>
```

Statusmeldungen (letzte Zeile der obigen Auflistung) nehmen insofern eine Sonderstellung ein, dass keine Anfrage des Clients vorliegt. Dennoch können sie in der gleichen Datenstruktur übertragen und verarbeitet werden.

Auf dieses Protokoll wird im Folgenden unter der Bezeichnung IRAI (*Incremental Revision Analysis Interface*) Bezug genommen. Eine Übersicht über alle Auszeichnungselemente und Attribute findet sich in Anhang A.

3.3.4 Übertragung von Modell-Daten

Zunächst wurde im Rahmen dieser Arbeit eine binäre Codierung zur Übertragung von Modell-Daten wie Knoten, Elementen etc. vorgenommen. Dadurch konnte einfach ein Array von reellen Zahlen mittels blockweisen Kopierens in den Textstrom eingefügt, und serverseitig wieder in ein Array zurück-kopiert werden. Dies funktionierte zwar gut, verletzte aber die Forderung der Plattformunabhängigkeit, da binäre Daten in verschiedenen Sprachen unterschiedlich repräsentiert sein können.

Modelldaten oder Berechnungsergebnisse können auch vollständig in XML-Notation formatiert werden, was jedoch zu großen und für Menschen schlecht lesbaren und unübersichtlichen Datenstrukturen führt.

Für die Modellierung von Produktmodell-Daten steht jedoch ein bereits etablierter, offener und relativ einfach umsetzbarer Standard zur Verfügung. Dieser Standard heißt STEP (vgl. Abschnitt 2.7.5).

Ähnlich wie bei XML definiert STEP zunächst nur ein bestimmtes Format zum Datenaustausch, die Semantik der Daten wird für verschieden Anwendungsbereiche durch ein Schema vorgegeben.

STEP-Daten sind grundsätzlich nach folgendem Prinzip aufgebaut:

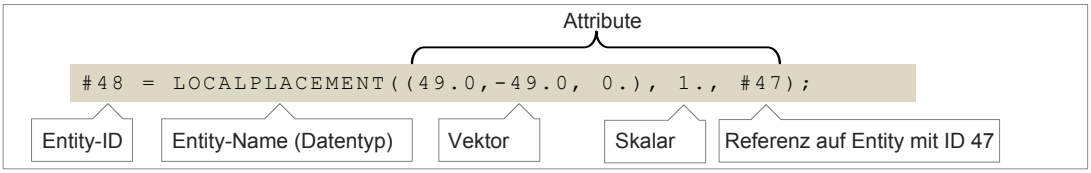


Abbildung 13: Aufbau einer STEP-Datenzeile

Das Schema für die STEP-Daten der hier zu definierenden Schnittstelle wird im Folgenden SFA (*STEP For Analysis*) benannt.

Das untenstehende Beispiel zeigt die Übertragung eines Kontrollpunktes und einer Einzellast als Argumente eines *ModelAdd*-Kommandos.

```
<ModelAdd><![CDATA[
#48= NODE((49.0,-49.0, 0.),1.);
#49= NODALLOAD(#48,(0.,0.,20.));
]]></ModelAdd>
```

Das vollständige SFA-Schema findet sich in Anhang B.

SFA ist an IFC angelehnt, jedoch wesentlich einfacher gehalten, mit Berücksichtigung spezieller Anforderungen der numerischen Analyse.

Eine Transaktion von IFC- oder allgemein STEP-Daten erfordert die Definition eines bestimmten Kontextes, wie in [Halfawy, Froese 2002] beschrieben:

> *„Moving beyond ad-hoc transactions.* While the IFC model standardizes the information content of an information exchange transaction, it offers no guidance to the context of these transactions. It is still left up to the two parties exchanging information to come up with ad-hoc agreements about what data are being exchanged, for what business purpose, with what constraints and obligations on each participant, etc. We are pursuing the formalization and possible standardization of data exchange protocols to support IFC-based transactions in distributed and heterogeneous environments. An accompanying paper in this proceedings focuses on the process of developing transaction-based interoperability protocols.“

Der erwähnte begleitende Artikel findet sich in [Froese 2003].

Im hier vorgeschlagenen Kommunikationsprotokoll wird dieser Kontext durch die XML-Datenstruktur erreicht, konkret im obigen Beispiel durch den XML-Tag *ModelAdd*.

3.3.5 Übertragung inkrementeller Modell-Änderungen

Eine Schnittstelle zur interaktiven numerischen Simulation muss in der Lage sein, Änderungen im Modell inkrementell zu übermitteln, wie zu Beginn des Kapitels erläutert.

Analog einem sog. „Diff“ auf Dateiebene (Abbildung 14) kann durch oben definiertes Protokoll eine Änderung eines Entity im STEP-Format übertragen werden.

```
92 - material->setSpecular( osg::Material::FRONT, Vec4f( 0.2f, 0.25f, 0.3f, 0.3f ) );
92 + material->setSpecular( osg::Material::FRONT, Vec4f( 0.4f, 0.3f, 0.03f, 0.3f ) );
```

Abbildung 14: Diff einer Quellcode-Datei

Anstelle der Zeilennummer (Abbildung 14 links) ist in STEP ohnehin die Entity-ID vorhanden (Abbildung 13). Anstelle des Minus- und Pluszeichens zum Entfernen und Hinzufügen einer Zeile (Abbildung 14, nach Zeilenangabe), wird der Kontext für die STEP-Modelldaten durch verschiedene XML-Tags definiert (Abbildung 15).

```
<ModelDataAdd><![CDATA[     #48= NODE((49.0,-49.0),1.);  ]]></ModelDataAdd>
<ModelDataChange><![CDATA[  #48= NODE((47.3,-49.0),1.);  ]]></ModelDataChange>
<ModelDataRemove><![CDATA[  #48;                         ]]></ ModelDataRemove >
```

Abbildung 15: Datenpakete zum Hinzufügen, Ändern und Löschen von Modelldaten

Theoretisch könnte der Kommando-Tag *ModelChange* vermieden werden, indem *ModelAdd* so interpretiert wird, dass wenn eine ID bereits vorhanden ist, und der Datentyp übereinstimmt, das betreffende Objekt mit den übertragenen Daten aktualisiert wird.

Möglich wäre auch, ein *Remove*-Kommando durch eine negative ID zu definieren, dann gäbe es nur einen Tag-Namen, um *Add*,- *Change*,- und *Remove*-Kommandos zu interpretieren.

Dieses Vorgehen würde aber die eigentliche Intention nur implizit zum Ausdruck bringen. Gewählt wurde hier die explizite und selbsterklärende Formulierung der Intention wie in Abbildung 15 dargestellt.

3.3.6 Asynchrone Simulation

Eine Simulation und Visualisierung in Echtzeit[43], wie in der numerischen Simulation in Computerspielen [Mezger et al. 2007], ist für Strukturmodelle des Ingenieurwesens nicht immer möglich, trotz Parallelisierung und enormer Leistungszuwächse der Hardware in den letzten Jahrzehnten.

Daher besteht der Anspruch für interaktive Simulation in dieser Arbeit darin, Analyse und Visualisierung so verzögerungsfrei wie möglich zu realisieren. Mit der Annahme, dass der entwerfende Ingenieur mit der Arbeit fortfahren möchte während eine Simulation läuft, ergibt sich zwangsläufig eine **asynchrone** Simulation. Dadurch ist es möglich, dass Analyseergebnisse bereits hinfällig sind, sobald sie zur Verfügung stehen. Eine Anwendung muss also vor der Umsetzung der Ergebnisvisualisierung prüfen, ob in der Zwischenzeit Modelländerungen eingetreten sind. Sinnvoll ist auch eine Möglichkeit zur Unterbrechung einer Analyse, wenn größere Modelländerungen eine komplette Neuberechnung erfordern.

3.3.7 Geometrisches Modell der IRAI-Schnittstelle

Grundsätzlich kann ein IFC-Modell direkt in ein Finite-Elemente-Modell konvertiert werden, wie in [Romberg et al. 2004] beschrieben. Somit kommt theoretisch ein IFC-Modell auch als Basis für die serverseitige Analyse in Betracht.

Allerdings ist das allgemeine Geometriemodell in IFC sehr komplex, was zu der Situation führt, dass Analyseergebnisse nicht immer eindeutig der ursprünglichen Geometrie zugeordnet werden können.

Als Beispiel ist in Abbildung 16 eine Stütze dargestellt, die im IFC-Modell durch Grundfläche und Extrusionsvektor gegeben ist.

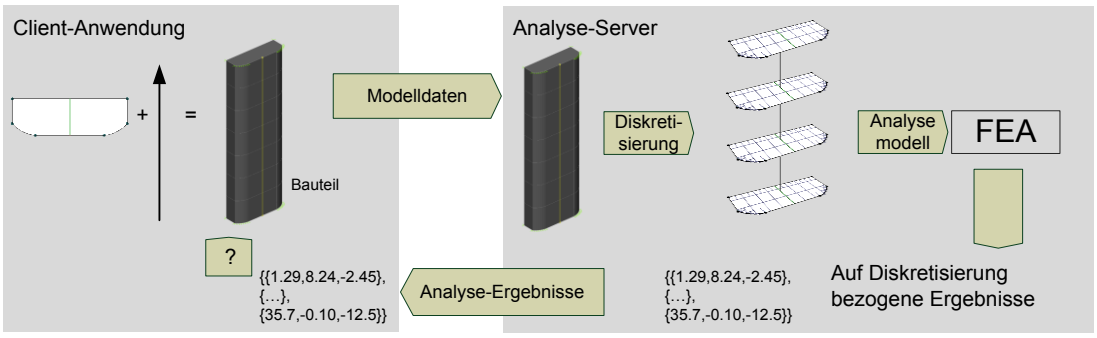


Abbildung 16: Zuordnungsproblem bei serverseitiger Diskretisierung

Serverseitig ist nun eine Diskretisierung erforderlich, da der Querschnitt durch Integrationspunkte im Analysemodell abgebildet werden muss.

[43] Der Begriff „Echtzeit“ meint in diesem Kontext die Aktualisierung per-Frame, also in jedem Aktualisierungszyklus der Eingabe und Visualisierung

Die Lage dieser Integrationspunkte und die Parameter zur Diskretisierung sind clientseitig nicht bekannt, sodass auch die Zuordnung der Ergebnisse zum ursprünglichen Modell (z.B. für die Aktualisierung der Grafik für eine Farbcodierung der Spannungen oder Dehnungen) nicht möglich ist.

Aus diesem Grund wird hier vorgeschlagen, die Diskretisierung von Querschnitten clientseitig durchzuführen. Somit benötigt das SFA-Schema nur eine einzige Art der Querschnittsrepräsentation (anstatt dutzender verschiedener Varianten wie in IFC). Diese wird im nächsten Kapitel näher erläutert.

3.4 Interoperabilität

Die Interoperabilität ist ein weiterer wichtiger Aspekt einer Entwurfsumgebung. Automatisierter Datenaustausch mit Anwendungen u. a. für die architektonische Modellierung wird immer wichtiger, da komplexe Projekte heutzutage kaum mehr durch den herkömmlichen Informationsaustausch über 2D-Pläne als Datei, oder gar in Papierform, bewältigt werden können. Diese Art von Datenaustausch stellt einen Bruch im Arbeitsfluss dar, durch die manuelle Neueingabe des gesamten Modells.

Eine vielversprechende Perspektive bietet das Konzept der Produktmodellierung bzw. des „Building Information Modeling“ (BIM), [Eastman et al. 2011], wie in Abschnitt 2.7 bereits näher erläutert. Dieser Ansatz gewinnt in der Forschung und auch in der Bauindustrie durch die *Industry Foundation Classes* (IFC) zunehmende Akzeptanz.

Der Produktmodell-Standard IFC soll als Lösungsansatz für die Interoperabilität der Entwurfsumgebung in dieser Arbeit zum Einsatz kommen. Im Gegensatz zur Schnittstelle für die numerische Simulation ist hier also keine neue Definition eines Schemas oder Protokolls erforderlich, sondern es sollen lediglich vorhandene Standards genutzt werden.

Die im folgenden Kapitel beschriebene prototypische Entwurfsanwendung verwendet intern ein aus verschiedenen Komponenten bestehendes Klassenmodell, das in ein IFC-Modell importiert und exportiert werden kann.

Dafür ist ein vollständiges IFC-Klassenmodell erforderlich. Dieses IFC-Klassenmodell ist in der EXPRESS-Datenmodellierungssprache definiert, ebenso wie das Datenmodell der IRAI-Schnittstelle (vgl. vorherige Abschnitte). Daher liegt es nahe, die Generierung von Quellcode aus EXPRESS Schema-Dateien zu automatisieren, anstelle einer Umsetzung von Hand. Auch die Anzahl von über 1000 Type- und Entity-Datentypen in IFC4 spricht für eine automatisierte Quellcode-Generierung.

Eine weitere Gemeinsamkeit zwischen der IFC- und der IRAI-Schnittstelle ist das Datenformat STEP[44]. Die Übersetzung von STEP-Daten in ein Objektmodell zur Laufzeit (Parsen) und das Erzeugen von STEP-Daten aus einem Objektmodell ist hier für beide Schnittstellen notwendig und wird durch den gleichen Parser-Generator realisiert. Details zur Umsetzung und Implementierung folgen im anschließenden Kapitel.

Es ergeben sich also zwei wesentliche Schnittstellen der hier konzipierten interaktiven Entwurfsumgebung:

[44] IFC-Daten können auch im XML-Format codiert werden, was jedoch zu erheblich größeren Dateien führt, und in der Praxis eher unüblich ist.

- Eine Schnittstelle zur Kommunikation mit einem Server für numerische Simulation (Abbildung 12), und
- Eine Schnittstelle zur interoperablen Kommunikation mit anderen Anwendungen im Rahmen eines Bauprojektes (Abbildung 17).

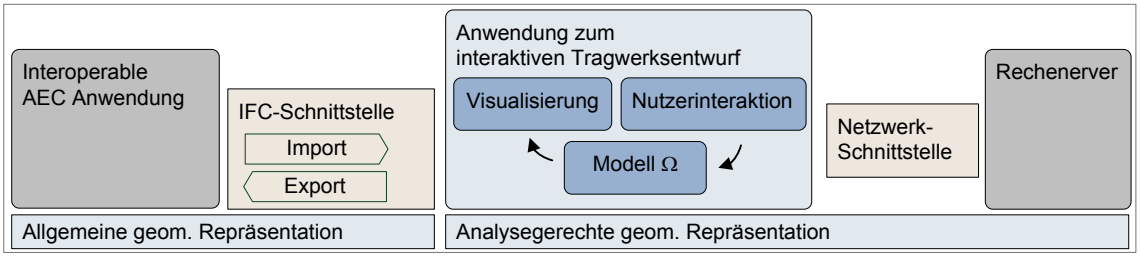


Abbildung 17: Entwurfsumgebung mit Schnittstellen zu anderen Anwendungen und zum Analyse-Server

In den folgenden Kapiteln soll die Konzeption dieser beiden wesentlichen Schnittstellen der Entwurfsumgebung im Detail entwickelt werden. Trotz der Unterschiedlichkeit der beiden Schnittstellen kann STEP als gemeinsames Datenformat genutzt werden. Somit können viele Software-Komponenten in beiden Bereichen genutzt werden, wie in den folgenden Abschnitten erläutert wird.

4 Umsetzung einer prototypischen Entwurfsumgebung

„Achieving ever-higher degrees of interactivity is worthwhile in its own right“ R. I. Mackie

4.1 Konzeption

Das Ziel der Umsetzung einer prototypischen Entwurfsumgebung schließt eine grafische Eingabe und Visualisierung ein. Zur grafischen Eingabe bieten sich zunächst existierende CAD-Programme wie AutoCAD an, die durch Plugins erweitert werden können. Leider ist damit jedoch keine interaktive Simulation möglich, da die Ergebnisdaten einer numerischen Simulation (FEA) ungeeignet sind für die (im Vergleich zu einem direkten OpenGL-Kontext erheblich eingeschränkten) Plugin-Schnittstellen von CAD-Programmen. Somit müssten die Ergebnisse in externen Anwendungen außerhalb der Modellierungsanwendung visualisiert werden, wodurch die Interaktivität verloren gehen würde, ähnlich wie bei zu langsamer Rückkopplung mit manuellen Zwischenschritten, wie bereits in Kapitel 3 gezeigt wurde.

Zur Umsetzung wird hier somit eine Softwarearchitektur vorgestellt, mit direkt eingebundenem OpenGL-Kontext und einer Netzwerkschnittstelle, hinter der serverseitig die numerische Simulation angeordnet ist (Abbildung 18). Die Serveranwendung kann auf der gleichen Maschine instanziiert werden wie die Entwurfsanwendung, oder auf einem speziell für numerische Berechnungen ausgelegten Server.

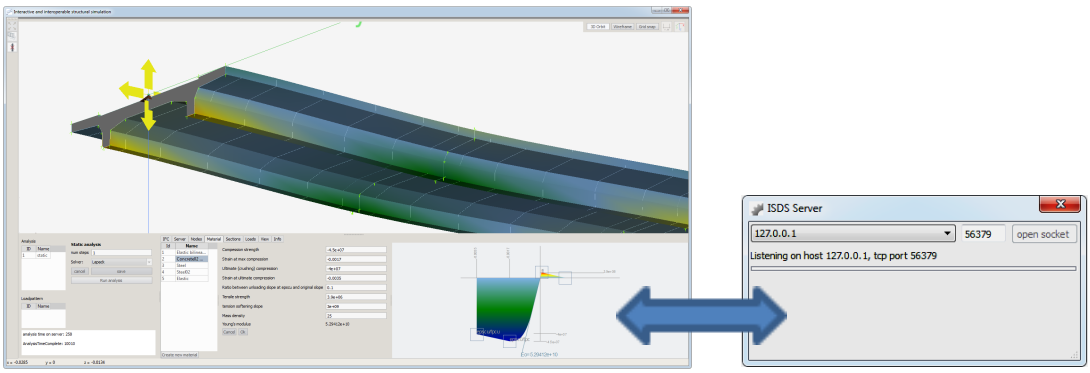


Abbildung 18: Anwendung zum Tragwerksentwurf, Schnittstelle zur numerischen Simulation

Auch die Unterstützung von IFC ist gegenwärtig in den etablierten CAD-Programmen üblicherweise mangelhaft[45]. Da die Interoperabilität im Sinne eines automatisierten und reibungslosen Datenaustausches zwischen Architekt, Ingenieur und anderen Fachplanern in diesem Kontext als wichtig angesehen wird, und außerdem das STEP-Format auch für die Kommunikation mit dem Simulations-Server eingesetzt werden sollte (vgl. Abschnitt 3.3.4), kam hier nur eine eigene Implementierung in Frage.

Serverseitig wird zur Finite-Elemente-Analyse OpenSees[46] [McKenna 1997], [Haukaas, Kiureghian 2007] verwendet.

[45] Selbst durchgeführte Versuche zum IFC-Export und Import von Modellen mit AutoCAD 2010 und Revit Structure 2012 ergaben erhebliche Probleme. So waren etwa in Revit die Koordinatensysteme ganzer Stockwerke nach dem Export und Re-Import um mehrere Meter verschoben.

[46] http://opensees.berkeley.edu, Abruf 02.10.2012

OpenSees (Open System for Earthquake Engineering Simulation) ist ein Software-Framework das an der UC Berkeley entwickelt wird. OpenSees wird hauptsächlich in der Erdbebenforschung eingesetzt, daher auch der Fokus auf Dynamik und nichtlineare Materialmodelle mit Berücksichtigung der Verformungs- und Schädigungshistorie. Der C++/Fortran Quellcode ist kostenlos verfügbar mit build-Skripten für verschiedene Zielplattformen wie Windows, Linux, OS X. Die Integration und Anpassung dieser Software in die Server-Anwendung wird in Abschnitt 4.4 beschrieben.

4.2 Clientseitige Implementierung

Die hier umgesetzte prototypische interaktive Modellierungsumgebung verfügt über eine interoperable IFC-Schnittstelle zum Import und Export von Bauwerksmodellen (Abbildung 19 unten Mitte).

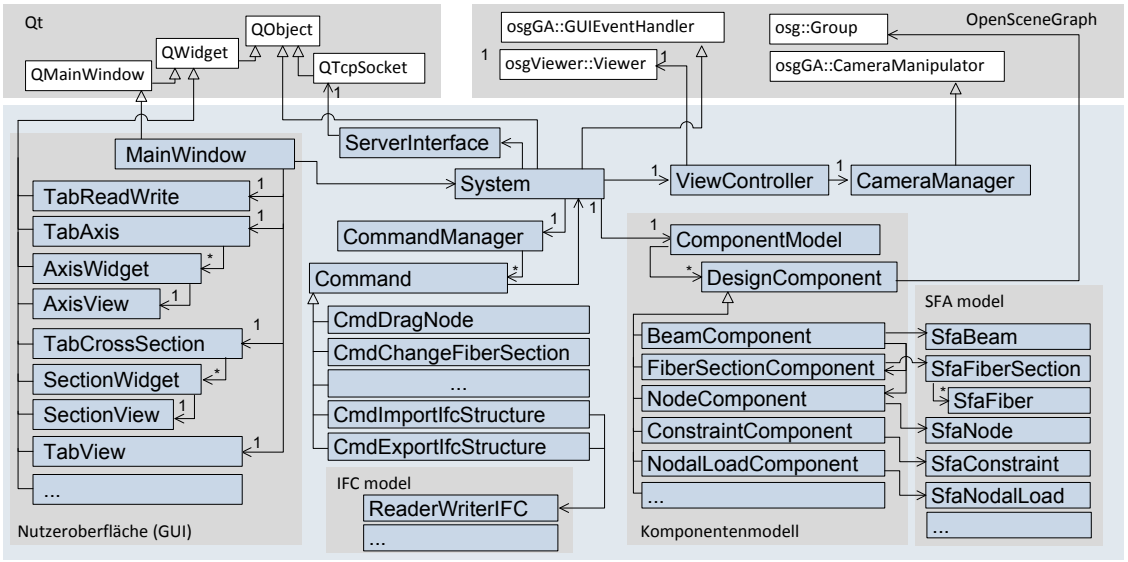


Abbildung 19: UML-Diagramm der wichtigsten Klassen der Entwurfs-Umgebung

Abbildung 19 zeigt eine Übersicht über die wichtigsten Klassen der clientseitigen Implementierung der Entwurfsumgebung. Es wurde eine konsequente Trennung des Codes für die Nutzeroberfläche (GUI-Graphical User Interface), das Komponentenmodell, und die Kommandos zur Änderung des Modells eingehalten. Die Klasse *System* steht im Mittelpunkt der Anwendung und hält Referenzen auf das Modell, den Kommando-Manager, und den View-Controller. Dagegen bestehen keine Referenzen von *System* oder aus dem Modell zur Nutzeroberfläche. Dadurch ist die Anwendung auch in einem anderen Kontext lauffähig.

Details zum Kommando-Manager werden in Abschnitt 4.2.7 noch näher erläutert.

Der View-Controller verwaltet den OpenGL-Kontext und allgemeine Einstellungen zur Visualisierung. Die Projektions- und View-Matrix bestimmen in OpenGL die perspektivische bzw. orthografische Projektion sowie die Blickrichtung, die Position und Skalierung (Zoom) einer gedachten Kamera in der 3D-Szene. Die GPU multipliziert alle Vertices der 3D-Scene mit der Projektions- und View-Matrix, wodurch eine räumliche Darstellung auf einem 2D-Bildschirm erzeugt wird.

Die Behandlung von Eingabe-Events (Mouse-move, push, scroll) und die entsprechende Übersetzung in eine View-Matrix übernimmt die Klasse *CameraManager*.

Die Klasse *ServerInterface* implementiert die im Abschnitt 3.3 beschriebene Netzwerkschnittstelle zur interaktiven numerischen Simulation.

OpenSceneGraph (Abbildung 19 oben rechts) dient als Abstraktionsschicht und Schnittstelle zum OpenGL-Kontext, und verwaltet alle grafischen Objekte in einer hierarchischen Datenstruktur, dem sog. Szene-Graphen. Dessen Knoten können grafische Primitive wie Zylinder oder Quader sein, oder selbst definierte Grafikobjekte durch Listen von Dreiecken oder Vierecken in Kombination mit Listen für Vertices, Farben und Normalenvektoren pro Vertex. Knoten des Szene-Graphen können aber auch ein Switch (An-Aus-Schalter) oder Transformationsknoten sein. Ein Switch kann alle Kind-Knoten aus der Szene ausblenden und wieder einblenden. Ein Transformationsknoten definiert über eine 4X4-Matrix Verschiebung, Rotation und Skalierung aller Kind-Knoten. Ein Transformationsknoten ist dann sehr hilfreich, wenn Objekte, die aus vielen Einzelknoten bestehen, als Ganzes in der Szene bewegt werde sollen, wie beispielsweise die Stabelemente in Abbildung 18. Einen sehr guten Überblick zu 3D-Grafik-Programmierung mit OpenSceneGraph gibt [Martz 2007]. OpenSceneGraph steht unter Open Source-Lizenz zum Download unter [OpenSceneGraph 2012] bereit.

Qt (Abbildung 19 oben links) ist ein ebenfalls in zentralen Bereichen der hier gezeigten Implementierung verwendetes Framework. Qt ist quelloffen und kann unter [Qt 2012] heruntergeladen werden. Zum Einstieg in die Qt-Programmierung und für Beispiele steht mit [Blanchette, Summerfield 2007] ein sehr gutes Buch zur Verfügung.

Mit Qt können Bedienelemente und Layout der grafischen Nutzeroberfläche in C++ geschrieben werden. Qt ist konsequent plattformübergreifend, setzt aber dennoch direkt auf den nativen Grafik-Schnittstellen des Betriebssystems auf, sodass die Elemente der Nutzeroberfläche ein gewohntes „Look & Feel“, bei gleichzeitig guter Performance haben. Besonders hilfreich ist auch der Signal-Slot-Mechanismus von Qt, mit dem eine Kommunikation zwischen beliebigen Objekten hergestellt werden kann, auch wenn sie keine direkte Referenz aufeinander haben (mehr dazu anhand von Beispielen im Verlauf des Kapitels).

4.2.1 Komponenten-Modell

Die prototypische Anwendung wurde für allgemeine räumliche Stabwerke konzipiert, mit allgemein polygonal umrandeten Querschnitten und verschiedenen nichtlinearen Materialmodellen.

Alle wichtigen Komponenten des Stabwerksmodells sind durch Klassen modelliert, die von *DesingComponent* abgeleitet sind (Abbildung 19). Diese Komponenten haben direkte Referenzen auf ihre Entsprechung im SFA-Modell (Abbildung 19 unten rechts). Es liegt in der Verantwortung der Komponenten, ihre jeweiligen SFA-Entities zu aktualisieren, sodass Änderungen im SFA-Modell mit dem Server zur numerischen Berechnung synchronisiert werden können.

Die Komponenten haben keine Referenzen auf das IFC-Modell (vgl. Abbildung 19), lediglich die Klassen *CmdImportIfcStructure* und *CmdExportIfcStructure* (Import und Export von IFC, siehe Abschnitt 4.2.4) sind in der Lage IFC-Entities in Komponenten zu übersetzen bzw. umgekehrt. So tangiert die Komplexität des IFC-Modells nicht die gesamte Anwendung, sondern nur diese beiden Klassen.

4.2.2 Implementierung des Produktdatenmodells IFC

Um in einer Anwendung auf IFC-Daten zugreifen zu können, bietet sich zunächst die Verwendung einer bestehenden Bibliothek, etwa der DLL[47] von TNO[48] an. Der Vorteil ist der sehr geringe Aufwand. Die mangelnde Erweiterbarkeit und Plattformunabhängigkeit sind jedoch erhebliche Nachteile, ebenso die fehlende Möglichkeit zur Geschwindigkeitsoptimierung (Parallelisierung) beim Verarbeiten von STEP-Daten. Schnelle Verarbeitung ist für die Analyse-Schnittstelle essentiell, vgl. Abschnitt 3.3. Eine Veröffentlichung des eigenen Quellcodes als Open Source ist auch nicht sinnvoll möglich, wenn wichtige Bestandteile davon kein Open Source sind. Damit schied diese Variante für die vorliegende Arbeit aus und eine eigene Implementierung der IFC wurde vorgenommen.

Die IFC umfassen über 1000 Klassen (aktuelle Version IFC4 RC4), veröffentlicht in der Datendefinitionssprache EXPRESS (vgl. Abschnitt 2.7.4). Bei so vielen Klassen ist eine manuelle Implementierung nicht praktikabel, sondern muss automatisiert aus dem Schema generiert werden. Zur automatisierten Übersetzung des EXPRESS-Schemas in Quellcode gibt es zwei Möglichkeiten:

- Quellcodegenerierung mittels einer existierenden Software wie „expressik Generator“[49]. Vorteil dabei ist der zunächst relativ geringe Aufwand zur Generierung des Quellcodes. Manuelle Nacharbeiten können jedoch schnell aufwendig werden, die Flexibilität ist eingeschränkt.
- Eigene Übersetzung des EXPRESS-Schemas in Quellcode.

Die zweite Methode bietet die größtmögliche Flexibilität, daher wurde sie hier gewählt. Diese Flexibilität kann unter anderem für eine effiziente Umsetzung eines STEP-Parsers genutzt werden (u. a. durch Auflistung aller Entity-Namen in einer Hash-Tabelle). Auch ein einheitliches Konzept zur Speicherverwaltung kann so einfach umgesetzt werden.

4.2.3 Übersetzung eines EXPRESS-Schemas in Quellcode

Wie in Abschnitt 2.7.4 beschrieben, basiert die Datendefinitionssprache EXPRESS auf dem Konzept der Objektorientierung. Dadurch ist es relativ einfach, ein solches Schema in eine objektorientierte Sprache wie C++ zu übersetzen (Abbildung 20).

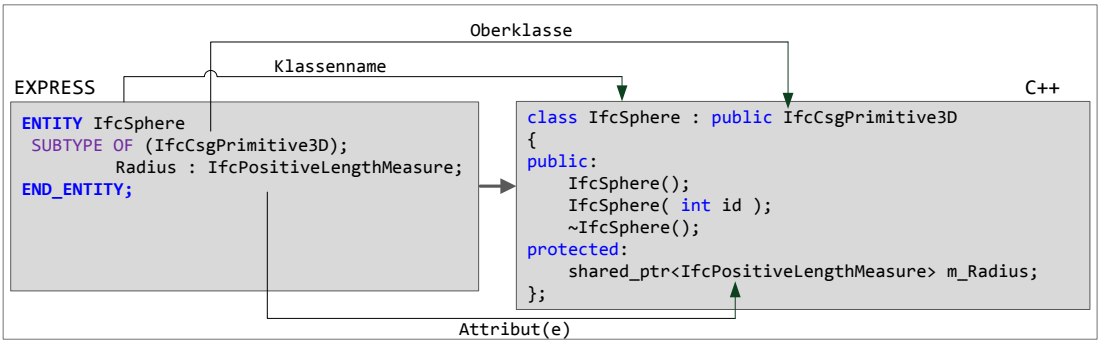


Abbildung 20: Übersetzung eines EXPRESS-Schemas in C++ Quellcode

[47] DLL: Dynamic Link Library für Windows®-Betriebssysteme

[48] Ifc Engine DLL, Toolbox for IFC Data, TNO Building and Construction Research, Delft, Niederlande, [TNO]

[49] „expressik Generator“, entwickelt an der University of Manchester, erzeugt C++ Quellcode aus EXPRESS-Schema-Dateien [Expressik Generator]

Mit regulären Ausdrücken[50] kann die Struktur eines Entity erfasst und in Einzelteile wie Klassenname, Oberklasse oder Attribute (Abbildung 20) zerlegt werden.

