Objektorientierte Dokumentation, Visualisierung und Auswertung der Planungs- und Ausführungsphasen eines Bauprojektes

An der Fakultät Bauwesen der Universität Dortmund zum Antrag auf die Eröffnung des Promotionsverfahrens vorgelegte

DISSERTATION

zur Erlangung des Grades eines Doktors

der Ingenieurwissenschaften

von

Dipl.-Ing Dirk Neethling

Essen,

November 2006

Vorwort

Die vorliegende Arbeit entstand waehrend meiner Taetigkeit als Bauingenieur, mit Schwerpunkt Bauinformatik, bei der Firma HOCHTIEF Construction AG.
Die praktischen Erfahrungen aus den Bereichen Terminplanung, Kalkulation und IT-intensiver Modellierung von Nachtraegen haben die Durchfuehrung der Arbeit massgebend unterstuetzt. Die Grundlagen des in der Arbeit aufgefuehrten Modells wurden weitgehend selbst entwickelt.
Die Erfahrungen deckten eine grosse Kluft zwischen kommerziell und teilweise in der Forschung vorhandenen Modellen und den Anforderungen an ein wirklich taugbares Modell zur Erfassung von Aenderungen eines Bauprojektes auf. Diese Arbeit moechte einen substantiven Beitrag liefern, diese Kluft zu schliessen.
Ich möchte folgenden Personen und Firmen meinen Dank für ihre Unterstützung während der Erstellung der Arbeit ausdrücken:

Prof. Dr.ès.sc.techn. N. Kohler, Universität Karlsruhe
Prof. Dr.-Ing. U. Blecken, Universität Dortmund
Prof. Dr.-Ing. K.-H. Schiffers, Universität Dortmund
Stefan Konz, Fa. Nemetschek
Christian Weyer, thinktecture
Dr. M. Mechnig, Fa. HOCHTIEF Construction AG
Fa. NETRONIC

Essen, 2007-09-09 Dirk Neethling

Tag der mündlichen Prüfung: 24.04.2007
Vorsitzender des Promotionsausschusses: Prof. Dr.-Ing. U. Blecken, Universität Dortmund
1. Gutachter: Prof. Dr.-Ing. K.-H. Schiffers, Universität Dortmund
2. Gutachter: Prof. Dr.ès.sc.techn N. Kohler, Universität Karlsruhe
Beisitzender: Prof. Dr.-Ing. M. Gralla, Universität Dortmund

ABBILDVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

Abb.	Abbildung
AEC	Architecture Engineering Construction
AG	Auftraggeber
API	Application Programming Interface
AOIM	Area of Interest Management
ASP	Active Server Pages
ASPX	ASP.NET
ATM	Asynchronous Transfer Mode
ASCII	American Standard Code for Information Interchange
BOT	Build Operate Transfer
bzw.	Beziehungsweise
CAD	Computer Aided Design
CERN	Center for European Particle Research
CLR	Common Language Runtime
COM	Component Object Model
CONSIMOBs	CONstruction SIMulation OBjects
CORBA	Common Object Request Broker Architecture
CSA	Critical Space Analysis
CSS	Cascading style sheet
2D	zweidimensional
3D	dreidimensional
4D	vierdimensional
DB	Database
DBMS	Database Managementsystem
DCOM	Distributed Component Object Model
Def.	Definition
d.h	das heißt
DIN	Deutsche Industrie Norm
DISCO	Discovery Protocol
DLL	Dynamic Link Library
DNS	Domain Name Server

DOM	Document Object Model
DSP	Digital Signal Processing
eBusiness	electronic Business
EDV	Elektronische Datenverarbeitung
ERF	Electro-Rheological Fluid
etc.	et cetera
exe	Executable File
Fa.	Firma
FIFO	First In First Out
FTP	File Transfer Protocol
GAEB	Gemeinsamer Ausschuss Elektronik im Bauwesen
Guids	Globally Unique Identifier
HMD	Head Mounted Device
HOAI	Honorarordnung für Architekten und Ingenieure
HTML	HyperText Markup Language
HTTP	HyperText transfer Protocol
i. Allg.	im Allgemeinen
i.d.R.	in der Regel
IIS	Internet Information Services
ISO/OSI	International Standardisation Organization – Open System-Interconnection
IP	Internet Protocol
IT	Informationstechnologie
JIT	Just In Time
JVM	Java Virtual Machine
KG	Kostengliederung (der DIN 276)
LAN	Local Area Network
Lab	Laboratory
LOD	Level of Detail
m	Meter
MIS	Management Information System
MTA	MultiThreaded Apartment

nD	n-Dimensional (mehr Dimensional)
NDR	Network Data Representation
.NET	Dot.NET Framework
Obj	Object
ODBC	Open Database Connectivity
OO	Objekt Orientiert
OS	Object Space
P.	Page
PDF	Portable Document Format
PDU	Protocol data unit
RDO	Remote Data Objects
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RTF	Rich Text Format
S.	Seite
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
sog.	so genannte
SQL	Structured Query Language
SSPI	Integrated Security
STA	SingleThreaded Apartment
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol - Internet Protocol
u.	und
u.a.	unter anderem
UDDI	Universal Description Discovery and Intergration
UDP	User Datagram Protocol
UML	Unified Modeling Language
URL	Uniform Resource Locator
usw.	und so weiter
UUID	Universally Unique Indentifier

VB	Visual Basic
VIRCON	Virtual Construction
VM	Virtual Machine
VOB	Verdingungsordnung für Bauleistungen
VRML	Virtual Reality Modeling Language
VR	Virtual Reality
WAP	Wireless Application Protocol
WBS	Work Break Down Structure
WIN32	Windows32
W3C	World Wide Web Consortium
WS	Webseite (kontinuierliche Seite, keine Seitenzahlen vorhanden)
WSDL	Web Service Discription Language
WWW	World Wide Web
z.B.	zum Beispiel
z.T.	zum Teil
z.Zt.	zur Zeit
X3D	Extensible 3D
XML	Extensible Markup Language
XPath	XML Path Language
XQL	Extensible Query Language
XSD	XML Structure Definitions
XSL	Extensible Stylesheet Language
XSLT	XSLT Transformation

ZUSAMMENFASSUNG

Das notwendige Projektmanagement bei aktuellen Bauprojekten größerer Ordnung wird gefördert, aber auch erschwert durch heutige, digitale Hilfsmittel. Die Analyse wie auch die Nachbetrachtung projektbezogener Daten sind mühselig und manchmal ohne eindeutigen Beleg. Wenn entscheidende Informationen wegen der Komplexität oder des Aufwandes einer Erfassung fehlen, so können wichtige Aussagen nicht eindeutig belegt werden. Eines der größten Probleme bleibt die Versionierung. Zum einen ist oft nicht klar, in welcher Reihenfolge Daten verändert wurden, zum anderen gehen vorherige Stände, die beispielsweise Zustände dokumentierten, gehen bei Aktualisierungen verloren.

Ein objektorientierter Ansatz eignet sich allerdings gut für die Modellierung von komplexen Projektmodellen. Das liegt daran, dass sowohl die einzelnen Entitäten als Objekte wie auch deren Beziehungen damit modelliert werden können. Allerdings muss ein solches Objektmodell, das zur Modellierung eines Projektes herangezogen wird, aufgrund der ständigen Veränderungen dynamisch erweiterbar sein.

Ein objektorientiertes Projektmodell weist als zentrale Eigenschaft eine oder mehrere Objekträume als Datenhaltung auf. Das Objekt beinhaltet alle Informationen und Methoden, welche für die Funktion des Objektes essentiell sind. Daher ist das funktionale Objekt auch die eigene Datenhaltung desselben. Das Objekt wird binär und im Zusammenspiel mit XML instanziert und serialisiert. Die Schablone des Objektes, die Klasse, wird auch binär oder als XMLSchema beschrieben und serialisiert. Die Klassen sind dynamisch (zur Laufzeit) erweiterbar.

In der Modellierung ist der objektorientierte Ansatz Komplex, vor allem in Anbetracht der dynamischen Komponente. Zur Überwindung der von Henckels beschriebenen Schwäche statischer Objektmodelle, wie das des IFC [Henckels, D., 2004. S. 24], wird eine Abstrahierung eingeführt, die das Ausführen von imperativen Sprachelementen durch deklarative (textbasierte) Befehle ermöglicht. Auf diese Weise wird die objektorientierte Vorgehensweise einem baufachlich Handelnden zugänglich gemacht. Dieser Lösungsansatz bietet entscheidende Vorteile durch das funktionale Ermessen, die objektzentrierte Speicherung, das Ereignismodell und die daraus erfolgende Objekt- und Klassen-

dynamik zur Behandlung von Änderungen einer Ausgangssituation.

Somit wird die eher prozessorientierte Methodik des Projektmanagements durch ein Objekt- und Ereignismodell ersetzt. Dieses Modell wird schon in der Planung, einem interaktiven Ereignismodell, aufgestellt. Die Objekte sind das Ergebnis des fachlichen Ermessens des Projektes.

Bei der Umsetzung des Objektmodells haben sich vier technologische Eigenschaften ergeben:

- Moderne, verteilte Protokolle ermöglichen die Bearbeitung und das Agieren auf Objekte in Unabhängigkeit von:
  - Raum,
  - Zeit,
  - Plattform und
  - Technologie.

  Dabei ist jedoch zu beachten, dass Aspekte wie Leistung und Stabilität von der Technologie abhängig sind.
- Die Eigenschaften und Methoden der Objekte spiegeln fachliche Spezialisierungen wieder. Die Summe dessen ist das integrierte, umfassende, funktionale Objekt.
- Das Klonen der Objekte bei Eigenschaftsänderungen sichert die Ausgangslage und ermöglicht die weitere Objektbearbeitung. Die Sicherungsmechanismen greifen nur auf das Objekt zu, andere Objekte werden nicht beeinflusst.
- Das Objektmodell bietet Strukturbäume, worüber navigiert und aggregiert werden kann.

In der Ausführung findet der Vergleich mit dem bei der Vergabe festgelegten Objektraum statt. Jegliche nennenswerte Änderung wird durch ein Ereignis gemeldet, und das System reagiert darauf durch die Bereitstellung eines neuen Objektes bestehend aus den Eigenschaften vor der Änderung sowie aller danach angebrachten Informationen. Die so entstehende Veränderungshierarchie dokumentiert die Abweichung von dem Ausgangszustand.

Die fachliche Innovation umfasst vier Bereiche:

- Das Agieren mit verteilten Objekten ermöglicht einen asynchronen, entfernten und

bisweilen interaktiven Zugriff mehrerer Prozesse auf den gleichen Datenbestand.

- Die Frage der Funktionalität gestaltet die Struktur und den Inhalt des Objektes (Attribute, Eigenschaften).
- Das dynamische Klonen von Objekten bei der Änderung derer Attribute und Eigenschaften ermöglicht die exakte Beschreibung der Änderungshistorie (gegenüber definierten Zuständen wie z.B. der Vergabe), da der Objektbaum den Umfang der Änderungen und deren Struktur darstellt.
- Die dynamische Vererbung (während der Laufzeit) ermöglicht eine Ausweitung des Objektmodells und die Gestaltung beliebiger Objektformen (Klassen). Damit unterliegt das Objektmodell nicht den Einschränkungen des IFC Modells.

Dieser Objektbaum wird zwingend Gegenstand aller Projektbeteiligten, da er eine strukturelle Veränderung einer jeweiligen vertraglichen Situation und Position darstellt. Somit ergibt sich die Möglichkeit einer exakten Bestimmung der Abweichung von der jeweiligen fachlich definierten Ausgangssituation (Planungsphase, Vergabe, Übergabe).

Objektmodelle sind gewissen Begrenzungen unterworfen. Das liegt zum einen an der Notwendigkeit der Erweiterung entlang hierarchischer Strukturen (Vererbung) [Henckels, D., 2004. S.10] und zum anderen daran, dass Typsicherheit zur Funktionstüchtigkeit dieser Modelle unerlässlich ist [Box, D.,, 2005. S. 2]. Ein dynamisches Objektmodell kann zwar die Hierarchien in einem Objektmodell nicht auflösen, jedoch können die problematischsten Einschränkungen eines starren Objektmodells überwunden werden. Kann allerdings der Problematik der erhöhten Komplexität und Datenpflege bei einem erweiterten Objektmodell nur mittels der verteilten Architektur des Objektmodells entgegengewirkt werden. Es bleibt also dieses grundsätzliche Problem eines objektorientierten Ansatzes bestehen. Die Kopplung des dynamischen Objektmodells mit fachlich zugespitzten Applikationen mittels Schnittstellen löst das Versionisierungsproblem, da diese Aufgabe dem Objektmodell zugeordnet und nicht den einzelnen Applikationen überlassen wird. Durch die Trennung des Objektmodells von den Applikationen werden die häufig auftretenden Probleme, deren Lösungen auf spezifischen fachlichen Applikationen aufbauen (z.B. CAD), umgangen. Ein weiterer Vorteil ist, dass ein solches Objektmodell in verschiedenen Sprachen und Programmierumgebungen realisiert werden kann.

AUFBAU DER ARBEIT

Diese Arbeit ist in vier Teile gegliedert (A, B, C und D), die jeweils in mehrere Abschnitte (I, II, III usw.) unterteilt sind.
Teil A behandelt die Grundlagen der EDV-technischen Projektmodellierung, wie sie üblicherweise in Firmen und Instituten praktiziert wird. Dadurch, dass diese Technologien in anderen Industriezweigen weiter fortgeschritten sind als im Baugewerbe, werden auch Entwicklungen in anderen relevanten Disziplinen im Anhang behandelt.

In Teil B wird ein neuer gesamtheitlicher Ansatz, konform der Aufgabenstellung, in der Projektmodellierung untersucht. Dieser Ansatz unterscheidet sich in drei Gesichtpunkten von herkömmlichen Entwicklungen:

- Der Kern eines Objektes wird nicht durch deren Geometrie, sondern durch dessen Funktion dargestellt.
- Das Objekt basiert auf einem Schema, d.h. einer Schablone (Template). Dieses Schema kann aus Gründen der Datenkonsistenz nach Bedarf dynamisch erweitert, jedoch niemals reduziert werden.
- Veränderungen an Objekten (d.h. Modifizierungen des Bauwerks) werden dokumentiert, indem das Objekt im Zustand unmittelbar vor der Modifizierung eingefroren, kopiert und dann mit den Änderungen versehen wird. Dieser Prozess ist EDV-technisch auch als Klonen (Cloning) bekannt, wobei die Tiefe des Klonens (depth of cloning) variabel gestaltet werden kann. Diese Tiefe des Klonens bestimmt, ob auch alle Kinder-Objekte geklont werden oder nur die äußere Objektschale geklont wird.

In Teil C wird das neu entwickelte Modell praktisch angewandt: der Bau eines Fakultätsgebäudes der Universität Dortmund wird simuliert. Die HOAI Phasen 1-5 sind berücksichtigt worden. Auf einige Spezialbereiche, die sich eher in höheren Phasen finden, wie z.B. Soll/Ist Vergleiche oder Mängel, wird kurz eingegangen.
In Teil D werden die Ergebnisse zusammengefasst, wird auf einige Schwachstellen des Modells hingewiesen und ungelöste Fragestellungen werden aufgeführt. Die Integration des Modells in herkömmliche Arbeitsprozesse wird auch skizziert.

EINLEITUNG

Es besteht ein Interesse, Bauprojekte hinsichtlich Entwurf, Terminen, Mengen und Kosten in einem virtuellen Projekt abzubilden. Diese Abbildung sollte nicht nur eine begleitende Dokumentation sein, sondern eine Basis für die projektbegleitende Planung und Steuerung darstellen. Somit agiert diese Art der Dokumentation als Objektmodell und Steuerungsinstrument einer integrierten Lösung.

Die Integration von EDV-technischen Entwicklungen im Planungs- und Bauprozess folgt der Entwicklung im Bereich Informatik mit einer gewissen Verzögerung und ist zudem von einigen Misserfolgen geprägt. Nach den missglückten Robotikversuchen der 70er und 80er Jahre wurden anfänglich zu große Hoffnungen in die Möglichkeiten des Internets gesetzt. Die Integration hin zu einem vollintegrierten Modell wurde oft als Archipellandschaft dargestellt (Inseln repräsentieren die fachlichen Bereiche), mit einem sich neigenden Meeresspiegel, der durch Entwicklungen in der Informatik gesenkt wird. Die Integration der fachlichen Bereiche versprach eine höhere Effizienz des Projekts und mehr Steuerungsmöglichkeiten. Dieses Bild namm allerdings keine Rücksicht auf das Interesse der Lieferanten von Technologien, die versuchen ihre Technologie-Inseln aus kommerziellen Gründen möglichst isoliert zu halten. Das beste Beispiel hierfür ist die nach wie vor wenig vorhandene Kommunikationsmöglichkeit zwischen Java-basierten und .NET-basierten Applikationen trotzt moderner Interprozess-Technologien wie XML Services und CORBA.

In der Vergangenheit wurden überwiegend prozessorientierte oder CAD[1] -basierte, objektorientierte Ansätze erarbeitet. Seit Einführung des CAD in Planungsbüros und -abteilungen stehen Daten als Quellen für anschließende Tätigkeiten zur Verfügung.

Gegenwärtig beschränkt sich die Nutzung von CAD-Daten weitgehend auf das Ausplotten und die Bereitstellung der Pläne für die nachfolgenden Beteiligten. Weiterentwicklungen wie 4DCAD werden später erwähnt. Einige Pilotprojekte sehen einen Datenaus-

[1] CAD – Rechnergestützter Entwurfsprozeß. Herkömmliche manuelle Entwurfsprozesse werden im Rechner abgebildet. Dadurch können auch zusätzlich Funktionen wie das Berechnen von Flächen und Volumina automatisiert werden. Änderungen können mit minimalem Aufwand durchgeführt werden.

tausch zwischen CAD-Anwendungen und anderen Applikationen vor. Dieser stark CAD-bezogene Ansatz bei der Projektmodellierung ist durch die geschichtliche Entwicklung des CAD als „Zeichenbrett-Ersatz“ und dessen zentrale Rolle bei der Projektarbeit zu verstehen.

Die übrigen Projektdaten wie Baubeschreibung, Raumbuch, Terminplanung, Ausschreibung, Verwaltung der Projektbeteiligten, Schriftverkehr etc. werden mit jeweils unabhängigen Softwarelösungen oder gar manuell erfasst.

Diese Lösungen sind für die Praxis aus folgenden Gründen nicht zufriedenstellend:

- Prozessorientierung dokumentiert nur das Ergebnis einer Planung und führt zu keinem Modell als Werkzeug für eine Planung.
- Der funktionale Charakter der Bauelemente wird der Prozessorientierung untergeordnet.
- Die Interaktion zwischen Prozessen ist äußerst schwierig und wird oft nur unlogisch modelliert, da ein einschlägiges Objektmodell fehlt.
- Innerhalb von Prozessen können Interaktionen meistens nur mit einfachen Regeln gekoppelt werden (z.B. in der Terminplanung mit Anfang - Ende und Ende - Ende).
- Die Objektmodelle sind überwiegend in ihrer Fachlichkeit beschränkt und beziehen sich auf technische Größen wie Ordnungszahlen[2], Positionen, Kostenarten oder Vorgängen.
- Die CAD Modellierung basiert üblicherweise auf ein CAD spezifisches Datenmodell, das sich i.d.R. für die Interaktion mit anderen Applikationen wenig eignet.
- Die CAD Modellierung erfordert relativ viele Informationen, da sie überwiegend geometrisch orientiert ist.
- Die EDV Umgebung ist teuer, komplex und erfordert eine umfangreiche Infrastruktur.
- Der Umgang mit CAD-basierten Systemen ist im Bauwesen unterschiedlich weit entwickelt, meist kompliziert und verhindert infolgedessen eine problemlose Interaktion, die eine Voraussetzung für hohe Akzeptanz in der Praxis wäre.
- Die Erweiterung herkömmlicher Objektmodelle geht oft mit einem neuen Software-

[2] Der Begriff Positionen führt zur Verwechslung mit ähnlichen Begriffen in anderen Baudisziplinen und wird demnach mit dem Begriff Ordnungszahl analog VOB ersetzt.

entwicklungszyklus einher. Dieser Zyklus setzt eine fachliche Ausarbeitung und eine Dokumentation der Anforderungen voraus. Ein solcher Zyklus umfasst etliche Monate oder gar Jahre und befriedigt meistens nur den kleinsten gemeinsamen Nenner an erwünschter neuer Funktionalität.

- Schnittstellen sind entweder sehr spezifisch und daher nicht anpassbar, oder werden so allgemein gehalten, dass der Aufwand an Transformationen zur Nutzung dieser Schnittstellen unpraktisch wird.
- Dynamische Schnittstellen (XML Dienste, Remoting, CORBA) werden im Allgemeinen noch nicht in kommerziellen Produkten angeboten. Das liegt and dem hohen Konfigurations- und Transformationsaufwand solcher Schnittstellen.
- Multidisziplinären Lösungen fehlt ein integriertes Ereignismodell. Allerdings bieten wenige API's im Bereich Bausoftware umfangreiche Ereignismodelle. Typischerweise beschränken sich mittels Diensten ansprechbare API's auf Eigenschaften und Methoden.
- Versionisierung der Objekte ist eine zentrale Anforderung für die lückenlose Dokumentation von Veränderungen. Eine automatische Versionisierung der Objekte gestaltet sich beim prozessorientierten Ansatz als schwierig und ist nicht Bestandteil heutiger Softwarelösungen.

Deshalb wird hier darüber reflektiert, wie mit Hilfe eines objektorientierten Datenmodells eine ganzheitliche, rechnergestützte Bauprojektabbildung entwickelt werden kann. Wichtig dafür sind das phasenübergreifende Merkmal und die dynamische Erweiterbarkeit des Modells.

Ein bisher beschrittener Weg war das Ausbauen der CAD-Funktionalität in 3D und der „vierten Dimension“ Zeit (4DCAD) [Clayton, M. et al, 2002. S. 229; Dawood, N. et al, 2003. S.124]. Dieser Lösungsansatz er scheint logisch und ist als zentrales Modellierungstool verbreitet. Eine Definition der Intention dieser Art Visualisierungstechnologie aus Dawood [Dawood N. et al, 2003. S. 123] lautet:

> *„There are two principal lines of development:*
> *To build on work on 4D planning where the process is visualised by building the 3D product model through time according to the critical path network. To build on work on the spatial configuration of the constructed product by applying those analytic principles to space use on site during construction-what we have dubbed critical space analysis (CSA).“*

Die Schwächen dieses Ansatzes sind oben dargestellt und besonders darin zu sehen, dass sich die Objektorientierung vornehmlich auf die Geometrie beschränkt. Das bedeutet, dass zur Datenerfassung eine geometrische Definition erforderlich ist. Objektänderungen werden als Datenstände, nicht jedoch als naturgemäße Historie mit einer Entwicklungshierarchie abgebildet.

In der professionellen Programmierung hat sich der objektorientierte Ansatz weitgehend durchgesetzt und ist bei komplexen EDV-Lösungen i.d.R. unumgänglich. Die hohe Akzeptanz dieser Methodik begründet sich dadurch, dass nach der Funktionalität eines Objektes gefragt wird, was dem Ingenieurdenken näher kommt. Bei den meisten IT Projekten wird ein Objekt als rechnertechnische Größe erfasst, z.B. ein Datensatz, eine Ordnungszahl oder ein Vorgang. Der objektorientierte Ansatz weist allerdings Schwächen auf, was an der starren Vererbungshierarchie und der Anforderung, dass Typinformationen vorab erforderlich sein müssen, liegt. Diese Schwächen wiederum verstärken die Unzulänglichkeiten von fachlich unzureichenden Modellen.

In der baubetrieblichen Informatik kann ein Objekt entweder geometrisch identifiziert werden (z.B. eine Tür) oder ein abstrakt funktionales Gebilde darstellen (z.B. ein Risiko). Die Abbildung (Geometrie, Benennungen, Risikoart) und Eigenschaften (z.B. Farben, Materialien, Vektor der Schadenshöhen und Eintrittswahrscheinlichkeiten) eines Objektes werden durch Attribute dargestellt. Als Methoden werden funktionale Möglichkeiten beschrieben, wie z.B. die Berechnung einer Oberfläche oder das Setzen einer Risikoverteilungskurve.

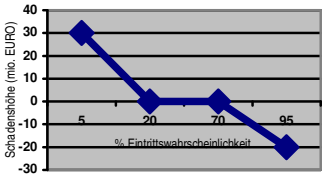
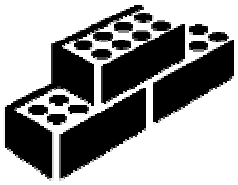
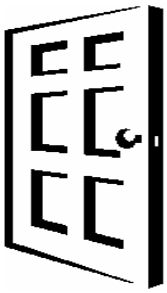
Beschreibung	Art	Attribut	Darstellung	Methode
Risikoart eines Bauprojektes	Abstrakt	Verteilung für Monte Carlo Berechnung		Setze Verteilung in Vektorform und speichere persistent
Mauerwerksziegel	gegenständlich, aber geometrisch nicht definiert	- DIN 18330 - DIN 276: KG 331 - Einbautermin 27.06.2002 (Soll) - beauftragt am 23.04.2001 - AN: Fa. Strabag AG		
Türblatt	gegenständlich	DIN 18 333 DIN 276 KG 344 „Dekor Buche natur“ „Fabrikat Westag“ 88,5/2,01 etc.		Berechne Oberfläche der Tür

Abb. 1: Verschiedene Objektarten

Die Abstrahierung der Bauelemente in ihren eher funktionalen als geometrischen Formen ermöglicht die Integration vieler verschiedener Informationen unabhängig von der Existenz einer geometrischen Definition, die für die jeweilige Funktion (Planung, Terminplanung, Kalkulation, Nachtragsmanagement, etc.) von Interesse ist. Dadurch soll das Objekt nicht auf eine spezifische Fachlichkeit ausgerichtet sein, sondern eher als Drehscheibe der Fachlichkeiten begriffen werden.

Eine Lösung, die dieses Ziel verfolgt, sollte sich zunächst vom CAD trennen und eine funktionale Ausrichtung aufweisen. Die geometrische Modellierung sollte durch ein CAD-System geleistet werden. Die Auslagerung spezialisierter Aufgaben, wie z.B. die geometrische Modellierung (AutoCAD), die Terminplanung (P3) oder eine Kostenschätzung (KUBUS) an dafür eigens konzipierten Applikationen. Das Zusammenspiel dieser Applikationen im Verbund ergibt die Stärke eines solchen Systems. Dadurch werden Aufgaben fachlich konzentriert, die mehrfache Eingabe von Daten vermieden und der Gesamtaufwand eines solchen Systems auf möglichst viele qualifizierte Teilnehmer verteilt.