```
"\s*([a-zA-Z0-9_]+)\s*:\s*([\s\[\]:\?\(\)a-zA-Z0-9_]+);\n*"
```

Erlaubte Zeichen des Attribut-Namens	Trennung durch Doppelpunkt	Kardinalität, wird anschließend genauer bestimmt	Erlaubte Zeichen des Datentyps	Abschluss durch Semikolon

```
Beispiel-Attribute (Treffer durch obigen Ausdruck):
Item : IfcGeometricRepresentationItem;
InnerBoundaries : OPTIONAL SET [1:?] OF IfcCurve;
```

Abbildung 21: Extrahierung der Attribute eines Entity

Abbildung 21 zeigt die Extrahierung von Attributen aus dem Schema eines zuvor aus dem Gesamtschema ausgeschnittenen Entity (Zeile 6 des Algorithmus 5.1). Der reguläre Ausdruck wird solange angewendet, bis keine Treffer mehr erzielt werden.

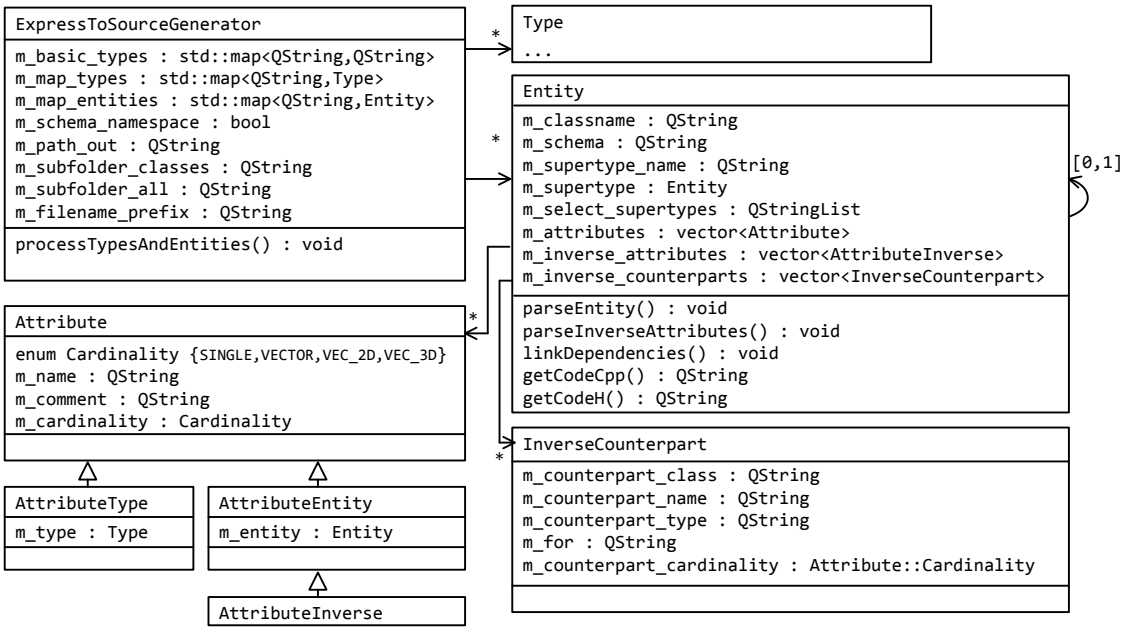


Abbildung 22: Klassenmodell des EXPRESS-Parsers

Algorithmus 4.1: ExpressToSourceGenerator::processTypesAndEntities() Übersetzung eines EXPRESS-Schemas	
Eingabe: Zeichenkette des EXPRESS-Schemas : schema_str	
1	rx_entity_begin := “\\bENTITY\\s+([a-zA-Z0-9_]+)\\s*”
2	rx_entity_end := “END_ENTITY;[\\r]*\\n*”
3	**while** einzelne Zeichenkette eines Entity kann mit rx_entity_begin und rx_entity_end gefunden werden **do**
4	Zeichenkette entity_str des Treffers in Schritt 3
5	Extrahiere Name (1. Treffer aus rx_entity_begin)
6	Instanziiere allgemeines Entity-Objekt e
7	Zuweisung des Entity-Namens und Schemas (entity_str) an e

[50] Suchmuster, mit dem in Zeichenketten nach spezifischen Abfolgen von Zeichen gesucht werden kann

```
8           Eintragen in die Map aller Entities m_map_entities, Entity-Name als Schlüssel
9       end while
10      Schritte 3-9 analog für TYPE-Datentypen
11      // Objekte für jeden Datentyp sind instanziiert, nun erfolgt die eigentliche inhaltliche Auswertung
12      foreach Entity e in m_map_entities do
13          e->parseEntity() nach Algorithmus 5.2
14      end for
15      Schritte 12-14 analog für TYPE Datentypen
16      foreach Entity e in m_map_entities do
17          e->linkDependencies()
18      end for
19      Schritte 16-18 analog für TYPE Datentypen
20      foreach Entity e in m_map_entities do
21          Schreibe .cpp-Datei mit Name e->m_classname und Inhalt e->getCodeCpp()
22          Schreibe .h-Datei mit Name e->m_classname und Inhalt e->getCodeH()
23      end for
24      Schritte 20-23 analog für TYPE Datentypen
```

Bis zum Schritt 9 wird lediglich der Name pro ENTITY/TYPE-Datentyp ausgewertet, ein Objekt vom Typ *Type* bzw. *Entity* (vgl. Abbildung 22) erzeugt, und in eine Map[51]-Datenstruktur eingetragen. Bei der eigentlichen inhaltlichen Auswertung ab Schritt 12, können Verweise auf Datentypen, etwa für Attribute oder Oberklassen, anhand dieser Liste gefunden und referenziert werden. Dadurch können sämtliche Informationen über einen Datentyp und dessen Erfordernisse für die Quellcodeerzeugung von dem entsprechenden Objekt abgefragt werden.

Der in Algorithmus 4.1, Schritt 13 aufgerufene Algorithmus ist im Folgenden dargestellt.

Algorithmus 4.2: Entity::parseEntity ()

Inhaltliche Auswertung eines Entity-Schemas mit Vererbung und Attributen

Membervariablen: nach Abbildung 22

```
1       rx_subtype :="SUBTYPE\\s*OF\\s*\\(\\s*([a-zA-Z0-9_]+)\\s*\\);"
2       m_supertype_name := Treffer aus Schritt 1
3       rx_inverse := "\\s*INVERSE\\s*\\n(\\s*([a-zA-Z0-9_]+)\\s*:\\s*([a-zA-Z0-
        9_\\?\\[\\]:\\s=\\(\\)]+);\\n*)+"
4       if Treffer in Schritt 3 do
5           parseInverseAttributes () // Auswerten der inversen Attribute, hier nicht dargestellt
6       end if
        // Parsen der Attribute: Erläuterung des Ausdrucks in Abbildung 21
7       rx_attribute := "\\s*([a-zA-Z0-9_]+)\\s*:\\s*([a-zA-Z0-9_\\s\\[\\]:\\?\\(\\)]+);\\n*"
8       while Treffer aus Schritt 7 vorhanden do
9           attribute_name : Treffergruppe 1 aus rx_attribute
10          attribute_type : Treffergruppe 2 aus rx_attribute
11          cardinality = Attribute::CARDINALITY_SINGLE;
12          rx_dim := "^(OPTIONAL|UNIQUE\\s)?(ARRAY|LIST|SET)\\s\\[([0-9]+)☹[0-9\\?]+)\\]\\sOF\\s*"
13          count_dim := 0
14          while Treffer mit rx_dim in attribute_type do
```

[51] Assoziatives Datenfeld. Über einen Schlüssel (hier: Zeichenkette) kann auf die zugeordneten Werte (hier: Type/Entity Objekte) zugegriffen werden.

```
15                    ++count_dim
16              end while
17              if count_dim == 1 then
18                    cardinality := VECTOR
19              else if count_dim == 2 then
18                    cardinality := VECTOR_2D
19              else if count_dim == 3 then
18                    cardinality := VECTOR_3D
19              end if
20              Attribute att // Referenz auf allgemeines Attribut-Objekt (vgl. Abbildung 22)
21              Zuweisung der in Schritt 11-19 ermittelten Kardinalität an das Attribut att
22              if attribute_name enthalten in Menge der elementaren Datentypen then
23                    att = new AttributePrimitive()// instantiiere Attribut für einen elementaren Datentyp
24              else if attribute_name enthalten in Menge der TYPE Datentypen (m_map_types) then
25                    att = new AttributeType()  // instantiiere Attribut für einen TYPE Datentyp
26              else if attribute_name enthalten in Menge der ENTITY Datentypen (m_map_entities) then
27                    att = new AttributeEntity()// instantiiere Attribut für einen ENTITY Datentyp
28              else Fehlermeldung
29              end if
30              Hinzufügen von att zur Menge der Attribute (m_attributes, vgl. Abbildung 22 )
31        end while
```

Die in Algorithmus 4.2, Schritt 5 aufgerufene Methode *parseInverseAttributes* ist ähnlich aufgebaut wie die Auswertung der Attribute in Schritt 7-30. Inverse Attribute haben immer ein „Gegenstück“-Attribut bei dem entsprechenden referenzierten Entity (*InverseCounterPart* in Abbildung 22). Zusätzlich wird in der Methode *parseInverseAttributes* vermerkt, wenn ein Attribut ein inverses Attribut als gegenläufige Referenz besitzt. Damit wird eine Methode im Quellcode aller Entities generiert, die aufgerufen werden kann, um die inverse Referenz herzustellen. Somit können beim Instanziieren eines IFC-Modells[52] alle inversen Referenzen automatisiert verknüpft werden[53].

Algorithmus 5.1 und 5.2 sind etwas vereinfacht dargestellt zwecks besserer Übersichtlichkeit. Der vollständige Quellcode zur Schema-Analyse und Quellcodegenerierung ist als Open Source unter dem Namen *IfcPlusPlus* veröffentlicht und befindet sich in einem frei zugänglichen Quellcode-Repository unter *http://code.google.com/p/ifcplusplus/.*

IfcPlusPlus kann somit genutzt werden, um beliebige (auch eigene) EXPRESS-Schema-Dateien in Quellcode zu übersetzen.

Der Weg vom abstrakten Datenmodell, beschrieben in einem Schema, über Quellcode bis zur konkreten Modell-Instanz ist in Abbildung 23 dargestellt.

[52] Inverse Attribute sind in den STEP-Daten nicht explizit vorhanden, müssen also separat hergestellt werden

[53] Wichtig ist dabei die Verwendung eines „schwachen“ Referenzzählers (std::weak_ptr anstelle von std::shared_ptr), um Ringreferenzen zu vermeiden, die zu Speicherlecks führen würden

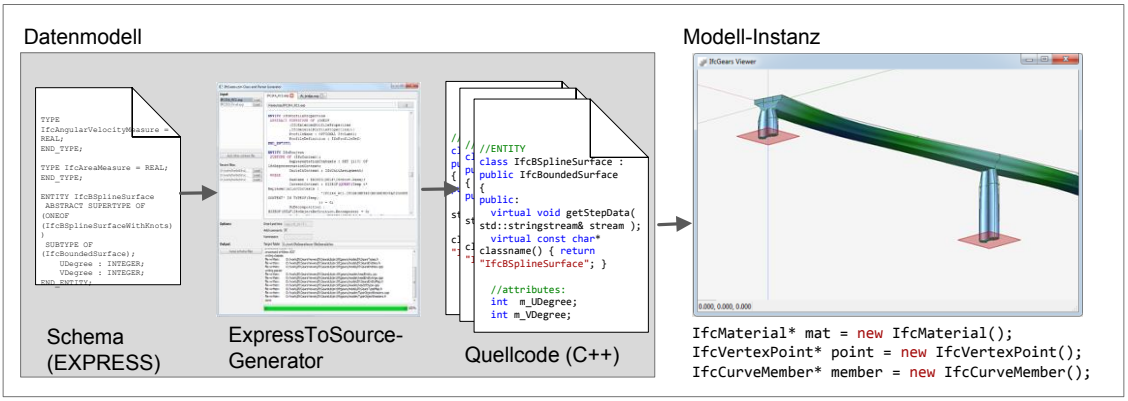


Abbildung 23: Übersetzung eines EXPRESS Schemas in Quellcode

Neben dem EXPRESS-Quellcode-Generator beinhaltet das genannte Open-Source-Projekt ein C++ Klassenmodell von IFC4 RC4 (erzeugt mit dem Generator), einen Parser zum Einlesen von IFC-STEP-Dateien, eine Klasse zum Exportieren der Daten, einen geometrischen Modellierer zur Übersetzung verschiedener geometrischer Repräsentationen in allgemeine Polyeder, sowie einen Viewer zum Anzeigen eines geladenen Modells (Abbildung 24).

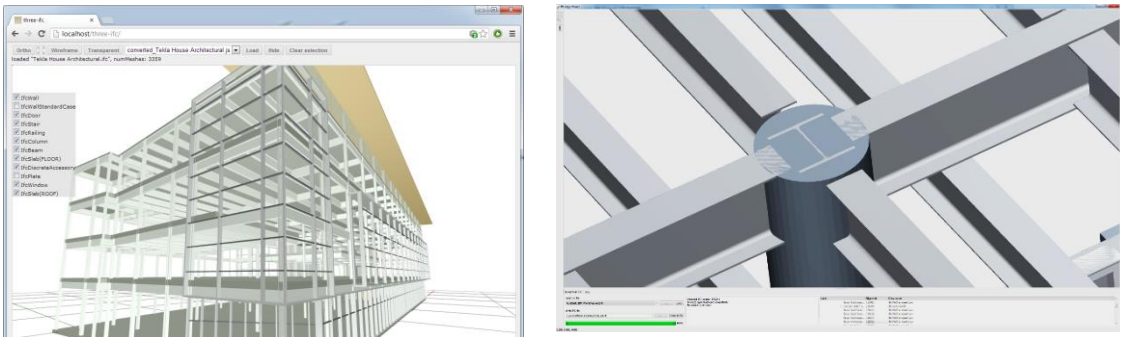


Abbildung 24: Visualisierung von IFC-Modellen mit IfcPlusPlus (links WebGL[54] , rechts: OpenSceneGraph)

Die Verarbeitungskette von STEP-Daten in *IfcPlusPlus* zeigt die Abbildung 25. Für die Auswertung boolescher Operationen wird eine Open-Source Bibliothek für CSG-Modellierung mit dem Namen *Carve* [Carve] verwendet (Abbildung 25 mittig).

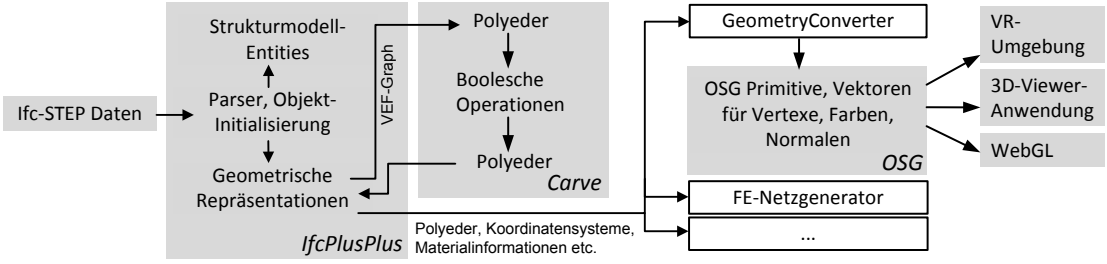


Abbildung 25: Verarbeitung von STEP-Daten in IfcPlusPlus

Der Quellcode-Generator *IfcExtender* wird auch zum Erzeugen des SFA-Klassenmodells und Parsers der Analyseschnittstelle verwendet, darauf wird in späteren Abschnitten noch eingegangen.

[54] WebGL: OpenGL-Schnittstelle für 3D-Grafik im Webbrowser ohne Installation von Plugins. IfcPlusPlus kam hier serverseitig zum Einsatz, um die geometrischen Repräsentationen aus IFC in ein Format für einen WebGL-Szenegraphen zu konvertieren

4.2.4 IFC-Schnittstelle für Strukturmodelle

Wie bereits in Abschnitt 2.7.8 beschrieben, werden Strukturmodelle in IFC redundant zum architektonischen Modell gespeichert. Als mögliche Strukturelemente umfasst die aktuelle Version IFC4 RC4 Stabelemente und Flächenelemente, wobei Letztere Scheiben-, Platten- oder allgemeine Schalenelemente sein können, eben oder gekrümmt. Volumenelemente sind also nicht vorgesehen und das ist auch der Grund, warum das Strukturmodell nicht kompatibel ist mit dem architektonischen Modell, denn dort sind üblicherweise praktisch alle Elemente durch eine volumetrische Repräsentation definiert.

Die hier umgesetzte Entwurfsanwendung unterstützt momentan nur Stabelemente, da bereits damit alle Aspekte der Interaktivität und Kommunikation mit dem Rechenserver gezeigt werden können.

Der Import eines IFC-Modells und Übersetzung in Komponenten des Strukturmodells (Abbildung 19) ist in Form eines Kommandos implementiert, das nach folgendem Algorithmus vorgeht:

Algorithmus 4.3: CmdImportIfcStructure::loadIfcStructure()

Import und Übersetzung in Komponenten des Strukturmodells

Eingabe: Dateiname und Pfad

```
1   Öffne Datei
2   std::map<int, shared_ptr<IfcPPEntity>  > enitities_map;
3   Objekt vom Typ IfcPlusPlus::ReaderWriterIFC parst den Inhalt der Datei und erzeugt Objekte -> entitties_map
4   std::vector<shared_ptr<IfcMaterialProfile> > material_profiles; // Querschnitte
5   std::vector<…> …// Analog Schritt 4 für Stabelemente, Lasten u.A.
6   foreach Entity e in entities_map do
7       shared_ptr<IfcMaterial> mat = dynamic_pointer_cast<IfcMaterial>(e);
8       if mat then
9           convertIfcMaterial( mat ); // direkte Konvertierung in ein MaterialComponent-Objekt
10          continue;
11      end if
12      shared_ptr<IfcMaterialProfile> mat_profile = dynamic_pointer_cast<IfcMaterialProfile>(e);
13      if mat_ profile  then
14  material_profiles.push_back(mat_profile); // Speicherung für anschließende Konvertierung in definierter Reihenfolge
15          continue;
16      end if
17      Analog Schritte 12-16 für Stabelemente, Lasten, u.A.
18  end for
19  foreach IfcMaterialProfile mp in material_profiles do
20      convertIfcMaterialProfile( mp );
21  end for
22  Analog Schritte 19-21 für Stabelemente, Lasten u.A.
```

Die Komponenten sind teilweise voneinander abhängig, z.B. die Komponente *FiberSectionComponent* hat eine oder mehrere Referenzen zu *MaterialComponent*. Da Eingabe-Entities (*entities_map*, Schritt 2) in der Liste jedoch keine bestimmte Reihenfolge aufweisen,

müssen zunächst bestimmte Entities wie *IfcMaterial* oder *IfcMaterialProfile* in getrennten Listen gefiltert (Schritt 13-17) und danach in kontrollierter Reihenfolge in entsprechende Komponenten konvertiert werden (Schritt 19-22).

Exemplarisch zeigt die Abbildung 26 die Modellierung von Kopplungen zwischen Strukturelementen und Lagerungsbedingungen, wie sie auch in der o. g. IFC-Schnittstelle zum Import und Export implementiert sind.

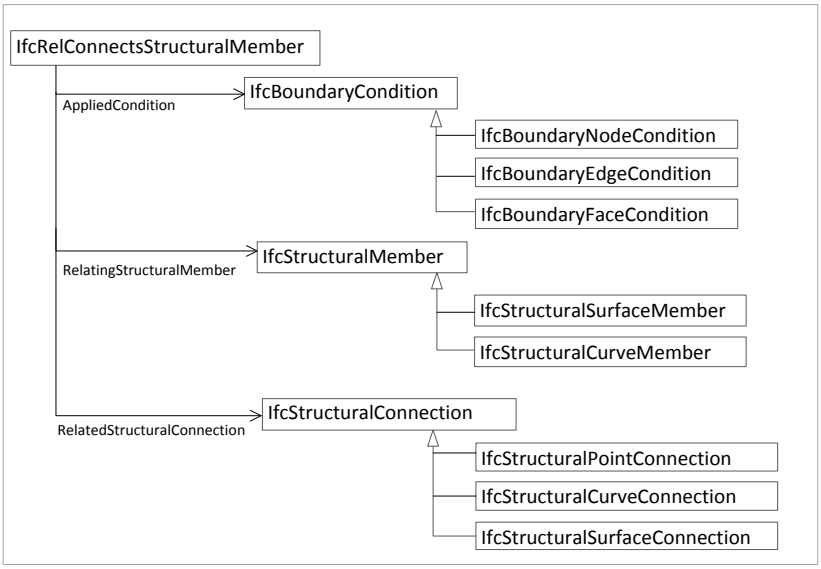


Abbildung 26: IFC-Entities zur Realisierung von Lagerungsbedingungen und Kopplungen

Die Abbildung 27 zeigt ein Beispiel zur Kopplung zweier Elemente anhand von STEP-Daten.

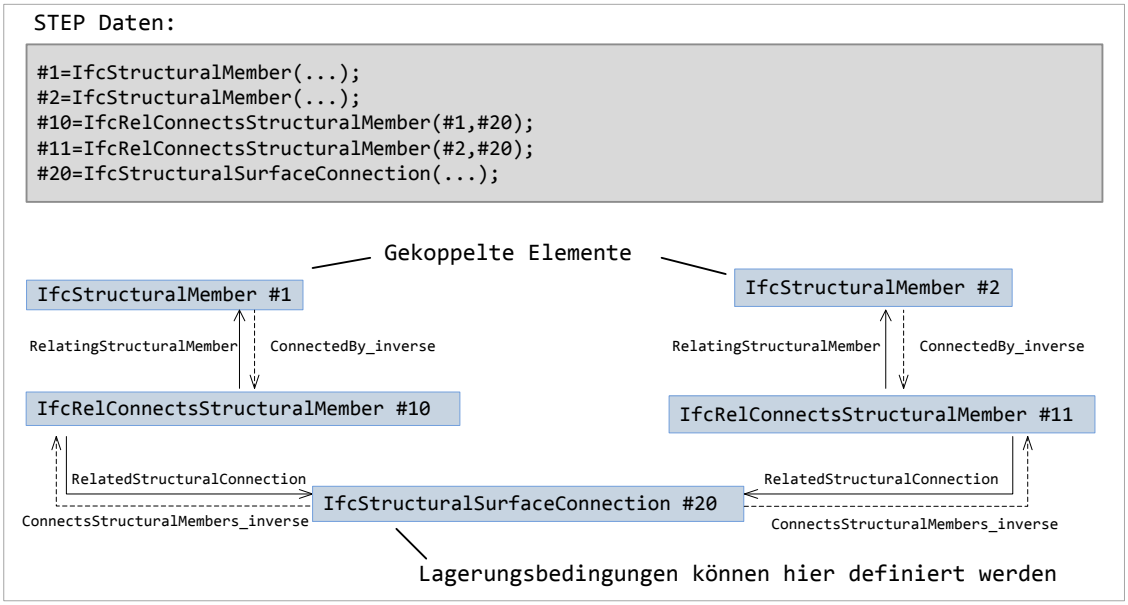


Abbildung 27: Realisierung von Lagerungsbedingungen und Kopplungen in IFC

Für eine nähere Beschreibung der IFC-Modellierung von Tragwerken wird auf [buildingSMART] verwiesen.

Der Export von Strukturmodellen erfolgt über die Klasse *CmdExportIfcStructure* (Abbildung 19), wobei hier die Reihenfolge der Komponenten keine Rolle spielt. Generell ist der Export einfacher, da immer nur eine Modellierungsvariante abgedeckt werden muss, im Gegensatz zum Import, wo

immer mit verschiedenen Varianten gerechnet werden muss (z.B. bei der Querschnittsdefinition), wenn der Standard dies erlaubt.

Die Verflechtung der Anwendung mit dem IFC-Modell wurde konsequent minimiert, sodass insgesamt nur 2 Klassen (*CmdImportIfcStructure* und *CmdExportIfcStructure*) die Schnittstelle der Anwendung zum IFC-Modell bilden (vgl. UML-Diagramm Abbildung 19).

4.2.5 Speicherverwaltung

Ein Konzept zur Speicherverwaltung ist ab einer gewissen Komplexität der Anwendung sinnvoll, denn das manuelle Löschen von Objekten kann sehr kompliziert und fehleranfällig werden [Kuehne, Martz 2007, Seite 32]. Dies gilt insbesondere in hierarchischen und vielfach untereinander verlinkten Datenstrukturen wie Szenegrafen oder Produktmodellen. Auch wenn verschiedene Bibliotheken in einer Anwendung verwendet werden, die jeweils ihre eigene Datenstruktur haben (etwa das Komponentenmodell, OpenSceneGraph, und das SFA-Modell in Abbildung 19), darf keine der Einzelbibliotheken Objekte eigenmächtig löschen, denn andere Bibliotheken könnten noch eine Referenz auf das Objekt halten und darauf zugreifen. Diese Problematik kann generell nur durch *Smart Pointer* oder einen *Garbage Collector*[55] gelöst werden.

Smart Pointer können in C++ grundsätzlich auf 2 verschiedene Varianten umgesetzt werden:

1.: Die zu verwaltenden Objekte haben ein Attribut zum Zählen der Referenzen. Ein Template-Objekt übernimmt das Inkrementieren/Dekrementieren des Zählers. Wenn der Zähler 0 erreicht, wird das Objekt gelöscht und der Speicher somit wieder freigegeben.

Beispiel (Speicherverwaltung in OpenSceneGraph):

```
osg::Geometry* my_geom = new osg::Geometry(); // normales Objekt
osg::ref_ptr<osg::Geometry> my_geom_ptr = my_geom;   // smart pointer
```

Die Template-Klasse *ref_ptr* ist der Smart Pointer, der den Referenzzähler des verwalteten Objektes *my_geom* in diesem Fall auf 1 erhöht. Die zu verwaltenden Objekte haben wie gesagt einen Referenzzähler als Attribut und löschen sich selbst, wenn der Zähler 0 erreicht. Diese Funktionalität ist in der Klasse *Referenced* zusammengefasst, von der die Klasse *Geometry* abgeleitet ist.

2.: Das Smart-Pointer-Objekt hat selbst einen Referenzzähler für beliebige zu verwaltende Objekte. Wenn Objekte an Methoden übergeben werden oder in Containern gespeichert werden, muss dies immer mitsamt dem Smart Pointer geschehen, da sonst der Zählerstand verloren geht.

Beispiel (Speicherverwaltung in C++ 11[56]):

```
std::shared_ptr<IfcCartesianPoint> my_point(new IfcCartesianPoint());
```

Container-Klassen wie Vektoren oder Mengen halten damit keine direkten Referenzen, sondern nur indirekt über die Pointer-Templates:

```
std::vector<osg::ref_ptr<IfcCartesianPoint> > my_container; // 1. Variante
```

bzw.

```
std::vector<std::shared_ptr<IfcCartesianPoint> > my_container; //2. Variante
```

[55] Ein Garbage Collector löscht automatisch nicht-referenzierte Objekte

[56] C++-Revision von 2011, standardisiert unter ISO/IEC 14882:2011

In beiden Fällen dekrementiert der Destruktor des Smart Pointers den Referenzzähler.

In der IFC-Implementierung der vorliegenden Arbeit wird die Variante 2 verwendet.

Vorsicht ist geboten zur Vermeidung von Ring-Referenzen. Sind zwei Objekte nur gegenseitig referenziert oder mehrere Objekte ringförmig, so bilden sie eine „Insel“, die eigentlich gelöscht werden könnte. Da der Referenzzähler aber nicht den Wert 0 erreicht, werden die Objekte nicht gelöscht, es handelt sich also um ein Speicherleck. Dies könnte bei den inversen Attributen von IFC-Objekten passieren. Es gibt jedoch die Möglichkeit, inverse Attribute durch „schwache“ Smart Pointer-Referenzen auszubilden. Diese erhalten den Zählerstand, verändern ihn aber selbst nicht.

Andere Konzepte zur Speicherverwaltung, wie Garbage-Collection in Java, scheitern ebenfalls an solchen Ring-Referenzen und bedürfen der Verwendung spezieller Referenzierungsobjekte.

4.2.6 Objektfang und Nutzerinteraktion

Für eine interaktive Simulation müssen möglichst intuitive Interaktionsmechanismen für den Nutzer gegeben sein. Eine sehr intuitive Form der Interaktion ist die Manipulation von Kontrollpunkten in einer 3D-Visualisierung durch ein Zeigegerät (etwa Maus)[57] (Abbildung 28).

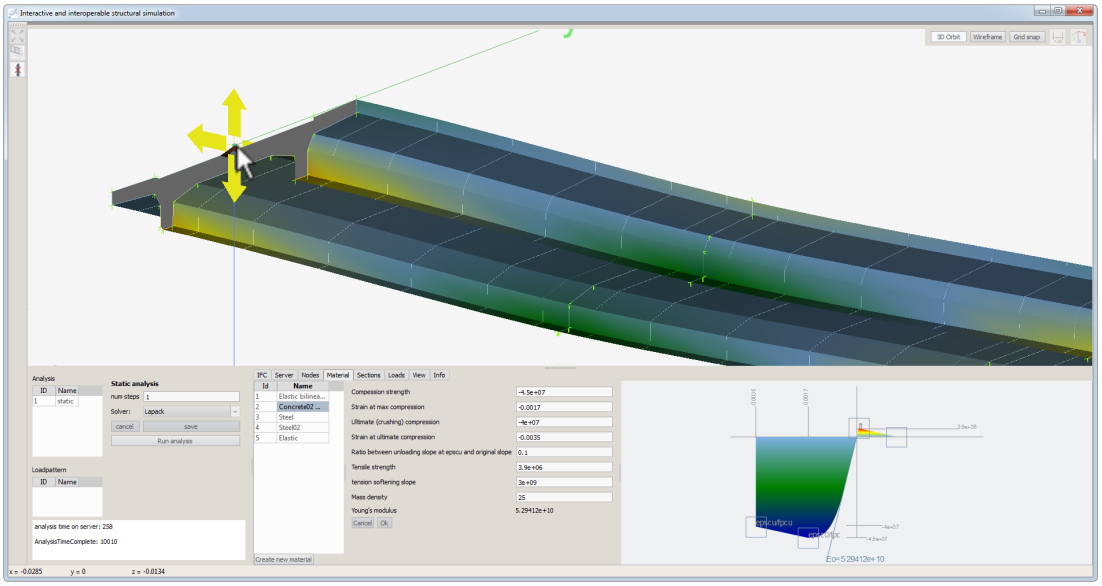


Abbildung 28: Interaktion mit Strukturmodell durch ein Zeigegerät

Ein sog. „Event“ wie die Bewegung eines Zeigegerätes wird vom Betriebssystem durch ein Objekt repräsentiert, das an die betreffende Anwendung weitergereicht wird. Durch Art und Zustand (Attribute) des Events kann entsprechend auf die Nutzereingabe reagiert werden. Im Falle eines 3D-Kontext wie in Abbildung 28 muss eine Anwendung allerdings selbst errechnen, ob sich Eingabe-sensitive Bereiche des Modells an der betreffenden Cursor-Position[58] befinden. Das Event-Objekt liefert zunächst nur die Bildschirmkoordinaten des Cursors, die nach dem unten stehenden Algorithmus in Modellkoordinaten umgerechnet werden, um danach für den Objekt-Fang genutzt zu werden. „Fang“ bedeutet hier, dass Eingabe-sensitive Objekte markiert (Abbildung 28) und für evtl. spätere Events gemerkt werden, sobald der Cursor in eine bestimmte Region über dem Objekt bewegt wird.

[57] Abgesehen von Konzepten zur Virtuellen Realität, cf. Kapitel 6

[58] Aktuelle Bearbeitungsposition am Bildschirm, in diesem Fall die Position des Zeigegeräts.

Algorithmus 4.4: **handleCursorMove**

Übergabeparameter: ea : osgGA::GuiEventAdapter,

Membervariablen: m_viewer : ViewController, m_component_model : ComponentModel

```
1   osg::Matrix matrix_model_screen;
2   multipliziere matrix_model_screen mit Viewport-Matrix (Bildschirm-Ausschnitt des 3D-Kontext)
3   multipliziere matrix_model_screen mit Projektions-Matrix des Viewers
4   multipliziere matrix_model_screen mit View-Matrix des Viewers
5   osg::Matrix matrix_screen_model := Inverse der Matrix matrix_model_screen
6   double cursor_x = ea.getX();
7   double cursor_y = ea.getY();
8   osg::Vec3d ray_origin( cursor_x, cursor_y, 0.0 );
9   osg::Vec3d ray_end( cursor_x, cursor_y, 1.0 );
10  // transformiere Bildschirm-Koordinaten des Events in Modell-Koordinaten
11  ray_origin = ray_origin * matrix_screen_model;
12  ray_end = ray_end * matrix_screen_model;
13  // Schnittpunkt des Cursor-Strahls mit aktueller Modellier-Ebene
14  osg::Vec3d intersect_point( m_system->getModellingPlane()->intersectPlaneLine( ray_origin, ray_end ) );
15  bool object_snapped = false;
16  double min_snap_dist = 0.05*0.05;
17  // finde Knoten in der Umgebung des Cursor-Schnittpunkts
18  K := Menge der Knoten-Komponenten
19  foreach node_component in { K } do
20          double d_square = computeSquareDistance( intersect_point, node_component );
21          if  d_square < min_snap_dist  then
22                  node_component->setSnapped( true );
23                  m_component_model->setComponentSnapped( node_component );
24                  // object snap highlighting
25                  if object snap symbol not yet set then
26                          set object snap object into scene graph
27                  end if
28                  adjust position of object snap symbol
29                  object_snapped = true;
30                  min_snap_dist = d_square;
31                  // don't break here, because other objects could be even closer
32          end if
33  end for
34  if object_snapped then
35          return
36  endif
37  Analog Schnitt des Cursor-Strahls mit allen Stabelementen
```

In diesem Algorithmus wird also nicht nur die Cursor-Position (*cursor_x*, *cursor_y, 0*) in Modellkoordinaten umgerechnet, sondern zusätzlich ein zweiter Punkt (*cursor_x, cursor_y, 1*), der einem Punkt senkrecht unter dem 2D-Cursor entspricht. Durch diesen Trick erhält man zwei 3D-Punkte, aus denen eine Geradengleichung aufgestellt werden kann, die einem gedachten Cursor-Strahl senkrecht durch das Modell entspricht.

Der Algorithmus 4.4 wird per-Frame durch das Event-Management von OpenSceneGraph aufgerufen.

Folgt auf ein Move-Event ein Click-Event, und ein Objekt ist bereits als „gefangen“ markiert, so kann dieses durch ein entsprechendes Kommando und wiederum darauf folgende Move-Events verschoben werden. Die dafür implementierten Algorithmen sind teilweise ähnlich dem Algorithmus 4.4.

4.2.7 Kommando-Management

Alle Änderungen am Modell werden als Kommando ausgeführt. Die Klasse *CommandManager* (Abbildung 19) ist dafür zuständig, alle Kommando-Objekte zu speichern, deren Ausführung zu starten, sie ggf. rückgängig zu machen (undo) oder zu wiederholen (redo).

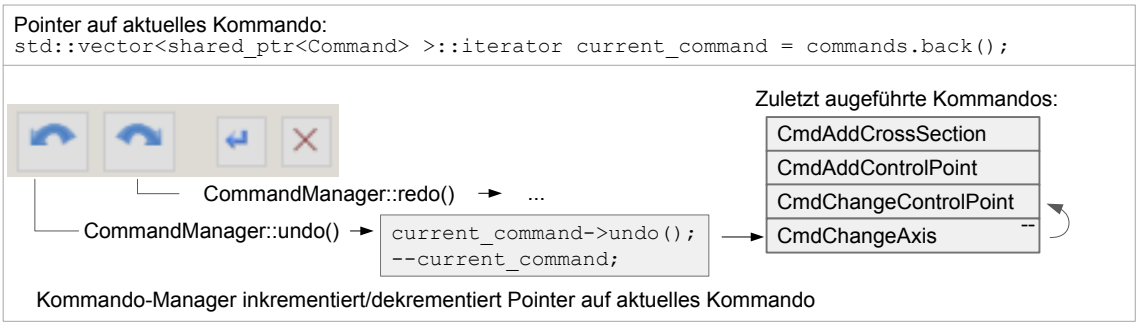


Abbildung 29: Kommando-Management

4.2.8 Nutzerinteraktion mit nichtlinearen Materialmodellen

Material-Parameter sollen hier ebenso sensitiv sein hinsichtlich einer Nutzerinteraktion wie die offensichtlichen Modellparameter wie Topologie und Geometrie der Bauteile.

Das bedeutet, dass der Entwurfsraum der potentiellen Möglichkeiten erweitert wird um die Dimensionen der Materialparameter.

Eine naheliegende Umsetzung ist die Interaktion durch herkömmliche Texteingabefelder (Abbildung 30 Mitte). Im Falle eines einfachen linear-elastischen Materialgesetzes können so die betreffenden Parameter Dichte, E-Modul und Querdehnzahl durch eine numerische Eingabe verändert werden. Im Falle eines nichtlinearen Materialgesetzes für Beton (Abbildung 30, Abbildung 32) treten entsprechend mehr Eingabefelder auf.

Eine grafische Darstellung der Spannungs-Dehnungs-Kurve eines Materialgesetzes bringt eine Verbesserung hinsichtlich der Abschätzbarkeit der Größenordnung der Eingaben. Auch dies ist in der prototypischen Anwendung implementiert (Abbildung 30 rechts).

Für eine vollständig interaktive Modellierung fehlt aber noch ein Baustein: die interaktive Manipulation der Materialobjekte. Dafür wurden hier Kontrollpunkte implementiert (Abbildung 30 rechts), die durch ein Zeigegerät verschoben werden können und ein visuelles Feedback geben, wenn der Cursor über den sensitiven Bereich bewegt wird.

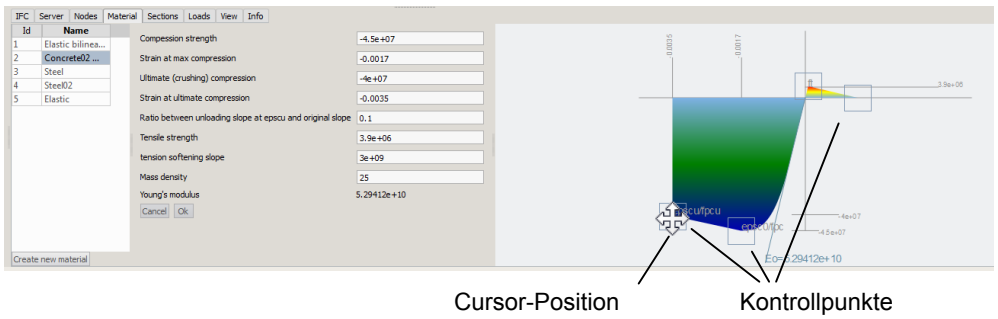


Abbildung 30: Interaktiver Material-Editor. Einheiten: [N], [m]

Die Spannungs-Dehnungs-Kurve wird bei einem Verschieben eines Kontrollpunktes per-Frame aktualisiert, ein Farbgradient zeigt dabei die Farbcodierung der Spannungen an, die auch für die Visualisierung des Strukturmodells verwendet wird (Abbildung 28 oben).

Optional kann über eine Checkbox die Aktualisierung des gesamten Strukturmodells per-Frame aktiviert werden, sodass die Auswirkungen der Veränderungen der Materialparameter sofort sichtbar werden, was jedoch nur bei kleineren Modellen[59] schnell genug möglich ist.

Da nun mehrere Controller einen Systemparameter verändern können (Textfeld und Kontrollpunkt), müssen diese bei Betätigung des jeweils anderen Controllers passiv aktualisiert werden. Dabei müssen Signale des jeweils anderen Controllers deaktiviert werden, damit kein rekursiver Aufruf und damit eine Endlosschleife entsteht (Abbildung 31).

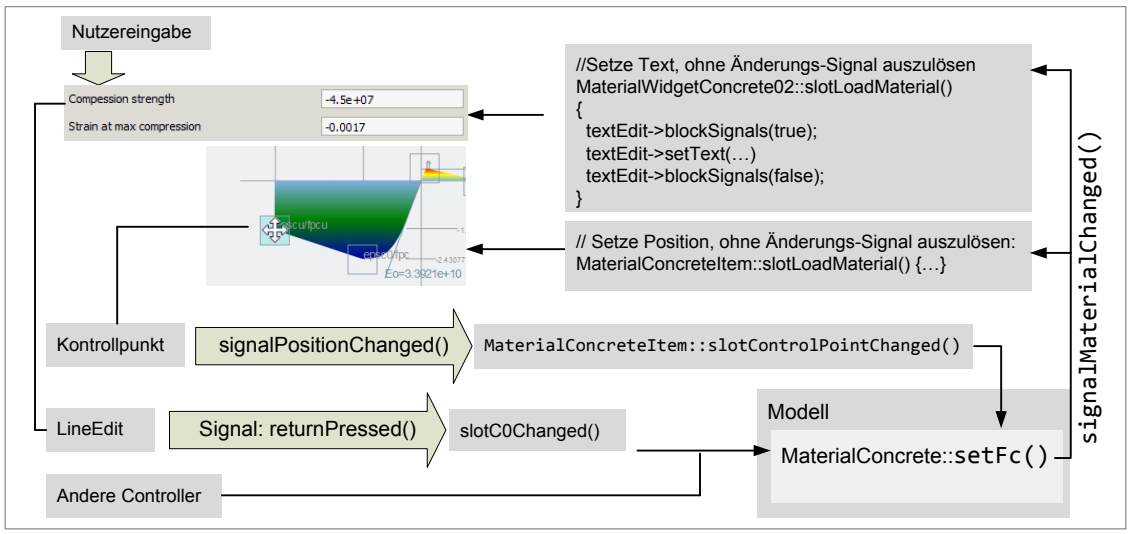


Abbildung 31: Controller-Events

4.2.9 Querschnittsmodellierung

Nichtlineares Materialverhalten unter Berücksichtigung der Belastungs-Historie kann numerisch nur punktuell modelliert werden, nicht für Flächen. Selbst wenn eine exakte Bestimmung des Flächenträgheitsmomentes eines Querschnittes möglich ist, kann ein nichtlineares Materialgesetz darauf nicht angewendet werden, denn im Querschnitt können gleichzeitig sehr unterschiedliche Spannungs- und Dehnungszustände auftreten. So können Teilbereiche bereits im plastischen Zustand sein, andere noch im linear-elastischen Zustand. Das einfache Grenzbiegemoment $M_{y,d} = W_y \cdot \sigma_{R,d}$ kann somit weder zur Schnittkraftermittlung[60] noch zur Bemessung herangezogen werden.

Ein eindimensional (punktuell im Querschnitt) modelliertes Materialgesetz kann jedoch durch Diskretisierung eines Querschnitts, z.B. in Rechtecke, auf beliebige polygonale Querschnitte angewendet werden. Dabei wird die exakte Fläche durch eine Menge von Rechteckflächen

[59] Die Geschwindigkeit der Berechnung hängt von der Anzahl der Freiheitsgrade und der Leistungsfähigkeit des Computers ab, vgl. Abschnitt 4.5

[60] Der Widerstand eines Stabelementes gegenüber Biegung, also Verdrehung eines Stabendknotens, geht in die Steifigkeitsmatrix einer FE-Analyse ein.

(Fasern) angenähert. Ein Rechteck wird dann durch seinen Mittelpunkt und die Fläche repräsentiert, zusammen mit einem entsprechenden Material-Objekt.

Als eine flexibel skalierbare und gleichzeitig einfach zu handhabende Datenstruktur wurde hier ein sog. Quadtree gewählt, also eine hierarchische Baumstruktur bei der jeder Knoten 4 Kindknoten hat (Abbildung 32).

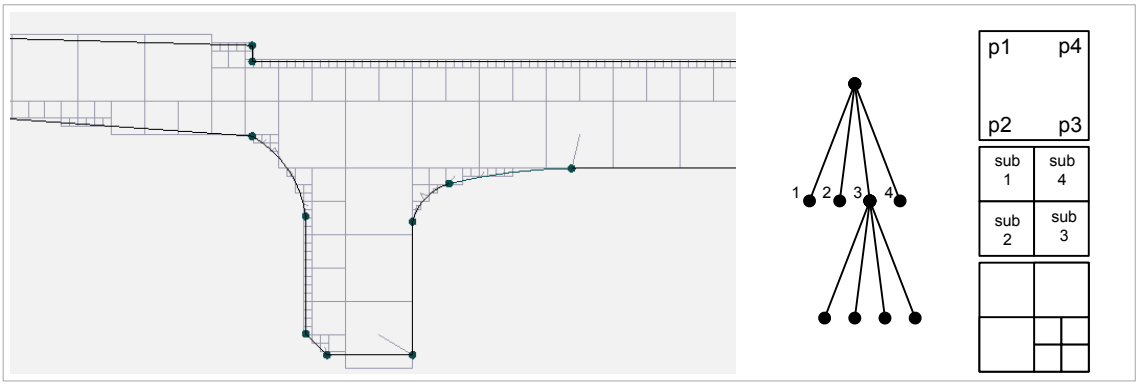


Abbildung 32: Querschnitts-Diskretisierung durch einen Quadtree

Zur Generierung des Quadtree aus einem beliebig polygonal umrandeten Querschnitt (mit Geraden und Kreisbögen) wurde auf Basis der in Abbildung 33 dargestellten Modellierung hier folgender rekursiver Algorithmus entwickelt[61]:

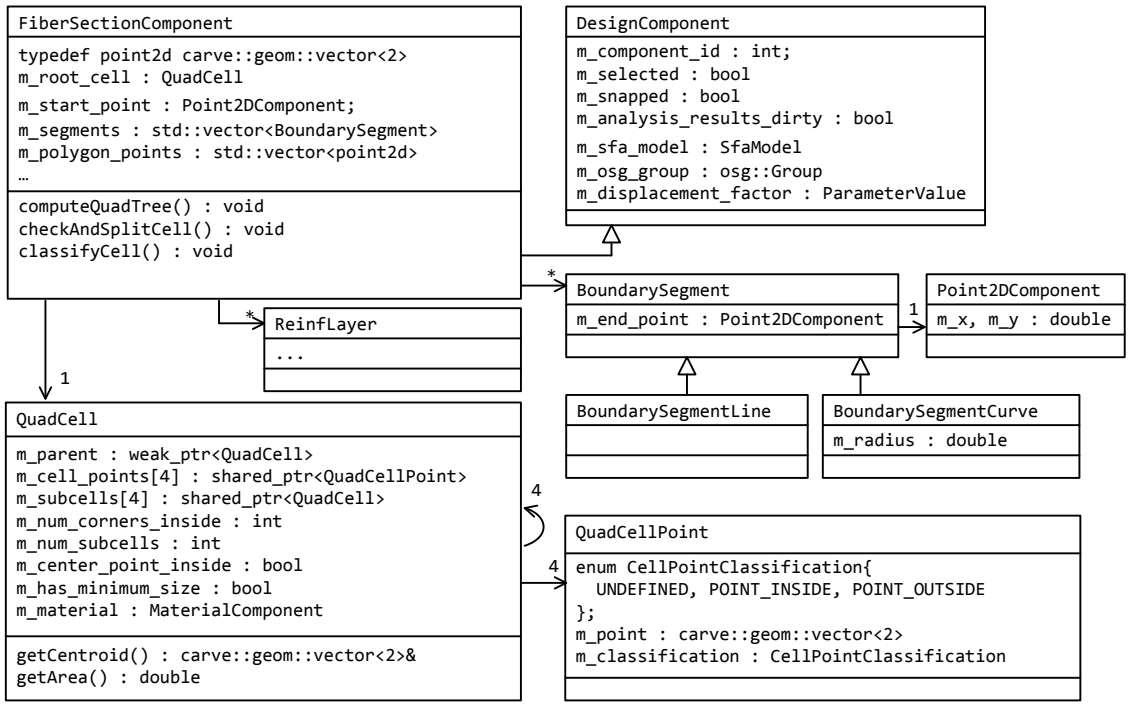


Abbildung 33: Aufbau der Querschnitts-Modellierung in UML-Darstellung

[61] Der Grund, warum keine bestehende Quadtree-Bibliothek verwendet wurde, ist, dass zur Einbindung einer solchen, mit der erforderlichen Datenkonvertierung in beide Richtungen, der Aufwand ähnlich hoch ist wie eine Eigenentwicklung

Algorithmus 4.5: **FiberSectionComponent::computeQuadTree()**

Übergabeparameter: pcell : QuadCell,

Membervariablen: m_polygon_points : std::vector<carve::geom::vector<2> > m_max_fiber_size : real, m_min_fiber_size : real

```
1       p1 := pcell->m_cell_points[0]
2       p2 := pcell->m_cell_points[1]
3       p3 := pcell->m_cell_points[2]
4       p4 := pcell->m_cell_points[3]
5       bool split_cell = false
6       real dx = abs(p4.x – p1.x)
7       real dy = abs(p2.y – p1.y)
8       if dx <= m_max_fiber_size && dy <= m_max_fiber_size then
9               parent_cell->m_has_minimum_size = true
10      end if
7       if !parent_cell->m_has_minimum_size then
8               split_cell = true
9       end if

10      int num_inside = 0
11      if isInside(m_polygon_points, p1 ) then
12              ++num_inside
13              cell_pt1.m_point_classification = POINT_INSIDE
14      else
15              cell_pt1.m_point_classification = POINT_OUTSIDE
16      end if
17      Schritte 11-16 analog für p2, p3, p4

18      parent_cell->m_num_corners_inside = num_inside
19      parent_cell->m_center_point_inside = isInside( m_polygon_points, center_point )

20      if parent_cell->m_has_minimum_size then
21              if num_inside == 4 then
22                      return
23              if num_inside == 0 then
24                      if !parent_cell->m_center_point_inside then
25                              return
26                      end if
27                      split_cell = true
28              end if
29      end if
30      if num_inside > 0 && num_inside < 4 then
31              split_cell = true
32      end if
34      if dx <= min_fiber_size && dy <= min_fiber_size ) then
35              return
36      end if
37      if split_cell then
38              checkAndSplitCell( parent_cell, polygon_points )
39      end if
```

Dieser Algorithmus wird mit einer einzelnen Zelle gestartet, die aus der Bounding Box des Polygons gebildet wird. Der Algorithmus prüft zunächst, ob die Mindestgröße für eine Zelle erreicht ist und ob anhand der Lage der Eckpunkte (innerhalb oder außerhalb des Polygons) eine Aufteilung der Zelle notwendig ist. Wenn ja, wird die Methode *checkAndSplitCell* aufgerufen, die durch den folgenden Algorithmus dargestellt ist.

Algorithmus 4.6: **FiberSectionComponent::checkAndSplitCell()**

Übergabeparameter: cell : QuadCell

Membervariablen: wie Algorithmus 6.1

```
1       berechne center_point (Mittelpunkt von cell)
2       berechne p5 (Mittelpunkt p1-p2 von cell)
3       berechne p6 (Mittelpunkt p2-p3)
4       berechne p7 (Mittelpunkt p3-p4)
5       berechne p8 (Mittelpunkt p1-p4)
6       berechne c1c( Mittelpunkt von Subzelle 1)
7       berechne c2c( Mittelpunkt von Subzelle 2)
8       berechne c3c( Mittelpunkt von Subzelle 3)
9       berechne c4c( Mittelpunkt von Subzelle 4)

        // keine Teilung wenn alle 4 Subzellen im Polygon liegen, und die min. Größe für die aktuelle Zelle erfüllt ist
10      if  cell->m_has_minimum_size then
11              int num_sub_inside = 0
12              if isInside(m_polygon_points, c1c ) then
13                      ++num_sub_inside
14              end if
15              Analog Schritt 13, 14 für Punkte c2c, c3, und c4c
16              if num_sub_inside == 4 then
17                      if  cell->m_num_corners_inside > 0 then
18                              return
19                      end if
20              end if
21      end if
22      QuadCell sub_cell1 = new QuadCell( p1, p5, center_point, p8, cell )
23      computeQuadTree( sub_cell1, polygon_points )
24      classifyCell( sub_cell1, 0 )

25      QuadCell sub_cell2 = new QuadCell( p5, p2, p6, center_point, cell )
26      computeQuadTree( sub_cell2, polygon_points )
27      classifyCell( sub_cell2, 1 )

28      QuadCell sub_cell3 = new QuadCell( center_point, p6, p3, p7, cell )
29      computeQuadTree( sub_cell3, polygon_points )
30      classifyCell( sub_cell3, 2 )

31      QuadCell sub_cell4 = new QuadCell( p8, center_point, p7, p4, cell )
32      computeQuadTree( sub_cell4, polygon_points )
33      classifyCell( sub_cell4, 3 )
34      end if
```

Die Methode *isInside* (Schritt 12) prüft, ob der übergebene Punkt innerhalb oder auf einer Kante oder einem Polygonpunkt liegt, andernfalls wird *false* zurückgegeben. Dazu wird die Bibliothek *Carve* verwendet.

Die Klassifizierung, ob eine Zelle innerhalb oder außerhalb des Querschnitts liegt, findet in der folgenden Methode statt:

Algorithmus 4.7: **classifyCell**

Übergabeparameter: cell : QuadCell, index : int

```
1       QuadCell parent_cell = cell->m_parent
2       if cell->m_num_corners_inside == 0 then
3               if cell->m_num_subcells > 0 then
4                       parent_cell->m_subquads[index] = cell
```

```
5                                   ++parent_cell->m_num_subcells
6                           end if
7                   end if