Die in dieser Arbeit aufgeführte Lösung besteht aus einem dynamisch erweiterbaren Objektmodell. Die Klassen werden mittels XML Schema definiert und können zur Laufzeit erweitert werden. Eine klassische Vererbung wird somit gewährleistet. Disziplinen wie die Terminplanung oder CAD werden mittels Diensten eingebunden. Diese Vorgehensweise erhält herkömmliche Arbeitsweisen und liefert dem Objektmodell alle wesentlichen Änderungen. Dienste garantieren auch einen Grad an Unabhängigkeit zwischen dem Objektmodell und den Applikationen. Aufgrund des klassisch objektorientierten Ansatzes können Auswertungen mit objektorientierten Berichtswesen gemacht werden. Das Objektmodell lässt sich in andere Lösungen integrieren und an beliebige Benutzeroberflächen anschließen.

A TEIL A: Baubetriebliche Informatik und Projektmodellierung

I Baubetrieblich relevante Informatik

Die moderne Informatik umfasst ein beachtliches Spektrum an Technologien und Verfahren. Sie ist in der heutigen Arbeitswelt und modernen Forschung zu einem unverzichtbaren Instrument avanciert. Inzwischen können vor allem technische Aufgaben nicht mehr ohne produktive Anwendungen gemeistert werden.

Generell lassen sich viele IT Technologien auch in den verschiedenen Phasen eines Bauprojektes einsetzen. Oft jedoch spielen andere Faktoren wie Kosten, Ergonomie, Prozessablauf, organisatorischer Aufwand und die technologischen Fähigkeiten der Mitarbeiter eine entscheidende Rolle. Diese Faktoren führten dazu, dass im Bauwesen ein weitaus geringerer Grad an Automatisierung stattgefunden hat als in anderen Industrien [Henckels, D., 2004. S. 3-5].

Hinzu kommt die geringe Bereitschaft der Bauindustrie in die Entwicklung von modernen IT Verfahren zu investieren. Die meisten Entwicklungen beschränken sich auf den Einsatz von Technologien, welche sich in anderen Industriezweigen wie dem Maschinenbau bewährt haben. Bei der Wahl geeigneter Technologien ist aber allem auf die in der Bauindustrie ungünstigen Rahmenbedingungen zu achten. Daher empfiehlt sich bei der Wahl von IT Technologien, möglichst auf bewährte und effiziente Verfahren zu setzen.

In den nächsten Kapiteln und im Anhang werden einige für das Bauwesen relevante Technologien aufgeführt und erläutert.

II Projektmodellierung

Üblicherweise werden Projektinformationen in unterschiedlichen Applikationen und deren Datenbanksystemen gespeichert. Oft handelt es sich auch um ein dateibasiertes Speichermedium. Solche Systeme sind nicht nur räumlich getrennt, sondern weisen unterschiedliche Formate auf und verlangen auch die Transformation der Daten in andere Strukturen. Solche Transformationen sind meistens nicht-trivial oder es fehlen Daten [Henckels, D., 2004. S. 8-9]. Somit entsteht bei einer dezentralen Bearbeitung der Projektdaten, wie es zwangsläufig in einem nicht-integrierten Objektmodell stattfindet, eine Verteilung der Speichermedien und Datenformen. Generell kann die Aussage gemacht werden, dass Termininformationen, geometrische Daten und Kostendaten eines Projektes in unterschiedlichen Systemen untergebracht sind.

1 Herkömmlicher Ansatz

Die erwähnte parallele Datenhaltung führt zu folgenden Problemen:

- Beim Auswerten der gedruckten Pläne werden Mengen fehlerhaft ermittelt.
- Es müssen mehrere Datenquellen abgeglichen werden (z.B. Raumbuch, Pläne).
- Die erneute Erfassung der Daten erfordert einen erneuten Aufwand, der sich beim Abgleich mit den anderen Datenquellen zeigt. Beispiel: Ein im Raumbuch aufgeführtes Türblatt wird nicht mehr angeboten, weil der Hersteller nicht mehr existiert.
- Es sind keine Zuordnung der ein- und ausgehenden Dokumente zu den einzelnen CAD-Objekten vorhanden, sondern lediglich die regelmäßig aktualisierte Version der Daten. Ein nachträgliches Aufklären (z.B. von Änderungen in Bezug auf deren Zeitpunkt und Verursacher) wird damit praktisch unmöglich.
- Datenquellen werden oft binär gespeichert. Ein externer Zugang ist daher meistens nicht möglich.
- Darüber hinaus wird die Chance vertan, aus einem Datenpool heraus operieren zu können und somit stets eindeutige und aktuelle Daten zu haben.

Alle mit der parallelen Datenhaltung einhergehenden Probleme können vermieden werden, wenn sämtliche Projektdaten in einem integrierten Objektmodell gespeichert würden. Meistens deutet dies auch auf eine einzige Datenbankhin. Moderne Datenbanken können über mehrere Server verteilt werden.

2 CAD-basierte Lösungen

3D CAD wird seit einigen Jahren in der Architektur und in geringerem Maße im Bauwesen eingesetzt. Durch diese Technologie wird die räumliche Perspektive des Objektes verstärkt herausgehoben. Es können Unpässlichkeiten zwischen verschiedenen Bestandteilen eines Produktes oder Projektes erkannt und somit ungenügend dimensionierte Öffnungen, unvollständige Sanitärleitungen oder ungenau spezifizierte Gewerke identifiziert werden [Clayton et al., 2002. S. 232 f.]. Fehler in der Dimensionierung können auch durch eine Zusammensetzung der einzelnen 3D Module eines Projektes auf der Terminschiene erkannt werden.

In Bauprojekten kommt der Einsatz einer 3D Modellierung weniger häufig vor. Das liegt daran, dass die Bauausführung mit 2D Zeichnungen eines gewissen Maßstabes erfolgt, vorzugsweise im Maßstab 1:50, ggf. M. 1:20. Dort würde ein 3D Modell der Bauausführung nur als Hilfsinstrument dienen und nicht als primäres Arbeitsmittel. Solche Visualisierungen von CAD Daten stellen immer einen optionalen Zusatz dar [Building Design and Construction, Vol.1, 2004. S. 12].
Dadurch, dass Baumaßnahmen finanziell und zeitlich immer knapp bestückt sind, ist der Aufwand einer solchen Modellierung kaum vertretbar. Erschwerend kommt hinzu, dass die vielen Änderungen während einer Ausführung in das 3D Modell eingearbeitet werden müssten ohne direkten praktischen Nutzen. Somit müssten Vollzeitkräfte für die Aktualisierung bereitgestellt werden. Ein ähnliches Problem betrifft die 4D CAD Modellierung. Nach ersten Einsätzen in der Baupraxis zeichnet sich ab, dass der Aufwand einer 3D Modellierung, im Gegensatz zur anvisierten Positionierung dieser Technologie, nicht im Verhältnis zum praktischen Nutzen steht. Anders als in der Kampfsimulation oder der Telechirurgie (Disziplinen, die sich mit lebensbedrohlichen Szenarien befassen) besteht in der Bauindustrie kein akuter Bedarf für eine Produktivitätsverbesserung.

Clayton [Clayton et al., 2002. S.229] beschreibt das anvisierte 4D CAD Konzept wie folgt:

> *„In the concept of ‚4D CAD', the 3D model is blended with construction schedule information to produce visual simulations of the construction process. The effect is similar to time-lapse photography of a construction site. Research has established the effectiveness of 4D CAD in aiding construction managers in solving scheduling problems. "*

Dawood [Dawood et al., 2002. S. 123] sieht einen zweifachen Sinn in einem 4D/VR Modell:

> *„To build on work on 4D planning where the process is visualised by building the 3D product model through time according to the critical path network. To build on work on the spatial configuration of the constructed product by applying those analytic principles to space use on site during construction – what we have dubbed critical space analysis (CSA). “*

In einem Versuch mit 4D CAD Methoden konnte ein Terminplan mit früheren Endterminen, weniger Vorgängen auf dem kritischen Weg, eine optimierte Ressourcenzuordnung und ein maximaler Puffer für Vorgänge erarbeitet werden [Clayton et al., 2002. S. 229]. Es können also Optimierungen trotz des großen Aufwandes erreicht werden.

Kohler [Kohler und Lockemann, 2003. S. 12] beschreibt die Entwicklung eines lose gekoppelten Gebäudemodells im Gegensatz zu dem oft in dem 4D CAD eingesetzten eng gekoppelten Gebäudemodell. Dieses Konzept der losen Kopplung wird in dieser Arbeit in jenem Sinne verfolgt, dass das Objektmodell dynamisch an den partizipierenden Applikationen angepasst wird und diese nicht in eine starre Hierarchie eingefügt werden. Somit entspricht es auch der Tendenz in der Informatik bezüglich verteilter Dienste [Box, D.,, 2005. S. 2].

Kohler [Kohler und Lockemann, 2003. S. 6]: „Der besondere Charakter der losen Kopplung ist, dass die Aufmerksamkeit vom vollständig integrierten Modell („top-down“) weg auf die zu integrierenden Teile gerichtet wurde („bottom-up“). Akzeptiert man eine nur lose Kopplung der Komponenten, dann bekommen diese größeres Gewicht.“ Kohler [Kohler und Lockemann, 2003. S. 7] untersucht drei Aspekte, die für die Architektur von Wichtigkeit sind:

- Welt der Geometrie / Räume,
- Welt der Akteure / Personen,
- Welt der (Bau-)Elemente / Materialien.

Eine von Kohler beschriebene, sehr interessante Innovation ist die Transformation der Räume eines Gebäudes in niedere Dimensionen mittels der Grafentheorie [Kohler und Lockemann, 2003. S. 8]. Dies geschieht durch die Zerlegung von Raumzellen und flächigen Bauteilen (in Teilen oder Facetten) auf solche Weise, dass jede Facet-

te genau eine Raumzelle auf der Vorder- und Rückseite hat.

Der so entstehende Graph (Volumenadjazenzgraph) weist Knoten als Raumzellen und Kanten als Facetten auf, die jeweils einen Raum auf der Vorderseite und der Rückseite verbinden. Dieser Graph bildet auch den Zusammenhang zwischen Volumen und Flächen. Durch diese Betrachtungsweise entsteht eine topologische Theorie architektonischer Gebäude. Die grundlegende Überlegung ist, dass jede zusammenhängende Menge ein zusammenhängendes Urbild haben muss [Kohler und Lockemann, 2003. S. 13].

Die Literatur, aktuelle Forschungen und kommerzielle Entwicklungen weisen objektorientierte Ansätze auf; oft wird von IFC oder IAI kompatiblen Lösungen gesprochen [Fischer, M., 1993. S. 4]. Es handelt sich hierbei um geometrische Modelle mit Attributen für Termin- und Kosteninformationen [Dawood, N. et al, 2003. S. 130; Clayton et al. 2002. S. 234]. Dieser Standard wurde von einem Gremium aus Industrievertretern beschlossen und in einigen Applikationen integriert (z.B. Microsoft Visio, Nemetschek Allplan). Der IFC Standard sieht eine Klassenbibliothek zur Modellierung von komplexen virtuellen Modellen vor.

Kohler verweist auf die Begrenzung des IFC Modells [Kohler und Lockemann, 2003. S. 8]. Er deutet an, dass die Vielfalt der geometrischen Klassen in der IFC Spezifikation ein Indiz dafür ist, dass ein einheitliches Konzept zur Modellierung der topologischen Relationen fehlt. In diesem Sinne bildet die IFC Spezifikation höchstens eine Beschreibung für den minimalen Dateninhalt einer Schnittstelle.

Weiterhin leidet der IFC, Standard so wie die W3C Standards an der Schwerleibigkeit internationalen Gremien, die sich aus staatlichen-, wirtschaftlichen- und Forschungsinstanzen zusammensetzen. Das führt zu langen Entwicklungszyklen und reduziert alle Erweiterungen auf Konsens niveau. Benutzer dieses Standards sind weitgehend gezwungen, sich innerhalb der Möglichkeiten des Standards zu bewegen. Die Umsetzung des Standards in Produkten der Hersteller ist nicht gesichert und in der Praxis stark variierend.

Der IFC Standard ist zwar auf Grund der zentralen Rolle des CAD im Bauwesen und der geschichtlichen Entwicklung verständlich, führt zu einer zu starken Fixierung auf die Geometrie von Baukörpern, die jedoch nur einen Teil der gesamten Merkmale eines Bauvorhabens ausmacht.

Ein weiterer Nachteil ist, dass die Fixierung auf das IFC Modell, im Gegensatz zu einem lose gekoppelten Modell [Kohler und Lockemann, 2003. S. 6], eine starre Hierarchie voraussetzt (nämlich die des IFC Modelles) Entwicklung und Bedeutung der partizipierenden Applikationen begrenzt. Diese Objekte weisen keine Versionierung außer dem Speichern von Projektständen auf. Ein selbständig agierendes Ereignismodell ist ebenfalls nicht vorhanden. Ein Grund hierfür könnte die Komplikation der Integration eines Ereignismodells in CAD sein.

Fischer [Fischer, 1993. S. 2] führte schon früh die Vorteile einer Integration der geometrischen und zeitlichen Komponenten auf. Dadurch lässt sich die Konstruktion des Baukörpers modellieren sowie Konflikte bei der Installation von Baukörpern ersehen. Diese Technik hat bis dato verstärkt im Anlagenbau (Abb. 2) Interessenten gefunden. Inzwischen ist dieser Ansatz ausgebaut worden, eine Kompatibilität mit dem IAI IFC Modell wird angestrebt. Dadurch sollte der Austausch von Objekten zwischen IFC kompatiblen Lösungen ermöglicht werden. Allerdings hat Kohler, wie oben erwähnt, auf die Begrenzung der IFC Spezifikation hingewiesen. Weiterhin fehlt dem 4DCAD Ansatz der Einsatz von Metaklassen bzw. Klassen zur Modellierung von dynamischen Objektwelten, wie es bei Willenbacher z.B. implementiert wurde [Willenbacher, 2002. S. 67].

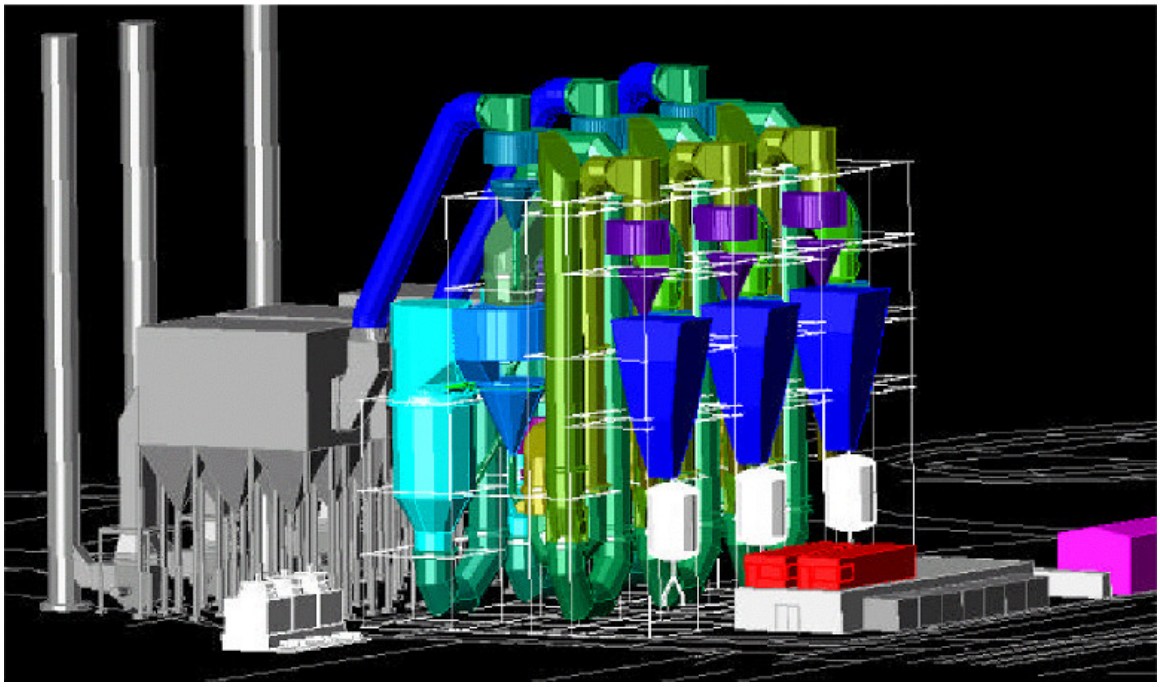


Abb. 2: 4DCAD Darstellung des Sequus Pharmaceutical Pilot Plant [Fischer, M., 1993. S.1].

Die Modellierung der Bauphase soll durch die Simulierung von Bauvorgängen zu Einsparungen führen. Es handelt sich um Erkenntnisgewinnung durch Simulation. Ähnliche Technologien werden beim Militär (Kampfsimulationen) und bei der Luftfahrt (Flugsimulator) eingesetzt.

Die Visualisierung von Baustrukturen dient der subjektiven Darstellung und erweitert die visuelle Informationsfülle. Dadurch wird ein zusätzlicher Sinn aktiviert, der bei der Beurteilung der Machbarkeit von Baumodellen hilfreich sein kann. Insbesondere können Entscheidungen im Projektmanagement dadurch verbessert werden. Diese direkte Kopplung zwischen Modellierungswerkzeugen und Entscheidungsprozessen wurde in der Vergangenheit oft abgelehnt.

Eine wichtige Errungenschaft der 3D Modellierung und insbesondere der VE-Net Technologien ist die Kollisionskontrolle. Die ersten Modelle kannten keine Kollisionskontrolle; das damit verbundene unfreiwillige Durchwandern von Gegenständen wurde sogar als besonderes Merkmal dargestellt (Abb. 3).

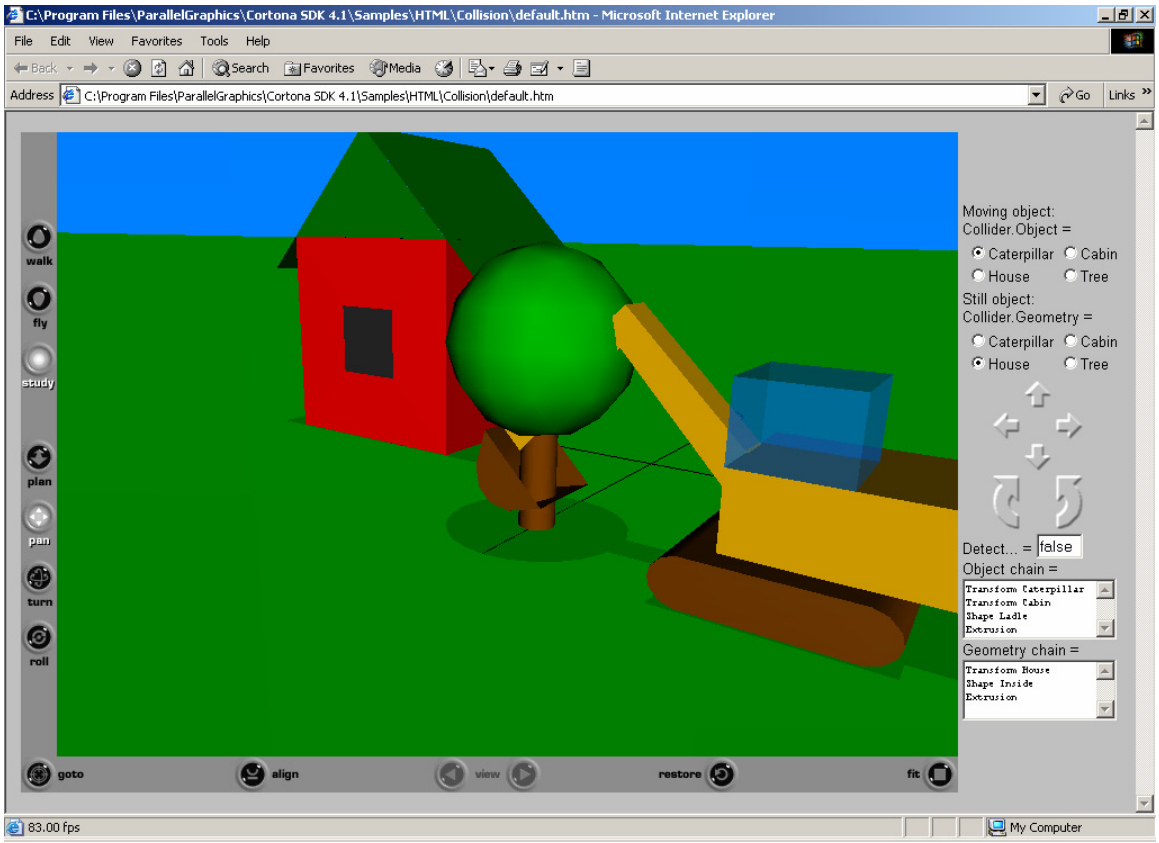


Abb. 3: Fehlende Kollisionskontrolle führt zu unlogischen Situationen [Parallelgraphics, 2002. WS. Samples\HTML\Collision].

Inzwischen bieten moderne 3D Protokolle wie X3D, Open Inventor, OpenGL und Java3D integrierte Methoden zur Kollisionskontrolle (Abb. 4).

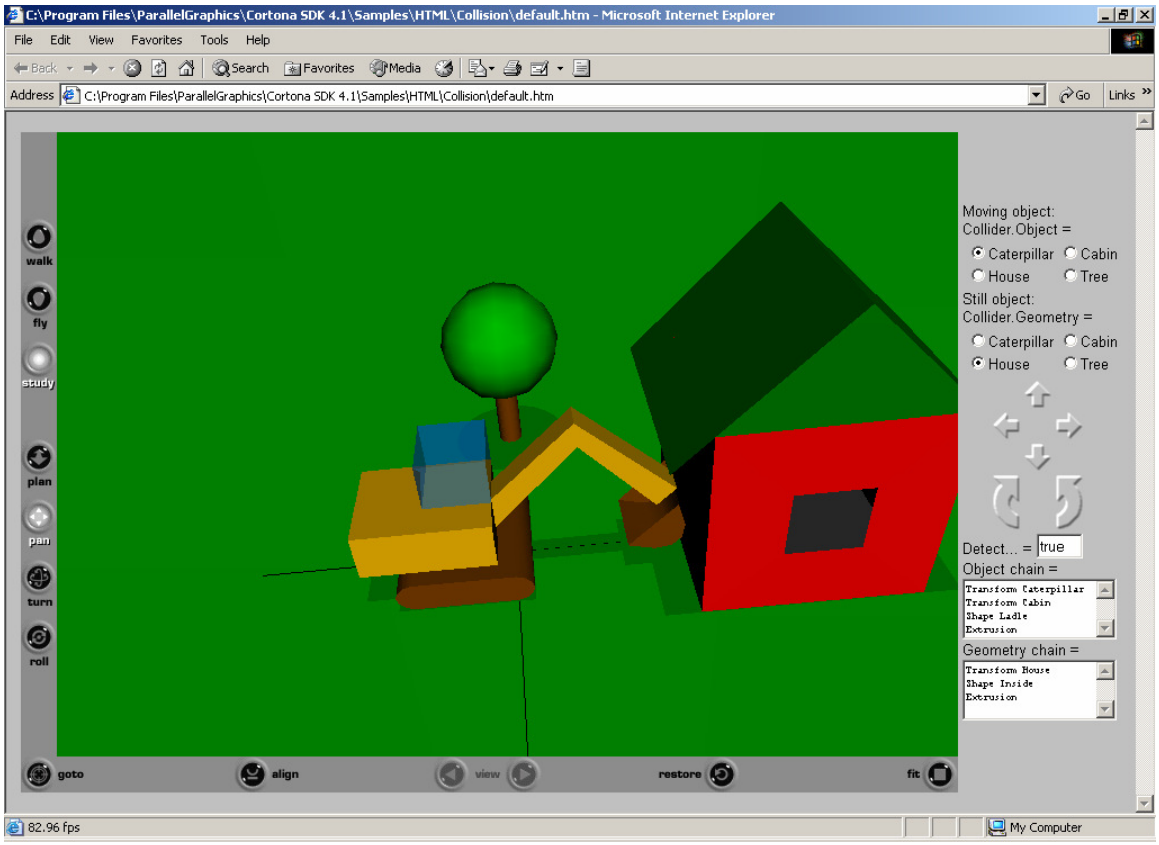


Abb. 4: Kollisionkontrolle erhöht den Realismus einer Darstellung [Parallelgraphics, 2002. WS. Samples\HTML\Collision]

Mit kollisionskontrolle ist die Einhaltung der Hierarchie der Objekte in einer 3D-Szene bezüglich Position und Integrität der Körperoberflächen gemeint. Es soll verhindert werden, dass Objekte durcheinander geschoben werden können, oder dass Personen (Avatare) oder Geräte z.B. durch Wände wandern können.

Manchmal ist eine Integration von Objekten jedoch erwünschenswert. Das tritt vor allem in Modellierungstools wie 3D CAD oder CosmoWorlds von Silicon Graphics auf. Auf diese Weise können Nahtstellen zwischen unregelmäßig geformten Körpern eruiert werden (Abb. 5).

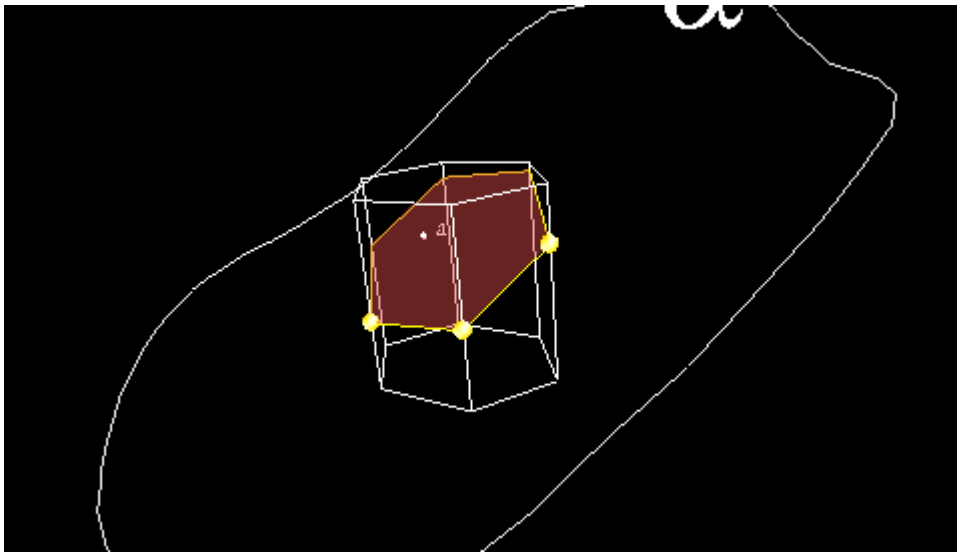


Abb. 5: Feststellung der Schnittstelle zweier Körper.

Bei der Kollisionskontrolle ist zu beachten, dass visuell ersichtliche Maße bei der starken Skalierung solcher 3D oder 4D Modelle nicht als Grundlage für Kollisionsentscheidungen gelten können. Eine eindeutige Aussage kann nur mittels einer Vektorberechnung erfolgen. Dazu reicht jedoch eine geometrische Darstellung nicht aus, sondern ein Tensor für das Objekt, also eine mathematische Größe wird benötigt. Eine Kollisionskontrolle mit dem bloßen Auge ist somit nur eine Annäherung, keine Entscheidungsgrundlage.

Zur Visualisierung geometrischer Formen von Objekten eignet sich die Internetfähige X3D (früher VRML) Technologie. Sie basiert auf der XML Syntax, ist plattformübergreifend, relativ einfach zu bedienen, steht der Öffentlichkeit zur Verfügung und reicht aus, um nahezu alle geometrischen Formen abzubilden. Abbildung 6 stellt eine vom Bauherrn erstellte Zeichnung eines Kavernenkomplexes dar. Diese Szene wurde in das VRML Format umgewandelt, leidet aber stark unter dem Hauptproblem des VRML Formats, dass Daten und Darstellung in einer Datei kombiniert sind. Dadurch werden unnötig viele Daten transportiert, mit der Folge verringerter Darstellungsleistung. So erklärt sich die Motivation zur Entwicklung des X3D Formats.

Als CAD ist die Szene allerdings wesentlich handlicher, da mit einer einfachen Konsole wie dem Cosmo Player die Navigation ermöglicht wird. Die Datenfülle erfordert ein Mindestmaß an Rechnerleistung zur Handhabung der Szene.

X3D ist leicht in einem dynamischen Objektmodell integrierbar, es existieren zahlreiche CAD-X3D Interfaces (d.h. Planer-Projektsteuerer Interfaces). Visuelle Kollisionskontrolle lässt sich mit dieser Technologie ermöglichen sowie externe mathematische Operationen auf Grund ihrer offenen Schnittstelle. Eine X3D Szene ist aus einer 3D CAD Zeichnung zu generieren. Moderne 3D CAD Applikationen wie Allplan 2005 von Nemetschek ermöglichen sogar einen COM-basierten Zugriff auf 3D CAD Objekte und deren Echtzeit Transformation nach VRML mittels einer weiteren Transformation nach X3D.

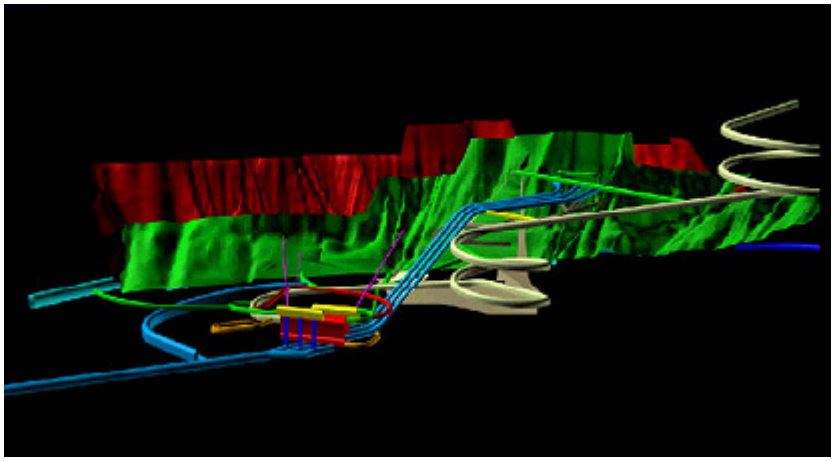


Abb. 6: VRML (VRML 97) Darstellung des Kavernenkomplexes Sawalkot, Kashmir [Unterlagen der Ausschreibung].

Indem sich die Geometrie der Objekte deren Definition (Klasse) unterordnet und daher lediglich ein Merkmal darstellt, wird Sie zu einer nicht mehr notwendigen Erfassungsbedingung für die Instanzierung eines Objektes.

Diese Abstrahierung von Bauwerkskörpern ermöglicht eine von der Geometrie unabhängige Datenerfassung.

3 Das phasenübergreifende Baumodell

Seit geraumer Zeit beschäftigt man sich mit dem Gedanken, eine größere Beeinflussbarkeit von Kosten und Zeit durch das Ineinandergreifen der Bauphasen zu erzielen. In den USA sind solche Modelle durch eine frühe Festlegung des Bauherrn auf eine ausführende Firma bereits in der Praxis etabliert. Die Ausführbarkeit der Planung wird in frühen Phasen geprüft. Hierdurch können komplizierte oder problematische Entwürfe frühzeitig optimiert und verändert werden. Dies führt zu einer höheren Kosten- und Terminsicherheit. Die Beeinflussbarkeit der Planung und damit der Kosten nimmt nach Erkenntnissen solcher Studien exponential mit der Phasenentwicklung ab. Der wirtschaftliche Druck in der heutigen Bauindustrie zwingt zu solchen Optimierungen.

Ein ähnliches System beschreibt Thabet [Thabet W., 2002. S. 1]. Dort kommt eine umfangreiche Bibliothek von 3D geometrischen Objekten zum Einsatz. Deren Zusammensetzung wird als Workflow dokumentiert; mit ihm werden Work Procedures automatisch erstellt. So können auch Konflikte bei der zeitlichen Reihenfolge untersucht werden.

In solchen Szenarien (Preconstruction-Phase, Machbarkeitsstudien) ist der Einsatz eines 4DCAD Systems durchaus sinnvoll, da die Umsetzung der Planung simuliert werden kann. Dabei können nicht nur geometrische Diskrepanzen, sondern auch geometrische Konflikte, die sich bei der zeitlichen Reihenfolge einer Montage ergeben, vermieden werden. Das Instrument würde sich auch bei der Prüfung von Work Procedures als brauchbar erweisen. Der Modellierungsaufwand ist jedoch in der Regel hoch.

III Modellierung der Geometrie

X3D (ein W3C Standard) gilt als Nachfolger des für das Internet entwickelten VRML Formates zur Beschreibung dreidimensionaler Körper und zur Navigation in einer solchen dreidimensionalen Welt (Abb. 7). Das ursprüngliche VRML 2.0 Format (Abb. 8) entstand aus den sehr erfolgreichen Arbeiten der Silicon Graphics Gruppe (Open GL). Trotz anfänglicher Begeisterung für dieses Format konnte es sich wegen der ursprünglich niedrigen Bandbreite des Internets nicht durchsetzen, da eine enorme Vergrößerung der VRML Objekte bei komplexen Formen und Welten entstand. Ein weiterer Grund für die Leistungsprobleme von VRML Welten lag in der Verquickung von Inhalt (Daten) und Form (Darstellung), wie es bei HTML Seiten üblich ist. Um diese Redundanz bei HTML zu eliminieren wurde die Entwicklung der XML/XSLT Semantik eingeführt. Bei modernen Webinhalten werden Inhalt (XML) und Form (XSLT) getrennt, das darstellende Medium (z.B. Browser) sorgt für deren Verquickung. Ähnlich verhält es sich mit X3D. Die Daten werden im XML Format transportiert (Abb. 9, 10 und 11) und die Darstellung wird dem Browser überlassen (Abb. 12). Allerdings gibt es z.Z. wenige X3D-fähige Browser.

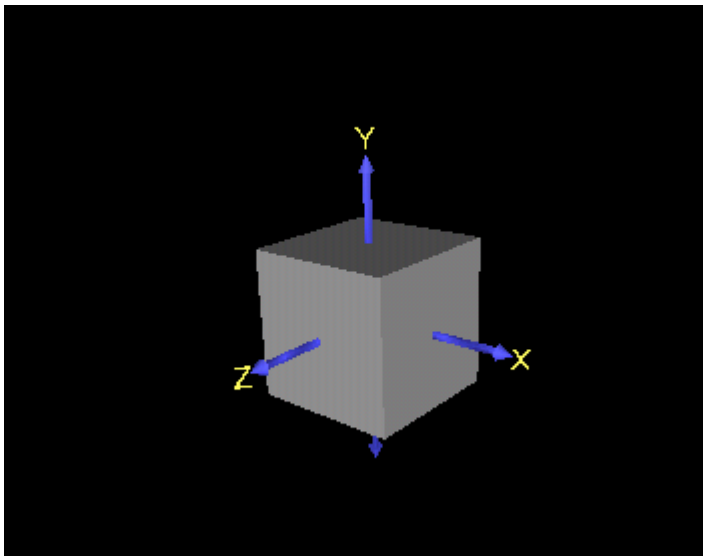


Abb. 7: Einfaches geometrisches Objekt in VRML bzw. X3D Format: Ein Quader [Kass, 2001. URL: http://www.web3D.org/TaskGroups/x3d/translation/ examples/ Conformance/Geometry/Box/default.x3d. WS].

```
#VRML V2.0 utf8
# X3D-to-VRML-97 XSL translation autogenerated by X3dToVrml97.xsl
# http://www.web3D.org/TaskGroups/x3d/translation/X3dToVrml97.xsl

# [X3D] VRML V3.0 utf8

# [head]
# [meta] image: default-front.jpg
# [meta] image: default-rear.jpg
# [meta] image: default-top.jpg
# [meta] image: default-bottom.jpg
# [meta] image: default-left.jpg
# [meta] image: default-right.jpg
# [meta] author: http://www.itl.nist.gov/div897/ctg/vrml/members.html
# [meta] filename: default.x3d
# [meta] disclaimer: This file was provided by the National Institute of Stan-
dards and Technology, and is part of the X3D Conformance Test Suite, available
at http://www.nist.gov/vrml.html The information contained within this file is
provided for use in establishing conformance to the ISO VRML97 Specification.
Conformance to this test does not imply recommendation or endorsement by the Na-
tional Institute of Standards and Technology. This software can be redistributed
and/or modified freely provided that any derivative works bear some notice that
they are derived from it, and any modified versions bear some notice that they
have been modified.
# [meta] translator: Michael Kass, NIST
# [meta] translated: 21 January 2001
# [meta] revised: 12 June 2002
# [meta] description: By default, the box is centered at the origin of the local
coordinate system, aligned with the coordinate axes, and measures 2 units in
each direction, from -1 to +1.
# [meta] url:
http://www.web3D.org/TaskGroups/x3d/translation/examples/Conformance/Geometry/Bo
x/default.x3d
# [meta] generator: Vrml97ToX3dNist, http://ovrt.nist.gov/v2_x3d.html
# [meta] generator: X3D-Edit,
http://www.web3D.org/TaskGroups/x3d/translation/README.X3D-Edit.html

# [Scene]

NavigationInfo {
  type [ "EXAMINE" "WALK" "FLY" "ANY" ]
}
Shape {
  appearance Appearance {
    material Material {
    }
  }
  geometry Box {
  }
}
```

Abb. 8: VRML Darstellung eines einfachen Quaders
[Kass, 2001. URL: http://www.web3D.org/TaskGroups/x3d/translation/ examples/ Conformance/Geometry/Box/default.x3d. WS].

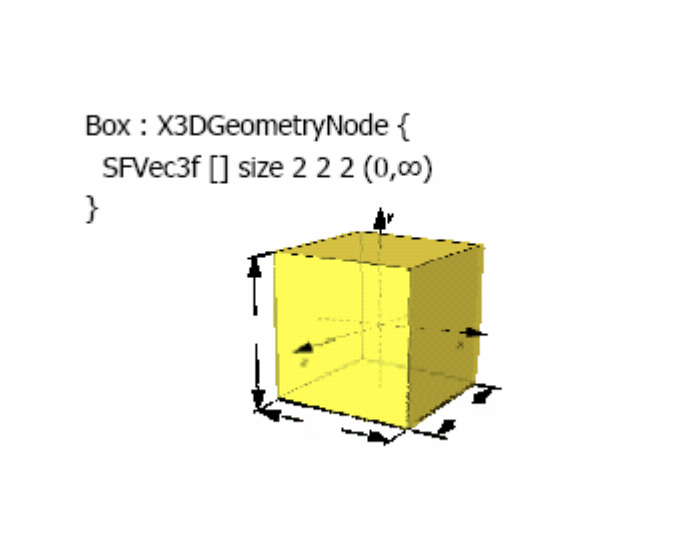


Abb. 9: X3D Darstellung eines einfachen Quaders [Corno, 2004. S. 5].

```
<X3D version="3.0">
- <head>
  <meta content="Figure03.01DefaultBox.x3d" name="filename" />
  <meta content="Figure 3.1, The VRML 2.0 Sourcebook, Copyright [1997] By
    Andrea L. Ames, David R. Nadeau, and John L. Moreland" name="author"
    />
  <meta content="Don Brutzman" name="translator" />
  <meta content="6 August 2000" name="revised" />
  <meta content="Default Box." name="description" />
  <meta con-
    tent="http://www.web3D.org/TaskGroups/x3d/translation/examples/Vrml2.0
    Sourcebook/Chapter03-Shapes/Figure03.01DefaultBox.x3d" name="url" />
  <meta content="X3D-Edit,
    http://www.web3D.org/TaskGroups/x3d/translation/README.X3D-Edit.html"
    name="generator" />
    </head>
- <Scene containerField="" class="">
  <NavigationInfo type="EXAMINE ANY" avatarSize="0.25 1.6 0.75" head-
    light="true" speed="1" visibilityLimit="0" bind="false" bindTime="-1"
    isBound="false" containerField="children" class="" />
- <Shape containerField="children" class="">
- <Appearance containerField="appearance" class="">
  <Material ambientIntensity="0.2" diffuseColor="0.8 0.8 0.8" emissive-
    Color="0 0 0" shininess="0.2" specularColor="0 0 0" transparency="0"
    containerField="material" class="" />
    </Appearance>
  <Box size="2 2 2" containerField="geometry" class="" />
    </Shape>
    </Scene>
    </X3D>
```

Abb. 10: X3D Syntax eines einfachen Quaders (Box)
[Brutzman, 2002. URL: http://www.web3D.org/TaskGroups/x3d/translation/examples/Vrml2.0Sourcebook/Chapter03-Shapes/Figure03.01DefaultBox.x3d. WS].

```
<X3D version="3.0">
- <head>
  <meta name="image" content="default-front.jpg" />
  <meta name="image" content="default-rear.jpg" />
  <meta name="image" content="default-top.jpg" />
  <meta name="image" content="default-bottom.jpg" />
  <meta name="image" content="default-left.jpg" />
  <meta name="image" content="default-right.jpg" />
  <meta content="http://www.itl.nist.gov/div897/ctg/vrml/members.html" na-
    me="author" />
  <meta content="default.x3d" name="filename" />
  <meta content="This file was provided by the National Institute of Standards
    and Technology, and is part of the X3D Conformance Test Suite, available at
    http://www.nist.gov/vrml.html The information contained within this file is
    provided for use in establishing conformance to the ISO VRML97 Specification.
    Conformance to this test does not imply recommendation or endorsement by the
    National Institute of Standards and Technology. This software can be redis-
    tributed and/or modified freely provided that any derivative works bear some
    notice that they are derived from it, and any modified versions bear some no-
    tice that they have been modified." name="disclaimer" />
  <meta content="Michael Kass, NIST" name="translator" />
  <meta content="21 January 2001" name="translated" />
  <meta content="12 June 2002" name="revised" />
  <meta content="By default, the box is centered at the origin of the local coor-
    dinate system, aligned with the coordinate axes, and measures 2 units in each
    direction, from -1 to +1." name="description" />
  <meta content="http://www.web3D.org/TaskGroups/x3d/translation/examples/
    Conformance/Geometry/Box/default.x3d" name="url" />
  <meta content="Vrml97ToX3dNist, http://ovrt.nist.gov/v2_x3d.html" na-
    me="generator" />
  <meta content="X3D-Edit, http://www.web3D.org/TaskGroups/x3d/translation/
    README.X3D-Edit.html" name="generator" />
    </head>
- <Scene containerField="" class="">
  <NavigationInfo type="EXAMINE WALK FLY ANY" avatarSize="0.25 1.6 0.75" head-
    light="true" speed="1" visibilityLimit="0" bind="false" bindTime="-1" is-
    Bound="false" containerField="children" class="" />
- <Shape containerField="children" class="">
- <Appearance containerField="appearance" class="">
  <Material ambientIntensity="0.2" diffuseColor="0.8 0.8 0.8" emissiveColor="0 0
    0" shininess="0.2" specularColor="0 0 0" transparency="0" container-
    Field="material" class="" />
    </Appearance>
  <Box size="2 2 2" containerField="geometry" class="" />
    </Shape>
    </Scene>
    </X3D>
```

Abb. 11: X3D Syntax nach dem X3D Conformance Test Suite
[Kass, 2002. URL: http://www.web3D.org/TaskGroups/x3d/translation/examples/
Conformance/Geometry/Box/default.x3d. WS].

Ferner kann durch spezielle Anweisungen großer Einfluss auf das Verhalten des Viewers genommen werden, um u.a. Bandbreite zu sparen. Somit findet eine größere Vereinheitlichung statt, womit die Größen, der über das Internet zu transferierenden Dateien reduziert werden können. Im Zusammenspiel mit den aktuell höheren Bandbreiten führt das zu realistischeren und praktischeren Darstellungen. Dadurch hat sich die Beliebtheit dieser Technologie erhöht. Zielanwendungen sind: kommerzielle Darstellungen von Produkten, industrielle Anwendungen bei Projekten (Anlagen, Bauvorhaben, Maschinen), Geo-Darstellungen (GEO Informationssysteme), 3D Veranschaulichungen eher abstrakter Daten (Internet Traffic), Simulierungen oder auch Echtzeit Darstellungen dynamischer (menschlicher Interaktionen – Avatars, militärische Szenarien – logistische Abläufe) und interaktiver Situationen (Medizin/Chirurgie, Entertainment).

Die Semantik der X3D ist XML-konform. Daher kann bei der Bearbeitung von X3D Objekten herkömmliche XML Technik eingesetzt werden. Für die Serialisierung von X3D Objekten bietet sich die gleiche Technik an wie bei der dynamischen Vererbung an; die Klasse des Objektes wird mit Hilfe von einem W3C Schema Fragment beschrieben; die Instanzierung des Objektes (das XML-förmige X3D Fragment) wird als Realisierung der Klasse gespeichert. Die Speicherung erfolgt in einer Vielzahl von Medien wie z.B. XML-Dateien, XML-konforme Datenbanken, objektorientierte Datenbanken oder auch relationale Datenbanken. Sehr gute Zugriffszeiten können erreicht werden, wenn solche Objekte im flüchtigen Speicher (RAM) abgelegt werden. Wichtig ist die Serialisierung der Geometrie zusammen mit der funktionalen Beschreibung.

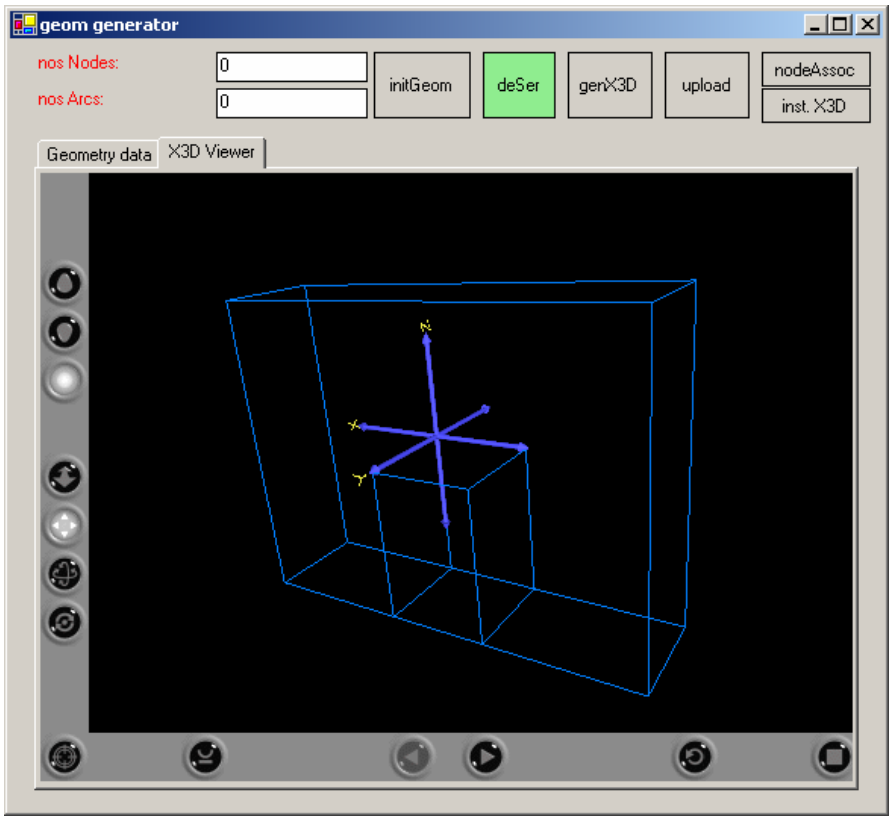


Abb. 12: Wand mit Türöffnung, eine mit X3D beschriebene Objektwelt des conObj Modells.

IV Objektorientierte Programmierung

1.1 Randbedingungen

Eine programmatische Umsetzung findet immer mit Hilfe der technologischen Möglichkeiten der Zeit statt. Zurzeit findet eine Ausrichtung zum VM (Virtual Machine) Konzept statt. Eigentlich war Visual Basic der Anfang dieser Ausrichtung, wurde jedoch in professionellen Kreisen nicht besonders mit Respekt bedacht. Es war eine für den Massenmarkt gedachte Sprache. Eine VM kapselt die für die Umsetzung nötigen Direktiven von den für die Kompilierung erforderlichen Anweisungen ab. Bei der bisherigen VM für Visual Basic (bis Version 6) äußerte sich dies darin, dass u.a. eine strikte Typisierung (strict typing) nicht erforderlich war. Bei der JVM (Java Virtual Machine) und der CLR (.NET Common Language Runtime) werden die Speicherverwaltung (Heap und Stack) und die Ausnahmebehandlung durch die Frameworks administriert [Richter, J., 2002. S. 453]. Dies befreit den Programmierer von Direktiven, die eigentlich mit der Administration des Betriebssystems und nicht der Ausführung logischer Sequenzen zu tun haben.

Typisierung wurde allerdings bei Java und .NET strikt und konsequent umgesetzt. Bei .NET liegt das zum Teil daran, dass ein Zusammenspiel von vielen Sprachen erreicht werden soll, was nur mit einer konsequenten Typisierung ermöglicht werden kann. Die starke Objektorientierung moderner VM Sprachen kommt dem Konzept der objektorientierten[3]-Modellierung sehr entgegen.

Der verstärkte Einsatz von XML bietet eine hervorragende und plattform-neutrale Schemasprache für die Erfassung und dynamische Verwaltung von Objekten und Klassensignaturen. Leider ist die XML–basierte, deklarative Sprache XPath bislang nicht als eigenständige Sprache geeignet. Daher sind alle Implementierungen der Klassen noch mittels einer imperativen Sprache wie C#, C++ oder Java durchzuführen.

In Anhang A.I wird eine kurze Einführung in die objektorientierte Programmierung aufgeführt. Zusätzlich werden fortgeschrittene Technologien wie .NET Remoting, SOAP, Multi-Threading und Serialisierung/DeSerialisierung beschrieben, welche für die Entwicklung des Objektmodells wichtig sind. Das Java Pendant JavaSpaces wird ebenfalls kurz erklärt, um zu unterstreichen, dass das hier aufgeführte Objektmodell auch mit der Java SDK entwickelt werden könnte und nicht auf .NET bezogen ist. Alle wichtigen Technologien der .NET Framework, die hier zum Einsatz kamen, finden sich, wenn auch in leicht unterschiedlicher Form, in der Java SDK wieder.

1.2 Das Objektmodell im verteilten Einsatz

Zur Veranschaulichung wird die einfache Stütze, beschrieben in Anhang A.I.1.2.3, in einem verteilten Einsatz untersucht. Die Stütze weist Dimensionen und Materialeigenschaften auf. Eine typische Situation sieht wie folgt aus:

- Ein Konstrukteur dimensioniert die Stütze als Teil des Tragwerks.
- Ein Ingenieur entscheidet über die erforderliche Materialgüte.

---

[3] „Objektorientiert“ – beschreibt eine Vorgehensweise in der Programmierung, bei der die Kapselung der Attribute und Methoden einer logischen Einheit entweder ein reales (z.B. Mensch) oder ein abstraktes (z.B. Ereignis) Objekt beschreiben. Somit werden keine Daten durch Funktionen geschleust, wie es bei einem funktionalen Ansatz der Fall wäre, sondern die Daten werden in logische Einheiten gekapselt und die Methoden, die für diese Einheiten in Frage kommen, werden gleich dort angesiedelt. Klassen beschreiben das Konstrukt dieser logischen Kapseln.

Diese Personen sitzen räumlich getrennt, deren Applikationen sind unterschiedlicher Natur. Der Konstrukteur arbeitet mit einem CAD System, der Statiker verwendet ein FE-Simulationsprogramm. Beide Applikationen bedienen sich des Objektmodells als Speicher. Der Konstrukteur wird die Eigenschaften

```
<height/>
<width/>
<length/>
```

bearbeiten und deren Werte auf jeweils 3.9, 0.2 und 0.3 setzen.
Der Statiker bemüht die Eigenschaft

```
<material>
```

und setzt dessen Wert nach Abschluss seiner Untersuchungen auf „B40“.
Wie funktioniert der gemeinsame Zugriff auf das Objekt *Stütze*?
Das Objekt *Stütze* wird im Objektmodell unter der IP-Adresse 10.3.10.44 gespeichert. Der Statiker befindet sich unter der IP-Adresse 10.50.90.56. Der Konstrukteur unter der IP-Adresse 10.2.5.45. Den Lokationen entsprechen drei verschiedene Standorte.