8                   if cell->m_num_corners_inside > 0 then
9                           if cell->m_center_point_inside then
10                                  parent_cell->m_subquads[index] = cell
11                                  ++parent_cell->m_num_subcells
12                          else
13                                  if cell->m_num_subcells > 0 then
14                                          parent_cell->m_subquads[index] = cell
15                                          ++parent_cell->m_num_subcells
16                                  end if
17                          end if
18                  end if
```

4.2.10 Nutzerinteraktion mit allgemein polygonalen Querschnitten

Der Querschnitts-Editor ist in die prototypische Modellierungs-Anwendung integriert (Abbildung 34). Er ermöglicht eine tabellarische Eingabe des Querschnitts, und eine interaktive Anpassung der Position der Kontrollpunkte über ein Zeigegerät (der Algorithmus zur Interaktion mit den Kontrollpunkten ist ähnlich dem Algorithmus 4.4). Die Quadtree-Repräsentation wird in Echtzeit neu berechnet und visualisiert[62]. Bei Bedarf kann der Diskretisierungs-Algorithmus über entsprechende Bedienelemente beeinflusst werden (minimale und maximale Kantenlänge der Zellen bzw. Fasern).

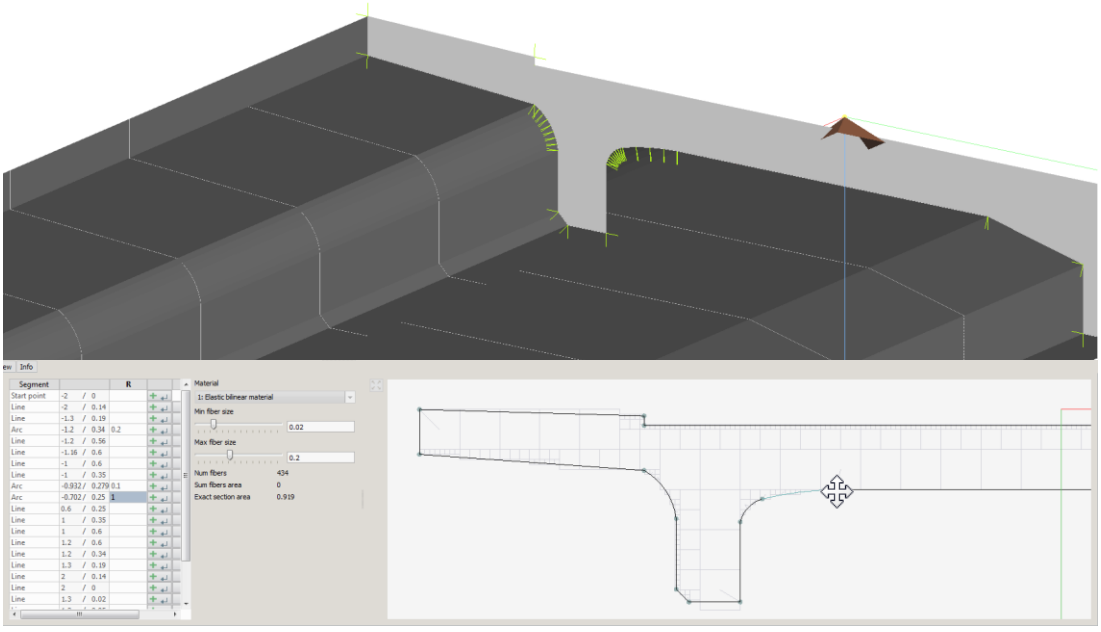


Abbildung 34: Querschnitts-Editor mit tabellarischer Eingabe (links unten), grafischer Interaktion durch Kontrollpunkte (rechts unten), und Steuerung der Diskretisierungsparameter (unten mittig)

[62] Mit „Echtzeit“ ist hier wieder die Aktualisierung „per-Frame“ gemeint.

Die **Interaktivität** bezieht sich beim Querschnitts-Editor also nicht nur auf die Modellparameter, wie die Lage der Querschnittspunkte, sondern auch auf Analyse-Parameter zur Steuerung der Genauigkeit der Diskretisierung. Für beides ist eine direkte visuelle Rückkopplung durch die Grafische Nutzeroberfläche implementiert, was eine interaktive Steuerung der Modellierung und Simulation ermöglicht.

Eine Verbesserung des Algorithmus wäre durch eine Quantifizierung der Über- bzw. Unterschätzung des tatsächlichen Flächenmoments $I_y = \int_A z^2\, dA \approx \sum A_i \cdot z_i^2$ und einer Reduktion des Fehlers auf einen vorgegebenen Wert möglich. Die Vorgabe des maximalen Fehlers würde die Vorgabe der minimalen und maximalen Fasergröße ersetzen. Da die Querschnitts-Diskretisierung hier aber nur einen Seitenaspekt darstellt, wurde dies nicht weiter umgesetzt.

Die Visualisierung der Spannungen/Dehnungen am Stabelement beruht nicht direkt auf der Faser-Repräsentation des Querschnitts, sondern dem tatsächlichen Umfangs-Polygon[63] (Abbildung 34). Um an diesen Polygon-Punkten Berechnungsergebnisse für die Farbcodierung zur Verfügung zu haben, wird eine zusätzliche Faser mit sehr geringer Fläche für jeden Polygonpunkt genau an dessen jeweiliger Position platziert.

Objektfang und Interaktion durch Zeigegeräte ist im Querschnittseditor für alle Kontrollpunkte des Querschnitts möglich.

Weitere Verbesserungsmöglichkeiten sind: In der vorliegenden Implementierung sind die Quadtree-Zellen quadratisch. Bei ungünstigen Seitenverhältnissen des umgebenden Polygons führt dies zu einer hohen Anzahl von sehr kleinen Fasern, um einen schmalen Streifen zu füllen, wie in Abbildung 32 oben. In diesem Fall ist die Kantenlänge der Fasern optimal für die Breite des Polygons, nicht für die Höhe. Rechteckige Fasern mit einem Breite/Höhe-Verhältnis innerhalb bestimmter Grenzen würden hier zu einer geringeren Anzahl von Fasern führen. Dies ist mit einigen Anpassungen in Algorithmus 4.5 bis Algorithmus 4.7 möglich, wurde hier aber noch nicht implementiert.

4.2.11 Bemessung

Da alle Stabelemente in Faserquerschnitte diskretisiert vorliegen, kann damit eine Querschnitts-Bemessung im Prinzip sehr einfach durch numerische Integration des vorhandenen Moments über alle Fasern i durchgeführt werden: $M_{Ed} = \sum A_i \cdot z_i \cdot \sigma_{E,d,i}$

Integration des maximal zulässigen Moments: $M_{Rd} = \sum A_i \cdot z_i \cdot \sigma_{R,d,i}$

Daraus ist ersichtlich, dass die nach Norm erforderliche Bemessung am Querschnitt in diesem Fall auf einen Spannungsvergleich in den Querschnitts-Integrationspunkten (Fasern) hinausläuft. Dies gilt für Stahlquerschnitte ebenso wie bewehrte Betonquerschnitte, da die Bewehrung sehr einfach durch einzelne Fasern modelliert werden kann.

Ist das Materialgesetz bereits unter Berücksichtigung der maximalen Spannung und Dehnung implementiert (und nicht etwa durch eine bilineare Kurve mit unendlich ansteigender Dehnung), so ist bereits der Spannungsnachweis an allen Integrationspunkten durch eine Konvergenz der FE-Analyse erbracht.

[63] Auch dieses ist streng genommen diskretisiert: Die exakte Randkurve besteht aus Linien und Kreisbögen, für die Quad-Tree-Diskretisierung und Visualisierung der Spannungen werden die Kreisbögen durch Polygone mit 15 Punkten pro Viertelkreis angenähert.

Für die vorliegende Arbeit und die prototypische Implementierung wurde die Bemessung jedoch nicht weiter verfolgt.

4.2.12 Modellierung der Elemente

Für die prototypische Anwendung wurden allgemeine räumliche Stabelemente implementiert. OpenSees unterstützt solche Elemente mit beliebig vielen Integrationspunkten entlang der Stabachse. An jedem Integrationspunkt kann ein Faserquerschnitt (vgl. Abschnitt 4.2.9) definiert werden, sodass auch Vouten etc. möglich sind (in der prototypischen Anwendung nicht implementiert).

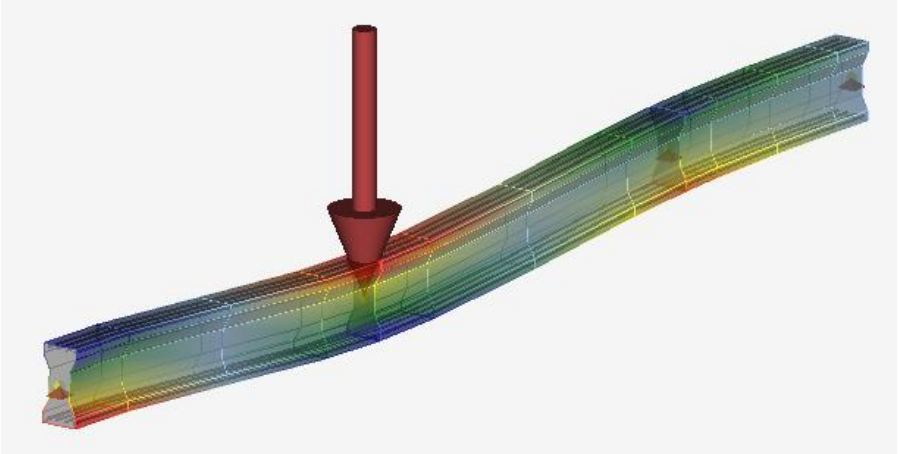


Abbildung 35: Stabelemente in semi-transparenter Darstellung

Serverseitig ist die Implementierung sehr einfach, es muss lediglich eine Konvertierung von SFA in ein entsprechendes OpenSees-Objekt vorgenommen werden.

Clientseitig werden alle für die Berechnung relevanten Daten in einem entsprechenden SFA-Objekt modelliert. Aufwendig ist vor allem die Implementierung der Visualisierung. Das kommt zum einen durch die in Form und Anzahl beliebigen Querschnitte entlang der Stabachse. Zum anderen sind die Integrationspunkte beliebig im Raum verschoben und verdreht, wie in Abbildung 35 gut sichtbar.

Um die Außenfläche eines Stabelements darzustellen, ist zunächst die Berechnung der Koordinaten aller Querschnittspunkte in allen Integrationspunkten erforderlich. Zusätzlich ist die Berechnung von Punktkoordinaten zur Darstellung der Bewehrung notwendig (Abbildung 36).

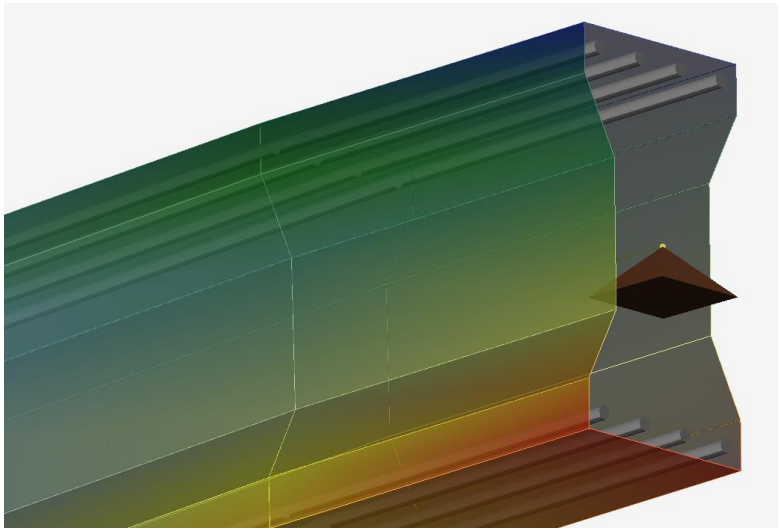


Abbildung 36: Darstellung der Bewehrung im Stabelement

In jedem Integrationspunkt muss die Verschiebung und Verdrehung auf alle Querschnitts- und Bewehrungspunkte angewendet werden.

Die eigentliche Oberflächendarstellung erfolgt durch Mengen von grafischen Primitiven, die durch OpenGL unterstützt werden. Dies sind im Wesentlichen Punkte, Linien, Dreiecke und Vierecke. Die in Abbildung 35 und Abbildung 36 dargestellten Stabelemente bestehen jeweils aus mehreren tausend solcher Primitive, die durch Indexfelder aus den zuvor berechneten Punkten zusammengesetzt werden. Für jede Fläche (Dreieck oder Viereck) ist die Berechnung eines Normalen-Vektors erforderlich, diese Vektoren werden in einem separaten Feld gespeichert und von OSG in den Speicher der GPU kopiert. Die Indizes der Normalen-Vektoren entsprechen denen der Punkte (Vertices) und da an einen Punkt angrenzende Primitive meist unterschiedliche Normalen haben, müssen auch die Punkte gedoppelt werden.

Wenn die Stabelemente als geschlossene Körper dargestellt werden sollen, müssen die Anfangs- und Endquerschnitte (in Abbildung 35 grau dargestellt, Abbildung 36 ohne Endquerschnitt) separat gerendert werden, da sie im Gegensatz zur Außenfläche aus einem allgemeinen, nichtkonvexen Polygon bestehen. Dazu ist eine sog. Tesselierung (Aufteilung eines allgemeinen Polygons in Dreiecke) erforderlich. Ein solcher Tesselator ist in OSG enthalten und musste daher hier nicht implementiert werden.

4.2.13 Synchronisierung des Analysemodells nach Änderungen

Das SFA-Modell besteht im Wesentlichen aus einer Klasse zur Verwaltung des Modells (Klasse *SfaModel*, vgl. Abbildung 37), sowie einer Menge von Entities. Werden einzelne Entities geändert, werden diese entsprechend markiert (*setFlagChanged()*, Abbildung 37). Anhand dieses Flags[64] kann jederzeit eine Synchronisierung des Analysemodells auf dem Server erfolgen. Ansonsten bleibt die Menge der Entities unverändert.

Wird eine Komponente (etwa ein Stabelement) gelöscht, so werden alle zugehörigen Entities aus der Menge der Entities ausgetragen, und in die Menge der gelöschten Entities eingetragen (Map *m_removed_entities* in Abbildung 37).

[64] Flag: Englisch für Flagge, Markierung

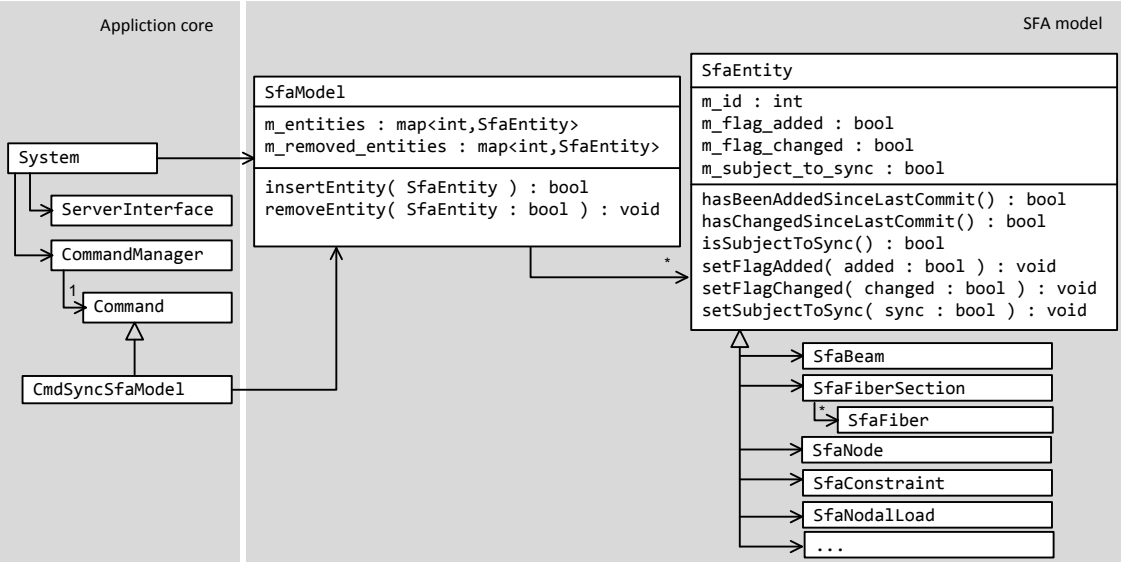


Abbildung 37: SFA-Modell

Vor einer Re-Analyse des Tragwerks, etwa nachdem Änderungen in einem Querschnitt vorgenommen wurden, wird die Synchronisierung des Modells durch einen Kommando-Aufruf durchgeführt. Dieses Vorgehen wird durch den untenstehenden Algorithmus beschrieben.

Algorithmus 4.8: **CmdSyncSfaModel**

Membervariablen: m_system : System

```
1   map<int, SfaEntity> removed_entities  = sfa_model->getRemovedEntities();
2   std::stringstream stream_removed;
3   stream_removed << "<ModelRemove><![CDATA["
4   foreach SfaEntity sfa_obj in removed_entities do
5       if sfa_obj->isSubjectToSync () then
6           stream_removed << "#" << sfa_obj->getEntityId() << ";";
7           sfa_obj->setSubjectToSync( true );
8       end if
9   end for
10  stream_removed << "</ModelRemove><![CDATA["
11  server_interface->sendData( stream_removed );
12
13  map<int, SfaEntity> all_entities  = sfa_model->getEntities();
14  std::stringstream stream_add;
15  stream_add << "<ModelAdd><![CDATA["
16  foreach SfaEntity sfa_obj in all_entities do
17      if sfa_obj->hasBeenAddedSinceLastCommit() then
18          sfa_obj->getStepLine( stream );
19          sfa_obj->setFlagAdded( false );
20          sfa_obj->setFlagChanged( false );
21          sfa_obj->setSubjectToSync( true );
22      end if
23  end for
24  stream_add << "</ModelAdd>
25  server_interface->sendData( stream_add );
26
27  std::stringstream stream_changed;
28  stream_changed << "<ModelChange><![CDATA["
29  foreach SfaEntity sfa_obj in all_entities do
30      if sfa_obj->isSubjectToSync () then
31          if sfa_obj->hasChangedSinceLastCommit() then
32              sfa_obj->getStepLine( stream );
33              sfa_obj->setFlagAdded( false );
```

```
34                                    sfa_obj->setFlagChanged( false );
35                              end if
36                        end if
37              end for
38              stream_change << “</ModelChange>
39              server_interface->sendData( stream_change );
40              end if
```

Zunächst werden demnach in den Schritten 1-11 alle aus dem Modell entfernten Entities durchlaufen und deren ID in einem Datenpaket (*ModelRemove*) an den Server gesendet.

Danach (Schritt 13-25) wird die Menge aller (aktuell im Modell enthaltenen) Entities durchlaufen und im Falle einer gesetzten Flag (*addedSinceLastCommitFlag*) dieses Entity in STEP-Repräsentation in ein Datenpaket (*ModelAdd*) eingefügt. In diesem Zuge wird das Entity in den Synchronisations-Mechanismus aufgenommen, indem wiederum ein Flag gesetzt wird (*setSubjectToSync*).

In den Schritten 27-40 wird nochmals die Menge aller Entities durchlaufen. Dabei werden in Schritt 30 und 31 diejenigen Entities herausgefiltert, die Gegenstand der Synchronisierung sind und sich geändert haben seit der letzten Synchronisierung. Falls beides zutrifft, wird das entsprechende Entity in STEP-Repräsentation übersetzt und in das Datenpaket *ModelChange* (Schritt 28) eingefügt.

Die Reihenfolge der Übertragung der Datenpakete *ModelRemoved*, *ModelAdd* und *ModelChange* ist nicht veränderbar, denn ein Entity kann aus dem Modell entfernt worden sein und gleich danach ein Entity mit gleicher ID eingefügt worden sein. Somit muss auch auf dem Server diese Reihenfolge eingehalten werden. Änderungs-Operationen können nur auf der Menge der aktuell im Modell vorhandenen Entities definiert sein, somit muss dieses Datenpaket zum Schluss übertragen werden. Wurde ein Entity geändert und gleich darauf gelöscht, so wird die Änderung durch den Algorithmus 4.8 nicht mehr übermittelt (das Löschen dagegen schon).

Wird ein Entity erzeugt und vor der nächsten Synchronisierung wieder gelöscht, so befindet es sich zum Zeitpunkt des Löschens nicht unter Synchronisation, wird nicht in die Menge der entfernten Entities eingetragen und ist somit bei Aufruf des Algorithmus 4.8 weder in der Menge der aktuellen Entities, noch in der Menge der gelöschten Entities enthalten.

Die Schritte 13-25 und 27-40 lassen sich auch in einer Schleife kombinieren, worauf hier aus Gründen der Übersichtlichkeit verzichtet wurde.

4.3 Aufbau des Datenmodells der Analyse-Schnittstelle IRAI

4.3.1 Schema-Definition

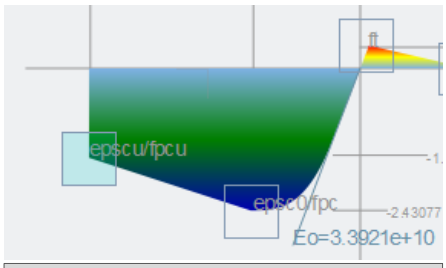
Die Schema-Definition der Simulations-Schnittstelle wurde hier als EXPRESS-Schema umgesetzt, der Aufbau der TYPE- und ENTITY-Datentypen lehnt sich an IFC an, weicht aber zugunsten wesentlich geringerer Komplexität und Eindeutigkeit hinsichtlich der Analyse (vgl. 3.3.7) davon ab.

4.3.2 Material

Für verschiedene Materialien wie Stahl oder Beton sind in SFA Entities definiert, angelehnt an die Materialmodelle in OpenSees. Ein Beispiel zeigt die Abbildung 38.

```
ENTITY SfaMaterialConcrete02
 SUBTYPE OF (SfaMaterial);
    CompressionStrength : SfaPressureMeasure;
    CompressionStrain : SfaStrainMeasure;
    UltimateStrength : SfaPressureMeasure;
    UltimateStrain : SfaStrainMeasure;
    UnloadingSlope : SfaRatioMeasure;
    TensileStrength : SfaPressureMeasure;
    TensionStiffening : SfaPressureMeasure;

DERIVE
    E0 : SfaPressureMeasure := 2.0*CompressionStrength/
CompressionStrain;
END_ENTITY;
```




```
TYPE SfaStrainMeasure = REAL;
END_TYPE;
```

Abbildung 38: EXPRESS-Modellierung von Beton in IRAI

Ein SFA-Materialobjekt kann im STEP-Format zur Kommunikation mit dem Analyse-Server verwendet werden, oder auch als anwendungsinternes Materialmodell. Dabei ist ein Wert wie die maximale Dehnung (*UltimateStrain* in Abbildung 38) nicht direkt durch eine reelle Zahl modelliert, sondern durch ein Objekt vom Typ *SfaStrainMeasure*[65] (Abbildung 38 unten rechts). Somit hat nicht nur der Attributname eine Semantik, sondern auch der Datentyp, wodurch u. U. Missverständnisse oder Fehler in der Programmierung oder Anwendung vermeiden werden können.

4.3.3 Querschnitte

In Abschnitt 4.2.9 wurde die Diskretisierung allgemeiner Querschnitte für eine nichtlineare Analyse erläutert. Die bei der Quadtree-Diskretisierung berechneten Fasern werden als Integrationspunkte für die Analyse verwendet. Faserquerschnitte können eindeutig in ein Analysemodell überführt werden (vgl. *FiberSection* in [McKenna 1997]), ebenso können die Berechnungsergebnisse gut auf das Ausgangsmodell abgebildet werden. Damit sind Faserquerschnitte eine gute Basis für die Querschnittsdefinition in SFA.

Jedes Faser-Objekt hat eine Referenz auf ein Material-Entity, dadurch sind zusammengesetzte Querschnitte und die Modellierung von Bewehrung möglich.

4.3.4 Bewehrung

Um Bewehrung in Querschnitten zu modellieren, sind keine weiteren Entities notwendig. Denn ein Bewehrungsstab (Abbildung 39) kann sehr einfach und mechanisch korrekt mit den vorhandenen Datentypen des Faserquerschnitts als Punkt repräsentiert werden (vgl. vorheriger Abschnitt).

[65] Dies kann in C++ auch als „typedef SfaStrainMeasure double“ implementiert werden

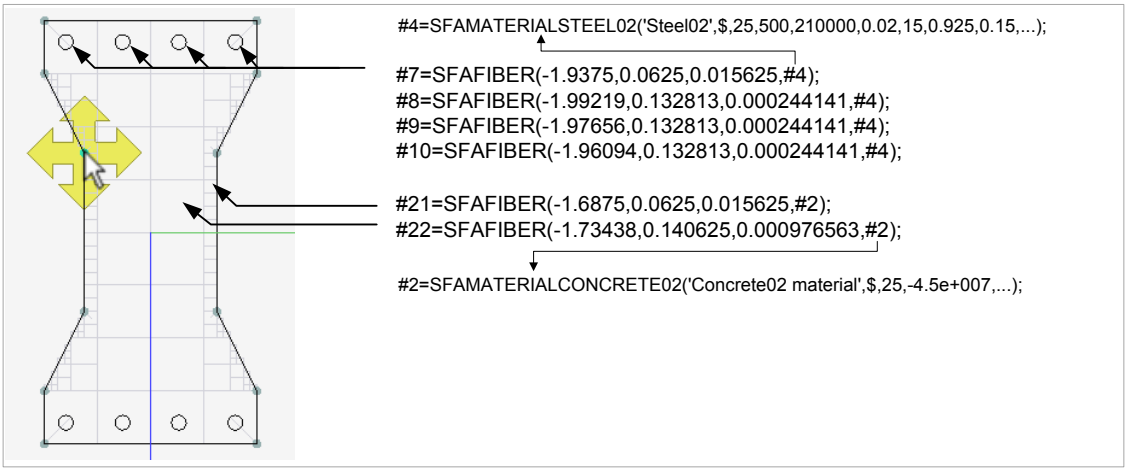


Abbildung 39: Modellierung der Bewehrung in SFA

Um als Anwender nicht jeden Bewehrungsstab einzeln positionieren zu müssen, sind im Komponentenmodell Objekte zur Handhabung einer ganzen Bewehrungslage implementiert (Klasse *ReinfLayer*, Abbildung 33), mit (interaktiv veränderbaren) Attributen Anfangspunkt, Endpunkt, Anzahl Bewehrungsstäbe, Material und Fläche pro Bewehrungsstab.

4.3.5 Lasten und Lastfälle

Entities zur Modellierung von Lasten sind in SFA zunächst auf Punkt- und Linienlasten für Stabelemente begrenzt, zumal für die prototypische Implementierung nur die von OpenSees unterstützten Lastarten sinnvoll sind.

Lastfälle sind als eine einfache Gruppe umgesetzt, die einen Faktor als Attribut haben, der auf alle zugehörigen Lastobjekte (genauer: Objekte, die von *SfaStructuralAction* abgeleitet sind) angewendet wird.

Da das Lastfall-Entity (*SfaStructuralLoadCase*) ebenfalls von *SfaStructuralAction* abgeleitet ist (Abbildung 40), kann ein ganzer Lastfall Bestandteil eines anderen Lastfalles sein. Somit lässt sich eine hierarchische Lastfall-Gruppierung realisieren.

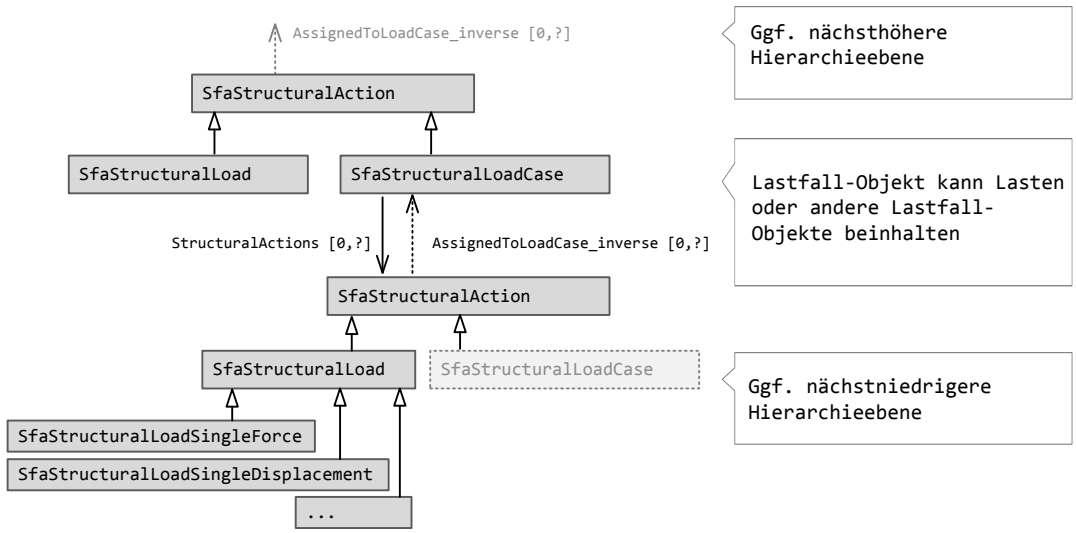


Abbildung 40: Lasten und Lastfälle in SFA

Linienlasten sind wie in IFC durch ein Entity (*SfaStructuralLoadLinearForce*) modelliert. Ein Vektor von Lastwerten (Entity *SfaLinearForceValue*) bestimmt den konstanten, trapezförmigen oder polygonalen Verlauf der Last.

4.4 Serverseitige Implementierung

4.4.1 Klassenaufbau

Der Aufbau des Rechenservers (Abbildung 41) ist wesentlich einfacher als der Aufbau der Modellierungsanwendung, da die Nutzeroberfläche nur aus einem kleinen Fenster zur Status-Anzeige besteht (Abbildung 18).

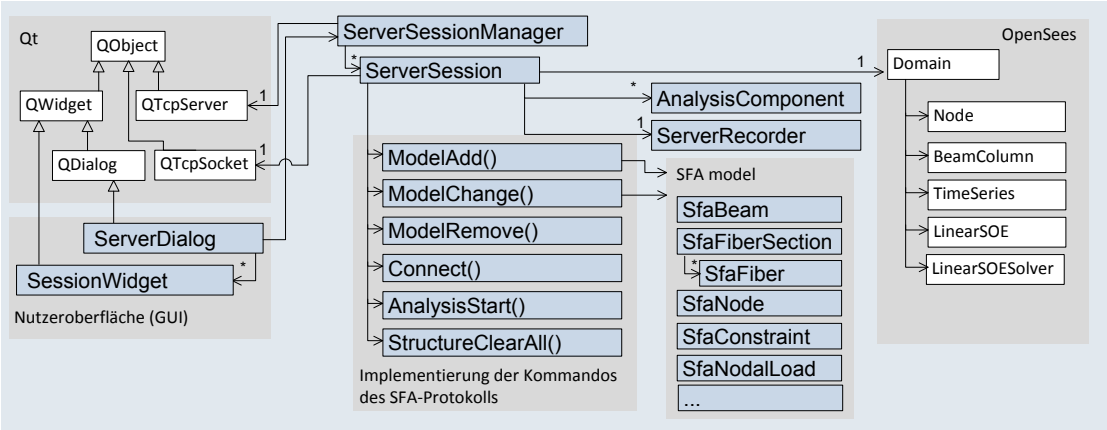


Abbildung 41: UML-Diagramm des IRAI-Analyse-Servers

Für die Netzwerkfunktionalität und die Nutzeroberfläche kommt wieder Qt zum Einsatz.

Die zentrale Klasse *ServerSessionManager* öffnet einen TCP-Port, über den mehrere Einzelverbindungen gleichzeitig angenommen werden können. Jede Einzelverbindung wird in einer sog. *Session* (Sitzung) verwaltet, die im Folgenden näher erläutert wird.

4.4.2 Session-Management

Verbindet sich eine Client-Anwendung über den TCP-Socket mit dem Server (und ist die Überprüfung der Benutzerkennung erfolgreich), wird eine Instanz einer *ServerSession*-Klasse (vgl. Abbildung 41) erzeugt. Diese Klasse übernimmt alle weiteren Auswertungen der Netzwerkdaten (Kommandos).

Eine Hauptaufgabe der *ServerSession*-Klasse besteht darin, die Kommandos des IRAI-Protokolls auszuwerten und auszuführen. Wie in Abschnitt 3.3.3 definiert, besteht jedes Kommando aus einem XML-Datenpaket, dessen Struktur zunächst überprüft und ausgewertet werden muss. Für jeden gültigen XML-Tag wird eine lokale Funktion ausgeführt (vgl. Abbildung 41 unten), die den jeweiligen Inhalt auswertet.

4.4.3 Auswertung des IRAI-Protokolls

Wie bereits in Abschnitt 3.3.4 beschrieben, sind Modelldaten des Strukturmodells in eine XML-Struktur eingebettet. Als Beispiel sei nochmals das Kommando zum Hinzufügen eines Knotens gezeigt:

```
<ModelAdd><![CDATA[     #48= SFANODE((0.0,-4.92,1.);     ]]></ModelAdd>
```

Abbildung 42: Kommando zum Hinzufügen von Analyse-Modelldaten

Ein beim Server ankommendes *ModelAdd*-Datenpaket führt zur Ausführung der entsprechenden Funktion (Abbildung 43), die als Argument die STEP-Daten erhält. Mit diesen Daten wird das Analysemodell aktualisiert.

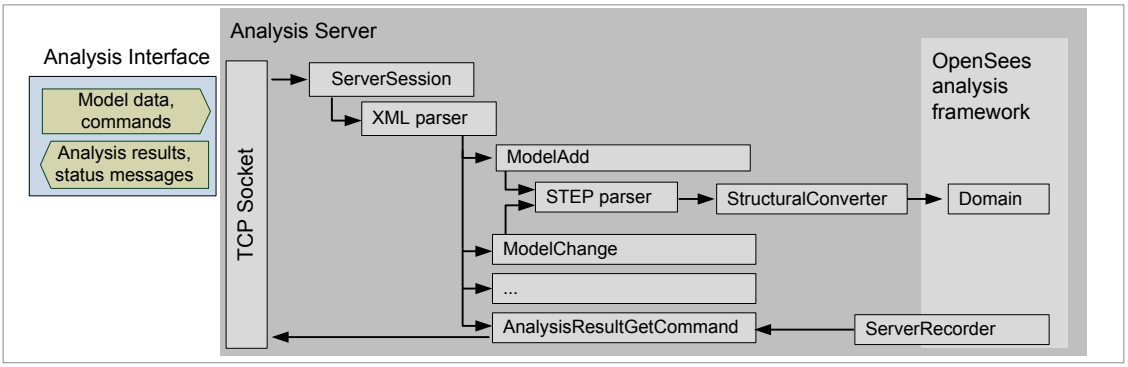


Abbildung 43: Interne Struktur des Analyse-Servers

Ähnlich wie beim Parsen von IFC-Modellen erfolgt das Parsen von SFA-Daten in mehreren Stufen. Im Gegensatz zum Laden von IFC-Daten wird hier von einem inkrementellen Hinzufügen bzw. Ändern von einzelnen oder mehreren Entities ausgegangen. Dabei muss das Parsen selbst nur für die Menge der hinzugefügten Entities erfolgen, das Herstellen der Referenzen jedoch auf der Menge der vorhandenen plus der hinzugefügten Entities. Dazu wurde hier folgender Algorithmus entwickelt:

Algorithmus 4.9: SfaStepDataAdd

Übergabeparameter: cdata : std::string, session : ServerSession