Die Methoden zur Modifikation der Eigenschaften, Dimensionen und Materialbezeichung sind trivial, werden im Objekt gekapselt und lauten:

```
public bool setDimensions(double height, double width, double length)
{
      this.height = (height>=0)?height:0;
      this.width = (width >=0)?width:0;
      this.length = (length >=0)?length:0;
      return true;
}
public bool setMaterial(string m)
{
      this.material = m;
      return true;
}
```

Der Konstrukteur wird sich der ersten Methode bedienen, der Statiker der zweiten. Die Aktualisierungsvorgänge können in zwei Formen erfolgen:

- Das Objekt wird serialisiert, mit dem SOAP Protokoll über die Leitung zur Verfügung gestellt, lokal anhand der Klasse wieder instanziert und verändert. Danach wird das Objekt wieder mit Hilfe von SOAP serialisiert, über die Leitung verschickt und auf der IP-Adresse 10.3.10.44 deserialisiert und im Objektmodell abgelegt.
- Die Methoden werden als Fernaufrufe (Remote Procedure Calls) mit dem SOAP Protokoll ausgeführt.

Als Anwendungen agieren Konsolen. Der Dienst, hier *remStuetze*, ruft die Klasse *stuetze* auf, instanziert das gewünschte Objekt und setzt die Eigenschaften. Dazu benutzt er die `serialize()` und `deSerialize()` Methoden der Klasse

`SoapFormatter`

im Namespace

`System.Runtime.Serialization.Formatters.Soap`

und setzt die Objektinformation wie folgt:

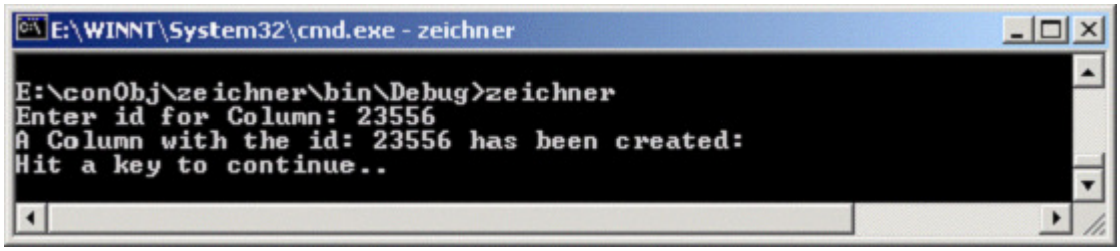
```
[WebMethod]
public int setDimensions(int id, double height, double width, double
length)
{
            stuetze obj = new stuetze(id);
            obj = obj.deSerialize(id);
            obj.height = height;
            obj.width  = width;
            obj.length = length;
            obj.serialize();
            return id;
}

[WebMethod]
public int setMaterial(int id, string material)
{
            stuetze obj = new stuetze(id);
            obj = obj.deSerialize(id);
            obj.material = material;
            obj.serialize();
            return id;
}
```

Die Konsolenapplikation *Zeichner*, gedacht als Applikation zur Dimensionierung eines Objekts, nutzt den WebDienst *remColumn*. Dieser Dienst stellt vier Methoden zur Verfügung:

- `createColumn` – Zeichner erzeugt ein Objekt (mit CAD)
- `setDimensions` – Objekt wird dimensioniert
- `setMaterial` – Statiker setzt Materialgüte
- `attributes` – Projektleiter informiert sich

Der Zeichner instanziert das Objekt mit der Methode createColumn:

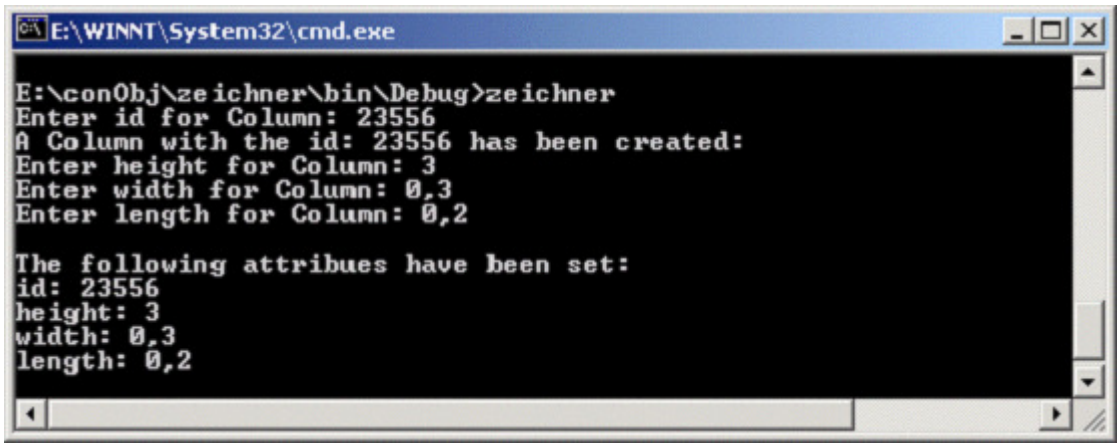


Das Ergebnis in der serialisierten Form des Objektes *rem* auf dem fernen Server ist wie folgt:

```
<SOAP-ENV:Envelope xmlns:xsi="...">
      <SOAP-ENV:Body>
            <a1:stuetze id="ref-1" xmlns:a1="...">
                  <id>23556</id>
                  <height>0</height>
                  <width>0</width>
                  <length>0</length>
                  <material xsi:null="1" />
            </a1:stuetze>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Bis jetzt existiert das Objekt; alle Eigenschaften, außer der des Materialls, sind noch nicht gesetzt worden. Material hat den Wert `null`.

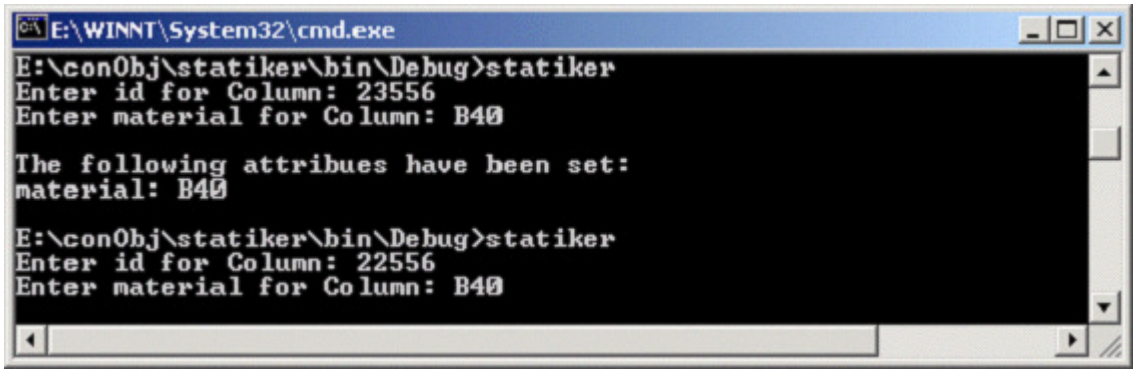
Der Zeichner dimensioniert jetzt das Objekt:



Die serialisierte Version des Objektes beinhaltet jetzt die Dimensionierung, die Eigenschaft Material jedoch bleibt auf `null` gesetzt:

```
<SOAP-ENV:Envelope xmlns:xsi="...">
      <SOAP-ENV:Body>
            <a1:stuetze id="ref-1" xmlns:a1="...">
                  <id>22556</id>
                  <height>3</height>
                  <width>0.3</width>
                  <length>0.2</length>
                  <material xsi:null="1" />
            </a1:stuetze>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

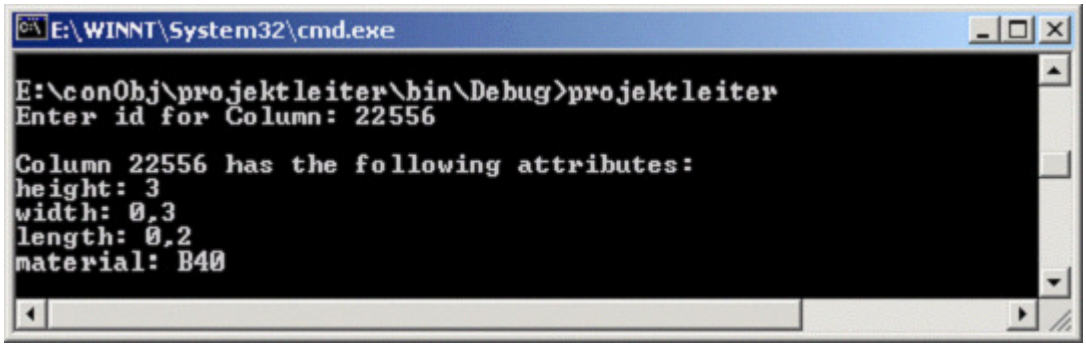
Der Statiker setzt die Materialeigenschaft der Stütze, um seinen berechneten Lasten zu genügen:



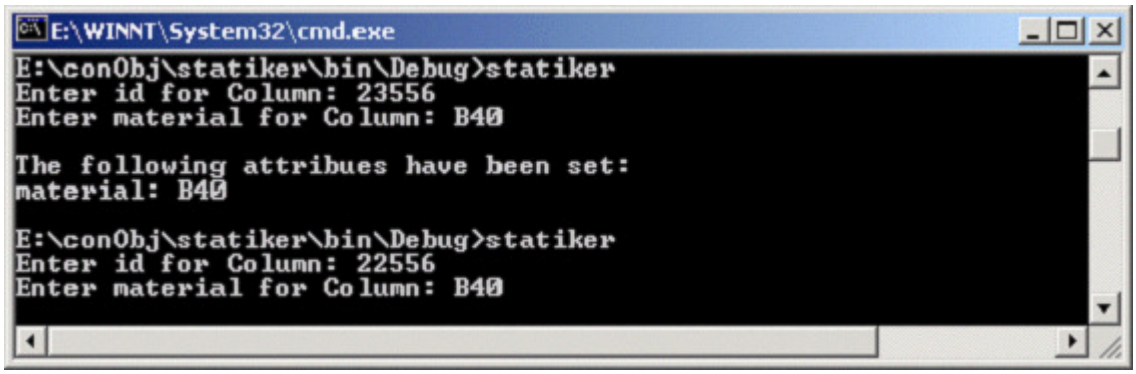
Die serialisierte Version des Objektes auf dem fernen Server ist wie folgt:

```
<SOAP-ENV:Envelope ...">
      <SOAP-ENV:Body>
            <a1:stuetze id="ref-1" xmlns:a1="...">
                  <id>22556</id>
                  <height>3</height>
                  <width>0.3</width>
                  <length>0.2</length>
                  <material id="ref-3">B40</material>
            </a1:stuetze>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Damit ist unsere einfache Stütze komplett. Der Projektleiter möchte gerne wissen, wie die Stütze aussieht:



Der Projektleiter ist zufrieden, der Architekt findet jedoch die unsymmetrische Form der Stütze nicht ansprechend und ändert deren Dimensionen auf Höhe 3, Breite 0,2 und Länge 0,2. Er teilt diese Entscheidung dem Bauleiter mit, der die Änderung anbringt (indem er die Stütze abreißt und eine neue aufbaut) und diese Informationen dem Zeichner als in-situ Zeichnungsinformationen und dem Vertragsmanager als Veränderung weitergibt. Der Zeichner fragt bei dem Statiker, der eine Materialänderung auf B45 empfiehlt, noch mal nach, da die vorletzte Spezifikation wenig Spielraum vorsah und teilt diese dem Vertragsmanager mit:



Die Versuchung läge jetzt nahe, die neuen Dimensionen und Materialangaben in das entfernte Objekt einzutragen und eine Veränderung zu melden. Dieser Nachtrag beruhte allerdings auf realen Informationen, die nicht mehr verifizierbar wären, da ein Eintrag der neuen Eigenschaftswerte den Stand des Objektes vor den Änderungen unkenntlich macht. Die vorherige Situation wäre im Nachhinein im System nicht mehr auffindbar.

Hier greifen die Mechanismen des Klonens von Objekten und bei einer Erweiterung der Klasseninformation die dynamische Vererbung.

V Virtual Reality Technologien

Virtuelle Technologien umfassen ein großes Spektrum an visuellen und telerobotischen Verfahren. Deren – nicht immer unumstrittener – Einsatz in der Praxis wird ein hoher Stellenwert beigemessen, da sie als Verstärker der Sinneswahrnehmungen einer Simulation dienen und somit deren Wert steigern. Im Wesentlichen geht es darum, die menschliche Interaktion mit virtuellen Gebilden (Bauprojekte, Körper, Gefechtsszenen, Produkte usw.) anzureichern, indem nicht nur der Verstand, sondern auch Sinnesorgane wie die Optik und das Gefühl (im Sinne Kraft und Verschiebung) angesprochen werden [Bar-Cohen,Y. et al., 2001. S. 1-2]. Dahinter verbirgt sich die Erkenntnis, dass bei einem körperlich gesunden Menschen bis zu 70 Prozent der Informationsaufnahme visuell übermittelt wird [Geiger, C., 2004. S. 97]. Virtual Reality wird von Forschern wie folgt definiert:

> „*... a high-end user interface that involves real-time simulation and interactions through multiple sensorial channels. These sensorial modalities are visual, auditory, tactile, smell, taste, etc.* “ [Burdea, 1993 cit. aus von der Heyde, M., 2001. S. 1]

Einige kombinative VR-Technologien wie *virtual immersion* und der bloße Einsatz von Erfassungsgeräten werden jedoch nicht als VR angesehen [von der Heyde, M., 2001. S. 2]. Dazu gehören auch die modernen animierten Filme wie „Titanic“„Find Nemo“ oder „iRobot“. Diese werden mit aufwendigen Simulationen und Computergrafiken erzeugt, gelten jedoch als Teil der Computergrafik Disziplin und nicht als VR-Applikationen [Vince, J., 1998. S. 3].

Virtual Reality Technologien werden in verschiedenen Szenarien eingesetzt. Sie dienen zur Steuerung von Robotern, zur Visualisierung eines Körpers (Baukörper, Mensch, Produkt), zur Simulation eines Vorgangs oder einer Interaktion mehrerer Entitäten (Verhaltenstherapie), zur Simulation eines Prozesses (Fertigungsanlage, Bauprozess), zur Einübung militärischer Prozesse wie Kampfhandlungen, oder zur Echtzeit Visualisierung einer Vielzahl von elektronischen Informationen im komplexen Zusammenhang (Unterwasservisualisierung, Neurochirurgie). Hierbei ist allerdings zu beachten, dass oft technische (z.B. 3D-Aerodynamik) oder organisatorische Erfordernisse (Interaktion zwischen Einheiten) die Triebkraft für die räumliche Darstellung von Objekten war. Der Close Combat Tactical Trainer (CCTT) der US Army stellt einen der größeren net-VE's dar. Der CCTT bildet das Herzstück des Combined Arms Tactical Trainer (CATT) Programs. [STRICOM via Singhal, S. and Zyda, M., 1999. S. 1].

Die Erfordernisse der US Militärs im Bereich Simulation haben der VR Technologie von Schlüssellieferanten wie IBM oder Silicon Graphics einen erheblichen technologischen Schub gewährt. Weiterhin ist Personal der US Navy im Web3D Konsortium vertreten. Laut Medienberichten liegt der Wehranteil an der Finanzierung der Robotik in den USA bei 80 Prozent.

Die einzelnen Themenbereiche werden kurz angerissen, um einen für die Projektsimulation relevanten Überblick zu ermöglichen.

Ein interessantes Nebenthema ist der Einfluss des eBusiness auf die Bauindustrie. In Building Design and Construction wird festgestellt, dass die Zunahme im Bereich eBusiness zu einer Abnahme von Einkaufszentren und Gewerbeflächen führt. Dies wird durch die Straffung der Just In Time (JIT) Lieferungsverfahren der verarbeitenden Industrie mit dem Einsatz von Backend eBusiness und Supply Chain Management Systemen verstärkt. Ob es eine ähnliche Erfahrung im Hinblick auf traditionelle Planungs- und Bauprozesse bei dem verstärkten Einsatz von VR-Technologien geben wird, ist abzuwarten. Versuche einer Mechanisierung des Bauwesens in den 70er und 80er Jahren durch den vermehrten Einsatz von Robotern sind häufig gescheitert, was an der komplexen Natur moderner Bauobjekte lag.

Der Einsatz von verteilten Systemen, künstlicher Intelligenz und Pattern Recognition ermöglicht jedoch die Überwindung dieser Barrieren, und mit dem Zusatz der VR-Technologien als Schnittstelle zwischen Mensch und Maschine wären komplexe Vorgänge nun durchführbar. Eine Verringerung der eher traditionell aufwändigen Planungs- und Bauprozesse durch den Einsatz von VR-Technologien, ist möglich und aus Sicht der Produktivität nötig, analog der Abnahme von kommerziellen Nutzflächen durch eine Zunahme in eBusiness.

Im Anhang A.II werden Einzelheiten zur Robotik, Aspekte einer net.VE Umgebung und Simulationsverfahren aufgeführt. Im Anhang A.II.2 werden CAD-basierte Technologien erläutert. In Teil C werden einige Aspekte der net.VE Umgebung, die zur Optimierung der Leistung des hier entwickelten Objektmodells eingesetzt werden können, besprochen.

VI Datenhaltung

1.1 Objektorientierte Datenhaltung

Objekte können u.a. in einer relationalen Datenbank als serialisierte Binärströme abgelegt werden. Diese serialisierten Objekte werden auf der Objektmodell-Ebene sinnvoll verwertet und dann über einen Dienst dem Client zur Verfügung gestellt.

Die Business-Logik liegt auf dem Server und verwaltet die Beziehungen zwischen den Objekten. Die Objekte werden mit den im Objekt gegenwärtigen Methoden *serialize()* und *deSerialize()* in einem Bytestream umgewandelt und dann mit einem Dienst an eine Datenbankprozedur übergeben. Die Datenbank beinhaltet keinerlei Relationen. Diese sind innerhalb der Objekte gekapselt und werden von der Business-Logik aufgearbeitet und genutzt.

Der Vorteil dieser Methodik beruht auf der großen Unabhängigkeit des Objektmodells von den Datenbankstrukturen. Daher ist ein Umstieg auf eine andere Datenbank, z.B. Oracle, ohne große Probleme möglich, es handelt sich lediglich um Tabellen und gespeicherte Prozeduren (Stored Procedures). Ein weiterer Grund ist die Möglichkeit der dynamischen Erweiterung der Meta-Ebene, oder Klassenebene, ohne Einsatz datenbankspezifischer Sprachen, wie z.B. Cache ObjectScript. In dieser Arbeit wurde eine solche datenbankunabhängige Vorgehensweise gewählt.

1.2 Objektorientierte Datenbank

Objektorientierte Datenbanken bieten hohe Leistungen, lohnen sich aber nur, wenn die Business-Logik auch objektorientiert gestaltet wurde. Objektorientierte Datenbanken setzen meistens auch den Einsatz proprietärer Sprachen wie Cache ObjectScript voraus. Eine Integration des Objektmodells in die Datenbank ist nötig, um die hohe Leistungsfähigkeit dieser Datenbanken zu nutzen. Java oder .NET verfolgen eine Einfachvererbung, objektorientierte Datenbanken verfolgen oft eine Mehrfachvererbung aufgrund der Nähe zu C++ oder C. Dieser Unterschied ist bei einer Implementierung zu beachten. Die Leistungssteigerung gegenüber herkömmlich relationalen Datenbanken liegt bei Faktor 20 oder mehr.

Ein gutes Beispiel einer objektorientierten Datenbank ist Cache von der Firma Intersystems. In der Version 5.0 dieser Datenbank wurde ein SOAP basierter WebServices Handler entsprechend Abbildung 13 eingebaut:

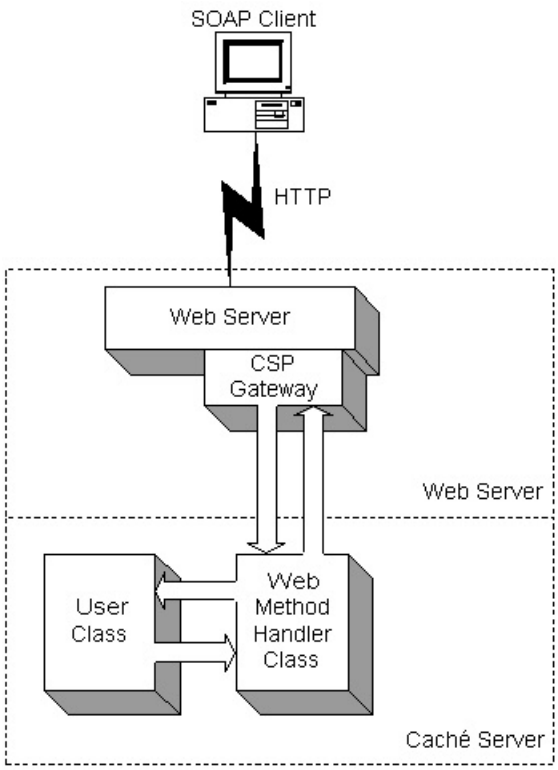


Abb. 13: SOAP Message Handling in Intersystems Cache WebServices Engine
[Cache Documentation Intersystems Corp., 2004. WS]

1.3 Objekt-Relationale Datenhaltung

Der stetig wachsende Einfluss der objektorientierten Programmierung hat auch zunehmend Einfluss auf existierende Speichermechanismen ausgeübt. Diese sind jedoch immer noch überwiegend relationaler Art. Auch komplexe Anwendungen wie SAP basieren auf relationale Datenbanken.

Daher kommt bei der objektorientierten Programmierung oft der Umstand vor, dass Objektmodelle (Business Layers) zur Speicherung zerlegt und einer relationalen Datenquelle zugeordnet werden müssen. Dieser Prozess ist aufwendig und fehlerträchtig. Hinzu kommt, dass eine Änderung im Objektmodell auch einer konsequenten Umsetzung in allen relationalen Strukturen wie Tabellen, Beziehungen und Abfragen bedarf. Dieser Umstand wird aus mehreren Gründen in Kauf genommen:

- Relationale Strukturen sind leicht lesbar, die Objektattribute kommen als Tabellenspalten vor; einzelne Objekte als Tabellenzeilen oder verknüpfte Tabellenzeilen.
- Relationale Strukturen mit einem hohen Normalisierungsgrad sind immer noch sehr leistungsfähig und skalierbar.
- Der Atomisierungsgrad relationaler Tabellen ist das Objekt-Attribut. Der Atomisierungsgrad einer Objektdatenbank ist unter Umständen das Hauptobjekt einer Anwendung, z.B. bei der objektorientierten CAD-Anwendung, der ganze zur Bearbeitung stehende Motorblock. Daher sind relationale Strukturen mit weit weniger Aufwand Multiuser-fähig.
- Relationale Datenbanken sind einfacher zu programmieren.
- Die relationale Abfragesprache SQL ist ISO standardisiert, weit verbreitet, einfach zu erlernen und sehr leistungsfähig. In der aktuellen Version sieht die SQL Spezifikation auch objektorientierte Szenarien vor. Allerdings wird diese Spezifikation nur teilweise umgesetzt.

Daher haben kommerzielle Entwickler relationaler Datenbanken wie Oracle, Microsoft und IBM die Fusion der relationalen und objektorientierten Ansätze angestrebt, mit sog. objektrelationalen Datenbanken. Der Grad der ernsthaften Parallelisierung beider Welten ist in den Produkten dieser Hersteller unterschiedlich ausgeprägt. Meistens handelt es sich um eine relationale Struktur mit einer objektorientierten Darstellung.

Inzwischen gibt es auch ernsthafte Versuche dieser Art in der Objektebene oberhalb der relationalen Datenbank. Dort findet die Zuordnung zwischen der objektorientierten Ebene und der darunter liegenden relationalen Struktur statt. Ein solcher Entwurf ist die ObjektSpaces Namespace der .NET Release 2.0 (in Alpha), die leider aus technischen Gründen nicht in der Release Version implementiert wurde.

In der Release-Version werden einige Klassen zur Verfügung gestellt, womit Objekte den jeweiligen Tabellen mittels XML-Schema Dateien zur Verfügung gestellt werden und auch Methoden zur Persistierung, Abfrage und Versionskontrolle der Objekte zugänglich sind. Der Vorteil dieser Methodik liegt darin, dass die darunterliegende relationale Struktur unverändert bleibt. Erste Performance Ergebnisse sind positiv. Die Objektserialisierung im Vergleich mit der herkömmlichen Serialisierung der binären Form des Objektes zeigt deutlich bessere Werte. Hinzu kommen die oben beschriebenen Vorteile der klaren Darstellung des Objektes in tabellarischer Form sowie die Möglichkeit, SQL Anweisungen in der relationalen Struktur auszuführen. Die Vorgehensweise der Objektserialisierung bzw. -deserialisierung setzt folgendes voraus:

- Die Objektattribute werden als Eigenschaften ausgewiesen.
- Es existiert ein **MappingSchema** (System.Data.MappingSchema – verknüpft die RSD- und OSD-Dateien, beschreibt die Verbindung zwischen der Klasse und der Datenquelle) des Objektes bestehend aus einer MappingSchema Datei (XML-Format], einer RSD-Datei [beschreibt die Tabellen, Spalten und Einschränkungen in der Datenquelle) und einer OSD-Datei (beschreibt die persistenten Eigenschaften der zu speichernden Klasse).
- Die Datenbank weist die in der RSD aufgeführten Datenstrukturen auf.

In Anhang A.III.1.3 wird am Beispiel einer Währungsrelation als Baustein eines objektorientierten Kalkulationsprogramms der Aufbau einer objekt-relationalen Objektpersistenz untersucht.

In Abbildung 14 wird ein Leistungsvergleich der verschiedenen Serialisierungsmethoden zu einem Objekt der Klasse `currency` aufgeführt. Dabei ist zu beachten, dass eine relationale Abfrage immer in der objekt-relationalen Zuordnung (Mapping) einen Leistungsverlust verursacht, da dort alle Attribute und Eigenschaften durchiteriert und aus dem relationalen Bestand gesetzt werden müssen.

Demgegenüber stehen bei einem objektorientierten Ansatz, soweit keine Mapping Klassen eingesetzt werden, die Objekte sofort nach der Abfrage zur Verfügung.

	Binär[4]	ObjectSpaces[5]	Relational[6]	Cache:SOAP[7]
Initialisation	3485	5000	2500	5000–9000
Insert/Update	650	50–130	10–20	20–80
Select	700	50–110	10–20	20–30

Abb. 14: Vergleich der Serialisierungsmechanismen für die Klasse currency. Die Versuche wurden auf einem IBM T21 ThinkPad, mit Windows2000 Server als Betriebssystem ausgeführt. Die .NET Version 2.0 (in Alpha) wurde für das Objektmodell eingesetzt. Intersystems Cache 5.0 kam als Objektdatenbank zum Einsatz. SQL2000 diente sowohl als relationale-, objektrelationale- und binäre Datenbank.

---

[4] Objekte können in der .NET Framework mit Klassen in der Namespace `System.Runtime.Serialisation.Formatters.Binary` als Binärstrom dargestellt und in einer Tabelle als BLOB gespeichert werden. Die Performance dieser handlichen Methode lässt sich leider schlecht skalieren, was bei größeren Objekten hilfreich wäre.

[5] Die Namespace `System.Data.ObjectSpaces` bietet sämtliche Klassen zum Einsatz dieser neuen Technologie. ObjectSpaces handhabt die Zuordnung zwischen relationalem und dem Objektmodell auf der Ebene des Objektmodells. Dadurch können auch ältere Versionen des SQL Server zum Einsatz kommen.

[6] Verbindung zur relationalen Datenbank SQL2000 mittels der Klasse genSpHandler, die wiederum auf dem `System.Data.OleDb` Namespace basiert. Die `System.Data.Sql` Namespace bietet eine Performancesteigerung um ein Sechsfaches (6x), jedoch unter Verlust der Parametermaskierung.

[7] Intersystems Cache bietet ab der Version 5.0 die Möglichkeit, Objekte und Objektmethoden mittels SOAP Protokoll anzusprechen. Die Anbindung an der .NET Framework erfolgt problemlos wie der Fall mit .NET Web Services. Durch die Vermittlung des Webservers ist jedoch mit Performanceverlusten zu rechnen.

VII Anforderungen an ein dynamisches Objektmodell

Ein Objektmodell muss kontemporäre Technologien berücksichtigen, um dessen Einsatz zu sichern. Die fachlichen Anforderungen ergeben sich aus der Problemstellung. Somit kann ein Satz an fachlichen und technischen Anforderungen an ein dynamisches Objektmodell aufgestellt werden.

1 Fachliche Anforderungen

- Die Integration diverser fachlicher Ausrichtungen wie z.B. Kalkulation, Planung, Terminplanung, Kostenkontrolle, Qualitätsmanagement findet immer auf Objektebene statt.
- Es wird angenommen, dass übergeordnete Schichten einer fachlichen Ausrichtung die Aggregation der Auswirkungen dieser Richtung auf den Objekten darstellt.
- Das Modell muss eine unbegrenzte Anzahl von Strukturen (DIN Normen, Gebäudemodelle, Phasenmodelle) ermöglichen. Die Objekte werden diesen Strukturen über dessen Knotenpunkte (nodes and leafs) zugeordnet, wo es fachlich sinnvoll ist. Dadurch ist das Objektmodell grundsätzlich phasenunabhängig. Die Knotenpunkte der Strukturen, die ebenfalls Objekte sind, können Methoden zur dynamischen Berechnung eines Phasenzustandes eines Objektes zugeordnet werden.
- Objekte sollten unabhängig vom Zustand ihrer Eigenschaften einsetzbar sein. Somit können auch Objekte ohne geometrische Form oder sonstige Eigenschaften sinnvoll eingesetzt werden.
- Existierende fachliche Applikationen für Kalkulation, Terminplanung, CAD usw. müssen weiterhin einsetzbar sein. Die Interaktion findet vorzugsweise über die API's statt, wenn nicht anders möglich über einen dateibasierten Datenaustausch.
- Die fachlich spezialisierte Anwendung ist ein fachliches Fenster auf diese Objektwelt.
- Die Interaktion eines fachlichen Sachbearbeiters mit dem Objektmodell ändert sich möglichst gar nicht oder nur minimal.

2 Technische Anforderungen

- Das Objektmodell muss alle wichtigen Aspekte einer modernen objektorientierten Programmumgebung widerspiegeln (Klassen, Vererbung, Attribute, Eigenschaften, Ereignisbehandlung (Delegates) auch innerhalb eines Objektes, Serialiserung/ deserialisierung, verteilte Technologien, dynamische Interaktion zwischen deklarativen (XML) und imperativen Sprachen).
- Das Objektmodell muss mit einem modernen Framework modelliert werden, um sicher zu stellen, dass es plattformunabhängig ist und an neuen IT-technischen Entwicklungen angepasst werden kann. Der Einsatz einer Virtual Machine bildet die Grundlage für die technische Umsetzung der Abstrahierung auf einer Meta-Ebene. Sie ermöglicht auch die Steuerung dieser Abstrahierung mit textbasierten Befehlen. Aus dieser Sicht kommt nur die Java Virtual Machine oder die .NET Framework in Frage.
- Das Objektmodell muss dynamisch (zur Laufzeit) erweiterbar sein.
- Grundsätzlich ist die Kommunikation mit anderen Teilnehmern mit verteilten Technologien wie XML Services oder .NET Remoting zu realisieren. Technologien die auf anderen Protokollen wie UDP aufsetzen sind auch einsetzbar, es ist jedoch darauf zu achten, dass die Konfiguration der Technologien programmatisch möglich sein muss.
- Die Modellierung sollte ausschließlich objektorientiert erfolgen, um wichtige Eigenschaften dieses Ansatzes, wie die Polymorphie, Vererbung, die Klassendefinition und die Kapselung nutzen zu können.
- Die Begrenzung des klassischen objektorientierten Ansatzes, die einer starren Klassendefinition, soll mit modernen Metatechnologien, die in Java wie auch der .NET Framework vorhanden sind, überwunden werden.
- Die Existenz von umfangreichen und professionell einsetzbaren API's kann vorausgesetzt werden. In nur wenigen Programmen sind solche API's vorhanden, das sollte jedoch keine Rolle bei der technischen Umsetzung spielen, da es ein erklärtes Ziel der überwiegenden Anzahl Softwarehersteller im Baubereich ist, dynamische Schnittstellen anzubieten.

- Das Objektmodell liest und schreibt Objektinformationen in die jeweiligen fachlichen Applikationen mittels derer API's. Somit findet die Transformation der Daten zwischen den Applikationen und dem Objektmodell auf Objektebene statt. Die Regeln dieser Transformation werden in der Konfiguration der dynamischen Anbindungen berücksichtigt.
- Die Konfiguration des Objektmodells sollte mittels W3C konformen Sprachen wie XML Schema erfolgen, um eine möglichst große Interoperabilität zu gewährleisten, z.B. mit den IFC Klassen.
- Aus obigem Grund ist das gewünschte Datenaustauschformat XML und nicht eine Binärform. Somit kann eine Weiterverwendung der Daten des Systems in anderen Applikationen gewährleistet werden.
- Die Ereignisbehandlung sollte entsprechend des objektorientierten Ansatzes möglichst auf Objektebene stattfinden. Dieses Ziel läuft zwar Leistungsüberlegungen zuwider, führt jedoch zu einer wartungsarmen und applikationsunabhängigen Lösung.
- Das Objektmodell muss grundsätzlich als API, unabhängig von einer Applikation entwickelt werden. Diese Trennung ist ein zentrales Charakteristikum des Modells.
- Die Datenhaltung sollte von dem Objektmodell getrennt werden. Somit ist theoretisch auch eine primitive Datenhaltung mit XML Dateien möglich. In der Praxis wird jedoch der Einsatz einer relationalen oder objektorientierten Datenbank vorausgesetzt.
- Objektveränderungen müssen kenntlich gemacht werden (entweder durch eine kumulative Historie innerhalb des Objektes oder durch einen Kopiermechanismus)
- Der Aufbau eines Regelwerks für die Steuerung der Objektveränderungen muss ermöglicht werden.
- Das Objektmodell darf grundsätzlich nicht an ein einzelnes Eingabemedium (Desktop, Web, Mobile) gebunden sein.
- Das Einbinden externer Applikationen, die einzelne Fachbereiche eines Projekts darstellen (z.B. Terminplanung, Konstruktion, Kalkulation) muss durch Unterstützung von verteilten Technologien ermöglicht werden.

Das Objektmodell wird in Teil B detailliert beschrieben. Im Wesentlichen werden die obrigen Anforderungen dadurch erfüllt, dass ein (zur Laufzeit) dynamischer Wechsel zwischen der Deklaration des Objektmodells (im XML Format) und der typsicheren Umsetzung (Binärformat mittels .NET Klassen) erreicht wird. Dieser Wechsel wird durch das Zusammenspiel der .Net Namespaces ISerializable und System.CodeDom ermöglicht.

Eine wesentliche Technologie ist die Serialisierung/deserialisierung von Objekten in Binärformat bzw. XML. Dadurch können Objekte beliebig zwischen einer deklarativen Sprache (XML) und einer imperativen Form (Binär) gewechselt werden. Das ermöglicht den Einsatz von typsicheren Methoden zur (zur Laufzeit) dynamischen Definition von Klassen und Instanzierung von Objekten.

B Neuer funktional-orientierter Ansatz der Projektmodellierung

I Voraussetzungen einer Lösung

1 Funktionale Objektorientierung

Eine weit verbreitete Schwäche der IT-Modellierung von Bauprojekten ist die Versionierung. Diese Schwäche wird wiederum durch die starke Markenorientierung der am Markt vorhandenen Lösungen verstärkt. RPC und Datenbankaufrufe sind umfangreich, die entsprechenden Protokolle sehr unterschiedlich und teilweise schwierig und deren Implementierung ressourcen-intensiv. Hinzu kommt, dass die meisten Lösungen keine offenen Schnittstellen unterstützen. Die Fülle neuerer Ankündigungen vermeintlicher XML Unterstützung basiert in Wirklichkeit meist auf der Tatsache, dass die überwiegende Anzahl der kommerziellen und Open Source Datenbanken XML-Schnittstellen zu den Daten liefern. Jedoch werden Regelwerke zur Datenverarbeitung in der Schicht oberhalb des Speichermediums implementiert. Ein XML-Auszug der Datenbank würde daher eine Sammlung von Daten ohne Zusammenhang liefern. Zwei Angelegenheiten bedürfen einer genaueren Betrachtung:

1. Ein übergreifendes Beschreibungsmodell, basierend auf einem offenen Standard (Public Standard), ist anzusetzen und so zu entwickeln, dass alle Teilnehmer ungehinderten Zugang haben und der Quellcode mit einfachen Mitteln lesbar und verständlich ist.
2. Versionierung ist wichtig, weshalb ein spezifischer Speicherort verwendet werden sollte. Alternativ können mehrere Implementierungen untereinander repliziert werden, um einen gemeinsamen Projektzugang zu gewähren.

2 Technische Vorraussetzungen

2.1 W3C XML Schema in .NET

XML Schema ist eine W3C Spezifikation, daher industrieübergreifend aktuell. Diese Technologie hat sich bereits durchgesetzt und wird verstärkt in hochkomplexen Systemen erfolgreich eingestellt, seien es XML-basierte-, relationale- oder objektorientierte Datenbanken.

Als deklarative Definitionsprache gedacht eignet sich XML für die Beschreibung von Objekten. Das ist ähnlich der Klassenbeschreibung. XML kann also als Ersatz für eine sprachgebundene Klassenbeschreibung dienen. Im Beispiel in 1.10.4 sieht eine Schema-Beschreibung des einfachen Objektes wie folgt aus:

```
[C#]
public class centralColumn{
   public double colHeight;
}
```

[Visual Basic]

```
Public Class centralColumn
   Public colHeight As Double
End Class
```

```
[XML Schema]
<?xml version="1.0" encoding="utf-16" ?>
<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="centralColumn">
            <xsd:complexType>
                  <xsd:sequence minOccurs="1" maxOccurs="1">
                        <xsd:element name=" colHeight" type="xsd:double" />
                  </xsd:sequence>
                  <xsd:attribute name="scope" type="xsd:string">
                        public
                  </xsd:attribute>
            </xsd:complexType>
      </xsd:element>
</xsd:schema>
```

Die Syntax ist zwar physikalisch umfangreicher als bei den jeweiligen imperativen Sprachen, gilt aber als Sprachen- und Plattform übergreifend. Es braucht keine Rücksicht auf die Syntax jeglicher zur Verfügung stehender Sprachen genommen zu werden, da in allen Sprachen eine Methode zur Behandlung einer XML Schema Datei etabliert werden kann.

2.2 .NET Objektserialisierung

Die .NET Framework bietet zusätzlich zur binären eine XML-basierte Serialisierung von Objekten in der using `System.Xml.Serialization` Namespace. Der Vorteil liegt in der Sprachen- und Plattformunabhängigkeit sowie der Möglichkeit, XML-basierte Technologien auch hier anzuwenden. Dadurch vergrößert sich die Anzahl der Möglichkeiten beim Umgang mit Objekten. Ein einfaches Beispiel der von .NET vorgesehenen XML Serialisierung ist eine Klasse mit nur einem public Attribut:

[C#]

```
public class centralColumn{
   public double colHeight;
}
```

Wenn eine Instanz dieser Klasse serialisiert wird, könnte es wie folgt aussehen:

```
<centralColumn>
   <colHeight>24.23</colHeight>
</centralColumn>
```

In diesem Fall hat das Attribut colHeight den Wert 24.23.

Diese in der .NET Framework vorgesehene Methodik ist sehr hilfreich und effizient. Eine Einschränkung, dass alle Klassen im Vorfeld bekannt sein müssen, was in dem hier vorgesehenen Objektmodell der dynamischen Vererbung und den damit zur Laufzeit generierten Klassen, nicht möglich ist. Daher eignet sich das .NET Verfahren nicht zur Serialisierung beliebiger Klassen.

3 Funktionale Beschreibung als Basis der Objektmodellierung

Ein Objekt im IT-technischen Sinne ist eine funktional abgeschlossene Informationseinheit und umfasst Attribute, Eigenschaften, Methoden und Ereignisbehandlungen. Die Funktion des Objektes entspricht einer fachlichen Vorstellung, sei es die Abbildung eines Benutzers, ein Vorgang in einem Terminplanungsprogramm, eine Position in einem Kalkulationsprogramm, eine Datenbankverbindung, ein Word Dokument, oder ein Bauobjekt.

Das Objekt beinhaltet alle für dessen Funktionalität benötigten Informationen (Attribute und Eigenschaften) und Vorgehensweisen (Methoden), um es als eine fachlich abgeschlossene Einheit zu betrachten. Interne und externe Änderungen werden mittels der Ereignisbehandlung kenntlich gemacht. Methoden zur Modifikation des Objekts oder anderer Objekte werden dann von den Ereignishandlern ausgeführt.

Die Objektmodellierung ist ein Teilgebiet der Informatik. Eine wichtige und hilfreiche Modellierungssprache ist UML (Unified Modelling Language). Mit ihrer Hilfe können anhand von Prozessbeschreibungen fachlich geschlossene funktionale Einheiten identifiziert werden, die wiederum als Objekte modelliert werden. Diese Methodik bietet den großen Vorteil, dass ein prozessübergreifendes, nicht-redundantes System geschaffen wird, worauf sich alle Prozesse beziehen können. Es können auch weitere Prozesse aufgebaut werden, die auf dem System basieren. Die Objekte können hinsichtlich Informationsbestand und Methoden erweitert werden, um wiederum neue Prozesse zu ermöglichen.

Somit können Prozesse als Handlungsweisen der Handelnden und die Objekte als Mittel und das Ergebnis der Handlungen gesehen werden. Dadurch ergeben sich folgende Vorteile:

- Eine objektorientierte Sicht von baubetrieblich relevanten Entitäten (Planungen, Gebäuden, Kalkulationen, Terminplänen, Ressourcen) verbessert die einheitliche Betrachtung aller Leistungen und Vorkommnisse eines Projektes und schafft eine Basis, die wiederum Prozessbeschreibungen erleichtert.
- Diese Sichtweise schafft desweiteren fachlich verständliche Einheiten mit integrierten Informationen (aus verschiedenen Disziplinen) und eignet sich daher auch für traditionell fachlich gesonderte Arbeitsweisen.

- Die fachliche objektorientierte Sichtweise, so sie sich zunehmend durchsetzen wird, passt natürlich aufgrund ähnlicher technischer Vorgehensweisen gut in die Applikation conobj.

3.1 Beschreibung der Geometrie

Bei den herkömmlichen IFC Implementierungen nimmt die geometrische Modellierung (meistens mittels CAD) eine herausragende Position ein. Das ist dadurch zu erklären, dass moderne Projekte fast ausschließlich mit Hilfe von einem CAD System geplant und modelliert werden, entweder in 2D oder 3D. Die baubetrieblich relevanten Informationen sind zum Teil in den Planinformationen (Dimensionen, Koordinaten, Materialien) enthalten. Einige IFC Implementierungen, die sich auch unter dem Begriff „Virtual Construction“ einordnen, sehen in der Verquickung einer 3D geometrischen Modellierung mit der Zeitschiene sogar eine „vierte Dimension“, auch 4D CAD genannt. Dadurch können vor allem geometrische Konflikte, die durch die zeitliche Abfolge von der Installation einzelner Komponenten entstehen, abgebildet werden (die Prozesskette einer Installation erfordert eine gewisse Reihenfolge der Tätigkeiten, die wiederum zum Konflikt mit bereits installierten Komponenten führen kann). Sinnvoll ist auch der Vergleich zwischen einer geplanten (Soll) und tatsächlichen (Ist) Bauausführung. Die starke Festlegung auf CAD hat wahrscheinlich auch den Hintergrund der Familiarität. CAD hat sich durch langjährigen Einsatz zum Quasistandard etabliert.

Die CAD basierten Lösungen erlauben eine detaillierte und exakte geometrische Bestimmung der Objekte, sind aber z.T. erheblich mit systemeigenen Begleiterscheinungen behaftet, die mit der Aufgabe der geometrischen Modellierung nichts gemein haben. Dadurch führt der Ansatz, dass CAD zum Standard geworden ist und dadurch eine solch zentrale Rolle inne hat, zu einer Blockade bei der Suche nach einer umfangreichen und interdisziplinären Lösung.

Denn eine Modellierung im funktionalen Sinne beantwortet lediglich die Frage nach der nötigen physischen Gegebenheit eines Objektes zur Erfüllung einer funktionalen Aufgabe. Faktoren bei der Bestimmung der Geometrie eines Objektes sind:

- architektonische Randbedingungen,
- Statik,
- Material,
- Kosten und
- Verfahrensweisen.

Bei der CAD Modellierung entsteht die Geometrie eines Objektes, was im Idealfall die obigen Faktoren berücksichtigt. Einige Systeme erlauben auch die Verknüpfung des geometrischen Gebildes mit Informationsquellen, welche die obigen Faktoren beschreiben; eine Interaktion ist jedoch nur schwer möglich.

Das liegt daran, dass CAD Systeme komplex sind und viel Aufwand zur Programmierung erfordern. Eine Integration des CAD mit anderen fachlichen Funktionen ist aufwendig und eigentlich nicht sinnvoll, da der Aufwand sich nicht auf die anderen fachlichen Funktionen bezieht, sondern direkt mit der Komplexität des CAD Systems zu tun hat.

Sinnvoller ist die Beschränkung des CAD's auf dessen Stärke: die elektronische Visualisierung von zeichnungstechnischen Handlungen. Alles darüber hinaus ist fachlich nicht sinnvoll, da verschiedene Fachlichkeiten vermischt oder unnötig voneinander abhängig gemacht werden. Die Statik steht in *Interaktion* mit der Haustechnik, ist jedoch nicht immer von ihr *abhängig*. Eine Untersuchung der räumlich-zeitlichen Konflikte kann auch mit anderen Mitteln als einer aufwändigen CAD Modellierung erreicht werden. Moderne internetfähige, geometrische Modellierungssprachen wie X3D können von Browsern interpretiert werden, die ein räumliches Bewusstsein haben.

Die geometrischen Informationen können von CAD Zeichnungen, mathematischen Formeln oder Vektor-Informationen stammen. Termininformationen sind im funktionalen Objekt vorhanden. Die Interaktion der räumlichen Integrität und der zeitlichen Installationsreihenfolge kann mit solch einfachen Mitteln ebenfalls untersucht werden.

Ein zusätzlicher Vorteil wäre, dass unterschiedliche Applikationen jeweils die geometrischen und terminlichen Informationen lieferten (entsprechend der fachlichen Disziplinen: Zeichner, Terminplaner) und die Untersuchung durch eine dritte Person erfolgen könnte. Die Daten existierten jedoch als Attribute in den relevanten Objekten.

3D CAD Modelle basieren häufig auf einem Drahtmodell des Objektes. Darüber wird mit aufwendigen Algorithmen erst eine geschlossene Fläche installiert und diese dann mit Hilfe von Rendering gerundet und gefüllt, um ein optisch möglichst perfektes und vollendetes Bild zu erreichen. Das 3D Bild hat wenig informativen Wert, dient meist einer optischen Funktion und wird oft im Bereich Marketing eingesetzt. Das grundlegende Modell ist jedoch eine Sammlung räumlicher Punkte, die mit Bögen verbunden werden [Vince, J., 1998. S. 32].
Diese Bögen können allerlei Art sein:

- gerade Linien,
- Kreissektoren,
- Polynome,
- Splines,
- Kegelschnitte,
- nicht-algebraische Funktionen (oder transzendente Funktionen).

Dadurch können Geometrien auch als Sätze von räumlichen Koordinaten und deren Verbindungskurven samt Art der Kurven beschrieben werden.

In Kapitel B.II.2.1 wird eine Implementierung einer solchen Lösung mit Linien und Kreissektoren als Verbindungskurven beschrieben.

3.2 Das Urobjekt

Das Urobjekt entspricht dem Begriff „object“ in C#/C++, Object in der. NET CLR, oder „object“ in Java. In der IFC Hierarchie wird ein Objekt wie folgt definiert [Adachi et al., 2002. WS]:

> *Defintion from IAI: An IfcObject is the generalization of any semantically treated thing or process within IFC. Objects are things as they appear - i.e. occurences.*
>
> *NOTE Examples of IfcObject include physically tangible items, such as wall, beam or covering, physically existing items, such as spaces, or conceptual items, such as grids or virtual boundaries. It also stands for processes, such as work tasks, for controls, such as cost items, for actors, such as persons involved in the design process, etc.*

Soweit existiert kein Konflikt zwischen dem hier beschriebenen Objektmodell und dem IFC Modell. Interessant ist auch die Enumeration der Objekttypen [Adachi et al., 2002. WS]:

```
TYPE IfcObjectTypeEnum = ENUMERATION OF
 (  PRODUCT,
PROCESS,
CONTROL,
RESOURCE,
ACTOR,
GROUP,
PROJECT,
NOTDEFINED);
 END_TYPE;
```

Der letzte Eintrag „NOTDEFINED“ verdeutlicht die Problematik des IFC Modells. Durch die starke Vordefinition und fachliche Festlegung wird das Modell inflexibel. Diese Einschränkung kann dann nur mittels Einsatz undefinierter Typen erfolgen, wie z.B. „NOTDEFINED“, die wiederum zu Entitäten im System führen, die außerhalb des Objektmodells und dessen Mechanismen liegen, d.h. als tote Masse.

Je nach Anteil dieser „NOTDEFINED“ Objekte, wird die Relevanz des IFC Modells und damit dessen Implementierung gemessen.

Das „ifcObject“ Objekt wiederum wird von der abstrakten Supertype „ifcRoot“ abgeleitet [Adachi et al., 2002. WS]:

> *Definition from IAI: The IfcRoot is the most abstract and root class for all IFC entity definitions that roots in the kernel or in subsequent layers of the IFC object model. It is therefore the common supertype all IFC entities, beside those defined in an IFC resource schema.*
>
> *The IfcRoot assigns the globally unique ID, and the ownership and history information to the entity. In addition it may provide for a name and a description about the concepts.*

```
HISTORY New entity in IFC Release 1.0
ISSUE See issue and change log for changes made in IFC Release 1.5 and 2x.
EXPRESS specification:
 ENTITY IfcRoot
 ABSTRACT SUPERTYPE OF  (ONEOF(IfcObject, IfcPropertyDefinition,
      IfcRelationship));
 GlobalId   :   IfcGloballyUniqueId;
OwnerHistory   :   IfcOwnerHistory;
Name   :   OPTIONAL IfcLabel;
Description   :   OPTIONAL IfcText;

UNIQUE
 UR1   :   GlobalId;
 END_ENTITY;
```

Zur Veranschaulichung der Formgebung eines Bauelementes nach den IFC Regeln, und dessen Implementierung im Objektmodell dient die IFC Klasse *ifcSectiondSpine* (aus der IFC Spezifikation – IFC 2.X) [Adachi et al., 2002. WS], auch Abbildungen 15 und 16:

Definition from ISO/DIS 10303-42-ed2:1999: A sectioned spine is a representation of the shape of a three dimensional object composed of a spine curve and a number of planar cross sections. The shape is defined between the first element of cross sections and the last element of this set.

> *NOTE A sectioned spine may be used to represent a surface or a solid but the interpolation of the shape between the cross-sections is not defined. For the representation of a solid all cross-section are closed curves.*

Definition from IAI: A sectioned spine (IfcSectionedSpine) is a representation of the shape of a three dimensional object composed by a number of planar cross sections,

and a spine curve. The shape is defined between the first element of cross sections and the last element of the cross sections. A sectioned spine may be used to represent a surface or a solid but the interpolation of the shape between the cross sections is not defined.

For the representation of a solid all cross section are areas. For representation of a surface all cross sections are curves. The cross sections are defined as profiles, whereas the consecutive profiles may be derived by a transformation of the start profile or the previous consecutive profile.

The spine curve shall be of type IfcCompositeCurve, each of its segments (IfcCompositeCurveSegment) shall correspond to the part between exactly two consecutive cross-sections.

New: The IfcSectionedSpine combines the functionality of the previous entities IfcAttDrivenTaperedExtrudedSegment, IfcAttDrivenMorphedExtrudedSegment, IfcAttDrivenTaperedRevolvedSegment, IfcAttDrivenMorphedRevolvedSegment.

NOTE Corresponding STEP entity: sectioned spine. Please refer to ISO/DIS 10303-42-ed2:1999, p. 282 for the definition of the formal standard. The cross sections are defined in IFC as IfcProfileDef. The position coordinate systems are added.

HISTORY New entity in IFC Release 2x.
The definition has changed after version 2x beta 1.
ISSUE: See issues and change log for changes made in IFC Release 2x.

Example of an IfcSectionedSpine

- *The SpineCurve is given by an IfcCompositeCurve with two Segments. The Segment[1] has a ParentCurve of type IfcPolyline and a Transition = CONTSAMEGRADIENT. The Segment[2] has a ParentCurve of type IfcTrimmedCurve and a Transition = DISCONTINUOUS.*
- *Each CrossSectionPosition lies at a start or end point of the Segments.*
- *Each CrossSection are inserted by the CrossSectionPositions. The first two cross sections are of type IfcRectangleProfileDef, the third is of type IfcDerivedProfileDef.*

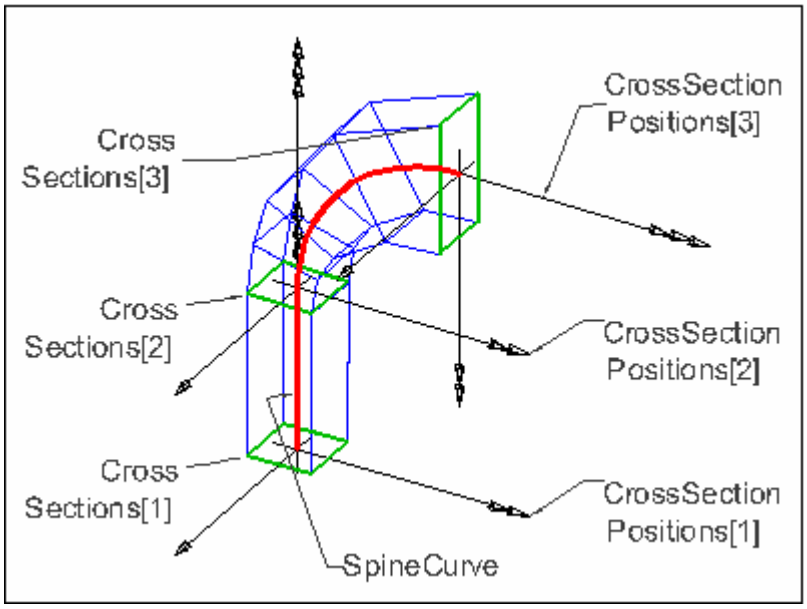


Abb. 15: Definition der Geometrie: IFCSectionedSpine [Adachi et al., 2002. WS]

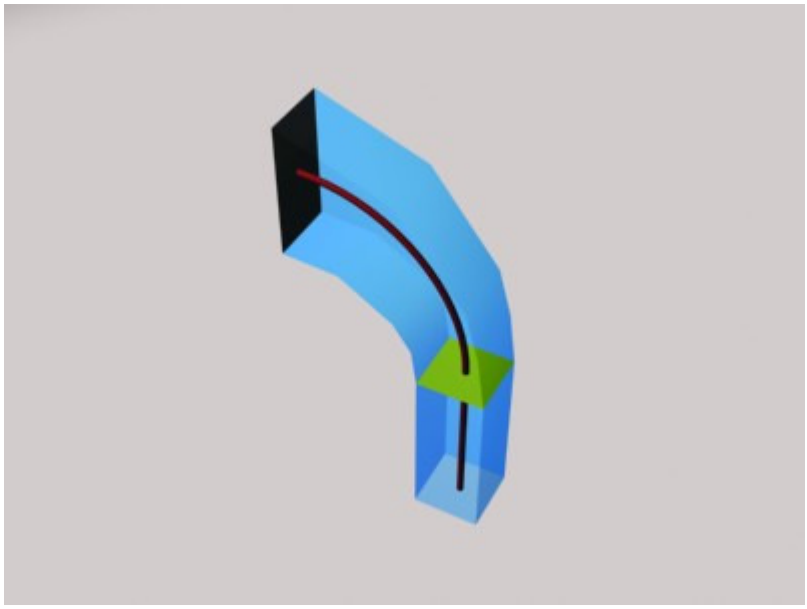


Abb. 16: Implementierung mittels X3D, nach VRML transformiert: IFCSectionedSpine [Adachi et al., 2002. WS]

Informal propositions

None of the cross sections, after being placed by the cross section positions, shall intersect.

None of the cross sections, after being placed by the cross section positions, shall lie in the same plane.

The local origin of each cross section position shall lie at the beginning or end of a composite curve segment.

EXPRESS specification:

```
ENTITY IfcSectionedSpine
 SUBTYPE OF (  IfcGeometricRepresentationItem);
 SpineCurve   :   IfcCompositeCurve;
CrossSections   :   LIST [2:?] OF IfcProfileDef;
CrossSectionPositions   :   LIST [2:?] OF IfcAxis2Placement3D;

DERIVE
 Dim   :   IfcDimensionCount :=  3;

WHERE
 WR1   :   SIZEOF(CrossSections) = SIZEOF(CrossSectionPositions);
WR2   :   SIZEOF(QUERY(temp <* CrossSections | CrossSections[1].ProfileType
<> temp.ProfileType)) = 0;
WR3   :   SpineCurve.Dim = 3;

END_ENTITY;

Attribute definitions:
SpineCurve    :    A single composite curve, that defines the spine curve.
Each of the composite curve segments correspond to the part between two
cross-sections.
CrossSections    :    A list of at least two cross sections, each defined
within the xy plane of the position coordinate system of the cross section.
The position coordinate system is given by the corresponding list CrossSec-
tionPositions.
CrossSectionPositions    :    Position coordinate systems for the cross sec-
tions that form the sectioned spine. The profiles defining the cross sec-
tions are positioned within the xy plane of the corresponding position co-
ordinate system.
Dim   :   The dimensionality of the spine curve is always 3.

Formal Propositions:
WR1   :    The set of cross sections and the set of cross section positions
shall be of the same size.
WR2    :    The profile type (either AREA or CURVE) shall be consistent
within the list of the profiles defining the cross sections.
WR3   :    The curve entity which is the underlying spine curve shall have
the dimensionality of 3.

References (1):
Name  Type  Referred through  Express-G
IfcGeometricRepresentationItem  Entity  Subtype
 Diagram 1

Inheritance graph

 ENTITY IfcSectionedSpine;
 ENTITY IfcSectionedSpine;
 SpineCurve   :   IfcCompositeCurve;
CrossSections   :   LIST [2:?] OF IfcProfileDef;
CrossSectionPositions   :   LIST [2:?] OF IfcAxis2Placement3D;

DERIVE
 Dim   :   IfcDimensionCount :=  3;

END_ENTITY;
```

Die Klasse `IfcSectionedSpine` hat folgende Vererbungskette [Adachi et al., 2002. WS]:

	IfcRoot
→	IfcObject
→	IfcResource
→	IfcGeometricModelResource
→	IfcGeometryResource
→	IfcGeometricRepresentationItem
→	IfcSectionedSpine

Diese abgeleiteten, abstrakten Klassen sind ein Kernbestandteil eines objektorientierten Systems. Wesentlich ist, dass eine strikte Vererbungshierarchie eingehalten wird, damit keine Sprünge und Inkongruenzen auftreten. Wenn dieses gesichert ist, ist die Detaillierung einer abstrakten Klasse aus fachlicher Sicht nicht entscheidend, sondern nur im Hinblick auf die praktische Handhabung. Wenige, dafür aber umfassend detaillierte, abstrakte Klassen verringern den Vererbungsaufwand bis hin zur Erstellung realer Klassen. Im hier aufgeführten Objektmodell wäre es z.B. möglich die Klasse `constructionobject` als abstrakte Klasse zu definieren, da in dieser Klasse wesentliche Basisinformationen eines im Baubereich sinnvoll einsetzbaren Objektes eingetragen sind.

Die Transformation vom hier beschriebenen dynamischen Objektmodell zum statischen IFC Modell erfolgt auf der Objektebene mittels einer XSLT Transformation. Dabei werden die Schemata des dynamischen Objektmodels (befindlich in dem jeweiligen Objekt selbst) und des IFC Models eingehalten. Eine Transformation kann entweder aus dem XML Format einer Geometrie oder dessen X3D Abbildung erfolgen.

3.3 XML-Schema-basierte Erzeugung einer Klasse

Folgendes UML Modell illustriert den Aufbau der virtuellen Basisklassen:

Abstract base Interface IConObj, base class conObj, and 1st level derived classes cAssembly cResource, and cDoc. 2nd level derived abstract classes cAssemblyPerm, cAssemblyTmp, cResourceHuman, cResourceMec, cResourceMat

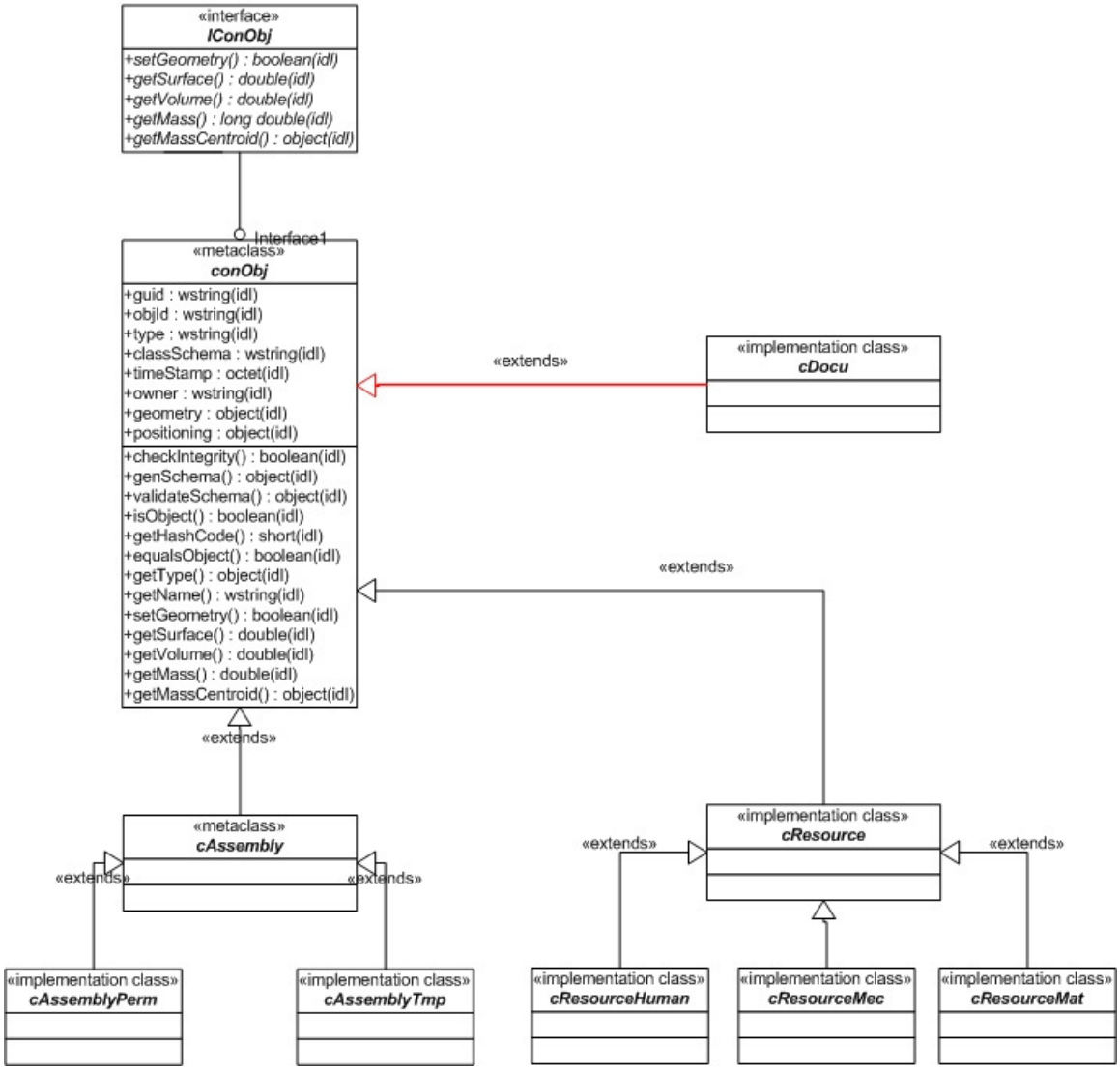


Abb. 17: UML Model von conObj und abgeleiteten, abstrakten Klassen

Diese Klassen können bei Bedarf den IFC Klassen zugeordnet werden. Zur Vermeidung der Eingrenzung des Modells durch eine zu starke Definition auf der abstrakten Ebene sollte darauf geachtet werden, dass die abstrakten Klassen nur systemimmanente Klassendefinitionen enthalten. Indem die abstrakte Ebene dort aufhört, wo projektspezifische und situationsbedingte Klasseninformationen benötigt werden und das Objektmodell beliebige Vererbungen vorsieht, können verschiedenste Objekttyp er durch Vererbung geschaffen werden.

Zurzeit ist eine einfache Vererbung vorgesehen. Dies entspricht den Regeln für C#, VB.NET und Java. Eine Mehrfachvererbung würde zu einem ähnlichen Problem führen wie es in der Praxis mit C++ öfter vorkommt: das Auftreten von mehreren Versionen derselben Attribute oder Methoden. Die Einfachvererbung sorgt für Kompatibilität mit C#, VB.NET, C++ und Java und durch die *Interfaces* können auch *Collection* (Aggregations-) Objekte eingesetzt werden, z.B. bei der Terminplanung, der Gruppierung in Leistungsbereiche, einer Kostengliederung.

Anhang B.I.3.3 beschreibt eine schema-basierte Erzeugung einer Klasse.

Der Vorteil einer objektorientierten Kapselung des Schema Konstrukts in einer modernen, objektorientierten Sprache ergibt sich durch die einfache Manipulation und die elegante Hantierbarkeit. Jedoch führt dies zu der Plattform-Abhängigkeit wie bereits früher beschrieben. Die Lösung würde die Implementation dieser Klassen in den Sprachen C#, VB.NET, MC++, C++ und Java vorsehen, da diese Sprachen objektorientierte Programmierung unterstützen und ein weites Spektrum abdecken.

In Anhang B.I.3.3 wird die entsprechende Implementierung in Java aufgeführt.

3.4 Dynamische Klassen und Objekte

XmlSchema eignet sich zur Beschreibung von Klassen. Sie ist jedoch nicht die einzige Methode und eigentlich nicht die beste. UML Diagramme sind eher geeignet zur Klassenbeschreibung. Sie lassen sich jedoch im Gegensatz zu XML Schema schlecht einbinden, was durch Frameworks wie .NET gut unterstützt wird. Hinzu kommt, dass UML eine starke Abstrahierung vorsieht, was zu Umsetzungsschwierigkeiten führt. Die Beschränkungen in XML Schema als Modellierungssprache liegen eher darin, dass das DOM (Document Objekt Model) in sich limitiert ist. Das spiegelt sich z.B. bei der Abbildung der Vererbung wider.

Genau diese Einschränkung führt auch zur erhöhten Komplexität bei den XQuery Lösungen, wo die Suche über mehrere Dokumente eher der Regelfall ist.

3.4.1 XML-Schema und XML

Im Verlauf dieser Arbeit wurde der Versuch unternommen, eine Applikationsschicht für die Instanzierung von Objekten im XML Format zu entwickeln. Das XML Format eignet sich naturgemäß für die Umsetzung von XML Schema basierten Beschreibugen. XML wird üblicherweise mit XML Schema validiert, beide Technologien wurden in Tandem durch das W3C entwickelt. Was vor allem für dieses Format spricht, ist dessen weite Verbreitung und Akzeptanz durch Software Hersteller, -Forscher und -Unternehmen. XML basierte Lösungen genügen generell dem Anspruch der Plattformunabhängigkeit. XML hat jedoch eine gravierende Schwäche gegenüber imperativen Frameworks wie Java oder .NET: das Format ist nicht typsicher. Laut Box [Box, D.,, 2003. S1] war ein Hauptmerkmal von .NET die Typsicherheit. Im Prinzip erfordert sie die genaue Bestimmung eines Klassenelements (Attribut, Eigenschaft, Methode, Ereignis) vor dessen Nutzung. Typsicherheit hat viele Vorteile, unter anderem das Verhindern von Ausnahmen, die auf unerlaubte Typkonvertierung zurückzuführen sind.

Clifton [Clifton, 2005. S. 1] hat eine „XML instantiation engine“ entwickelt, um XML Strukturen mit XML Schema zu beschreiben und in .NET zu instanzieren. Diese Vorgehensweise beruht auf der .NET Klasse `System.Xml.XmlDataDocument`. Die Dokumentation dieser Klasse ist wie folgt [.NET Framework Class Library, System.Xml Namespace, 2003. WS]:

> *“Allows structured data to be stored, retrieved, and manipulated through a relational DataSet.”*

Diese Klasse, welche die Umwandlung einer XML Schema Struktur in einem XML Datenstrom ermöglicht, wurde jedoch primär als Ergänzung für das relationale DataSet entwickelt und nicht als Instanzierungsmotor für XML Schema. Somit stellt die hier vorgeschlagene Lösung eine unbeabsichtigte Erweiterung des Einsatzspektrums der Klassen dar.

Diese Vorgehensweise hat viele Vorteile, und wird u.a. zunehmend in der automatischen Konfiguration von GUI's, Container, Ereignisse, Workflows und Plug-in Architekturen eingesetzt. Sie hat sich jedoch nach einem gründlichen Versuch als Lösungsansatz für die in dieser Arbeit beschriebene Problemstellung aus folgenden Gründen nicht bewährt:

- Dadurch, dass XML auf Objektebene eingesetzt wird ist die Typsicherheit nicht gewährleistet.
- Die Identifizierung von Klassenelementen (Attribute, Eigenschaften, Methoden, Ereignisse) in diesem Model erfolgt über Tags und nicht über den Typ. Es fehlt die gesamte Struktur System.Type.
- Die Leistung einer XML-basierten Lösung ist deutlich niedriger als bei einer imperativen Lösung.
- Elemente einer XML Struktur sind nicht automatisch sicher für multithreaded Operationen.
- Die Strukturierung eines Objektes in den grundlegenden Elementen (Attribute, Eigenschaften, Methoden und Ereignisse) erfolgt über Tags und ist nicht systemimmanent.

3.4.2 Dynamische Code Generierung

Die .NET Framework bietet in der XmlSerialization Namespace 4 Klassen für das dynamische Generieren von Code an (Abb. 18):

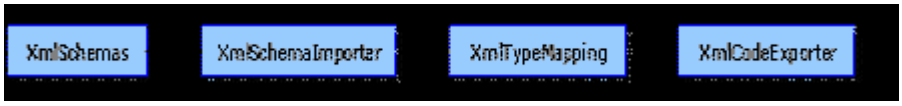


Abb. 18: Klassen in XML Serialization Namespace, die bei der automatischen Objektgenerierung eingesetzt werden [Cazzolino, D., 2005. S. 3]

Diese Klassen werden in folgender Sequenz eingesetzt, um aus den XML Schema Definitionen der Klassen dynamische Objekte zu generieren:

- Das Schema wird geladen.
- Eine Serie von Mappings wird für die obersten XSD Elemente generiert.
- Diese Mappings werden zu dem System.CodeDom.CodeDomNamespace exportiert.

Die XML Schema Datei einer Klasse wird in einer XML Schema Collection des Konstruktors der XmlSchemaImporter Klasse aufgenommen.

```
XmlSchemaImporter importer = new XmlSchemaImporter( schemas );
```

Der Instanz-Importer kann jetzt verwendet werden, um einen XmlCodeExporter zu initialisieren. Dazu wird eine Namespace deklariert:

```
CodeNamespace ns = new CodeNamespace( targetNamespace );
XmlCodeExporter exporter = new XmlCodeExporter( ns );
```

Im Folgenden wird der Code mit einem CodeDomProvider des CodeDom Namespaces generiert:

```
CodeDomProvider provider = new CSharpCodeProvider();
using(StreamWriter sw = new StreamWriter(targetFile, false))
{
      provider.CreateGenerator().GenerateCodeFromNamespace(
                  ns, sw, new CodeGeneratorOptions());
}
```

Der Exporter kann jetzt verwendet werden, um den Code in folgenden Formaten zu exportieren:

```
C#: CSharpCodeProvider
VB.NET: VBCodeProvider
```

Der generierte Code wird in einer Datei zwischengelagert. Diese Datei kann serialisiert und in einer Datenbank gespeichert werden. Diese Datei gilt als Basis für die dynamische Instanzierung von Objekten einer in XML Schema definierten Klasse.

In Anhang B.I.3.4 wird Code zur dynamischen Generierung und Kompilierung der Klassen, sowie deren Instanzierung aufgeführt. Dieses Vorgehen erfolgt über eine statische Methode der Klasse ***objmodel***.

Der `CSharpCodeProvider` bietet Zugang zu der `ICodeGenerator` Schnittstelle. Durch diese Schnittstelle kann Funktionalität des C# Compilers angesprochen werden.
Der Aufruf

```
ICodeCompiler compiler = provider.CreateCompiler();
```

gibt einen C# Compiler zurück. Der Compiler wird mit zusätzlichen Assembly - Bindungen initialisiert. Die Anweisung

```
cp.GenerateExecutable = false
```

weist den Compiler an, eine dll zu generieren. Die Compiler Methode

```
CompileAssemblyFromFile
```

instanziert nun die Klasse. Somit ist es möglich, dynamisch und mit minimaler Verzögerung, ein dynamisches Objekt aus einer XML Schema Klassenbeschreibung zu generieren. Dieses dynamische Objekt weist alle Vorteile einer imperativen Sprache auf:

- Interaktion mit System- und Netzwerk/Internet-Ressourcen
- Ereignisse (Auslösung und Behandlung)
- Ausführen von Methoden
- Eigenschaften (Properties) mit Anweisungsblöcken
- Verteilte Aufrufe (Remoting)
- Automatisiertes Ausführen von Methoden
- Serialisierung

Zusammenfassend führt folgende Sequenz zu dynamischen Objekten:

1. Fachliche Beschreibung von Objekten als Klassen in XML Schemata als abstrakte oder vererbte Klasse
2. Generierung von Code (C# oder VB.NET)
3. Kompilierung und Instanzierung
4. Einsatz in einem verteilten Szenario

Laut Cazzulino [Cazzulino, D., 2004. S. 5] bieten sich beim Customizing eines automatischen Code Generierungstools folgende beiden Vorgehensweisen an:

- *Add (potentially many) attributes to the XSD root <xs:schema> element that would be understood by my processor to apply customizations, similar to the typed dataset approach.*
- *Use built-in XSD extensibility through schema annotations to allow arbitrary customizations. It simply adds types, to a sort of code generation pipeline, to execute after the basic generation has taken place.*

Die automatisierte Generation von Code durch den CodeDomProvider führt zu Klassenbeschreibungen mit Einschränkungen. Ein typisches Code Generierungstool wie Xsd.exe stellt Attribute als Felder dar. Die Nachteile überwiegen:

- Lange switch Anweisungen
- Endlos viele Attribute
- Unwartbare Codes
- Mangel an Erweiterbarkeit

Die zweite Vorgehensweise ist robuster und berücksichtigt Erweiterbarkeit von vornherein. XSD ermöglicht eine solche Erweiterbarkeit mit dem <xs:annotation> Element. Dieses Element kann ein Kind von fast jedem Element eines Schemas sein. Diese Funktionalität zusammen mit dessen <xs:appinfo> Kind Element ermöglicht dem Entwickler die genaue Spezifizierung, welche beliebige Erweiterung ausgeführt werden sollte und dessen Reihenfolge. Ein solches erweitertes Schema könnte wie folgt aussehen:

```
<xs:schema elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>
      <Code xmlns="http://weblogs.asp.net/cazzu">
        <Extension Type= "XsdGenerator.Extensions.
            FieldsToPropertiesExtension,
            XsdGenerator.CustomTool" />
      </Code>
    </xs:appinfo>
  </xs:annotation>
```

Damit eine Erweiterung alle Möglichkeiten beinhaltet, und zum Beispiel auch Eigenschaften generiert werden können, müssen sog. Interfaces definiert werden:

```
public interface ICodeExtension
{
  void Process( System.CodeDom.CodeNamespace code,
                System.Xml.Schema.XmlSchema schema );
}
```

Beim Parsen einer XSD Datei (Klassendefinition) wird analog des Parsens einer XML Datei die XPath Abfragesprache eingesetzt.

Mit dieser Vorgehensweise kann die abstrakte Klasse ***conobj*** zur Laufzeit in ein dynamisches Objekt umgewandelt werden. Die Vorgehensweise ist folgendermaßen (Abb. 19):

Beschreibung	Methode/Ressource	Ergebnis
Klassendefinition	XML Schema	conobj.xsd
Validierung der XML Schema Klassendefinition	`ValidationCallbackOne`	`ValidationEventArgs`
Einlesen und Generierung einer CodeNamspace	statische `Process` Methode der Klasse objmodel	`CodeNamespace`
Generierung von Code	`GenerateCodeFromNamespace`	conobj.cs
Kompilieren und	`CompileCode`	`CompilerResults`
Instanzierung eines Objekts	`CompiledAssembly`	`assembly.CreateInstance`

Abb. 19: Vorgehensweise bei der Umwandlung einer abstrakten Klasse conobj in ein dynamisches Objekt, zur Laufzeit.

Das dynamische Objekt wird wie folgt instanziert:

```
object obj =
assembly.CreateInstance("conobj.classAttributesGeneric", false);
```

Jetzt können Eigenschaften gesetzt

```
obj.GetType().GetField("guid").SetValue(obj,conobj_guid);
```

oder angefragt werden:

```
obj.GetType().GetField("guid").GetValue (obj);
```

Methoden können auch aufgerufen werden:

```
object o =
  obj.GetType().GetMethod("initialise", flags).Invoke(obj, null);
```

flags beschreibt die BondingFlags Optionen, die bei dem Aufruf berücksichtigt werden müssen, z.B.