```
1   std::map<int,shared_ptr<SfaEntity> > map_existing = model->getMapEntities();
2   std::map<int,shared_ptr<SfaEntity> > map_existing_and_new( map_existing );
3   std::vector<std::string> step_lines;
4   // Zerlegen des Datenstroms in einzelne Zeilen (eine Zeile pro Entity):
5   splitIntoStepLines(step_lines)
6   std::vector<shared_ptr<SfaEntity> > vec_new_enties;
7   // Parsen der Datenzeilen (ID und Datentyp), Instanziierung der Objekte (ohne Auswertung der Attribute):
8   readStepLines(step_lines, vec_new_entities)
9   Kopieren der neuen Entities in die Map map_existing_and_new
10  // Parsen der Attribute nur der neuen Entities, Verlinken der Referenzen auf Basis aller Entities
11  readEntityArguments( vec_new_enties, map_existing_and_new );
12  // Initialisieren der inversen Attribute:
12  resolveInverseAttributes();
13  // Auswerten der Modelldaten:
14  std::vector<shared_ptr<SfaFiberSection> > vec_fiber_sections;
15  std::vector<shared_ptr<SfaBeamElement> > vec_beam_elements;
16  std::vector<shared_ptr<SfaStructuralLoadCase> > vec_load_cases;

17  for SfaEntity sfa_ent in vec_new_enties do
18      if Sfa_ent instance of SfaBeamElement then
19          Hinzufügen zu vec_beam_elements für spätere Auswertung
20          continue;
21      end if
22      if Sfa_ent instance of SfaNode then
23          Erzeugen eines Knoten-Objekts für OpenSees, Hinzufügen zur Domain
```

```
24                          continue;
25                  end if
26                  Analog Auswertung für Material-, Last und andereEntities.
27          end for
28          Übersetzen der Komponenten in vec_fiber_sections, vec_beam_elements, vec_load_cases
```

Der Algorithmus zum Ändern von Modelldaten („ModelChange“) unterscheidet sich von dem obigen Algorithmus nur geringfügig. Im Wesentlichen muss dabei in Schritt 9 darauf geachtet werden, dass ein Entity mit der selben ID[66] bereits in der Map *map_existing* enthalten ist, wenn die neuen Entities hinzugefügt werden. Ein einfaches Überschreiben des entsprechenden Wertes in der Map würde dazu führen, dass alle damit verlinkten Entities eine Referenz auf das alte Objekt behalten würden, das Modell wäre somit nicht korrekt aktualisiert und sogar inkonsistent durch mehrfach vorhandene Objekte und deren ID’s. Dieses Problem wurde gelöst, indem für alle Entities eine Methode generiert wurde, die als Übergabeparameter ein Objekt des selben Typs erwartet, und alle Attributwerte von diesem übernimmt:

```
void T::setEntity( shared_ptr<SfaEntity> other );
```

Die Methode *setEntity* funktioniert damit ähnlich wie ein Kopierkonstruktor, nur dass das Objekt selbst, und damit auch die Referenzen darauf, gültig bleiben.

Das *ModelChange*-Kommando führt somit tatsächlich nicht zum Hinzufügen neuer Objekte, sondern nur zum Erzeugen eines neuen Objekts des gleichen Typs als Vorlage für die Aktualisierung des vorhandenen Objekts mit gleicher ID.

Die Schritte 17 ff unterscheiden sich ebenfalls, denn auch die OpenSees-Objekte sollen nicht neu erzeugt, sondern inkrementell aktualisiert werden.

4.4.4 Modellierung der Ergebnisdaten

In Abschnitt 3.3 wurde bereits erwähnt, dass STEP im Prinzip gut geeignet ist, nicht nur Modelldaten in einem maschinen- und menschenlesbaren Format abzubilden, sondern auch Ergebnisdaten numerischer Simulationen. Ein einfaches Beispiel ist der Verschiebungsvektor eines Knotens:

```
#100002=SFARESPONSENODE($,1,(1.47854e-04,-3.67128e-02,0),#452);
```

Die Attribute dieses Entities sind ein optionaler Name („$“), ein Zeitstempel für dynamische Analysen, ein Vektor der Verschiebungen und eine Referenz auf den zugehörigen Knoten.

Das entsprechende EXPRESS-Schema für alle Response-Entities findet sich in Anhang B.

Nicht nur einfache Vektoren wie in obigem Beispiel lassen sich in STEP abbilden, sondern auch mehrdimensionale Vektoren zur Modellierung von Spannungstensoren oder anderen Ergebnisgrößen. Die EXPRESS-Notation mehrdimensionaler Attribute schreibt sich folgendermaßen:

```
Tensor0:  REAL;
Tensor1:  LIST [0:?] OF REAL;
Tensor2:  LIST [0:?] OF LIST [0:?] OF REAL;
Tensor3:  LIST [0:?] OF LIST [0:?] OF LIST [0:?] OF REAL;
```

Die entsprechende Repräsentation in STEP wird durch Schachtelung von Vektoren erreicht:

[66] ID: eindeutiger Identifikator

```
Tensor0:    1.4
Tensor1:    (1,-3,0)
Tensor2:    ((1,-3,0), (1,-3,0), (1,-3,0))
Tensor3:    (((1,3,0), (1,3,0), (1,3,0)), ((1,-3,0), (1,3,0), (1,3,0)))
```

Die obigen Datentypen finden folgendermaßen ihre Entsprechung in C++:

```
Tensor0:    double tensor0;
Tensor1:    std::vector<double> tensor1;
Tensor2:    std::vector<std::vector<double> > tensor2;
Tensor3:    std::vector<std::vector<std::vector<double> > > tensor3;
```

Die Übersetzung des EXPRESS-Schemas in ein objektorientiertes Modell (C++-Quellcode) übernimmt die Anwendung *IfcExtender* (vgl. Abschnitt 4.2.3). Durch die Kompatibilität des SFA-Schemas zu IFC sind keine Änderungen notwendig (beide sind in EXPRESS definiert). Ebenfalls ohne spezielle Anpassungen kann der SFA-Parser generiert werden, um SFA-Modell-Entities und SFA-Response-Entities aus STEP-Daten zu instanziieren. Auch das Erzeugen von STEP-Daten aus einem instanziierten Modell erfolgt durch Methoden, die ohne Änderungen mit *IfcExtender* generiert werden können.

Die Response-Entities werden durch den Parser als Objekte instanziiert und können einfach als Attribute im Komponentenmodell eingefügt werden, so dass Knoten, Stabelemente etc. ihre Visualisierung aktualisieren können. Es ist keine Umwandlung in andere Datenstrukturen notwendig, da die Daten der Response-Entities bereits als (ggf. mehrdimensionaler) Vektor vorliegen.

Die Response-Entities können auch in das SFA-Modell der Komponenten eingefügt und durch Referenzen und inverse Attribute mit dem übrigen Modell verlinkt werden. Dies ist jedoch zur Laufzeit nicht notwendig, und wäre am sinnvollsten zur Dokumentation des Projektes in einer STEP-Datei. Darin können Modelldaten ebenso wie Ergebnisdaten in einem einheitlichen Format archiviert werden.

4.4.5 Paralleles Lesen von STEP-Daten

In der in Abschnitt 4.2.2 beschriebenen IFC-Implementierung werden Daten in mehreren Schritten eingelesen:

1. Ausschneiden von Kommentaren.
2. Zerlegen des Datenstroms in einzelne Zeilen (1 Zeile pro Entity).
3. Parsen der Datenzeilen (ID und Datentyp), Instanziierung der Objekte (ohne weitere Attribute).
4. Parsen der Attribute, Initialisierung der Objekte.

Die Schritte 3 und 4 müssen nacheinander erfolgen, da im Schritt 4 Referenzen zu anderen Objekten hergestellt werden. Dazu müssen alle Objekte bereits existieren, auch wenn die Attribute noch nicht initialisiert sind.

Die Schritte 3 und 4 sind jeweils gut parallelisierbar, da die Menge der IFC-Objekte einfach unterteilbar ist und unabhängig voneinander ausgewertet werden kann. Schritt 2 könnte mit etwas mehr Aufwand auch parallel verarbeitet werden durch Unterteilung des Datenstroms in n Blöcke (n entspricht der Anzahl paralleler Prozesse). Schritt 2 macht jedoch ohnehin nur wenige Prozent der Verarbeitungszeit des Parsens aus und so wäre hier der Aufwand nicht gerechtfertigt.

Durch eine Präprozessor-Anweisung für OpenMP[67] wird Schritt 4 parallel ausgeführt:

```
const int num_objects = vec_empty_objects.size();
#pragma omp parallel for shared(vec_empty_objects, map)
for( int i=0; i<num_objects; ++i )
{
  ReaderContainer* c = vec_empty_objects[i];
  readIfcEntityArgs( c->m_obj, c->m_type_enum, c->args, map );
}
```

In diesem Beispiel ist also im Wesentlichen nur eine Zeile Präprozessor-Code notwendig, sowie das Setzen des Compiler-Flags „-openmp“ um den kompletten Schleifeninhalt auf n Prozessoren zu verteilen. Die eigentlich zeitaufwändigen Operationen befinden sich in der Funktion *readIfcEntityArgs()*, die somit parallel ausgeführt werden können. Dies ist möglich, da zwar Referenzen zu anderen Entities hergestellt werden, aber über diese Referenzen hier noch keine Zugriffe erfolgen. Obiges Beispiel ist vereinfacht dargestellt, die tatsächliche Umsetzung enthält noch eine Textstrom-Variable zur Fehlerprotokollierung und einen entsprechenden Zugriffsschutz (Barriere), damit nicht mehrere Threads gleichzeitig auf eine gemeinsame Variable zur Speicherung von Fehlermeldungen zugreifen[68].

Auf einer Vierkern-CPU (Intel Core2 Quad @2.4 GHz) konnten so folgende Beschleunigungen in Abhängigkeit der Modellgröße erreicht werden:

	Anzahl Entities	Seriell [ms]	Parallel [ms]	Skalierung
Modell 1	1414	69	48	1.44
Modell 2	17176	403	269	1.50
Modell 3	184489	2447	1218	2.01
Modell 4	1033909	12568	4538	2.77
Modell 5	5276285	53826	10076	2.68

Eine Skalierung um 2.7 für größere Modelle ist plausibel, da die Schritte 1 und 2 in jedem Fall sequentiell ausgeführt werden, und ein gewisser Aufwand für die Verwaltung der Threads erforderlich ist.

4.4.6 Komprimierung und Verschlüsselung der Datenübertragung

Eine **Komprimierung** der Modell- und Ergebnisdaten über TCP ist grundsätzlich möglich und ab einer gewissen Datenmenge auch sinnvoll, um die Antwortzeiten des Analyseservers so gering wie möglich zu halten. Testweise wurde hier eine Komprimierung mittels zLib[69] implementiert.

Dabei wurde nur der CDATA-Bereich der Datenpakete komprimiert. Dies geschieht aus folgenden Gründen: zum einen ist dies bei weitem das größte Datenvolumen. Zum anderen ist zur

[67] OpenMP: Open Multi-Processing. Offener Standard für parallele Programmierung für C++ und Fortran, http://openmp.org

[68] Der komplette Quellcode ist einsehbar unter http://code.google.com/p/ifcplusplus/

[69] Standardbibliothek unter Open Source Lizenz für Komprimierungen nach dem Deflate-Algorithmus

Dekomprimierung die Länge des Datenpaketes erforderlich (der TCP-Socket liefert nur einen kontinuierlichen Datenstrom). Die Länge müsste also zusätzlich (unkomprimiert) übermittelt werden, was einen zusätzlichen Aufwand bedeuten würde.

Im Falle einer Komprimierung kann dies als Attribut in der XML-Struktur angegeben werden:

```
<ModelAdd compression="deflate"><![CDATA[
Compressed STEP data
]]></ModelAdd>
```

Da der Analyse-Server hier hauptsächlich über den lokalen Host bzw. über Local Area Network (LAN) angebunden wurde, ergab sich kein signifikanter Geschwindigkeitsvorteil; der zusätzliche Aufwand zur Komprimierung hielt sich mit dem geringeren Aufwand der Datenübertragung in etwa die Waage.

Eine **Verschlüsselung** ist hauptsächlich dann sinnvoll, wenn eine Verbindung über das Internet besteht. Dadurch können übertragene Modelldaten abgesichert werden gegen eine mögliche Ausspähung auf Knotenpunkten der Verbindung zwischen Client und Server. Auf Basis des TCP-Protokolls könnte hier mit relativ geringem Aufwand eine ebenfalls standardisierte Technik wie SSL[70] verwendet werden. Dies wurde jedoch im Rahmen dieser Arbeit nicht weiter untersucht oder umgesetzt.

Die Sicherstellung der Datenintegrität wird durch das TCP-Protokoll bereits gewährleistet. TCP ist verbindungsorientiert und zuverlässig, d. h. alle gesendeten Daten kommen vollständig und in der richtigen Reihenfolge beim Empfänger-Socket an.

4.4.7 Rollback und Fehlerbehandlung

Tritt serverseitig ein inkonsistenter Modellzustand ein, wird eine entsprechende Meldung an den Client geschickt (<FatalError message="…" />). Dies kann beispielsweise der Fall sein, wenn sich ein Änderungskommando auf ein zuvor gelöschtes Entity bezieht. Dann hat eine Client-Anwendung die Möglichkeit, ein *ModelClear*-Kommando an den Server zu schicken, und danach das komplette Modell (nicht inkrementell) zu übermitteln.

Eine solche Neu-Übermittlung ist auch erforderlich, wenn der Serverprozess durch einen schweren Laufzeitfehler beendet und neu gestartet werden muss. Dann wäre theoretisch auch eine automatische Session-Wiederherstellung möglich. Dies wurde hier aber nicht implementiert.

4.4.8 Persistentes Analysemodell

Im Gegensatz zum klassischen Konzept der Eingabedatei erlaubt die hier vorgeschlagene Schnittstelle ein persistentes Analysemodell, das nicht nach jeder Änderung vollständig neu aufgestellt werden muss. So können viele Objekt-Instanzen bei Modelländerungen unverändert bleiben und Speicherbereiche zur Aufstellung von Gleichungssystemen können erhalten bleiben. Somit besteht viel Potential zur Beschleunigung der Re-Analyse.

4.4.9 Anpassungen des FE-Frameworks OpenSees

Ursprünglich wurde das Framework OpenSees als Konsolen-Anwendung entwickelt. Das zeigt sich daran, dass bei einem Fehler eine entsprechende Fehlermeldung in die Standardausgabe gegeben wird, und danach mit *exit()* der komplette Prozess beendet wird. Egal in welche Art Anwendung OpenSees dabei integriert ist, z.B. eine grafische Anwendung, sie wird durch *exit()* samt aller

[70] SSL: Secure Socket Layer

Threads sofort beendet, was natürlich generell nicht erwünscht ist – außer bei einer reinen Konsolenanwendung.

Im Rahmen dieser Arbeit wurde ein Fehlermanagement durch *Exceptions* implementier. Diese können an Stellen „geworfen“ werden, an denen ein Fehler identifiziert wird (z.B. Anfangsknoten eines Stabelementes ist ein Null-Pointer). Eine Exception kann dort „aufgefangen“ werden, wo der entsprechende Fehler behandelt werden kann, etwa in einem Kommando zur Änderung eines Stabelements.

Alle *exit()*-Anweisungen (ca. 1800) in OpenSees wurden hier durch globales Suchen & Ersetzen mittels eines regulären Ausdrucks[71] in Exceptions überführt, sodass der Text der Fehlermeldung erhalten bleibt:

```
if (theEigenvalues == 0) {
    opserr << "Domain::getEigenvalues - Eigenvalues were never set\n";
    exit(-1);
  }
```

wird zu

```
if (theEigenvalues == 0) {
    throw OpenSeesException( "Domain::getEigenvalues - Eigenvalues were never
    set\n", __func__ );
  }
```

Eine weitere Anpassung ist das Speichermanagement. Hier besteht in OpenSees bereits die Klasse *TaggedObject*, die sich als Ansatzpunkt für ein Speichermanagement eignet. Von ihr sind die Klassen *FE_Element*, *DomainComponent* und von diesen wiederum direkt oder indirekt praktisch alle OpenSees-Klassen abgeleitet. So konnten hier durch Einfügen eines Referenzzähler-Attributes und der Methoden *ref()* und *unref()* praktisch alle OpenSees-Klassen vorbereitet werden zur Speicherverwaltung durch OpenSceneGraph-Smart-Pointer Templates. Da in einem FE-Framework ebenso wie im Szenegraphen große hierarchische Datenstrukturen auftreten können (die meisten FE-Objekte haben eine Entsprechung im Szenegraphen zur Visualisierung), ist auch die Analogie in der Speicherverwaltung sinnvoll.

Normalerweise und in seiner ursprünglichen Form wird OpenSees per Eingabedatei als Konsolenanwendung gestartet. Alle Objekte werden einmal erzeugt, in die Domain eingefügt, berechnet, und die Ergebnisse in Textdateien geschrieben. Anschließend wird das zentrale Domain-Objekt gelöscht. Dieses wiederum löscht im Destruktor kaskadenartig die meisten bestehenden OpenSees-Objekte. Dabei ist eine saubere Bereinigung des gesamten Speichers und eine sichere Vermeidung von Speicherlecks nicht besonders wichtig, denn der (per Konsole gestartete) Prozess wird anschließend ohnehin beendet, und der gesamte Speicher wieder freigegeben.

Ein sauberes Bereinigen des Speichers ist in einer Anwendung, die als Serverprozess über mehrere Tage und Wochen läuft, und unzählige Sessions abarbeitet, jedoch offensichtlich von großer Wichtigkeit.

[71] Suchmuster in VS-Notation: „opserr:b*\<\<:b*{\"[a-zA-Z0-9\\\:\-:b]+\"};\n:b+exit\([:b\-0-9]*\);“
Treffer ersetzen durch: „throw OpenSeesException(\1, __func__);“

Komplexe Anwendungen mit großen hierarchischen Datenstrukturen, wie Szenegraphen oder FE-Modelle, sind ohne ein sauberes Konzept zur Speicherverwaltung nur schwer umsetzbar [Kuehne, Martz 2007].

Als Beispiel für die Umsetzung der Speicherverwaltung in OpenSees sei die Klasse *StaticAnalysis* genannt:

Im ursprünglichen Zustand hatte diese Klasse verschiedene Attribute mit unterschiedlichen komplexen Datentypen: *ConstraintHandler*, *DOF_Numberer*, *AnalysisModel*, *EquiSolnAlgo*, *LinearSOE*, *EigenSOE*, *StaticIntegrator*, und *ConvergenceTest*. Die Objekte mit diesen Datentypen werden dem Konstruktor übergeben und im Destruktor gelöscht. Dies kann zu Problemen führen, denn die Klasse *StaticAnalysis* kann nicht wissen, ob irgendwo anders noch Referenzen auf eines der Objekte bestehen und ob auf diese Objekte noch zugegriffen werden soll.

Zur Vermeidung dieses Problems wurden alle o. g. Attribute durch das Template *osg::ref_ptr<T>* umschlossen, und die *delete*-Anweisungen aus den Destruktoren entfernt.

Die geänderte Version von OpenSees steht in einem Online-Repository zur Verfügung:

https://code.google.com/p/sfa/

4.5 Reaktionszeit der Datenübertragung und der numerischen Analyse

Eine schnelle Antwortzeit der numerischen Simulation ist entscheidend für einen interaktiven Tragwerksentwurf. Daher sollen hier zur Verifizierung des gewählten Ansatzes die Reaktionszeiten für verschiedene Modellgrößen ausgewertet werden.

Wie im vorigen Abschnitt beschrieben, erfolgt das Lesen der STEP-Daten (Modelldaten und Ergebnisdaten) weitgehend parallel, sodass dies bei einem Mehrkern-Prozessor keinen „Flaschenhals“ im interaktiven Revisions-Zyklus darstellt. Zur Datenübertragung zwischen Client-Maschine und Analyse-Server diente ein herkömmliches 1Gb-Netzwerk (LAN).

Zur Zeitmessung wurde ein Rahmentragwerks-Modell implementiert (aufrufbar in der prototypischen Anwendung), das aus einer variablen Anzahl von Knoten und Stäben besteht. Clientseitig und serverseitig wurden mehrere Zeitmesspunkte implementiert, um den zeitlichen Ablauf einer Simulation in allen Phasen zu erfassen (Abbildung 44, Abbildung 45).

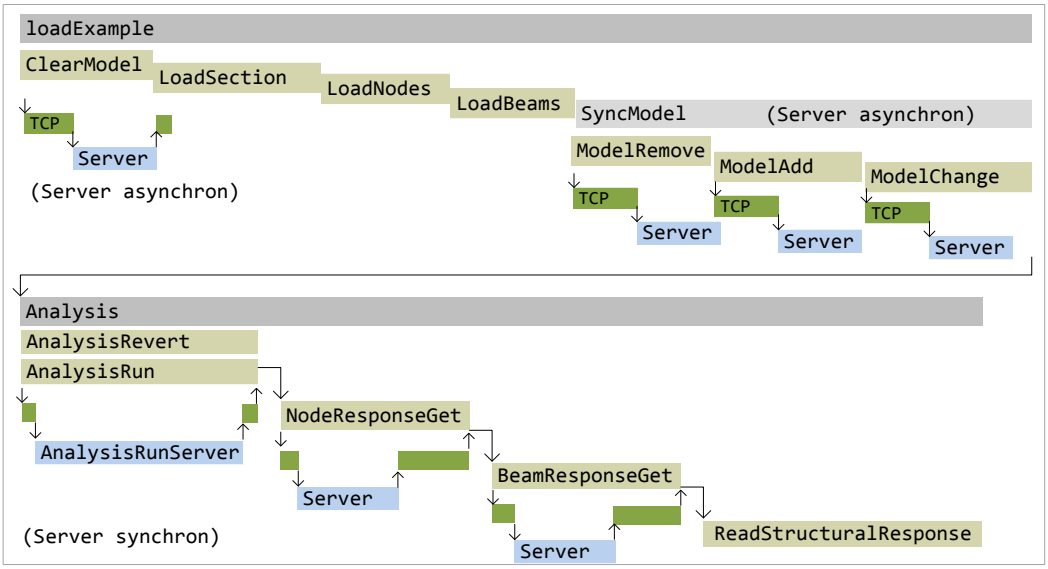


Abbildung 44: Zeitliche Abfolge der IRAI-Datenpakete und Berechnung

Die gemessenen Ergebnisse zeigen die oben dargestellten Berechnungsphasen in Abhängigkeit der Modellgröße, von 41 bis 7501 Knoten und 40 bis 7500 Stabelementen. Dies entspricht einer Anzahl an Freiheitsgraden (x-Achse in Abbildung 45) von 246 bis 45006.

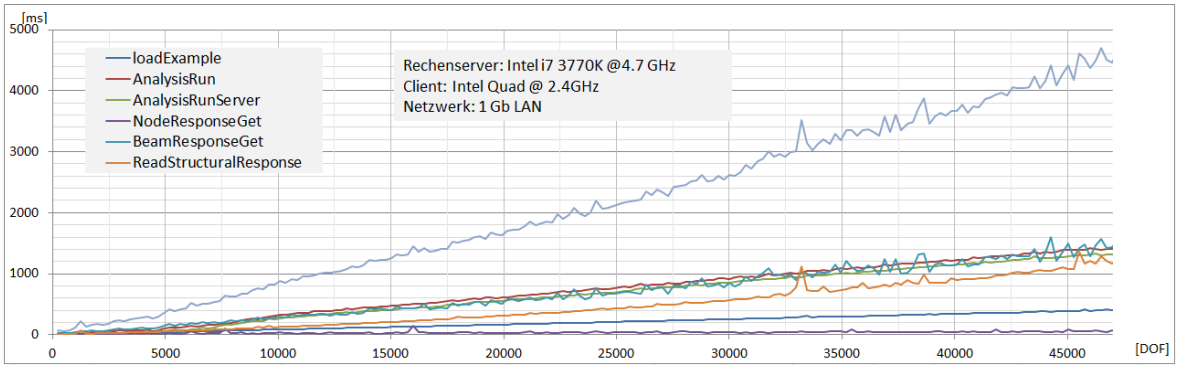


Abbildung 45: Dauer der Analyse und der Datenübertragung in Abhängigkeit der Modellgröße

Der Querschnitt ist für alle Elemente der gleiche, ein einfacher Rechteckquerschnitt mit 12 Fasern und linear-elastischem Materialgesetz. Der geometrische Aufbau des Rahmentragwerks ist hier nicht entscheidend. Ausschlaggebend für die Dauer der Datenübertragung und der numerischen Analyse ist lediglich die Anzahl der Knoten, Elemente und Freiheitsgrade.

Wählt man die Grenze für intuitives und damit „schnelles“ Feedback der Ergebnis-Visualisierung mit einer halben Sekunde, so entspricht dies auf der angegebenen Hardware einer Modellgröße von etwa 7000 Freiheitsgraden, also 1170 Knoten und ebenso vielen Stabelementen.

Subjektiv kann dies durch Verschieben eines Knotens in einem Stabwerksmodell der entsprechenden Größe bestätigt werden: die Visualisierung der Spannungen wird mit geringen Verzögerungen nachgeführt. Beim Absetzen des Knotens an einer Position wird auch die Verschiebung visualisiert, sodass eine intuitive Abschätzung der Auswirkungen des zuletzt ausgeführten Kommandos möglich ist.

Bei nichtlinearen Materialgesetzen erhöhen sich die Rechenzeiten erheblich, sodass interaktiver Tragwerksentwurf nur mit entsprechend kleineren Modellen möglich ist.

4.6 Validierung der numerischen Analyse

Die Kapselung der numerischen Analyse in einem Server hat den Vorteil, dass dieser jederzeit als gesamte Einheit getestet werden kann. Um die Zuverlässigkeit der Berechnungsergebnisse sicherzustellen, kann der Server mit der Lösung verschiedener Systeme beauftragt werden, von denen die Lösung entweder analytisch, durch Laborversuche oder durch Messungen an realen Bauwerken bekannt ist.

Während die Validierung der numerischen Simulation bei herkömmlicher Softwarearchitektur Teil der Qualitätssicherung eines Herstellers ist, kann dies bei der serverbasierten Simulation auch gut extern vorgenommen werden.

Beispielhaft ist in Abbildung 46 ein System mit nichtlinearer Push-over-Analyse dargestellt und in Abbildung 47 der Vergleich der Ergebnisse, der in diesem Fall eine gute Übereinstimmung zeigt und somit eine erfolgreiche Validierung.

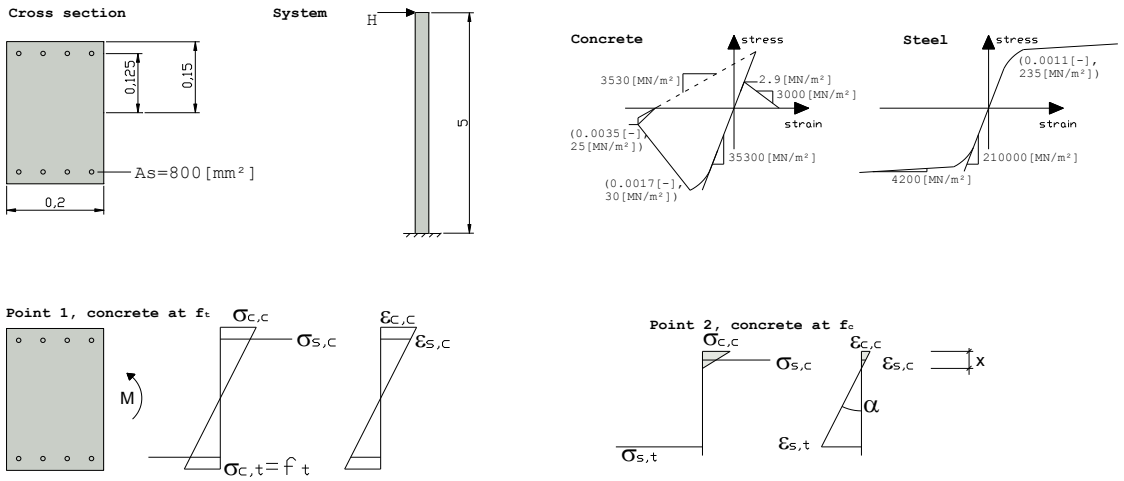


Abbildung 46: Testsystem und analytische Lösung

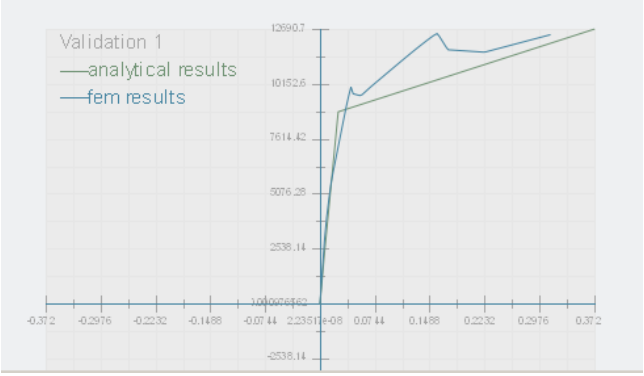


Abbildung 47: Gegenüberstellung der analytischen und numerischen Berechnungsergebnisse. Einheiten: [m], [N]

Grundsätzlich sind zwei Arten der Validierung in der o. g. Architektur möglich:

- **Per-Session**: Die Client-Anwendung prüft einen Server durch ein einfaches statisches System mit bekannten Ergebnissen zu Beginn einer Session.
- **Bei Software-Release:** Eine unabhängige Institution prüft eine Server-Schnittstelle durch eine große Anzahl statischer Systeme mit bekannten Ergebnissen, entweder analytisch oder durch Messungen. Diese Prüfung kann unregelmäßig, oder nach Aktualisierungen der serverseitigen Software erfolgen.

4.7 Einschränkungen der prototypischen Implementierung

Clientseitig wie serverseitig werden nur Stabelemente unterstützt, d. h. keine Schalenelemente und keine Volumenelemente.

Lasten können nur als Punktlasten modelliert werden. Auch OpenSees unterstützt gegenwärtig nur Knotenlasten und gleichmäßige Streckenlasten über die gesamte Stablänge.

Lastfallkombinationen, Teilsicherheitsbeiwerte und Nachweise der Tragsicherheit und Gebrauchstauglichkeit wurden nicht implementiert.

Die Nutzerinteraktion ist implementiert für nichtlineare Materialmodedelle und allgemein polygonal umrandete Querschnitte von Stabelementen. Das Verschieben von Knoten im 3D-Modell durch ein Zeigegerät ist nur in einer Ebene (x-z-Ebene) möglich.

Die erarbeitete IFC-Implementierung kann grundsätzlich alle Daten verarbeiten, die dem Daten-Schema IFC4 entsprechen. Jedoch wurde hier nur ein Import von Materialien, Lasten und Elementen der *Structural Analysis Domain* vorgenommen. Elemente eines architektonischen IFC-Modells in ein Strukturmodell zu überführen, war nicht Gegenstand der prototypischen Implementierung.

5 Anwendungsbeispiele

5.1 Textbasiertes Anwendungsbeispiel

Das erste Anwendungsbeispiel ist ein Minimalbeispiel, das die Funktionsweise und Vorteile des IRAI-Protokolls als menschenlesbares (und nicht nur maschinenlesbares) Format verdeutlichen soll.

Das statische System und die Materialien entsprechen dem Validierungsmodell aus Abschnitt 4.6, Abbildung 46.

Verbindungsaufbau mit dem IRAI-Server:

```
<Connect       ProtocolType="SFA"       ProtocolVersion="1"       UserId="109"
Password=”5eb63bbbe01eeed093cb22bb8f5acdc3” />
```

```
<ConnectResponse SessionId="16684"></ConnectResponse>
```

Damit ist der Verbindungsaufbau abgeschlossen. Modelldaten können nun gesendet werden:

```
<ModelAdd><![CDATA[
HEADER;
FILE_SCHEMA(('SFA1'));
ENDSEC;
DATA;
#1=SFASTRUCTURALLOADCASE(#2,(),(#24));
#2=SFATIMESERIESCONSTANT(1);
#3=SFASTRUCTURALANALYSISMODELSTATIC('static',$,1);
#4=SFASTRUCTURALANALYSISMODELTRANSIENT($,$,1,0.1,1e-008,6);
#5=SFAMATERIALBILINEAR('Elastic bilinear material',$,25,0.002,3e+007,1.5e+009);
#6=SFAMATERIALCONCRETE02('Concrete02          material',$,25,-4.5e+007,-0.0017,-4e+007,-
0.0035,0.1,3.9e+006,3e+009);
#7=SFAMATERIALHYSTERETIC('Steel',$,25,500,0.00238095,550,0.025,500,0.03,-500,-
0.00238095,-550,-0.025,-500,-0.03);
#8=SFAMATERIALSTEEL02('Steel02',$,25,500,210000,0.02,15,0.925,0.15,0,0,0,0);
#9=SFAMATERIALELASTIC('Elastic',$,25,3e+007);
#10=SFAFIBERSECTION((#25,#26,#27,#28,#29,#30,#31,#32));
#19=SFABOUNDARYNODECONDITION($,SFABOOLEAN(.T.),SFABOOLEAN(.T.),SFABOOLEAN(.T.),SFABOOLEA
N(.T.),SFABOOLEAN(.T.),SFABOOLEAN(.T.));
#20=SFANODE((0,0,0),#19);
#21=SFANODE((5,0,0),$);
#22=SFABEAMELEMENT($,$,$,(#10,#10,#10,#10,#10,#10,#10,#10),#20,#21,#23);
#23=SFADIRECTION((0,0,1));
#24=SFASTRUCTURALLOADSINGLEFORCE($,#21,$,$,0,0,500000,0,0,0);
#25=SFAFIBER(-0.1,-0.3,0.04,#5);
#26=SFAFIBER(-0.1,-0.1,0.04,#5);
#27=SFAFIBER(0.1,-0.1,0.04,#5);
#28=SFAFIBER(0.1,-0.3,0.04,#5);
#29=SFAFIBER(-0.1,0.1,0.04,#5);
#30=SFAFIBER(-0.1,0.3,0.04,#5);
#31=SFAFIBER(0.1,0.3,0.04,#5);
#32=SFAFIBER(0.1,0.1,0.04,#5);
ENDSEC;
END-ISO-10303-21;
]]></ModelAdd>
<AnalysisRevert analysisId="3"/>
<AnalysisRun analysisId="3"/>
```

Die Bestätigung der direkt ausführbaren Kommandos wird unmittelbar gesendet:

```
<ModelAddResponse />
<AnalysisRevertResponse />
```

Anschließend wird die Analyse ausgeführt. Ist diese beendet, wird die entsprechende Bestätigung an den Client gesendet:

```
<AnalysisRunResponse analysisTime="9" />
```

Die Client-Anwendung kann nun die Berechnungsergebnisse anfordern:

```
<NodalResultGet nodeId="all" />
<ElementalResultGet elementId="all" />
```

```
<NodalResultGetResponse timeStep="1"><![CDATA[DATA;
#100000=SFARESPONSENODE($,1,(0,0,0),#20);
#100001=SFARESPONSENODE($,1,(0,0,0.0868056),#21);
ENDSEC;
]]></NodalResultGetResponse>
<ElementalResultGetResponse timeStep="1"><![CDATA[
DATA;
#100002=SFARESPONSEBEAM($,1,((0,0,0),(0.32065,0,0),(1.02075,0,0),(1.97675,0,0),(3.02325,
0,0),(3.97925,0,0),(4.67935,0,0),(5,0,0)),((0,0,0,0,0,0,0),(0,2.62919e-
020,0.000715953,0,0,0),(0,1.92515e-019,0.00622757,0,0,0),(0,-1.65022e-
020,0.019305,0,0,0),(0,-1.39382e-020,0.0391433,0,0,0),(0,2.04001e-
020,0.0610089,0,0,0),(0,-1.78816e-
021,0.0785134,0,0,0),(0,0,0.0868056,0,0,0)),(#100003,#100004,#100005,#100006,#100007,#10
0008,#100009,#100010),(),#22);
#100003=SFARESPONSEFIBERSECTION($,1,(3.35649e+007,2.18829e+007,2.18829e+007,3.35649e+007
,-2.18829e+007,-3.35649e+007,-3.35649e+007,-2.18829e+007));
#100004=SFARESPONSEFIBERSECTION($,1,(3.24775e+007,1.82583e+007,1.82583e+007,3.24775e+007
,-1.82583e+007,-3.24775e+007,-3.24775e+007,-1.82583e+007));
#100005=SFARESPONSEFIBERSECTION($,1,(3.01033e+007,1.03443e+007,1.03443e+007,3.01033e+007
,-1.03443e+007,-3.01033e+007,-3.01033e+007,-1.03443e+007));
#100006=SFARESPONSEFIBERSECTION($,1,(2.40364e+007,8.01215e+006,8.01215e+006,2.40364e+007
,-8.01215e+006,-2.40364e+007,-2.40364e+007,-8.01215e+006));
#100007=SFARESPONSEFIBERSECTION($,1,(1.72935e+007,5.76452e+006,5.76452e+006,1.72935e+007
,-5.76452e+006,-1.72935e+007,-1.72935e+007,-5.76452e+006));
#100008=SFARESPONSEFIBERSECTION($,1,(1.11337e+007,3.71124e+006,3.71124e+006,1.11337e+007
,-3.71124e+006,-1.11337e+007,-1.11337e+007,-3.71124e+006));
#100009=SFARESPONSEFIBERSECTION($,1,(6.62277e+006,2.20759e+006,2.20759e+006,6.62277e+006
,-2.20759e+006,-6.62277e+006,-6.62277e+006,-2.20759e+006));
#100010=SFARESPONSEFIBERSECTION($,1,(4.55672e+006,1.51891e+006,1.51891e+006,4.55672e+006
,-1.51891e+006,-4.55672e+006,-4.55672e+006,-1.51891e+006));
ENDSEC;
]]></ElementalResultGetResponse>
```

Knotenbezogene Berechnungsergebnisse sind einfach und selbsterklärend (Zeile 2 und 3 in obigem Textblock). Die Entities *SfaResponseBeam* und *SfaResponseFiberSection* sind im Datenmodell (Schema in Anhang B) ersichtlich.

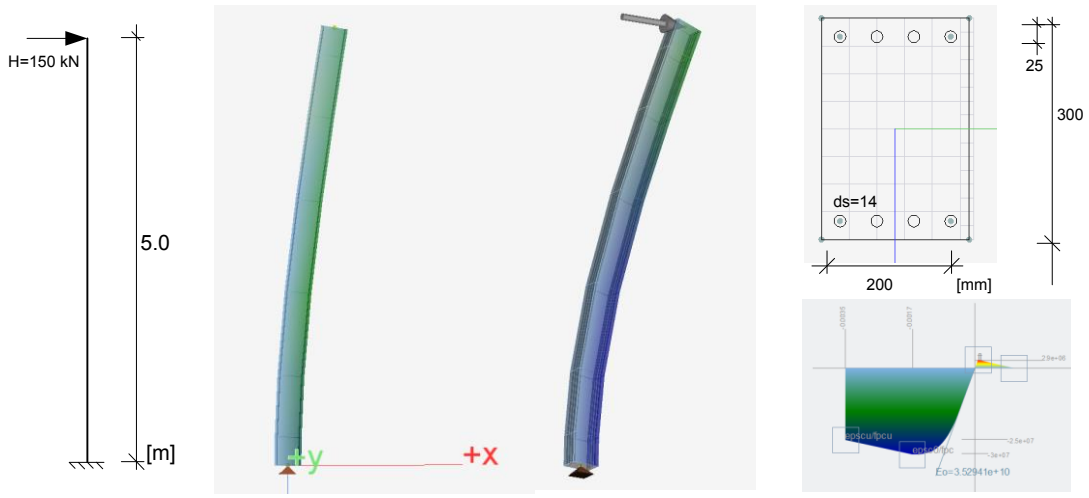


Abbildung 48: System, Spannungen, Verformung, Querschnitt und Material des Anwendungsbeispiels

Durch Erhöhen der Last am oberen Knoten entsteht die Last-Verschiebungskurve in Abbildung 47. Der Aufbau eines Datenpaketes zum Ändern der Last ergibt sich durch den Name des Datenpakets (ModelChange), und die geänderte Knotenlast im STEP-Format als Inhalt:

```
<ModelChange><![CDATA[
#24=SFASTRUCTURALLOADSINGLEFORCE($,#21,$,$,0,0,5500,0,0,0);
]]></ModelChange>
```

Danach folgt ein Datenpaket zur Anforderung einer Simulation:

```
<AnalysisRevert analysisId="3"/>
<AnalysisRun analysisId="3"/>
```

5.2 Grafisches Anwendungsbeispiel

Ein fiktives Rahmentragwerk soll als Anwendungsbeispiel für die clientseitige prototypische Modellierungsanwendung (Abschnitt 4.2) dienen. Das statische System ist ein dreidimensionales Rahmentragwerk wie in Abbildung 49 dargestellt. Materialgesetze für Stahl und Beton entsprechen dem vorigen Beispiel und dem Beispiel zur Validierung (Abbildung 46).

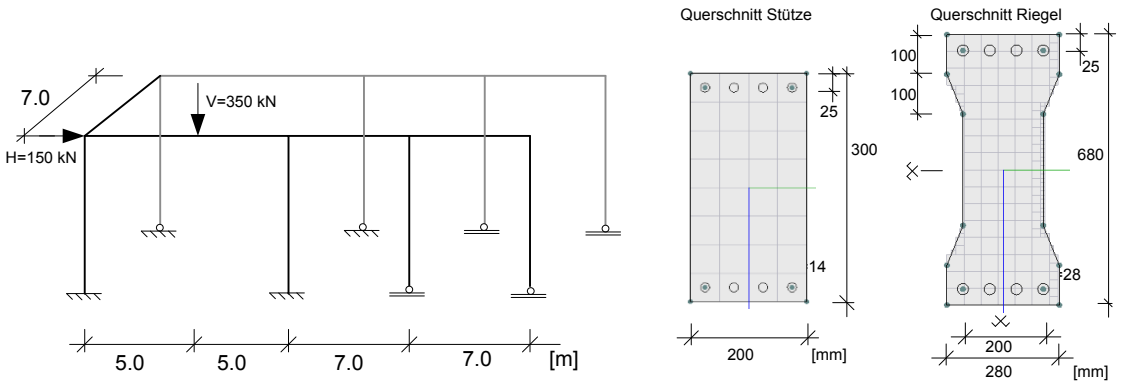


Abbildung 49: Abmessungen und Querschnitte des Beispielsystems

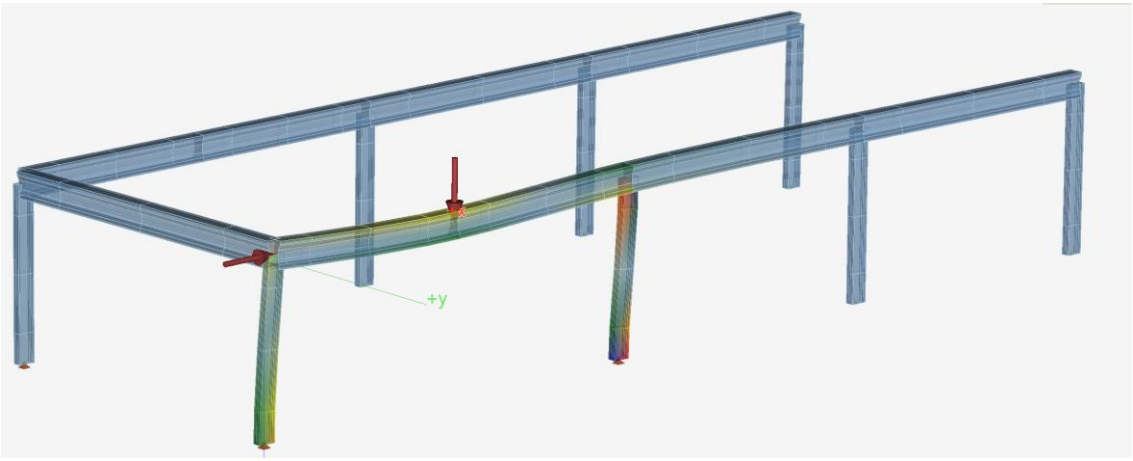


Abbildung 50: Spannungen und Verformungen des Beispielsystems

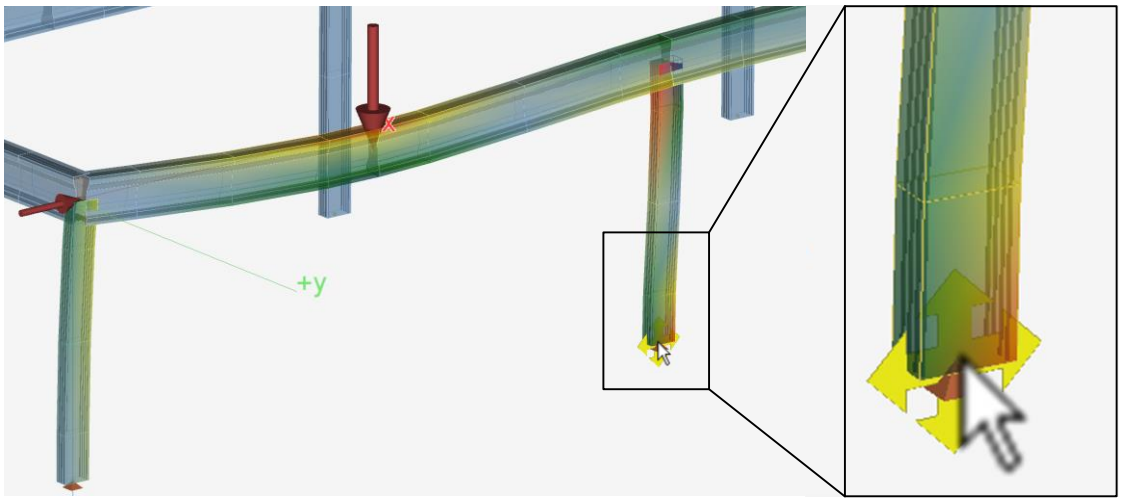


Abbildung 51: Verschieben eines Knotens

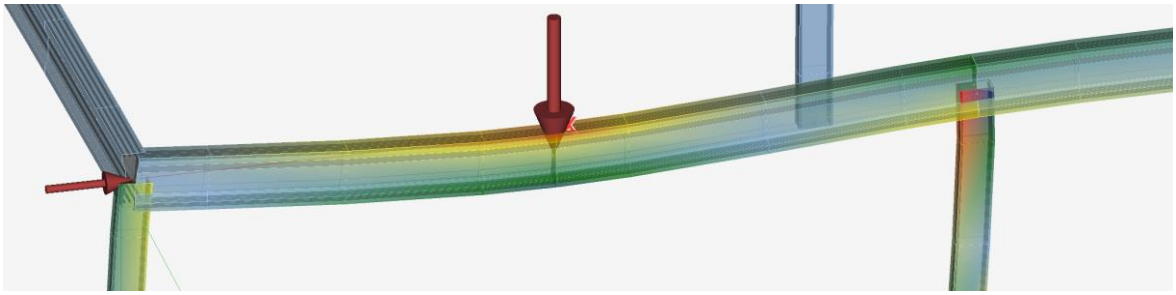


Abbildung 52: Verformung des Riegels

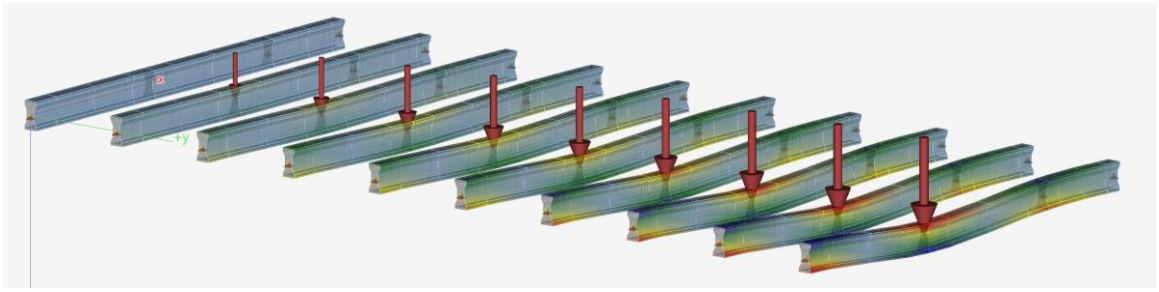


Abbildung 53: Riegel unter verschiedenen Belastungsstufen

Berechnete Daten zum Anwendungsbeispiel:

Anzahl Fasern Stützenquerschnitt	45 (8 Stützen)
Anzahl Fasern Riegelquerschnitt	175 (7 Riegel)
Anzahl Integrationspunkte in Stablängsrichtung	5
Anzahl Material-Objekte (nichtlinear, jeweils eigener Dehnungszustand und Historie)	8*5*45 + 7*5*175 = 7925 Objekte
Anzahl Knoten	18 (108 Freiheitsgrade)
Anzahl festgehaltener Freiheitsgrade	19
Berechnung des Systems serverseitig	0.016 sec
Zeit zur Übertragung des Systems, Berechnung, Abfrage der Ergebnisdaten und Aktualisierung der Visualisierung	0.058 sec

Die Berechnungen wurden serverseitig auf einem Intel i7 3770K @ 4.7 GHz Prozessor und 8 GB Hauptspeicher durchgeführt. Der Server war mit dem Client-Computer über ein lokales 1 Gbit Netzwerk verbunden.

In diesem Beispiel sind Zeitspannen zur Lösung des Gleichungssystems serverseitig nicht entscheidend, da die Anzahl der Freiheitsgrade sehr gering ist. Bei Re-Analyse nach Verschieben eines Knotens sind die Rechenzeiten sogar unterhalb von 10 Millisekunden. Eine genauere Analyse der Antwortzeiten der Simulation findet sich in Abschnitt 4.5.

Mit diesem Beispiel ist somit eine interaktive Entwurfsfindung möglich. Knoten können verschoben werden um die optimale Position zu finden. Dies ist auf eine sehr intuitive Weise möglich, denn die Spannungen werden mit nur sehr geringer Verzögerung während des Verschiebens visualisiert. Wird das Verschieben-Kommando beendet, so wird zusätzlich die Verformung des Systems visualisiert, wie in Abbildung 52 dargestellt.

6 Interaktiver Tragwerksentwurf in einer VR-Umgebung

6.1 Randbedingungen

Eine VR-Umgebung (Virtuelle Realität[72]) besteht aus mindestens einer, meistens aber mehreren stereoskopischen[73] Projektionsflächen, um Objekte dreidimensional zu visualisieren. Im Gegensatz zu herkömmlicher 3D-Grafik, bei der dreidimensionale Objekte auf einem zweidimensionalen Bildschirm dargestellt werden, „schweben“ die Objekte in einer VR-Umgebung scheinbar mitten im Raum. Eine VR-Umgebung, die aus mehreren Projektionsflächen besteht (Abbildung 54) wird als CAVE bezeichnet. Dabei kann sich eine Person inmitten der 3D-Projektion frei bewegen und durch Eingabegeräte wie einen 3D-Tracker (eine Art räumlicher Joystick) mit dem Modell interagieren.

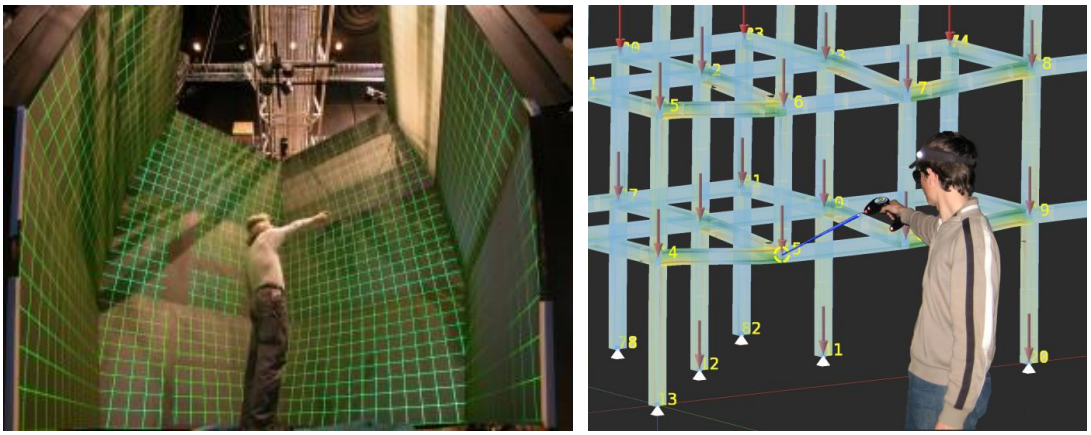


Abbildung 54: StarCAVE der UCSD (Quelle links: Jürgen Schulze, UCSD), interaktive FE-Simulationsanwendung

Die meisten Forschungsprojekte, die sich mit VR befassen, nutzen die Technik zur Visualisierung von Daten oder Modellen. Nutzerinteraktion ist meist auf Navigation im Modell beschränkt. Es gibt jedoch auch einige Projekte mit Interaktion und numerischer Simulation. Ein Projekt mit numerischer Strömungssimulation von Wasserkraft-Turbinen wird in [Ruprecht et al. 1999] beschrieben. Visualisierung und Interaktion erfolgt in einer VR-Umgebung. Die numerische Strömungssimulation wird bei Modelländerungen gestartet und die Visualisierung aktualisiert. Durch Veränderung von diversen Parametern des Simulationsmodells kann intuitiv der Strömungsverlauf optimiert und der Wirkungsgrad erhöht werden.

In [Rüppel, Schatz 2010] wird eine VR-Umgebung als Trainingsumgebung zur Brandbekämpfung mit Echtzeit-Brandsimulation beschrieben. Damit können in einer sichereren Umgebung verschiedene Brandszenarien simuliert und Brandbekämpfung geübt werden.

---

[72] „Virtuell“ bedeutet in diesem Kontext „so gut wie“. Virtuelle Realität ist also eine möglichst überzeugende Abbildung der Realität durch Effekte der Illusion, wie beispielsweise die Illusion eines räumlichen Bildes durch Stereoskopie.

[73] Jedes Auge erhält ein eigenes Bild aus einer eigenen Perspektive, dadurch entsteht der Eindruck eines räumlichen Objektes. Die Bilder werden beide auf die gleiche Leinwand projiziert, entweder kurz nacheinander oder unterschiedlich polarisiert. Eine Brille übernimmt dann durch Shutter bzw. Polfilter die Trennung für das jeweilige Auge. Es gibt auch unterschiedliche Möglichkeiten der Bildtrennung ohne Brille, dies wird als „autostereoskopisch“ bezeichnet.

In der vorliegenden Arbeit wurde auch eine interaktive VR-Simulation realisiert, mit dem interaktiven Tragwerksentwurf mit Finite-Elemente-Analyse als Zielrichtung (Abbildung 54 rechts). Durch direkte Interaktion mit einem virtuellen Tragwerk soll so ermöglicht werden, intuitiv eine gute Lösung einer tragwerksplanerischen Aufgabe zu erzielen.

Im Rahmen eines DAAD-geförderten Forschungsaufenthaltes an der UC San Diego ergab sich die Gelegenheit, eine der weltweit größten und modernsten CAVEs (*StarCAVE*, Abbildung 54 links) zu nutzen und die genannten Konzepte darin zu realisieren.

6.2 Komponenten der VR-Simulationsanwendung

Die CAVE-Anwendung mit Modellierung, Visualisierung und Nutzerinteraktion wurde unter der Bezeichnung *DirectFEA* entwickelt. Zur Finite-Elemente-Analyse wurde OpenSees[74] [McKenna 1997], [Haukaas, Kiureghian 2007] verwendet.

Wie hier in der CAVE-Anwendung realisiert, kann der OpenSees-Quellcode direkt in eine Anwendung integriert werden, in diesem Fall als statische Bibliothek verlinkt (Abbildung 55). Somit ist ein direkter Zugriff auf Objekt-Ebene möglich, der (vergleichsweise langsame) Umweg über die Eingabedatei und Ausgabedateien für Spannungen und Verschiebungen kann so vermieden werden. Das hat den Vorteil, dass nicht bei jeder Änderung im Modell eine vollständige Eingabedatei geschrieben werden muss, auch wenn die Änderung nur einen einzigen Knoten betrifft. Das Schreiben und Parsen von Ein- und Ausgabedateien in jedem Frame wäre mit den vielen Festplattenzugriffen zu langsam für eine interaktive Simulation.

Eine Übersicht über die Komponenten der Anwendung gibt Abbildung 55.

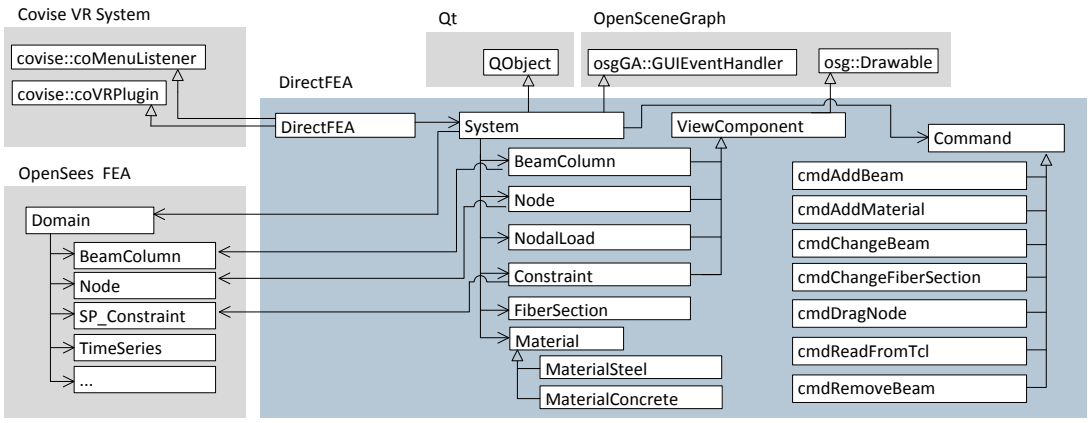


Abbildung 55: UML-Diagramm der Anwendung „DirectFEA“

Als zentrale Schnittstelle zur VR-Umgebung dient die Klasse *DirectFEA*, dort werden Events wie Nutzereingaben aus der COVISE-Umgebung[75] verarbeitet und an die Klasse *System* weitergeleitet. Dort wiederum werden Referenzen auf die Komponenten des Modells (Knoten, Stabelemente, Lasten, Lagerungen) verwaltet, wie auch die Ausführung von Kommandos. Die Komponenten wie Stabelemente halten Referenzen zu den entsprechenden OpenSees-Objekten zur numerischen Simulation und auch zu den Objekten des Szenegraphen. So können nach erfolgreicher FE-

[74] http://opensees.berkeley.edu

[75] Collaborative Visualization and Simulation Environment. Framework zum Betrieb des CAVE-Clusters, entwickelt an der Universität Stuttgart

Berechnung die Visualisierungsdaten zu Verschiebungen und Spannungen pro Element aktualisiert werden.

6.3 Interaktion mit dem virtuellen Modell

Die Hardware der VR-Umgebung *StarCave* beinhaltet einen 3D-Tracker, dessen Position und Rotation ständig überwacht wird. Im Falle einer Bewegung oder einer Betätigung eines der Bedienelemente kann das entsprechende Event über eine Schnittstelle des COVISE-Systems (Abbildung 55) empfangen und verarbeitet werden. Im Gegensatz zu einer normalen Desktop-Anwendung müssen die Koordinaten des Zeigegerätes nicht von 2D-Bildschirmkoordinaten in 3D-Modellkoordinaten umgerechnet werden (wie es in der Anwendung aus Kapitel 3 der Fall ist). Aus der Position und Rotation des Trackers kann direkt eine Geradengleichung aufgestellt werden[76], die einem imaginären Strahl entspricht (Abbildung 54 rechts, Abbildung 56). Zur Interaktion mit Modellkomponenten wie Knoten oder Stabelementen muss nun zunächst berechnet werden, ob ein bestimmter Mindestabstand unterschritten ist. Dazu wird eine Abstandsberechnung „Punkt-Gerade“ für alle Knoten des Modells vorgenommen. Wenn ein Knoten nah genug am Strahl liegt, wird dieser als „gefangen“ gemerkt und visualisiert (gelber Kreis in Abbildung 54 rechts). Kommt als nächstes ein Klick-Element der „Verschieben“-Taste des Trackers, wird die Differenz der Position berechnet, und der Knoten entsprechend verschoben. Dadurch folgt der Knoten der Bewegung des Trackers, bis erneut die „Verschieben“-Taste gedrückt wird. Das Analysemodell wird bei jeder Bewegung aktualisiert, die neuen Berechnungsergebnisse (Spannungen) werden visualisiert.

Liegt kein Knoten nah am Strahl, wird eine Abstandsberechnung „Geradenabschnitt-Gerade“ zwischen allen Stabelementen und dem Tracker-Strahl durchgeführt. Wird ein Stabelement „getroffen“, wird dieses als „gefangen“ visualisiert (gelbe Umrandung). Durch Betätigen der „Löschen“-Taste wird das gefangene Element aus dem Modell entfernt und das Analysemodell mitsamt Visualisierung entsprechend aktualisiert, wie in Abbildung 56 dargestellt.

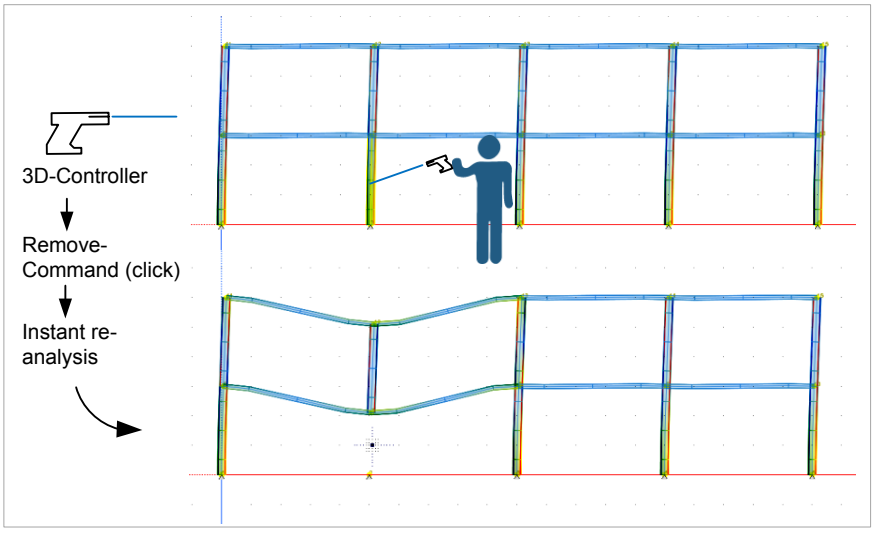


Abbildung 56: Interaktion mit dem Modell (Auswählen + Entfernen) in der VR

Die Anwendung umfasst folgende Funktionen zur Interaktion:

[76] Im Gegensatz zu einem 3D-Modell auf einem Bildschirm sind die Koordinaten des Eingabegerätes direkt im Modell-Koordinatensystem verfügbar, und müssen somit nicht umgerechnet werden

- Laden eines Rahmentragwerks mit Lagerungen und Lasten.
- Hinzufügen von Knoten und Elementen.
- Löschen von einzelnen Knoten oder Elementen.
- Verschieben von Knoten.
- Berechnung und Visualisierung eines Erdbeben-Beschleunigungs-Zeitverlaufs.

Nach allen Änderungen im Tragwerksmodell durch den 3D-Tracker wird eine sofortige Berechnung und Visualisierung der Auswirkungen vorgenommen. Bei den getesteten Modellen (Stabwerksmodelle mit ca. 600 Freiheitsgraden) waren die Antwortzeiten der Simulation ausreichend klein, um Berechnung und Visualisierung per-Frame zur realisieren.

6.4 Diskussion der VR-Anwendung

Die offensichtlichen Vorteile der Anwendung *DirectFEA* sind:

- Die räumliche Projektion des Tragwerksmodells erlaubt eine gute und intuitive Beurteilung der Simulationsergebnisse wie Spannungen und Verformungen.
- Die Interaktion mit räumlich sichtbaren Elementen ist sehr intuitiv, verglichen mit einem herkömmlichen Arbeitsplatz, bei dem immer eine mentale „Übersetzung“ von Bewegungen eines 2D-Eingabegerätes zur 2D-Projektion eines Bildschirms und außerdem die Interpretation der 2D-Projektion als 3D-Modell erforderlich ist.

Bestehende Defizite:

- Räumliche Eingabegeräte der Cave (3D-Tracker) sind zu unpräzise, um Knotenpositionen, Querschnittswerte oder andere Parameter genügend genau zu definieren. Als Ergänzung sind zumindest herkömmliche Eingabegeräte wie Maus und Tastatur notwendig, wodurch der Vorteil der räumlichen Interaktion wieder verloren geht.
- Die gezeigte Anwendung ist eine Insellösung, es besteht keine Interoperabilität zu anderen Anwendungen.
- Die komplette Anwendung ist als COVISE-Plugin realisiert, inklusive der FEA-Software. Die Anwendung wird beim Start auf alle 17 Knoten des CAVE-Clusters verteilt. Jeder Knoten ist für das Rendern einer Projektionsfläche (je 2 Projektoren) zuständig. Normalerweise ist das Rendern der jeweiligen Perspektive der aufwendigste Teil einer Cave-Anwendung, im Fall der interaktiven FE-Simulation ist das jedoch die numerische Berechnung. Die Perspektive ist auf allen Knoten unterschiedlich, die FE-Simulation jedoch identisch, deswegen wäre es eigentlich sinnvoll, von einem dedizierten Rechenserver aus die Simulationsergebnisse auf alle anderen Knoten zu verteilen. Für die realisierten Beispiel-Modelle (Stabwerke mit <1000 Freiheitsgraden) war diese verbesserte Implementierung jedoch nicht notwendig, und konnte auch aus zeitlichen Gründen im Rahmen dieser Arbeit nicht mehr realisiert werden.

Insgesamt lässt sich sagen, dass die Vorteile der räumlichen Visualisierung und der direkteren Interaktion mit dem Modell momentan nicht den technischen Aufwand zum Betrieb einer CAVE rechtfertigen, jedenfalls nicht für die praktische Anwendung. Das Problem der ungenauen Positionierung von 3D-Controllern erschwert eine effiziente Modellierung.

Dennoch macht es Spaß und bringt zusätzlichen, neuen Erkenntnisgewinn, mit einem virtuellen Modell zu interagieren. Betrachtet man räumlich, wie sich ein Tragwerk in einem Erdbeben, also einem aufgezeichneten Beschleunigungs-Zeitverlauf verhält, und wie sich Änderungen etwa durch

zusätzliche Elemente auswirken, ergibt sich ein guter Lerneffekt. Der Lerneffekt ist erheblich höher, wenn man mitten im virtuellen, interaktiven Tragwerksmodell steht, verglichen mit einem herkömmlichen Bildschirm.

7 Diskussion der Ergebnisse

SFA als Datenmodell zur serverbasierten Simulation

Die gezeigte Umsetzung und die Anwendungsbeispiele belegen, dass das hier entwickelte Protokoll auf Basis von XML und STEP zur Kommunikation mit einem Rechenserver geeignet ist. Verifiziert wurde dies für Stabwerke kleiner und mittlerer Größe. Anwendbar ist das Konzept prinzipiell auch auf große Modelle und auch auf andere Tragwerksarten bzw. Elementarten.

Die wesentlichen Möglichkeiten und Eigenschaften des vorliegenden Netzwerkprotokolls zur numerischen Simulation lassen sich folgendermaßen benennen:

- Inkrementelle Modelländerungen (Hinzufügen/Ändern/Löschen) können auf alle Bestandteile des Modells angewendet werden, ohne Neuaufstellung des gesamten Analysemodells (persistentes Analysemodell im Speicher des Simulations-Servers).
- Einheitliche, bidirektionale Schnittstelle für Modelldaten, Kommandos wie „Berechnung starten“ und Abfrage von Berechnungsergebnissen.
- Durch die Netzwerkfähigkeit kann ein Rechenserver auf dem lokalen PC, auf einer dedizierten Maschine im Unternehmensnetzwerk, oder über das Internet angebunden werden.
- Echte Plattformunabhängigkeit. Client-Anwendungen können in einer beliebigen Sprache implementiert sein. Insbesondere sind Anwendungen im Webbrowser möglich. Dabei kann die Visualisierung und Nutzerinteraktion nicht nur auf herkömmliche Weise in einer Desktop-Anwendung implementiert sein. Auch neue Internet-Technologien wie HTML5 eignen sich als technische Basis für Anwendungen zum Tragwerksentwurf, die numerische Simulation ist dann über die IRAI-Schnittstelle auf einem speziell dafür ausgelegten Rechenserver gekapselt.

Im Kontext der interaktiven Entwurfsraumsuche ist der erste der genannten Punkte der Wichtigste, während die anderen eher softwaretechnische Vorteile (gegenüber der herkömmlichen, dateibasierten Ein-und Ausgabe) darstellen.

Da für die Zielsetzung der interaktiven Entwurfsraumsuche in der Tragwerksplanung möglichst kurze Antwortzeiten essentiell sind, spielen jedoch auch die rein softwaretechnischen Aspekte eine wichtige Rolle.

Unter der Voraussetzung einer serverbasierten Simulation, bei einer größeren Anzahl von Revisions-Zyklen (etwa zur Suche eines Optimums), ist ein inkrementelles Kommunikationsprotokoll unbedingt sinnvoll. Denn ohne ein solches muss für jede Revision, selbst wenn sich nur eine einzige Koordinate eines Knotens geändert hat, das gesamte Modell zum Server transferiert und dort neu aufgestellt werden. Mit dem hier vorgestellten Protokoll kann eine beliebige Änderung im Modell inkrementell auf ein serverseitiges, persistentes Analysemodell angewendet werden.

Die Funktionsweise des Gesamtkonzeptes konnte durch eine prototypische Implementierung eines Rechenservers und einer grafischen Anwendung zur Interaktion mit einem Tragwerksmodell gezeigt werden.

Eine Absicherung der Kommunikation durch Verschlüsselung oder Techniken zur serverseitigen Parallelisierung der Gleichungslöser wurden hier nicht angewendet. Diese Konzepte sind jedoch auch nachträglich integrierbar.

Vermutlich werden in Zukunft rechenintensive Anwendungen wie die Finite-Elemente-Analyse als Dienstleistung über das Internet angeboten und genutzt werden. Möglich wäre auch ein reiner Gleichungslöser als Dienstleistung, wobei dann die Komplexität der FEA-Implementierung bei der Client-Anwendung verbleibt. Das hier entwickelte Konzept kapselt diese Komplexität in einer Serveranwendung, und bietet eine einfach zu implementierende Schnittstelle durch das STEP-Datenformat.

Interaktive Simulation und mathematische Optimierung

Interaktive Simulation mit Nutzerinteraktion und schneller Re-Analyse und Visualisierung hat gegenüber einem Optimierungsmodell mit entsprechenden Optimierungsalgorithmen vor allem die Vorteile der größeren Flexibilität und des geringeren Aufwands zum Erstellen des Modells. Der Aufwand für die Aufstellung eines Optimierungsmodells ist erheblich, der Aufwand für die Entwurfsraumsuche durch Nutzerinteraktion besteht lediglich auf der Seite der Softwareentwicklung, für den Anwender besteht kein zusätzlicher Aufwand im Vergleich zu herkömmlicher Modellierung.

Ein weiterer Vorteil der interaktiven Entwurfsraumsuche ist der Lerneffekt, der durch die intuitive Rückkopplung der Visualisierung entsteht. Denn der Zusammenhang zwischen Entwurfsparametern und Auswirkungen auf das Tragwerk wird so direkt begreifbar.

SFA als Archivierungsformat

STEP ist ein standardisiertes Datenformat, das maschinenlesbar und menschenlesbar ist ohne spezielle Software (die möglicherweise in 20 Jahren nicht mehr lauffähig ist). Es ist daher ideal als Archivierungsformat für Ergebnisse numerischer Simulationen.

Die Modelldaten und Berechnungsergebnisse können im STEP-Format zusammen in einer Datei gespeichert werden. Referenzen zwischen Modell- und Ausgabe-Entities bleiben so erhalten.

Gegenüber XML als ebenfalls standardisiertes und sowohl menschen- als auch maschinenlesbares Format hat STEP den Vorteil, dass die Daten für Menschen sehr viel intuitiver verständlich sind. Hinzu kommt, dass das Datenvolumen pro Informationsmenge wesentlich geringer ist.

Produktmodellierung

Wie bereits zuvor ausgeführt, ist der Einsatz einer interaktiven numerischen Simulation zur Entwurfsfindung von Tragwerken möglichst früh im Planungsprozess sinnvoll.

Dies ist nur mit einem reibungslosen Datenaustausch mit Anwendungen anderer Planungsdisziplinen möglich. Ein offener Standard wie IFC bietet die Möglichkeit einer solchen Interoperabilität.

Die in dieser Arbeit konzipierte und umgesetzte IFC-Implementierung *IfcPlusPlus* bildet mit Klassenmodell- und Parser-Generator auch die technische Basis für das Datenmodell des Kommunikationsprotokolls zwischen Client und Rechenserver. Somit hält sich der Implementierungsaufwand in Grenzen, bzw. kann doppelt genutzt werden. Als ein Baustein zur Verbesserung der Interoperabilität und veröffentlicht als Open Source[77], kann *IfcPlusPlus* auch in

[77] https://code.google.com/p/ifcplusplus/

beliebigen anderen Anwendungen verwendet werden. In mehreren Projekten in Industrie und Forschung kommt die Software bereits zum Einsatz.

Die in Abschnitt 5 gezeigte prototypische Implementierung unterliegt der Beschränkung, dass ein importiertes IFC-Modell bereits ein Strukturmodell enthalten muss. Ein reguläres, architektonisches IFC-Modell ist dafür nicht ausreichend und muss gesondert (manuell) erstellt werden. Der Vorgang des manuellen Aufstellens eines abgeleiteten Modells wird in [Eastman et al. 2008, Seite 19] explizit als „nicht BIM“ definiert: „Models that allow changes to dimensions in one view that are not automatically reflected in other views“. Das trifft offensichtlich zu auf ein redundantes, dimensionsreduziertes Strukturmodell.

Momentan ist die Verwendung eines dimensionsreduzierten Strukturmodells eine pragmatische Lösung, da in der Praxis nach wie vor üblicherweise mit dimensionsreduzierten Tragwerksmodellen gearbeitet wird. In der numerischen Simulation gibt es jedoch einen Trend, die Übereinstimmung der Simulationsmodelle mit der Wirklichkeit immer mehr zu erhöhen. Dies bezieht sich auf (nichtlineare) Materialmodelle, probabilistische Lastmodelle, aber auch auf die geometrische Modellierung. Dieser Trend in der geometrischen Modellierung könnte in eine vollständig volumetrische Modellierung von Tragwerken resultieren. Denn Flächen oder Linien ohne räumliche Ausdehnung (dimensionsreduzierte Strukturmodelle, vgl. Abschnitt 2.7.8) existieren in der Realität nicht. Ein Vorteil wäre dann, dass das Aufstellen eines Analysemodells aus dem architektonischen Modell weniger aufwendig ist, da praktisch keine Abstraktionen mehr notwendig sind. Bei einem volumetrischen Gesamtsystem mit realitätsnaher Modellierung der Geometrie und nichtlinearem Materialverhalten sind keine Annahmen zum statischen System mehr notwendig, also beispielsweise, ob eine Kopplung gelenkig ist oder eingespannt. Denn bei einer hinreichend realitätsnahen Modellierung eines Tragwerkes bildet sich der Kraftfluss selbständig aus, ebenso wie im realen Bauwerk.

Der Ingenieur wird dadurch keineswegs überflüssig. Die Aufgaben verschieben sich nur von der Routinearbeit des Aufstellens eines Analysemodells hin zu Entwurf und Modellierung von Tragwerken.

Literaturverzeichnis:

[Anderheggen, Pedro 2005] Anderheggen, E.; Pedro, C.: E-Teaching and E-Learning Structural Design. In *Proceedings of International Conference on Computing in Civil Engineering* (2005).

[Bazjanac 2004] Bazjanac, V. (2004): Building energy performance simulation as part of interoperable software environments. In Building and Environment 39 (8), Pages 879–883.

[Blachowicz, Wieja 2006] Blachowicz, T.; Wieja, M.: Remote-control and clustering of physical computations using the XML-RPC protocol and the open-Mosix system, 2006, arXiv:cs/0603111 Abruf 22.09.2012

[Blanchette, Summerfield 2007] Blanchette, J.; Summerfield, M.: C++-GUI-Programmierung mit Qt 4. Die offizielle Einführung. München, Boston [u.a.]: Addison-Wesley Verlag, 2007.

[Bletzinger et al. 2005] Bletzinger, K.-U.; Wüchner, R.; Daoud, F.; Camprubí, N.: Computational methods for form finding and optimization of shells and membranes. In *Computer Methods in Applied Mechanics and Engineering* 194 (2005), Pages 3438–3452.

[buildingSMART] buildingSMART International Ltd., http://www.buildingsmart-tech.org, Abruf: 02.10.2012

[Carve] Sargeant, T.: Carve – An efficient and robust library for boolean operations , http://www.carve-csg.com, Abruf: 16.10.2012.

[Campbell 2007] Campbell, D. A. Building information modeling: the Web3D application for AEC. In *Proceedings of the twelfth international conference on 3D web technology* (2007). ACM, New York. Pages 173-176.

[Chen, Lin 2008] Chen, H.; Lin, Y.: Web-FEM: An internet-based finite-element analysis framework with 3D graphics and parallel computing environment. In *Advances in Engineering Software* 39 (1) (2008), Pages 55–68.

[Chen et al. 2005] Chen, P.; Cui, L.; Wan, C.; Yang, Q.; Ting, S.; Tiong, R.: Implementation of IFC-based web server for collaborative building design between architects and structural engineers. In Automation in Construction 14 (2005), Pages 115–128.

[ComputerWeekly]: Single Google search uses 1,000 servers. Available online at http://www.computerweekly.com/news/2240088495/Single-Google-search-uses-1000-servers, Abruf: 6.07.2012.

[Cottrell et al. 2009] Cottrell, J. Austin; Hughes, Thomas J. R.; Bazilevs, Yuri: Isogeometric Analysis: Toward Integration of CAD and FEA. 1st: Wiley Publishing. 2009

[Daoud 2005] Daoud, F. Formoptimierung von Freiformschalen – Mathematische Algorithmen und Filtertechniken, Dissertation (2005), Lehrstuhl für Statik der Technischen Universität München

[DIN EN ISO/IEC 1995] Deutsches Institut für Normung e.V. (Hrsg.): DIN EN ISO/IEC 7498-1:Kommunikation offener Systeme: Basis-Referenzmodell: Basismodell (ISO/IEC 7498-1:1994). Berlin, Beuth-Verlag, 1995

[Eastman, Augenbroe 1998] Eastman, C.; Augenbroe, F. (1998): Product Modeling Strategies for Today and the Future (1998), Construction Informatics Digital Library http://itc.scix.net/paper w78-1998-20.content

[Eastman et al. 2008] Eastman, C., Teicholz, P., Sacks, R., Liston, K.: BIM Handbook. A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers, and Contractors: John Wiley & Sons, Inc., 2008

[Eastman et al. 2011] Eastman, C.; Teicholz, P.; Sacks, R.; Liston, K. (2011): BIM Handbook. A guide to building information modeling for owners, managers, designers, engineers, and contractors. 2nd edition, Wiley John + Sons, 2011

[Eissen, Stein 2006] Eissen, S. M.; Stein, B. (2006): Realization of web-based simulation services. In *Computers in Industry*, 57 (3) (2006), pp. 261–271

[Expressik Generator] http://mint.cs.man.ac.uk/expressik/express2cpp/, Abruf:10.10.2012.

[Fenves et al. 2000] Fenves, S. J.; Rivard, H.; Gomez, N.: SEED-Config: a tool for conceptual structural design in a collaborative building design environment.In *Artificial Intelligence in Engineering*, Volume 14, issue 3 (2000), p. 233-247. ISSN: 0954-1810

[Fliegner 2003] Fliegner, C.: Umsetzung der IFC mit Java und XML am Beispiel der Tragwerksplanung. Bauhaus-Universität Weimar, Diplomarbeit, 2003

[Froese 2003] Froese, T.: Future Directions for IFC-based Interoperability. In *ITCON 8 (Special Issue IFC — Product models for the AEC arena)* (2003), Pages 231–246

[Gerold 2010] Gerold, F. Integrative structural design. In*Computing in Civil and Building Engineering, Proceedings of the International Conference*, 2010, Nottingham, UK, Nottingham University Press, Paper 287, p. 573, ISBN 978-1-907284-60-1

[Gielingh 1988] Gielingh, W.: General AEC reference model (GARM). An aid for the integration of application specific product definition models. Construction Informatics Digital Library, http://itc.scix.net/data/works/att/w78-1988-165.content.pdf, Abruf: 4.11.2012

[Göddecke et al. 2005] Göddecke, D.; Strzodka, R.; Turek, S.: Accelerating Double Precision FEM Simulations with GPUs. In: *Proc. ASIM* (2005)

[Gumm, Sommer2002] Gumm, H.; Sommer, M.: Einführung in die Informatik. 5. Auflage (2002), Oldenbourg Wissenschaftsverlag, ISBN 978-3486256352

[Gu, Chan 1995] Gu, P.; Chan, K.: Productmodelling using step. In *Computer-Aided Design* 27 (1995), Pages 163–179.

[Halfawy, Froese2002] Halfawy, M. R.; Froese, T.: Modeling and implementation of smart AEC objects: an IFC perspective. CIB w78 conference — Distributing Knowledge in Building, Aarhus School of Architecture, Denmark (2002), Pages 1–8

[Haukaas, Kiureghian 2007] Haukaas, T.; Der Kiureghian, A.: Methods and Object-Oriented Software for FE Reliability and Sensitivity Analysis with Application to a Bridge Structure. In *ASCE J. Comput. Civil Engrg.*, 21 (3) (2007), Pages 151–163

[ISO 10303 2004] ISO 10303-11:2004 Industrial automation systems and integration -- Product data representation and exchange -- Part 11: Description methods: The EXPRESS language reference manual

[Kicinger et al. 2005] Kicinger, R.; Arciszewski, T.; Jong, K.: Evolutionary computation and structural design: A survey of the state-of-the-art. In *Computers & Structures* 83 (2005), Pages 1943–1978.

[IFC4 Dokumentation 2012] IFC4 RC4 Documentation, buildingSMART International Ltd, http://www.buildingsmart-tech.org/ifc/IFC2x4/rc3/html/index.htm Abruf: 10.10.2012.

[Kost 2003] Kost, B., Optimierung mit Evolutionsstrategien, Verlag Harri Deutsch, Frankfurt am Main, 2003. ISBN 3-8171-1699-3

[Krafczyk et al. 2007] Krafczyk, M.; Tölke, J., Fahrig, T.: Ein Prototyp für verteilte, interaktiv-kooperative Simulationen zur Beschleunigung von Entwurfszyklen im Konstruktiven Ingenieurbau. In Uwe Rüppel (Ed.): Vernetzt-kooperative Planungsprozesse im Konstruktiven Ingenieurbau (2007): Springer Berlin Heidelberg, Pages 273–293. ISBN: 978-3-540-68104-5.

[Kuehne, Martz 2007] Kuehne, B; Martz, P. (2007): OpenSceneGraph Reference Manual v2.2: Skew Matrix Software and Blue Newt Software.

[Lagler 2012] Lagler, D.: How to make a PIXAR movie: Deformable Bodies, 2012, http://wwwcg.in.tum.de/fileadmin/user_upload/Lehrstuehle/Lehrstuhl_XV/Teaching/WS10_11/Proseminar_Seminar/Talks/Daniel_Lagler.pdf, Abruf: 25.11.2012

[Lebegue et al. 2007] Lebegue, E; Gual, J.; Arthaud, G.: IFC-BRIDGE V2 Data Model, 2007, http://www.buildingsmart-tech.org/future-extensions/ifc_extension_projects/current/ci2, Abruf: 10.10.2012

[Liang et al. 1999] Liang, J.; Shah, J.J; Souza, R.; Urban, S.D: Synthesis of consolidated data schema for engineering analysis from multiple STEP application protocols. In *Computer-Aided Design* 31 (1999), Pages 429–447.

[Liebich 2009] Liebich, T.: IFC 2x Edition 3 Model Implementation Guide, Version 2.0 (2009), buildingSMART International Ltd.

[Mackie 2000] Mackie, R.I: An object-oriented approach to fully interactive finite element software. In *Computers & Structures*, Volume 77, Issue 5, 21 (2000), Pages 461–474

[Mahinthakumar 2002] Mahinthakumar, G.: A Hybrid MPI-OPENMP Implementation of an implicit Finite Element Code on Parallel Architectures. In *The International Journal of High Performance Computing Applications* 16 (4) (2002), Pages 371–393.

[Martz 2007] Martz, P. (2007): OpenSceneGraph Quick Start Guide: Computer Graphics Systems Development Corporation.

[McKenna 1997] McKenna, F.: Object-Oriented Finite Element Programming: Framework for Analysis, Algorithms and Prallel Computing. Dissertation (1997). University of California, Berkeley.

[Mezger et al. 2007] Mezger, J.; Thomaszewski, B.; Pabst, S.; Straßer, W.: A Finite Element Method for Interactive Physically Based Shape Modelling with Quadratic Tetrahedra. 2007. http://nbn-resolving.de/urn:nbn:de:bsz:21-opus-29305 Abruf 23.11.2012

[Mora et al. 2004] Mora, R.; Bedard, C.; Rivard, H.: A Framework for Computer-Aided Conceptual Design of Building Structures. ICCCBE, (2004), Weimar, Bauhaus-Universität

[Mora et al. 2008] Mora, R.; Bédard, C.; Rivard, H.: A geometric modelling framework for conceptual structural design from early digital architectural models. In *Advanced Engineering Informatics* 22 (2), Pages 254–270. (2008)

[Mulder 1999] Mulder, J. Wijkj and Liere R.: A survey of computational steering environments. In *Future Generation Computer Systems (1999)* 15(1), 119-129.

[OCIT Developer Group 2011] OCIT Developer Group: OCIT-C Center to Center Transport Protokol. Open Communication Interface for Road Traffic Control Systems, 2011, http://www.ocit.org/specs_public/OCIT-C/Version_1/OCIT-C_Protokoll_V1_R1.pdf Abruf: 22.04.2012

[OpenSceneGraph 2012] OpenSceneGraph project website, http://www.openscenegraph.org, Abruf: 9.10.2012

[Peng 2004] Peng, J.: Building finite element analysis programs in distributed services environment. In *Computers & Structures* 82 (22) (2004), Pages 1813–1833.

[Qt 2012]Qt project website, http://qt-project.org, Abruf:9.10.2012.

[Rank 2007] Rank, E.: Quo vadis Computational Engineering. In *SOFiSTiK Seminar 2007*, München, Sofistik GmbH Oberschleißheim

[Rank et al. 2012] Rank, E.; Kollmannsberger, S.; Schillinger D.; Sorger, Ch; Zander, N.: Geometric models, numerical analysis and computational steering in structural engineering In: *Proceedings of the 14. International Conference on Computing in Civil and Building Engineering* (ICCCBE 12), Moscow, Russia, 2012.

[Richter 2009] Richter, T.: Konzepte für den Einsatz versionierter Objektmodelle im Bauwesen. Verlag der Bauhaus-Universität Weimar, 2009 (Schriftenreihe Informatik in Architektur und Bauwesen). ISBN: 978-3-86068-403-0

[Romberg et al. 2004] Romberg, R.; Niggl A.; van Treeck, C.; Rank, E.: Structural Analysis based on the Product Model Standard IFC. Proceedings ICCCBE Weimar (2004).

[Rönneblad 2003] Rönneblad, A.: Product Models for Concrete Structures. Licentiate Thesis (2003), Lulea University of Technology, Sweden

[Roos 1965] Roos, D.: AN INTEGRATED COMPUTER SYSTEM FOR ENGINEERING PROBLEM SOLVING. Proceedings of AFIPS, November/December 1965, Seiten 151-159, New York

[Ruprecht et al. 1999] Ruprecht, A., Eisinger, R., Göde, E., Rainer, D.: Virtual Numerical Test Bed for Intuitive Design of Hydro Turbine Components. In: *Hydropower into the Next Century*, Gmunden, Austria (1999)

[Rüppel, Schatz 2010] Rüppel, U.; Schatz, K.: BIM-Based Virtual Training Environment for Fire-Fighters. In: *Proceedings of the 13th International Conference on Computing in Civil and Building Engineering & 2008 International Conference on Information Technology in Construction*, ISBN 978-1-90784-60-1 Page 23, Nottingham, UK, 2010

[Sanguinetti et al. 2012] Sanguinetti, P.; Abdelmohsen, S.; Lee, JaeMin; Lee, JinKook; Sheward, H.; Eastman, C.: General system architecture for BIM: An integrated approach for design and analysis. In *Advanced Engineering Informatics* (2012).

[SAP2000 ModelAlive] CSI, Computers and Structures, Inc. on Youtube: http://www.youtube.com/watch?v=QyJCmZ0cziU, Abruf: 22.5.2012

[Spearpoint 2007] Spearpoint, M.: Transfer of Architectural Data from the IFC Building Product Model to a Fire Simulation Software Tool. In *Journal of Fire Protection Engineering* 17 (2007), Pages 271–292.

[Sudarsan et al. 2005] Sudarsan, R.; Fenves, S.J; Sriram, R.D; Wang, F.: A product information modeling framework for product lifecycle management. In *Computer-Aided Design* 37 (2005), Pages 1399–1411.

[Tauscher 2011] Tauscher, E.: Vom Bauwerkinformationsmodell zur Terminplanung. Ein Modell zur Generierung von Bauablaufplänen. Dissertation (2011), Bauhaus-Universität Weimar.

[Terdalkar 2006] Terdalkar, S.; Rencis, J.: Graphically driven interactive finite element stress reanalysis for machine elements in the early design stage. In *Finite Elements in Analysis and Design* 42 (2006), Pages 884–899.

[TNO]: Ifc Engine DLL, http://www.ifcbrowser.com/ifcenginedll.html, Abruf:10.10.2012.

[Tulke 2010] Tulke, J.: Kollaborative Terminplanung auf Basis von Bauwerksinformationsmodellen. Dissertation (2010), Bauhaus-Universität Weimar

[van Treeck 2004] van Treeck, C.: Gebäudemodell-basierte Simulation von Raumluftströmungen. PhD thesis, Lehrstuhl für Bauinformatik, Technische Universität München (2004)

[VDI 3633 1992] VDI 3633 VDI - Richtlinien (Hrsg. Verein Deutscher Ingenieure): Simulation von Logistik-, Materialfluß- und Produktionssystemen - Grundlagen. VDI 3633. Blatt 1. - Entwurf. Düsseldorf. Oktober 1992.

[Wall et al. 2008] Wall, W. A.; Frenzel, M. A.; Cyron, C.: Isogeometric structural shape optimization. In *Computer Methods in Applied Mechanics and Engineering* 197 (2008), Pages 2976–2988.

[Wan et al. 2004] Wan, C.; Chen, P.; Tiong, R.: Assessment of IFC for Structural Analysis Domain. In *Itcon* Vol. 9 (2004), Page. 75.

[Wassouf 2010] Wassouf, Z.: Die Mortar Methode für Finite Elemente hoher Ordnung. Technical University Munich, Munich. Chair for Computation in Engineering (2010). http://d-nb.info/1001246683/34 Abruf: 1.10.2012

[Weiss 2007] Weiss, A.: Computing in the clouds. In *87ikipedia*, vol. 11, no. 4, 2007, Pages 16–25.

[Yang, Xu 2004] Yang, Q.Z; Xu, X.: Design knowledge modeling and software implementation for building code compliance checking. In *Building and Environment* 39 (2004), Pages 689–698.

[Yang 2004] Yang, Z.: A web-based platform for computer simulation of seismic ground response. In Advances in Engineering Software 35 (5) (2004), Pages 249–259.

[Zhao, Liu 2008]Zhao, W.; Liu, J.K: OWL/SWRL representation methodology for EXPRESS-driven product information model. In *Computers in Industry* 59 (2008), Pages 580–589.

Verzeichnis der Algorithmen:

Abkürzungen:

AEC	Architecture, Engineering, Construction
API	Application Programming Interface, Schnittstelle zwischen Programmen bzw. Programm-Bibliotheken
Analysemodell	Im Kontext dieser Arbeit ist Analysemodell gleichbedeutend mit Simulationsmodell, vgl. „Simulation“
BIM	Building Information Modeling
Bounding Box	Umgebendes Rechteck bzw. Quader eines zwei-bzw. dreidimensionalen geometrischen Objekts
Brep	Boundary Representation: Geometriebeschreibung durch Oberflächen. Üblicherweise sind Oberflächen ebene Polygone, können aber auch gekrümmte Flächen sein, beschrieben etwa durch NURBS
CAD	Computer-Aided Design
CAVE	Cave Automatic Virtual Environment. VR-Umgebung mit mehreren räumlichen (stereoskopischen) Projektionsflächen
CFD	Computational Fluid Dynamics
Cloud Computing	Im Kontext der numerischen Simulation ist eine Cloud eine auf die jeweilige Simulationsaufgabe spezialisierte Server-Infrastruktur, auf die über definierte Protokolle zugegriffen werden kann
CSG	Constructive Solid Geometry. Beschreibung geometrischer Objekte durch Primitive wie Kugeln oder Zylinder. Komplexe Objekte werden mit Hilfe von booleschen Operationen und Primitiven als Operanden definiert.
CUDA	Compute Unified Device Architecture, von Nvidia entwickelt
DOF	Degree Of Freedom = Freiheitsgrad, also Bewegungsmöglichkeit. Ein Punkt im dreidimensionalen Raum hat 6 Freiheitsgrade, 3 Translationen und 3 Rotationen
DXF	Drawing Interchange File Format
Early binding	Beim sog. „early binding“ werden alle Referenzen eines Objektmodells vollständig beim Laden des Modells hergestellt. Im Gegensatz dazu werden beim „late binding“ Referenzen erst bei Bedarf hergestellt, was zur Laufzeit wesentlich weniger performant ist.
FEA	Finite-Elemente-Analyse. Methode zur Aufteilung komplexer statischer Systeme in eine Menge von Elementen, deren Geometrie und Spannungszustand durch eine glatte und integrierbare Funktion beschreibbar ist.
FLOPS	Floating Point Operation Per Second
FTP	„File Transfer Protocol“. Netzwerkprotokoll der Anwendungsschicht, zur

	Übertragung von Dateien
Frame-Rate	Anzahl von Aktualisierungszyklen eines Grafik-Kontextes. Mit Frame-Raten von > 30 Frames/sec werden Animationen als flüssig wahrgenommen
Freiheitsgrad	Dimension einer möglichen Bewegung eines Objektes, etwa eines Knotens. In einem dreidimensionalen kartesischen Koordinatensystem hat ein Knoten 3 Verschiebungs-Freiheitsgrade, und 3 Rotations-Freiheitsgrade
GLSL	OpenGL Shading Language, Sprache zur Programmierung verschiedener Stufen der OpenGL-Render-Pipeline
GPU	Graphics Processing Unit, Prozessor einer Grafikkarte
GPGPU	General Purpose GPU
GUI	Graphical User Interface: grafische Nutzeroberfläche
GUID	Globally Unique ID, global eindeutiger Identifikator
http	„Hypertext Transfer Protocol“. Netzwerkprotokoll der Anwendungsschicht, allgemein zur Übertragung von Daten, hauptsächlich verwendet zur Übermittlung von Webseiten an Browser
ID	Identifikator
IFC	Industry Foundation Classes
IGES	„Initial Graphics Exchange Specification“. Herstellerunabhängiges Format zum Austausch von CAD-Daten, hauptsächlich angewendet in der Automobilindustrie.
IP	Internet Protocol
LAN	Local Area Network
Map	Assoziatives Datenfeld. Über einen Schlüssel (z.B. Zeichenkette) kann auf die zugeordneten Werte (elementare oder komplexe Datentypen) zugegriffen werden.
Modell	Im Kontext dieser Arbeit ist mit Modell ein Computermodell gemeint. Vgl. „Simulation“
Mutex	Mutual Exclusion (Engl. Für gegenseitiger Ausschluss). Es gibt verschiedene Möglichkeiten, einen gleichzeitigen Zugriff auf einen Speicherbereich auszuschließen, z.B. durch ein globales Objekt das vor dem Zugriff auf eine gemeinsame Ressource angefragt wird, weitere Anfragen führen zu einem Pausieren des Threads.
OOP	Objektorientierte Programmierung. Dabei handelt es sich um ein Konzept zur Strukturierung von Software. Attribute und Methoden werden in einem Objekt zusammengefasst und nach außen gekapselt.
OpenCL	Open Cumputing Language, spezifiziert von der Khronos Group
OpenGL	Open Graphics Library: API zur Darstellung von 3D-Szenen in Echtzeit

OpenSees	„Open System for Earthquake Engineering Simulation“. Objektorientiertes Framework für nichtlineare und dynamische FE-Berechnungen. Der Quellcode wird als Open Source an der UC Berkeley entwickelt (http://opensees.berkeley.edu)
OpenMP	Open Multiprocessing, multi-Plattform-API zur Parallelisierung unter C/C++ und Fortran
OSG	OpenSceneGraph
Parser	Software zum Zerlegen und Interpretieren von (textbasierten) Eingabedaten
Prozess	Ausführungs-Kontext des Betriebssystems für eine Anwendung. Ein Prozess bekommt Ressourcen wie Speicher und CPU-Zeit zugeteilt und kann parallel zu anderen Prozessen ausgeführt werden.
SDAI	Standard Data Access Interface, nach ISO 1030-22
Semantik	“Bedeutung” von Wörtern oder Symbolen. Ein semantisches Informationsmodell unterscheidet sich von einem rein geometrischen Modell z.B. eines Quaders dadurch, dass der Quader eine Wand darstellt.
Simulation	„Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.“ [VDI-Richtlinie 3633] Vgl. „Modell“
Skalierung	Die Skalierung im Kontext einer parallelisierten Berechnung ist ein Maß für die Beschleunigung. Die Skalierung ist immer kleiner als die Anzahl der parallelen Recheneinheiten, da ein gewisser Overhead der Verwaltung, sowie für die Aufteilung der Prozesse (Fork) und die Zusammenführung der Ergebnisse (Join) entsteht
SOAP	Auf XML basierendes Netzwerkprotokoll zur Übertragung von Daten und Kommandos (Remote Procedure Calls). SOAP Protokolle werden meist auf TCP oder http aufgesetzt. Die Datenstruktur eines SOAP Protokolls kann in einem WSDL-Schema definiert werden.
STEP	„Standard for the Exchange of Product model data“, nach ISO 1030
Spezifikation	Technische Beschreibung
Standard	Spezifikation, die sich in einem Bereich allgemein durchgesetzt hat
Tcl	„Tool command language“, Open Source-Skriptsprache die Variablen, Kontrollstrukturen und eigene Kommandos unterstützt
TCP	„Transmission Control Protocol“, Netzwerkprotokoll der Transportschicht, verbindungsorientiert und zuverlässig, d. h. Datenpakete kommen vollständig und in der richtigen Reihenfolge an.
Thread	Ausführungsstrang innerhalb eines Prozesses. Mehrere Threads können parallel laufen, nutzen aber denselben Speicher („shared memory“).
Token	Trennzeichen bei (textbasierten) Eingabedaten, z.B. Komma oder Semikolon
Topologie	Im Falle eines Gebäudemodells bezeichnet die Topologie die räumliche Lage

	von geometrischen Elementen und deren Nachbarschaftsbeziehungen zu anderen Elementen
UML	Unified Modeling Language
VEF-Graph	Vertex-Edge-Face Graph
Voxel	Eine Voxel-Repräsentation (Volumetric Pixel) ist die räumliche Entsprechung einer Pixelgrafik wie JPEG oder PNG. Voxel sind in einem regelmäßigen Raumgitter angeordnet. Ein skalarer Wert pro Voxel kann beispielsweise die Dichte oder die Farbe angeben. Durch Octree-Datenstrukturen (Baum mit jeweils 8 Kindknoten) können Voxel lokal verfeinert werden, ohne das gesamte Gitter zu verfeinern, was in erheblich größerem Speicherbedarf resultieren würde mit $O(n^3)$
VR	Virtual Reality. „Virtual“ bedeutet „so gut wie“, virtuelle Realität ist also eine Nachbildung bzw. grafische Modellierung der Realität
VS	Visual Studio, Entwicklungsumgebung von Microsoft
WebGL	OpenGL-Schnittstelle für 3D-Grafik im Webbrowser ohne Installation von Plugins
WSDL	Web Service Description Language. Protokoll-Definitionssprache für Netzwerkdienste
XML	Extensible Markup Language

A Datenstruktur des *Incremental Revision Analysis Interface (IRAI)*

Tag-Name	**Attribute**		**Beschreibung**
Connect	customerId password		Aufbau einer Verbindung zum Analyse-Server
ConnectResponse	sessionId		Antwort des Servers auf Connect-Request. Bei erfolgreicher Verbindung wird eine Session-ID geliefert
Compression	type	“zLib”	Anweisung, alle folgenden Daten komprimiert zu senden und empfangen
CompressionResponse	success	“true”|”false”	Bestätigung
ModelAdd			Im CDATA-Abschnitt werden Modelldaten im SFA-STEP-Format übertragen. Elemente werden zum bestehenden Analysemodell hinzugefügt
ModelAddResponse			Bestätigung
ModelChange			Format analog ModelAdd, Elemente im Modell werden geändert
ModelChangeResponse			Bestätigung
ModelRemove			Format analog ModelAdd, Elemente im Modell werden gelöscht
ModelRemoveResponse			Bestätigung
ModelRemoveAll			Löschen des gesamten Modells
ModelRemoveAllResponse			Bestätigung
ModelGetAll			Abfrage des gesamten Modells im STEP-Format
ModelGetAllResponse			Antwort-Datenpaket
AnalysisRun	analysisId	Ganzzahl[78]	
AnalysisRunResponse	analysisTime		Bestätigung

[78] Mögliche Eingabeformate: { 1; 1-5; 1,5,10; }

AnalysisRevert	Wie AnalysisRun		Zurücksetzen des Verschiebungszustandes und der Belastungs-Historie der Materialien
AnalysisRevertResponse			Bestätigung
AnalysisTerminate			Abbruch der aktuellen Analyse
AnalysisTerminateResponse			Bestätigung
NodeResponseGet	nodeId	Ganzzahl, “all“	Anforderung Analyse-Ergebnisse für Knoten. Datenformat STEP, Entity „SfaNodalResponse“
	timeStep	Ganzzahl	
NodeResponseGetResponse			Analyse-Ergebnisse für Knoten
BeamResponseGet	elementId	Ganzzahl, “all“	Anforderung Analyse-Ergebnisse für Knoten. Datenformat STEP, Entity „SfaBeamResponse“, „SfaShellResponse“
	timeStep	Ganzzahl	
BeamResponseGetResponse			Analyse-Ergebnisse für Elemente
Sync			Anforderung einer Antwort, sobald das aktuelle Kommando abgearbeitet ist
SyncResponse			Antwort zu „Sync“

B EXPRESS-Schema für das Datenmodell zum inkrementellen Transfer von Strukturmodellen

Das Schema ist an das von IFC4 angelehnt. Hier sind nur die von IFC abweichenden und für das Strukturmodell wesentlichen Teile des SFA-Schemas dargestellt. Das vollständige Schema sowie eine Software zur Übersetzung in C++ Quellcode findet sich in einem Open Source Repository unter https://code.google.com/p/sfa/

Um den Namensraum einheitlich abzugrenzen, wurde wie in IFC ein Präfix „Sfa“ an alle TYPE- und ENTITY- Namen angefügt.