```
BindingFlags flags = (BindingFlags.NonPublic | indingFlags.Public |
BindingFlags.Static | BindingFlags.Instance |
BindingFlags.DeclaredOnly);
```

Analog können Konstruktoren des Objekts deklariert werden:

```
ConstructorInfo constructorInfo Obj = obj.GetType().GetConstructor
(BindingFlags.Instance | BindingFlags.Public, null,
CallingConventions.HasThis, types, null);
```

Diese Vorgehensweise wird wesentlich interessanter bei der Möglichkeit, Delegates zu deklarieren und Ereignishändler zuzuweisen. Damit werden folgende Szenarien ermöglicht:

- Objekte können auf Änderungen innerhalb sich selbst reagieren
- Objekte können Ereignisse, die außerhalb des Objekts auftreten, wahrnehmen und darauf reagieren
- Objekte können Timer-gesteuert Methoden ausführen.

Auch können mit der Anweisung:

```
ompilerParameters cp = new CompilerParameters(new string[]
{"System.dll", "System.Xml.dll"}, exepath, false);
```

externe Dienste (XML Services) oder verteilte Objekte (.NET Remoting) dynamisch eingebunden werden. Somit erhöhen sich die Vielfalt und die Auswirkung von Objektmethoden erheblich.

Die Namespace XmlSerialization, die sich in ähnlicher Form in der Java Sprache wiederfindet, ermöglicht den Übergang von einer fachlichen Objektbeschreibung zu einem programmatisch manipulierbaren Objekt. Dadurch ist die Kette dynamischer geworden und bildet einen Kreislauf (Abb. 20).

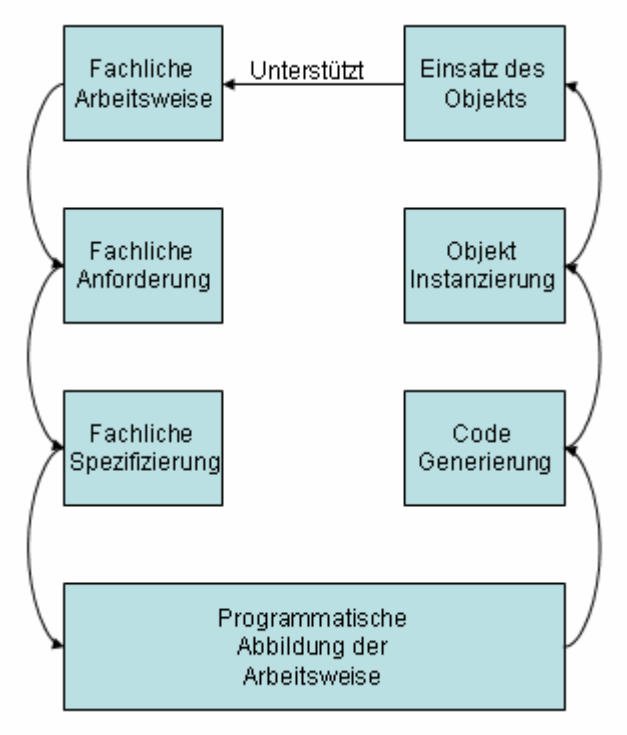


Abb. 20: Zyklus: Dynamische Unterstützung der fachlichen Arbeitsweise

Der Grad der Dynamisierung hängt nur noch von folgenden Randbedingungen ab:

- Umfang der fachlichen Spezifikation
- Rechenkapazität
- Leistung der .NET Framework
- Kompression des eingesetzten XML Schemas
- Leistung der Anwendung, die das instanzierte Objekt einsetzt

Als Beispiel dient die dynamische Erweiterung der Basisklasse

```
ProtoClass.constructionobject
```

Es soll eine neue Eigenschaft für Mängel angegeben werden. Eigentlich würde eine solche Eigenschaft in der Praxis aus einer Liste von Mängeln bestehen. Allerdings beschränkt sich das Beispiel auf einen einfachen Typus: *System.String*. Dadurch kann eine einfache Beschreibung der Mängel angegeben werden. Die Implementierung der Liste würde analog der Dokumentationsliste stattfinden und ist daher technisch uninteressant.

Cazzulino [Cazzulino, D., 2005. S. 5] beschreibt eine Vorgehensweise zur generischen Code Generierung mittels XML Schema. Eine zentrale Stellung nimmt dabei die Erweiterbarkeit des Verfahrens durch Implementierung der ICodeExtension Interface ein. Für jede erwünschte Erweiterung (Extension), kann eine Implementierung geschrieben werden, die dieses Interface einbindet. Somit kann das Verfahren auf alle Code Variationen wie Eigenschaften, Methoden, Konstruktoren und andere Klassenelemente ausgebaut werden. Die Klasse zur Erweiterung der Generierung von Eigenschaften wird folgend angegeben. Zu beachten ist, dass der Xsd.exe und das Standardverfahren über XmlCodeExporter Begrenzungen hinsichtlich Code Aufbau und Funktionalitäten haben. Diese Vorgehensweise wird in Anhang B.I.3.4.2 auf der Code-Ebene aufgeführt.

Mit der System.Reflection Namespace kann diese Klasse (siehe Anhang B.I.3.4.2) instanziert und die Eigenschaft Maengel abgefragt bzw. gesetzt werden (Abb. 21 und 22):

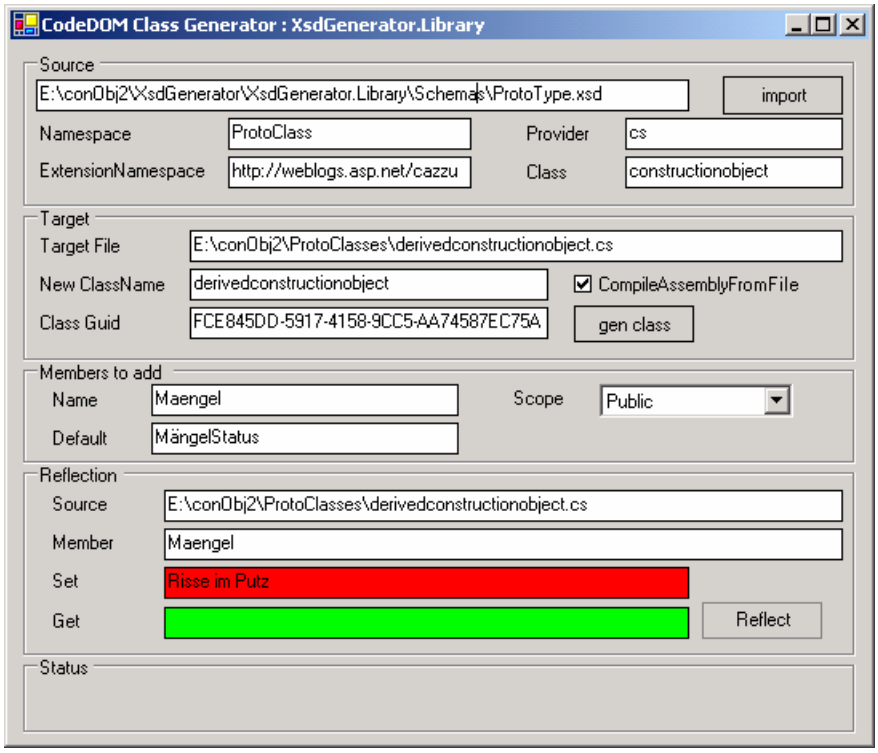


Abb. 21: Applikation zur dynamischen Generierung von einer abgeleiteten Klasse

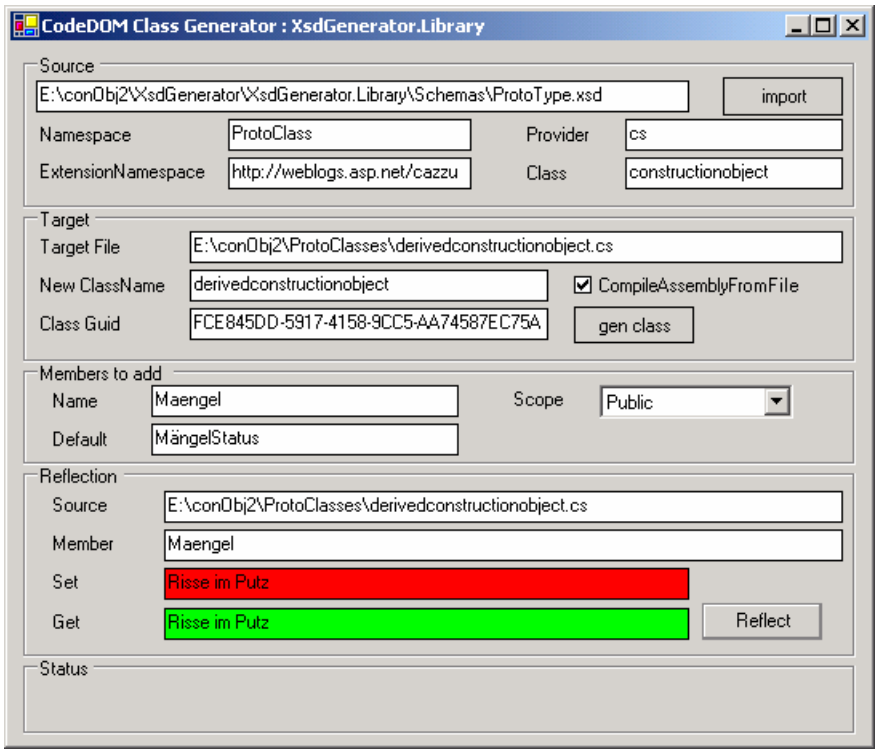


Abb. 22: Nutzung der Klasse mittels der System.Reflection Namespace. Set Wert (Rot) wird mit Get ermittelt und angezeigt (Grün).

Diese neue Klasse (ProtoClass.derivedconstructionobject) kann jetzt in das Objektmodell geladen werden, als Ableitung der Basisklasse ProtoClass.constructionobject (Abb. 23). Durch die automatisierte Verteilung wird diese Information in den Terminplan mittels des Dienstes *p3eWrapper* geschrieben (Abb. 24).

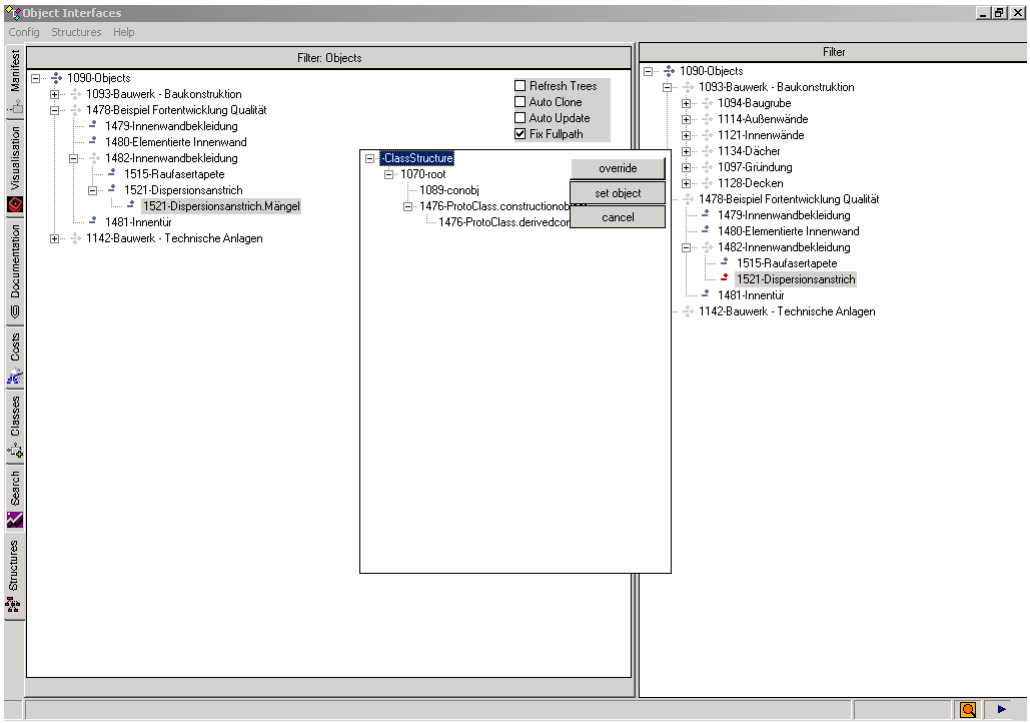


Abb. 23: ProtoClass.derivedconstructionobject von ProtoClass.constructionobject angeleitet.

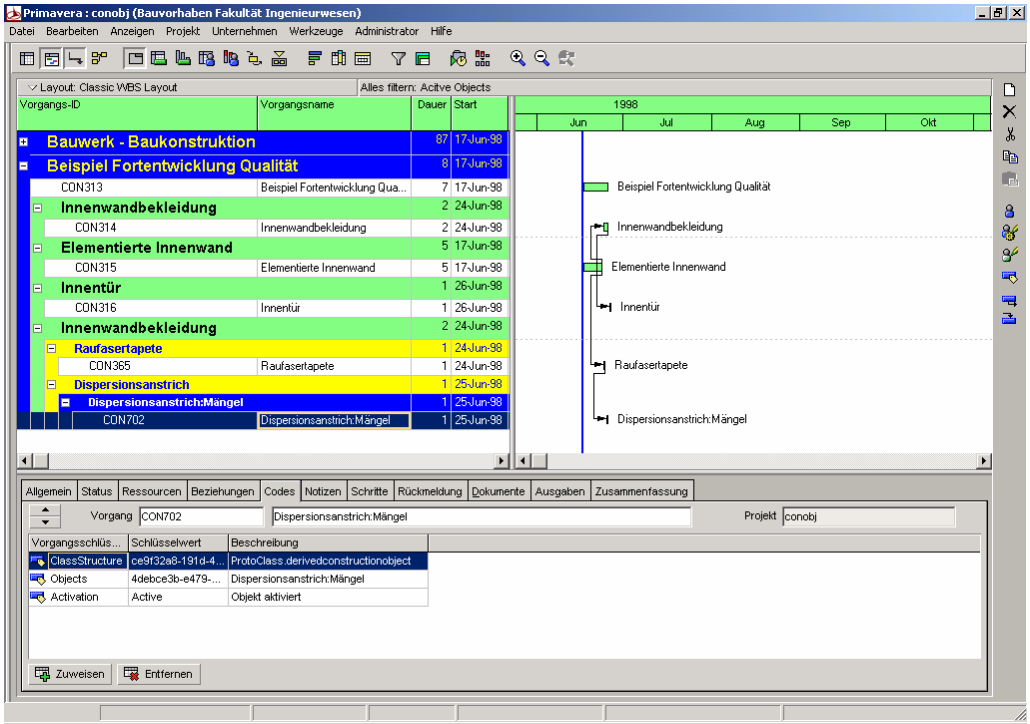


Abb. 24: Terminierung des Objekts Dispersionsanstrich:Mängel, vom Typ ProtoClass.derivedconstructionobject.

3.5 Prototyp einer dynamischen Klasse

Zur Entwicklung des dynamischen Systems wurde eine Musterklasse geschrieben. Diese Klasse sollte folgende Anforderungen erfüllen:

- minimale Anzahl Eigenschaften
- interne Ereignisbehandlung
- Timer Methoden zur Aktualisierung von Eigenschaften
- Methoden zur Serialisierung
- Schnittstellen zur Aktualisierung externer referenzierender Applikationen

Diese Musterklasse wird in Anhang B.I.3.5 aufgeführt.

In ILDASM kann das Manifest dieser Klasse als Baumstruktur veranschaulicht werden (Abb. 25). Die Attribute zur XmlSerialisierung der Klasse sind in der Baumstruktur ersichtlich.

Die Applikation, welche dynamische Objekte dieser Art einsetzt, sollte auch die Schnittstelle `IConstructionObject` einsetzten, um sicher zu stellen, dass die wesentlichsten Änderungen in den Objekten in der Applikation ersichtlich werden:

```
using System;
using System.Windows.Forms;

namespace ProtoClass
{
/// <summary>
/// IConstructionObject defines a contract between a dynamic
/// class instantiation and the application utilising that class
/// </summary>
      public interface IConstructionObject
      {
            TextBox textBox_ObjectId
            {
                  get;
                  set;
            }