```
ENTITY SfaFiber;
       X : REAL;
       Y : REAL;
       Area : REAL;
       Material : SfaMaterial;
END_ENTITY;

ENTITY SfaFiberSection;
       FibersList : LIST [1:?] OF SfaFiber;
END_ENTITY;



ENTITY SfaMaterial
 SUPERTYPE OF (ONEOF
              (SfaElasticMaterial
              ,SfaMaterialConcrete02
              ,SfaMaterialBilinear
              ,SfaMaterialHysteretic
              ,SfaMaterialSteel02));
       Name : OPTIONAL SfaLabel;
       Description : OPTIONAL SfaText;
       MassDensity : SfaMassDensityMeasure;
END_ENTITY;

ENTITY SfaMaterialBilinear
 SUBTYPE OF (SfaMaterial);
       PlasticStrain : SfaStrainMeasure;
       YieldStress : SfaPressureMeasure;
       PostYieldingSlope : SfaPressureMeasure;
END_ENTITY;

ENTITY SfaMaterialConcrete02
 SUBTYPE OF (SfaMaterial);

       CompressionStrength : SfaPressureMeasure;
       CompressionStrain : SfaStrainMeasure; (*strain at compression strength*)
       UltimateStrength : SfaPressureMeasure; (*stress at ultimate (crushing) strain*)
       UltimateStrain : SfaStrainMeasure; (*ultimate (crushing) strain*)
       UnloadingSlope : SfaRatioMeasure; (*ratio between unloading slope at epscu and
original slope*)
       TensileStrength : SfaPressureMeasure;
       TensionStiffening : SfaPressureMeasure; (*tension stiffening slope*)

DERIVE
       E0 : SfaPressureMeasure := 2.0*CompressionStrength/CompressionStrain; (*initial
young's modulus*)

END_ENTITY;


ENTITY SfaMaterialHysteretic
 SUBTYPE OF (SfaMaterial);

       YieldStressPositive : SfaPressureMeasure;
       StrainAtYieldStressPositive : SfaStrainMeasure;

       UltimateStressPositive : SfaPressureMeasure;
       StrainAtUltimateStressPositive : SfaStrainMeasure;

       CrackingStrainPositive : SfaStrainMeasure;
```

```
       StressAtCrackingStrainPositive : SfaPressureMeasure;

       YieldStressNegative : SfaPressureMeasure;
       StrainAtYieldStressNegative : SfaStrainMeasure;

       UltimateStressNegative : SfaPressureMeasure;
       StrainAtUltimateStressNegative : SfaStrainMeasure;

       CrackingStrainNegative : SfaStrainMeasure;
       StressAtCrackingStrainNegative : SfaPressureMeasure;

END_ENTITY;

ENTITY SfaMaterialSteel02
 SUBTYPE OF (SfaMaterial);

       YieldStress : SfaPressureMeasure;
       InitialStiffness : SfaPressureMeasure;
       HardeningRatio : SfaRatioMeasure;  (*     (Esh/E0)   *)
       R0 : REAL;   (* exp transition elastic-plastic  *)
       cR1 : REAL;  (* coefficient for changing R0 to R  *)
       cR2 : REAL;  (* coefficient for changing R0 to R  *)
       a1 : REAL;   (* coefficient for isotropic hardening in compression  *)
       a2 : REAL;   (* coefficient for isotropic hardening in compression  *)
       a3 : REAL;   (* coefficient for isotropic hardening in tension  *)
       a4 : REAL;   (* coefficient for isotropic hardening in tension  *)

END_ENTITY;


ENTITY SfaMaterialElastic
 SUBTYPE OF (SfaMaterial);
       YoungsModulus : SfaPressureMeasure;
END_ENTITY;


ENTITY SfaResponse
 ABSTRACT SUPERTYPE OF (ONEOF
       (SfaResponseNode,SfaResponseFiberSection,SfaResponseBeam));
       Name : OPTIONAL SfaLabel;
       TimeStep : OPTIONAL REAL;
END_ENTITY;

ENTITY SfaResponseNode
 SUBTYPE OF (SfaResponse);
       Displacements : LIST [1:3] OF REAL;
       CorrespondingNode : SfaNode;
END_ENTITY;

ENTITY SfaResponseFiberSection
 SUBTYPE OF (SfaResponse);
       FiberValues : LIST [1:?] OF REAL; (* same order as fibers in SfaFiberSection *)
END_ENTITY;

ENTITY SfaResponseBeam
 SUBTYPE OF (SfaResponse);
    IntegrationPoints : LIST [1:?] OF LIST [3:3] OF REAL;
       Displacements : LIST [1:?] OF LIST [6:6] OF REAL;
       Stress : LIST [1:?] OF SfaResponseFiberSection;
       Strain : LIST [1:?] OF SfaResponseFiberSection;
       CorrespondingBeam : SfaBeamElement;
END_ENTITY;



ENTITY SfaSolver
  ABSTRACT SUPERTYPE OF (ONEOF
       (SfaSolverBandSPDLinLapack
       ,SfaSolverProfileSPDLinDirectSolver
       ,SfaSolverSuperLU));
END_ENTITY;

ENTITY SfaSolverProfileSPDLinDirectSolver
  SUBTYPE OF (SfaSolver);
  (*Profile SPD - Direct profile solver for symmetric positive definite matrices*)
END_ENTITY;

ENTITY SfaSolverBandSPDLinLapack
```

```
  SUBTYPE OF (SfaSolver);
  (*Band SPD - Direct solver for banded symmetric positive definite matrices*)
END_ENTITY;

ENTITY SfaSolverSuperLU
  SUBTYPE OF (SfaSolver);
END_ENTITY;

ENTITY SfaStructuralAnalysisModel
 ABSTRACT SUPERTYPE OF (ONEOF
       (SfaStructuralAnalysisModelStatic
       ,SfaStructuralAnalysisModelTransient));
       Name : OPTIONAL SfaLabel;
       PredefinedType : SfaAnalysisModelTypeEnum;
       NumLoadIncSteps : INTEGER;
END_ENTITY;

ENTITY SfaStructuralAnalysisModelStatic
 SUBTYPE OF (SfaStructuralAnalysisModel);
END_ENTITY;

ENTITY SfaStructuralAnalysisModelTransient
 SUBTYPE OF (SfaStructuralAnalysisModel);
       TimeIncrement : REAL;
       ConvergenceTestTolerance : REAL;
       ConvergenceTestNumIterations : INTEGER;
END_ENTITY;


ENTITY SfaStructuralAction
  ABSTRACT SUPERTYPE OF (ONEOF
       (SfaStructuralLoad,
       SfaStructuralLoadCase));
INVERSE
       AssignedToLoadCase : SET [0:?] OF SfaStructuralLoadCase FOR StructuralActions;
END_ENTITY;


ENTITY SfaStructuralLoad
 ABSTRACT SUPERTYPE OF (ONEOF
       (SfaStructuralLoadStatic,
       SfaStructuralLoadTransient))
  SUBTYPE OF(SfaStructuralAction);
       Name : OPTIONAL SfaLabel;
END_ENTITY;


ENTITY SfaStructuralLoadCase
 SUBTYPE OF (SfaStructuralAction);
       TimeSeries : SfaTimeSeries;
       SelfWeightCoefficients : OPTIONAL LIST [3:3] OF SfaRatioMeasure;
       StructuralActions : SET [0:?] OF SfaStructuralAction;
END_ENTITY;


ENTITY SfaLinearForceValue;
       Position : SET [1:2] OF SfaLengthMeasure;
       LinearForceX : OPTIONAL SfaLinearForceMeasure;
       LinearForceY : OPTIONAL SfaLinearForceMeasure;
       LinearForceZ : OPTIONAL SfaLinearForceMeasure;
       LinearMomentX : OPTIONAL SfaLinearMomentMeasure;
       LinearMomentY : OPTIONAL SfaLinearMomentMeasure;
       LinearMomentZ : OPTIONAL SfaLinearMomentMeasure;
END_ENTITY;

ENTITY SfaStructuralLoadLinearForce
 SUBTYPE OF (SfaStructuralLoadStatic);
       Element : OPTIONAL SfaElement;
       LoadValues : SET [1:?] OF SfaLinearForceValue;
END_ENTITY;


ENTITY SfaStructuralLoadPlanarForce
 SUBTYPE OF (SfaStructuralLoadStatic);
       PlanarForceX : OPTIONAL SfaPlanarForceMeasure;
       PlanarForceY : OPTIONAL SfaPlanarForceMeasure;
       PlanarForceZ : OPTIONAL SfaPlanarForceMeasure;
END_ENTITY;
```

```
ENTITY SfaStructuralLoadSingleDisplacement
 SUPERTYPE OF (ONEOF
        (SfaStructuralLoadSingleDisplacementDistortion))
 SUBTYPE OF (SfaStructuralLoadStatic);
        DisplacementX : OPTIONAL SfaLengthMeasure;
        DisplacementY : OPTIONAL SfaLengthMeasure;
        DisplacementZ : OPTIONAL SfaLengthMeasure;
        RotationalDisplacementRX : OPTIONAL SfaPlaneAngleMeasure;
        RotationalDisplacementRY : OPTIONAL SfaPlaneAngleMeasure;
        RotationalDisplacementRZ : OPTIONAL SfaPlaneAngleMeasure;
END_ENTITY;

ENTITY SfaStructuralLoadSingleForce
 SUPERTYPE OF (ONEOF
        (SfaStructuralLoadSingleForceWarping))
 SUBTYPE OF (SfaStructuralLoadStatic);
        Node : OPTIONAL SfaNode;
        Element : OPTIONAL SfaElement;
        Position : OPTIONAL SfaLengthMeasure;
        ForceX : OPTIONAL SfaForceMeasure;
        ForceY : OPTIONAL SfaForceMeasure;
        ForceZ : OPTIONAL SfaForceMeasure;
        MomentX : OPTIONAL SfaTorqueMeasure;
        MomentY : OPTIONAL SfaTorqueMeasure;
        MomentZ : OPTIONAL SfaTorqueMeasure;
END_ENTITY;

ENTITY SfaStructuralLoadSingleForceWarping
 SUBTYPE OF (SfaStructuralLoadSingleForce);
        WarpingMoment : OPTIONAL SfaWarpingMomentMeasure;
END_ENTITY;

ENTITY SfaStructuralLoadStatic
 ABSTRACT SUPERTYPE OF (ONEOF
        (SfaStructuralLoadLinearForce
        ,SfaStructuralLoadPlanarForce
        ,SfaStructuralLoadSingleDisplacement
        ,SfaStructuralLoadSingleForce
        ,SfaStructuralLoadTemperature))
 SUBTYPE OF (SfaStructuralLoad);
END_ENTITY;

ENTITY SfaNode;
        Coordinates : LIST [1:3] OF SfaLengthMeasure;
        Support : OPTIONAL SfaBoundaryNodeCondition;
INVERSE
        NodalResponse : SfaResponseNode FOR CorrespondingNode;
END_ENTITY;


ENTITY SfaTimeSeries
 ABSTRACT SUPERTYPE OF (ONEOF
        (SfaTimeSeriesConstant
        ,SfaTimeSeriesPath));
END_ENTITY;


ENTITY SfaTimeSeriesConstant
 SUBTYPE OF (SfaTimeSeries);
        Factor : REAL;
END_ENTITY;

ENTITY SfaTimeSeriesPath
 SUBTYPE OF (SfaTimeSeries);
        Factor : REAL;
        TimeValues : SET [0:?] OF REAL;
        LoadFactors : SET [0:?] OF REAL;
END_ENTITY;
```

C 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 31. Januar 2013 Fabian Gerold

D Über den Autor

Allgemeine Angaben

Name:	Fabian Gerold
Geburtstag und –ort:	11.05.1978 in Kressbronn am Bodensee

Schulbildung

1985 – 1996	Waldorfschule Wangen i.Allg.
1996 – 1998	Waldorfschule Freiburg i.Br.

Wehrdienst

Januar - Oktober 1999	Grundwehrdienst

Studium und beruflicher Werdegang

Oktober 2000	Beginn des Studiums Bauingenieurwesen an der HTWG Konstanz
August 2002 - Januar 2003	Praxissemester bei Dr. Hutarew & Partner Consulting Engineers, Pforzheim
März - September 2003	Praxissemester bei Kirchhoff+Schleith Straßenbau GmbH, Konstanz
Oktober 2004	Diplom: "Dreidimensionale Modellierung von Gebäuden mit der Methode der Finiten Elemente"
Oktober 2004	Beginn des Masterstudiums Konstruktiver Ingenieurbau an der HTWG Konstanz
März - Juni 2006	Masterarbeit: "Numerische Optimierung im Konstruktiven Ingenieurbau "
2005 - 2006	Lehrauftrag für Bauinformatik und Technische Mechanik an der HTWG Konstanz
Seit November 2006	Wissenschaftlicher Mitarbeiter und Doktorand am Lehrstuhl für Informatik im Bauwesen von Prof. Dr.-Ing. K. Beucke an der Bauhaus-Universität Weimar

Wettbewerbsbeiträge

2005 "Architektur- und Ingenieurbaupreis Hamburg 2005"

Erster Preis als Teamarbeit mit den Architekten Johannes Oelschläger und Jan Voswinckel.

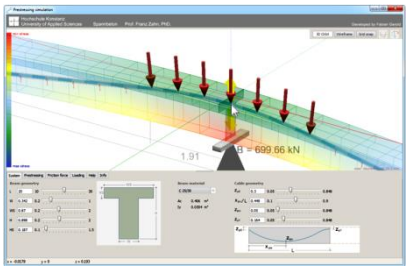
Weitere Informationen: www.hamburgtower.com

2010 Auf IT gebaut 2010, www.aufitgebaut.de

Ausgelobt vom Bundesministerium für Wirtschaft und Technologie und dem Hauptverband der Deutschen Bauindustrie e.V.

Erster Preis in der Kategorie Bauingenieurwesen. Beitrag: "E-Learning in der Tragwerksplanung". Download der Software:

www.dictionary.bi.htwg-konstanz.de/PrestressedBeam/



Publikationen

[Werkle, Gerold 2005] Werkle H., Gerold F., Modellierung stabartiger Bauteile bei Flächentragwerken, *Sofistik Seminar 2005*, Nürnberg, Sofistik GmbH Oberschleißheim

[Gerold, Koch 2007] Gerold, F.; Koch, C., OpenSource im Bauwesen: Ingenieurgerechtes Pre- und Postprocessing in der Numerischen Simulation.“ In: Merkel, A. (Hrsg.); Schütz, R. (Ed.); Wießflecker, T. (Ed.): *Forum Bauinformatik 200*7. Graz: Verlag der Technische Universität Graz, 2007.

[Gerold 2007] Gerold, F.: „Numerische Optimierung im Konstruktiven Ingenieurbau“. In: Merkel, A.; Schütz, R.; Wießflecker, T. (Hrsg.): *Forum Bauinformatik 2007*. Technische Universität Graz, September 2007. - ISBN 987-3-902465-86-3

[Siffling et al. 2007] Siffling, M.; Katz, C.; Gerold, F., Optimierung mit dem Programmmodul OPTIMA. *Sofistik Seminar 2007*, München, Sofistik GmbH Oberschleißheim

[Gerold 2010] Gerold, F., 2010. Integrative Structural Design. In: *Computing in Civil and Building Engineering, Proceedings of the International Conference*, W. TIZANI (Editor), 30 June-2 July, Nottingham, UK, Nottingham University Press, Paper 287, Page 573, ISBN 978-1-907284-60-1

[Werkle et al. 2011] Werkle, H.,Michaelsen, S.,Francke, W.,Denk, H.,Gerold, F.,Lumpe, G.,Möller, G.,Schulz, G.,Mathcad in der Tragwerksplanung, Springer Vieweg, Wiesbaden, 2. Auflage, 2011

[Gerold et al. 2012] Gerold, F., Beucke, K., Seible, F., Integrative Structural Design. In: *Journal of Computing in Civil Engineering*, 26(6), 720–726 (2012). doi: 10.1061/(ASCE)CP.1943-5487.0000180

[Lhotzky et al. 2013] Lhotzky F., Gerold F., Noack M., Nutzung neuer Web-Technologien (HTML5) für Ingenieuranwendungen am Beispiel einer interaktiven Spundwand-Simulation. In: *Bauingenieur 2-2013*, Seite 73-77, 2013