            TextBox textBox_TickCounter
            {
                  get;
                  set;
            }
            void updateGeometry();
            void callUrl(string xml);
      }
}
```

Die dynamischen Objekte können sich somit auf das Vorhandensein der Eigenschaften

- `textBox_ObjectId`
- `textBox_TickCounter`

und Methoden

- `updateGeometry();`
- `callUrl(string xml);`

verlassen. Diese Eigenschaften der Applikation bzw. deren Methoden werden aus den Eigenschaften der dynamischen Objekte bei Änderungen derer Werte angesprochen.

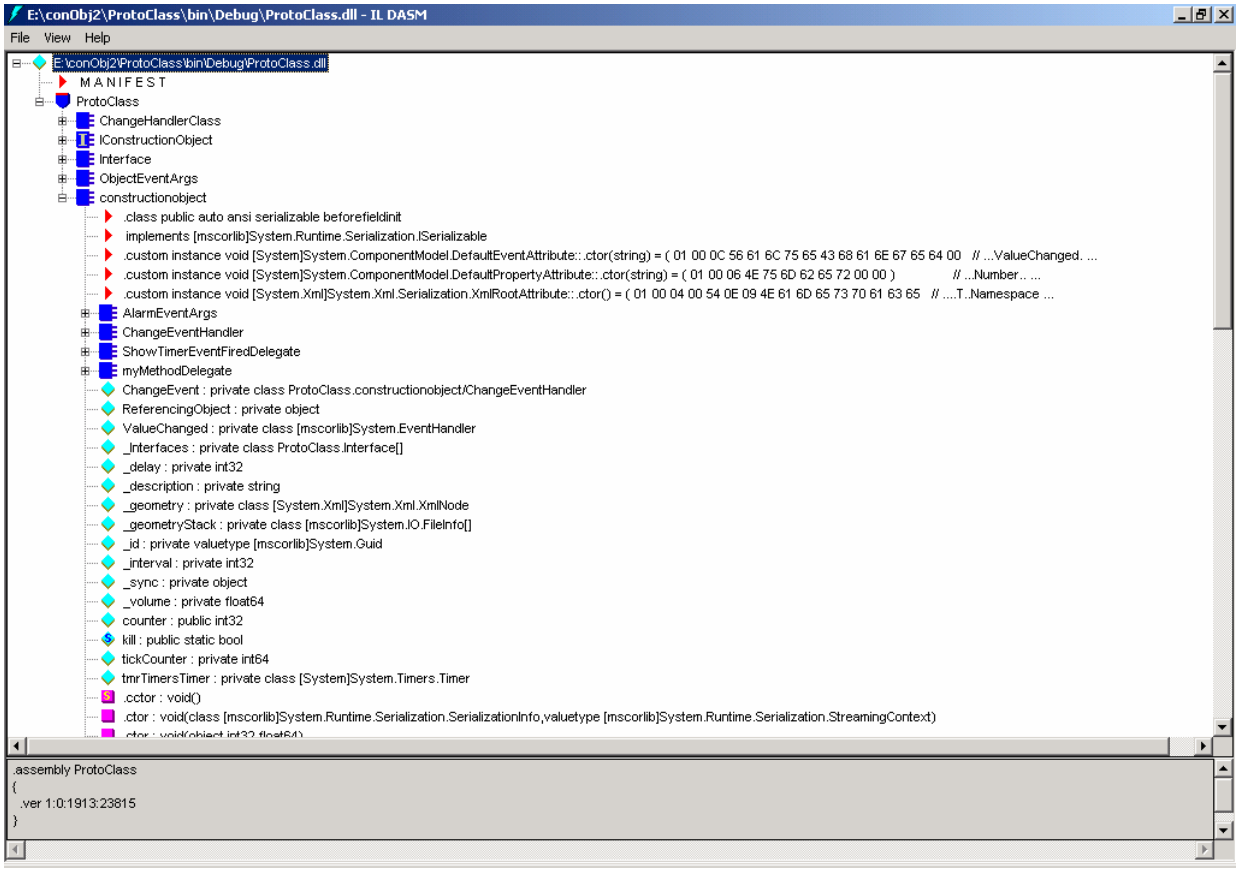


Abb. 25: Struktur der Klasse in ILDASM.

II Trennung der funktionalen und der geometrischen Modellierung

1 Herkömmliche geometrische Modellierung

Seitens Industrie und Wissenschaft wurde seit Anfang der 80er–Jahre in mehrfach groß angelegten Initiativen eine umfangreiche Klassenbeschreibung aller baurelevanten Entitäten unternommen. Das bekannteste Beispiel ist die IFC Klassenbibliothek. Dadurch, dass die CAD-Technologie, vorrangig vom Maschinenbau getrieben, schon weit entwickelt war, lehnten sich objektorientierte Modellierungsansätze vorwiegend an CAD Technologien an. Ein aktuelles Beispiel ist die 4DCAD Technik [Dawood, N., 2002. S. 124 f.]. Dieser Ansatz ist plausibel und entspricht dem Wunsch, industrienahe Lösungen und rasch einsetzbare Technologien zu entwickeln. Vorteile einer stark CAD-Basierten Technik sind:

- Anlehnung an existierende Standards
- Großer Benutzerkreis
- CAD-integrierte Bearbeitung (Edit) und Anschauung (Viewing)
- Existierende CAD Tools bieten eine Vielzahl von Funktionalitäten
- CAD stellt eine Schnittstelle zur Baupraxis (Planung, Ausführung) dar

1.1 Trennung von Geometrie und Funktion

Die Verquickung der CAD-orientierten Objektmodellierung mit der CAD Technik setzt eine geometrische Definition der Objekte voraus. Bauelemente werden als geometrische Gebilde verstanden. Mittels 4D Technik können diese auch mit einer Zeitschiene versehen werden, durch die der Zustand zu einem jeweiligen Zeitpunkt auch festgehalten wird.

Die während der Projektdauer auftretenden Erscheinungsformen eines Bauelementes können jedoch nicht nur geometrisch deutbar, sondern auch abstrakt erfolgen. Die Reihenfolge der Instanzen eines solchen Elementes kann auch durch Planungsbrüche und Änderungen verändert werden, es treten Mischformen von virtuellen und geometrischen Gebilden auf. Während der frühen Phasen eines Projektes wird mit abstrakten Konzepten (Funktionalität, Raum, Nutzen usw.) gearbeitet. Während der Planungsausführung konkretisieren sich die geometrischen Ausmaße und werden z.T. zur Leistungs-Ausschreibung durch Sondervorschläge zur Disposition gestellt. Darüber hinaus sind Vertragsformen, die die Planung als Bestandteil der Bauleistung vorsehen (z.B. BOT-Projekte) auch vertreten.

Während der Ausführung treten oft gravierende Änderungen auf, die Mehraufwendungen und Verzögerungen der Bauausführung nach sich ziehen können. Solche Zwischenstände sind umfassend zu dokumentieren, um sie professionell bearbeiten zu können.
Die Frühphasen eines Projektes sind wichtig, da die dort gefällten Entscheidungen Auswirkungen auf die späteren und teureren Planungs- und Ausführungsphasen haben.
Durch die Entscheidung, eine strikte Trennung zwischen der geometrischen Darstellung (eigentlich nur ein Attribut oder Eigenschaft eines Bauelementobjektes) und der funktionalen Beschreibung (das, was das Objekt im Kontext des Projektes bezwecken soll) einzuhalten, wird der Zwang zur geometrischen Modellierung eines Projektes, die in den früheren Phasen praktisch nicht realisierbar ist, aufgehoben. Diese Relativierung der Geometrie erleichtert auch die technische Beschreibung von Veränderungen. Danach kann ein beliebiges System zur geometrischen Modellierung eingesetzt werden, natürlich auch CAD. Die geometrische Modellierung kann zu der ihr angemessenen Zeit erfolgen, dann nämlich, wenn die funktionale Beschreibung eines Objektes vorhanden ist und eine Form (Geometrie) zur Realisierung dieser funktionalen Beschreibung gefunden worden ist. Die Funktionalität dominiert die Geometrie.

Die Objekthistorie kann mit Hilfe von bewährten objektorientierten Techniken beschrieben und behandelt werden; sie kann dynamisiert werden (dynamische Vererbung), um die Veränderungshistorie abzubilden. So kann eine Baumstruktur der Objekthistorie samt dazugehöriger Klassenhistorie erstellt werden, die auch der Entwicklung des Teils eines Projektes dokumentiert.

2 Methoden der geometrischen Beschreibung

2.1 Theoretische Methoden

Die Mathematik liefert etablierte Methoden zur Beschreibung mehrdimensionaler Körper. Die Ausgangsfestlegung ist immer die eines Achsensystems bzw. einer Betrachtung der Transformation eines Achsensystems in ein anderes. Transformationen sind eine wesentliche Vorraussetzung dafür, dass ein geometrisches Modell als praxistauglich bewertet werden kann, da geometrische Körper aus verschiedenen Gründen in unterschiedlichen Achsensystemen untersucht werden müssen. Zusätzlich zu Referenzpunkten in einem Achsensystem können auch Funktionen zur Beschreibung von Oberflächen entwickelt werden. Die theoretische Beschreibung fester Körper ist auch hilfreich bei der Vergabe von Verhaltensregeln an solchen Körpern, z.B. das elastisch-plastische Materialverhalten (Elastizitätstheorie) oder Betrachtungen zur Energetik (Kinetik).

2.1.1 Allgemein

Als erweiterbar und jedoch nicht übermäßig kompliziert erweist sich die Methode, einen Körper als Zusammensetzung von eindeutigen Punkten mit Kurven zu definieren und ihn mathematisch und grafisch zu beschreiben. Ein solches Gebilde wird als geometrisches Objekt in die dynamisch vererbten, funktionalen Objekte integriert. Auf diese Weise können Körper jeglichen Ausmaßes und jeglicher Geometrie erstellt werden ohne Einsatz eines Modellierungsinstruments wie CAD.

CAD verfolgt einen ähnlichen Ansatz. Es werden bei eher grafisch orientierten Benutzeroberflächen grafische Interpolationen, z.B. die Umwandlung eines „Drahtmodells“ in ein Flächenmodell, hinter den Kulissen ausgeführt und damit der Eindruck eines Volumens geschaffen.

Eine Methode zur Schaffung eines solchen Drahtmodells soll nachfolgend erklärt werden und einige Fälle der beliebigen Möglichkeiten eruiert werden. In dieser Arbeit beschränken sich die Verbindungsarten auf Geraden und Kreissektoren (Teile von Kreisen). Das Achsensystem verhält sich relativ zu einem dem Projekt zentralen System und erlaubt beliebige Transformationen der beschriebenen Körper.

2.1.2 Konstruktion der Körper

Die Körper bestehen ausschließlich aus Punkten, die mit Hilfe von definierten Kurven verbunden werden. Die Kurven werden je nach Art mit Hilfe von vorher definierten Punkten konstruiert; durch Nummerierung der schon vorhandenen Punkte erfolgt die Assoziation der Punkte mit den Kurven.

Diese Aktionen finden in dem geometrischen Objekt statt. Das Objekt wird danach in dem funktionalen Objekt integriert. Diese Vorgehensweise entspricht der in dieser Arbeit verfolgten Trennung von der funktionalen und geometrischen Modellierung. Der Schwerpunkt soll auf der funktionalen Modellierung liegen, da diese den baubetrieblichen Zustand genauer modelliert als eine geometrische Beschreibung.

Die geometrische Beschreibung auch eines komplexen Objektes ist letztlich nur ein Attribut des funktionalen Objektes und dient der Visualisierung.

Die Entwicklung der theoretischen Methode und deren Umsetzung werden detailliert in Anhang B.II.2.1.2 aufgeführt.

Das Schreiben des Drahtmodells in eine X3D Datei und dessen Transformation nach VRML (der z.Z.. von Browsern unterstützte Standard) geschieht in der Klasse `x3d` im Namespace `inc.render`. Zur Transformation von X3D in VRML wird das von X3D Konsortium ausgegebene Stylesheet X3dToVrml97.xsl eingesetzt [Brutzman, D., 2002. URL: http://www.web3D.org/TaskGroup/x3d/translation/X3dToVrml97.xsl. WS]. Es entspricht dem *fourth draft matching x3d-compromise.dtd*.

Das Programm ObjectViewer (was als Desktop Anbindung an das Objektmodell zur Veranschaulichung der Objekte dient) besitzt eine Methode zum Einlesen einer mit Koordinaten und Linien beschriebenen XML Datei in die Klasse `geom`. Ein Beispiel ist die Datei upload.xml die in Klasse Anhang B.II.2.1.2 aufgeführt wird.

Diese Datei wird in einer Instanz von `geom` eingelesen, und das aus `geom` generierte Drahtmodell einer Wand sieht wie in Abbildung 12 aus. Eine Instanz dieser Klasse fungiert als Attribut in einem funktionalen Objekt.

Die Ausarbeitung einer Klasse für die dreidimensionale Methode der Kurven dient der Untersuchung von 3D Simulationsverfahren und der Bereitstellung einer Basisversion zur geometrischen Visualisierung eines Objektes. In der Praxis werden des Öfteren 3D CAD Systeme wie Nemetschek Allplan eingesetzt. Diese bieten auch die Möglichkeit des Datenaustausches in verschiedenen Formaten, u.a. VRML oder X3D.

Moderne CAD Applikationen verwenden ähnliche Klassenstrukturen, um Objekte drei dimensional abzubilden (siehe Abb. 26), außer, die Informationen zu den enthaltenen Flächen werden in dem Objekt selbst festgehalten werden.

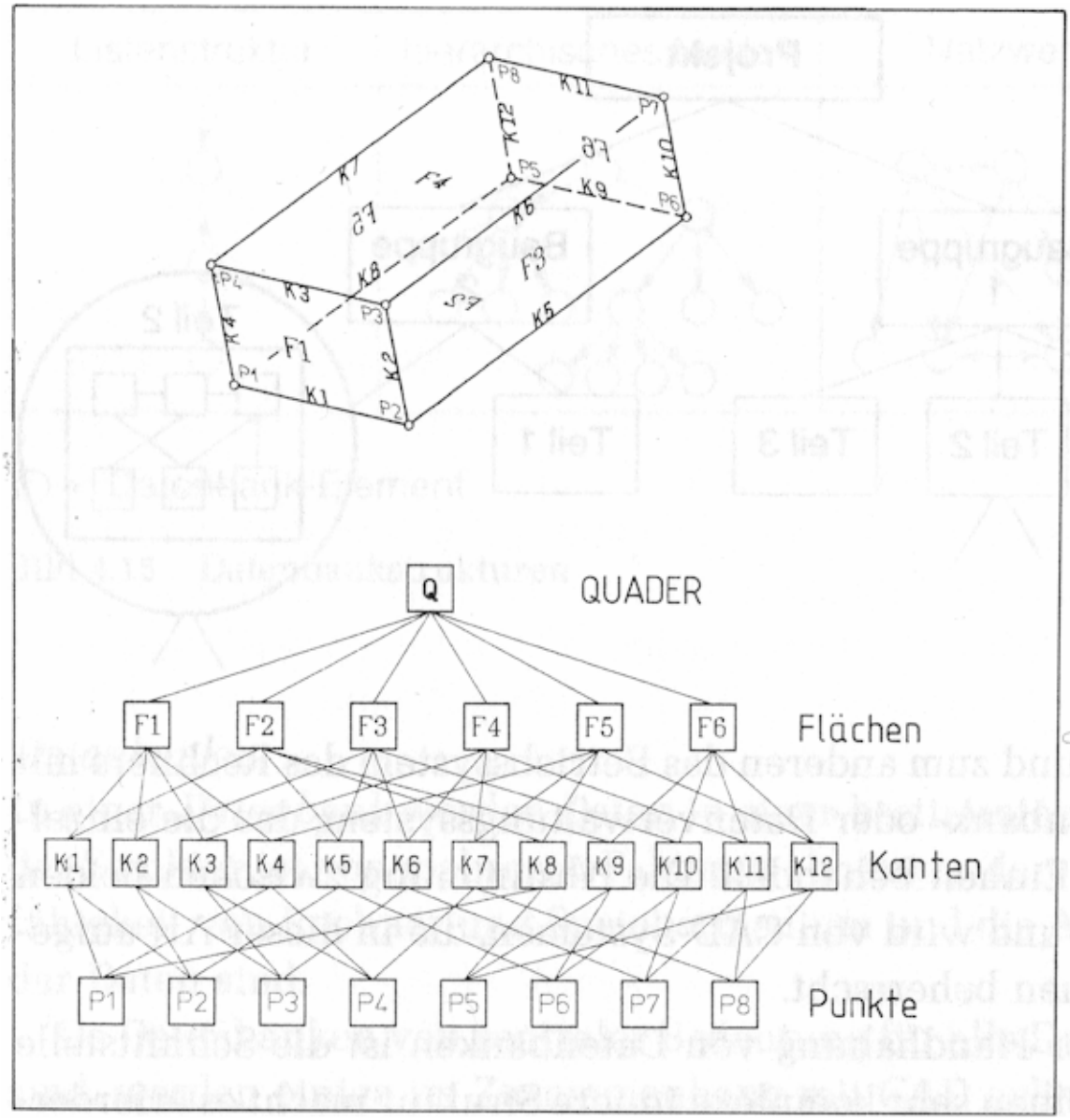


Abb. 26: Reduktion eines Körpers in drei Vektoren mit jeweils Informationen zu Punkten, Verbindungslinien (Kanten) und Flächen) [Helmerich, R. und Schmidt, P., 1985. S. 124] .

2.2 3D CAD

Moderne CAD Anwendungen wie Nemetschek Allplan 2005.1 (.NET basierend) oder AutoCAD sehen eine COM-Schnittstelle zur Automatisierung von Objekten vor. Allplan soll in diesem Objektmodell beispielhaft eingesetzt werden. Prinzipiell ist der Einsatz eines jeden CAD Systems mit einer objektorientierten Schnittstelle und genügend Funktionalität möglich. Die Anbindung an das CAD System erfolgt analog der Anbindung an andere Applikationen wie P3e. Allerdings stellt sich in der Praxis immer wieder heraus, dass solche Schnittstellen aufgrund deren internen Architektur Einschränkungen unterlegen sind, was eine saubere Trennung des Objektmodells von der Schnittstelle fast unmöglich macht.

Diese COM-Schnittstelle kann mit einem Dienst ausgebaut werden und dient somit als XML-basierter Dienst für die JIT(Just In Time)-Lieferung von Objekten. Dadurch, dass die Objekte in einem XML-Format geliefert werden, besteht die Möglichkeit, sie mittels eines geeigneten XSLT in X3D zu transformieren und mit dem von Don Brutzman entwickelten X3dToVrml97.xsl von X3D nach VRML97 (2.0) zu transformieren und in einem VRML 2.0 konformen Steuerelement darzustellen (Abb. 12). Alternativ können die in Allplan vorhandenen Objekte mit einer dll direkt in VRML umgewandelt werden.

Leider ist diese COM Schnittstelle (NAllCOM) nur begrenzt fertig gestellt worden. Es können nur folgende Funktionalitäten ausgeführt werden:

- Allplan aufrufen,
- aktives Projekt auswählen,
- aktives Teilbild auswählen und
- primitive Zeichnungsobjekte setzen.

Die Komplettierung dieses Systems wird mit der Kopplung der NOI Schnittstelle erreicht. Die NOI Schnittstelle steht für Native Object Interface.

2.2.1 Native Object Interface

Die NOI wird meistens in unmanaged C++ erweitert und beschreibt die programmatische Erweiterung des Programms mittels einer Ansammlung von .lib und header Dateien (.h). Diese Dateien beschreiben die Klassen und Objekte und stellen Schnittstellen zur Implementierung von Aufrufen auf diese Objekte zur Verfügung. Folgende Header Dateien und libs werden typischerweise in einer AddOn Applikation mittels des Linkes eingebunden:

- mallpl.lib
- Eing2000FC.lib
- Ctrl2000FC.lib
- Geom2000FC.lib
- Allg2000FC.lib
- NemAll_NOI_ARCH_Objects10.lib

Includes:

- ..\noi_header;
- ..\nem_header;
- ..\Eing2000FC;
- ..\Geom2000FC;
- ..\Ctrl2000FC;
- ..\Allg2000FC

Die Datei NAllComServer.tlb ermöglicht den Zugriff aus einer nativen Umgebung wie C++ auf die NAllCOM Schnittstelle. Tlb Dateien oder Typelibraries wie NAllComServer.tlb werden typischerweise bei der COM Serialisierung eingesetzt [Box, D., 1998. S. 40]. Der Zugriff auf die NOI ermöglicht den nativen Zugriff auf die CAD Objekte in der Applikation Allplan 2005 (Anhang B.II.2.2.1).

2.2.2 Allplan als Objektserver

Mit der Kombination NOI und NAllCOM Server dient Allplan 2005 als JIT Objektgeometrie-Server. Die einzige Bedingung ist, dass die Objektbezeichnung in Allplan 2005 mit der in dem Objektmodell verwendeten Bezeichnung übereinstimmen muss, damit das Objektmodell das Objekt in Allplan 2005 finden kann. Da der Dienst auch schreiben kann (NOI in Allplan 2005 setzen), kann die Objektbezeichnung in der Objekthierarchie des Objektmodells in Allplan sichtbar gemacht werden.

Der Architekt vergibt dem Objekt eine Geometrie, und das Objektmodell zieht diese geometrische Information im JIT-Verfahren ab. Dabei ist der Zustand der Geometrie allein für die (automatische) Objektversionisierung (mittels Klonens) von Interesse. Somit werden im Objektmodell die geometrischen Zustände festgehalten, was in Allplan nicht mehr zu erfolgen braucht.

Im Normalfall jedoch wird der Objektmodellierer lediglich fertige Geometrien aus dem Allplan 2005 Projekt seinen Objekten zuordnen. Diese Objekte können mit den Mitteln der NAIICOM Schnittstelle nur als sämtliche Objekte eines Teilbildes transparent gemacht werden. Eine weitere Differenzierung bietet diese Schnittstelle nicht. Durch den Einsatz der NOI Schnittstelle jedoch kann die Granularität der Auswahl erhöht werden. Diese Kombination aus NAIICOM und neue Schnittstelle jedoch kein kohärentes, geschlossenes und im objektorientierten Sinne gekapseltes System dar. Vielmehr stellt Sie sich als eine Kombination aus

- einer COM Schnittstelle (objektorientiert)
- und einem Windows Handle, der einen Aufruf auf eines mit der NOI Schnittstelle geschaffenen Module ausführt, dar.

Diese Kombination stellt zurzeit die bestmöglich technische Lösung dar und ermöglicht den höchsten Grad der Granularität bei der Zuordnung von Geometrien. Die NOI Schnittstelle unterstützt allerdings nicht alle Objekttypen. Unterstützt werden folgende Typen:

- `(lib, "NemAll_NOI_Base_Objects10.lib")`
- `(lib, "NemAll_NOI_GEO_Objects10.lib")`
- `(lib, "NemAll_NOI_Common_Objects10.lib")`
- `(lib, "NemAll_NOI_BASIS_Objects10.lib")`
- `(lib, "NemAll_NOI_ARCH_Objects10.lib")`
- `(lib, "NemAll_NOI_BIE_Objects10.lib")`

Es sollte daher darauf geachtet werden, mit welchen Objekttypen und mit welcher Kompartmentalisierung die Zeichnungen aufgebaut werden. So stellen z.B. Architekturobjekte anders als Ingenieurobjekte keine Volumeninformationen zur Verfügung.

Die Arbeitsteilung zwischen dem NAllCOM Server und der NOI erfolgt wie folgt:

1. Die Applikation wird über NAllCOM aufgerufen.
2. Das aktive Projekt wird gesetzt.
3. Das aktive Teilbild wird gesetzt.
4. Der Benutzer hat jetzt die Möglichkeit, alle Objekte und deren Attribute soweit vorhanden abzurufen, zu untersuchen und eine Liste von erwünschten Objekten auszuwählen. Folgende Attribute vom Typ `CNOI_FreeAttributePtr` werden unterstützt:
   - `Area = Oberfläche des Körpers`
   - `Volume = Volumen des Körpers`
   - `Text1 bis Text5`
   - `NOI_UUID`
   - `ClassName`
5. Die NOI_UUIDs (Objektidentifikationen) der ausgewählten Objekte werden jetzt als Filter in Form eines delimitierten String Objekts zusammengestrickt und einer entsprechenden Methode des Remote Servers übergeben. Diese Methode iteriert die Objekte des aktiven Teilbilds und filtert sie anhand des übergebenen NOI_UUID Stacks heraus. Somit können die Objekte dediziert angesprochen und samt Attributen herausgefiltert werden.

Diese Vorgehensweise ist nicht leistungsfähig, da eine Zwischenspeicherung auf einem lokalen Datenträger benötigt wird. Bei dem Herauslesen eines Bildes kommen folgende Schritte vor, womit die Plattenzugriffe besonders langsam werden:

1. RemoteAllplanObject wird instanziert (.NET Remoting: Server Call, HTTP Formatter, IIS Hosted).
2. Zugriff über die managed Assembly *cplus02* ermöglicht das Aufrufen von Methoden des unmanaged VRMLServer Objekts.
3. Interne Allplan Berechnungen werden über NOI aktiviert.
4. Zwischenspeicherung der Ergebnisse erfolgt auf lokalen Datenträgern.
5. Weiterleitung der Daten nach Serialisierung über den HTTP Channel im *RemoteAllplanObject.*
6. Aufbereitung der Daten und Zuweisung an Attribute des aktuellen Objektes in der *conobj* Applikation.

Der *VRMLServer* simuliert in Wirklichkeit einen Menüaufruf in der geöffneten Allplan Applikation. Die Win32 Methode `SendCopyDataMessage` ermöglicht das Senden von Botschaften über Prozessgrenzen hinweg. Mit dieser Methode macht der VRMLServer einen Aufruf des vorinstallierten Menüeintrags (PlugIn) in der Allplan Applikation. Der Filterstring sowie Applikationsname, Version und dll (*VRMLServer*) werden mit dieser Methode übergeben. Der Call Stack wird in Anhang (B.II.2.2.2) aufgeführt.

In der Consumer Klasse des *VRMLServer* Utility wird der Aufruf mit der folgenden Methode entgegengenommen und ausgewertet:

```
OnCopyData(FM_COPYDATASTRUCT *pCds)
```

Insbesondere:

```
COPYDATASTRUCT *pData = (COPYDATASTRUCT *)pCds->VPtr[0];
CString *myData = (CString *) pData->lpData;
```

Dieser Pointer kann dann einer Verifizierungsmethode weitergegeben werden, um den Datenzyklus zu prüfen:

```
XMLwriter->writeDllMessageString(*myData);
```

`*myData` enthält den Filterstring der jetzt mittels des Delimeters gesplittet werden kann und durch den Aufruf der Filterfunktion die gewünschten Objekte in folgenden Medien wiedergibt:

- X3D
- VRML
- XML

Aufgrund der noch unvollständigen Implementierung des MessageMaps in Allplan 2005 ist die Schnittstelle noch nicht in der Lage, das Argument `RemoteDLLArgumentString` zu verarbeiten.

Damit jedoch einzelne Elemente einer Zeichnung ausgewählt und einem Objekt zugeordnet werden können, kann ein Umweg über das FileSystem eingesetzt werden. Dieser Umweg wird in Abbildung 27 dargestellt. Über die NAllCOM Schnittstelle wird ein Teilbild angesteuert. Die dazu genutzten API's bzw. Module werden schematisch in Abbildung 28 aufgeführt.

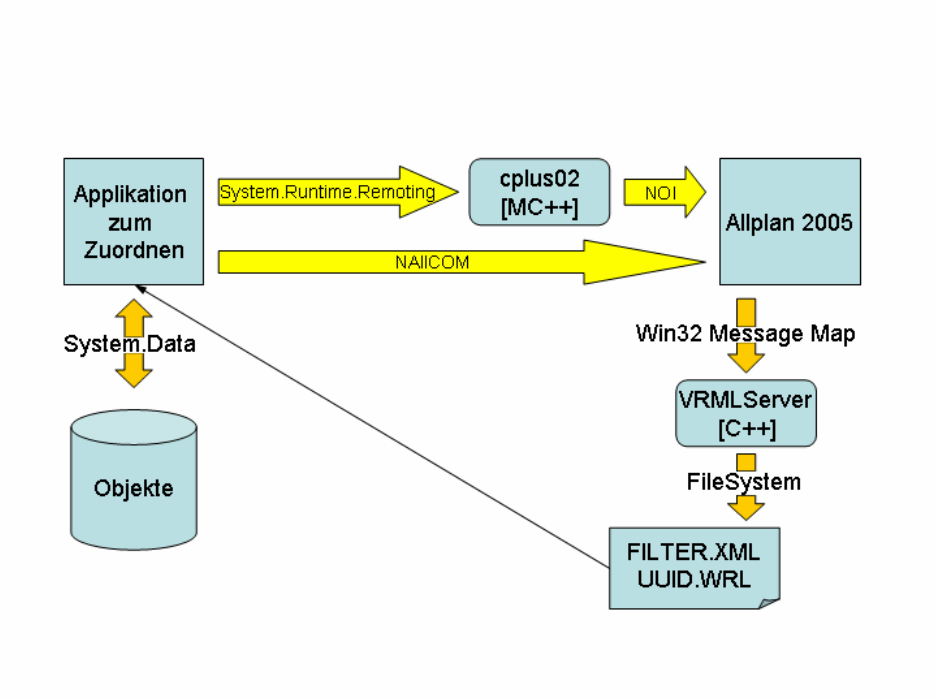


Abb. 27: Schematik des FileSystem-basierten Zugriffs auf die Elemente eines Teilbilds in Allplan 2005.

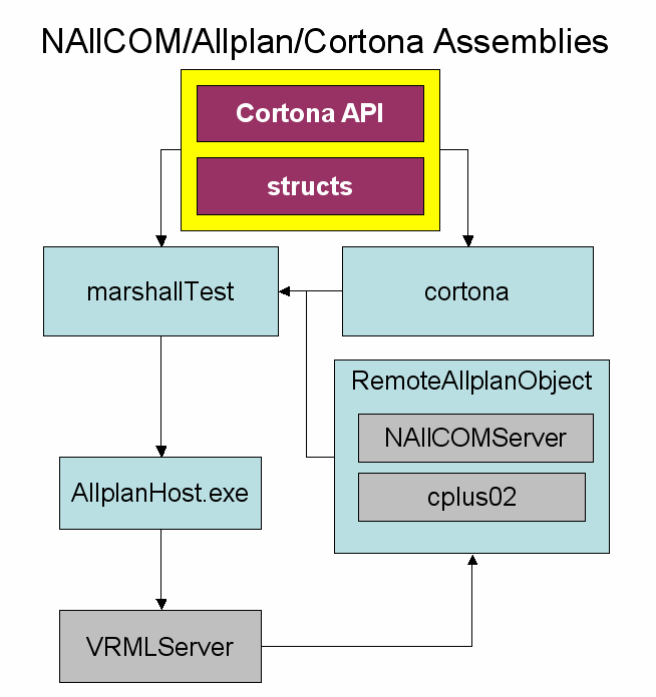


Abb. 28: API's bzw. Module für den pseudo-automatischen Remote Zugriff auf 3D CAD Objekte in Allplan 2005.

Die Objekte eines Teilbilds können mit der statischen Methode

```
void getTypeFilterFromActiveAllplanFile(bool UseHashTable, Filter-
Strings_t &filter, UUIDVector &__v, AttributeStrings &__ats)
```

und einer zusätzlichen Methode verwendet werden, um die UUID's der Elemente auszulesen:

```
NOIID.DrawingUUIDAsString()
```

Weitere Attribute, soweit vorhanden, können wie folgt ausgelesen werden

```
CNOI_FreeAttributePtr pattrArea =
CNOI_Service_FreeAttributes::GetAttributeByNumber(pDB, vec[i],229);
CNOI_FreeAttributePtr pattrVolume =
CNOI_Service_FreeAttributes::GetAttributeByNumber(pDB, vec[i],223);
CNOI_FreeAttributePtr pattrText1 =
CNOI_Service_FreeAttributes::GetAttributeByNumber(pDB, vec[i],501);
CNOI_FreeAttributePtr pattrText2 =
CNOI_Service_FreeAttributes::GetAttributeByNumber(pDB, vec[i],502);
CNOI_FreeAttributePtr pattrText3 =
CNOI_Service_FreeAttributes::GetAttributeByNumber(pDB, vec[i],503);
CNOI_FreeAttributePtr pattrText4 =
CNOI_Service_FreeAttributes::GetAttributeByNumber(pDB, vec[i],504);
CNOI_FreeAttributePtr pattrText5 =
CNOI_Service_FreeAttributes::GetAttributeByNumber(pDB, vec[i],505);
```

wobei die Methode

```
CNOI_String getValue(CNOI_FreeAttributePtr &pAttr)
```

den Zeiger `CNOI_FreeAttributePtr` nutzt, um die Werte der Attribute zu ermitteln.

2.2.3 Zuordnung eines 3D CAD Objekts

Die Informationen der 3D CAD Objekte werden in die Datei FILTER.XML aufgenommen und in dem Verzeichnis abgelegt, welches anhand des NOIID für das Teilbild erzeugt wurde (siehe Abb. 29). Somit hat die aufrufende Applikation oder das Objekt Zugriff auf alle sich in einem Teilbild befindenden Elemente. Diese können dann zur Auswahl (Abb. 30) und Zuordnung (Abb. 32) zu einem Objekt aufgezeigt werden. Abbildung 31 stellt einen Dialog aus Allplan 2005 mit Informationen zu einem Teilbild dar.

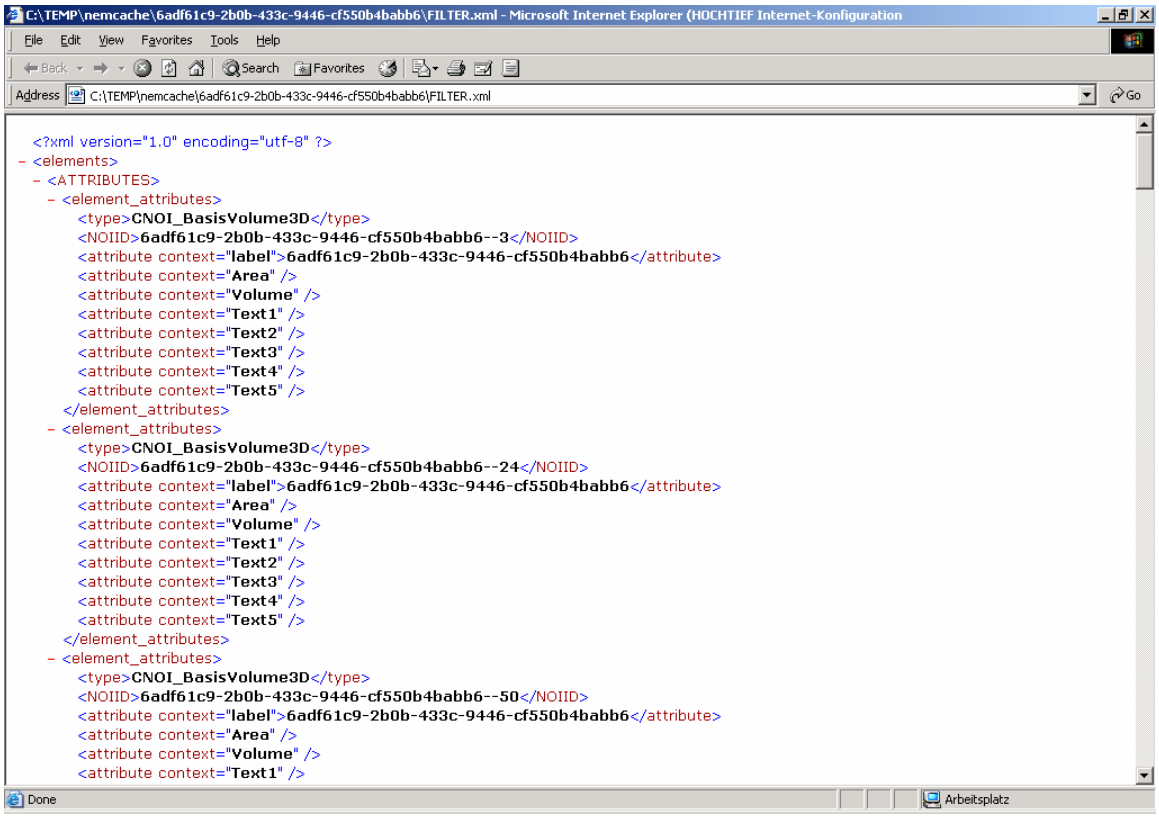


Abb. 29: Datei FILTER.XML mit den in dem Teilbild 3D_EG.A_Kerne erhaltenen Elementen, wird in Verzeichnis C:\TEMP\nemcache\6adf61c9-2b0b-433c-9446-cf550b4babb6 von VRMLServer.dll abgelegt.

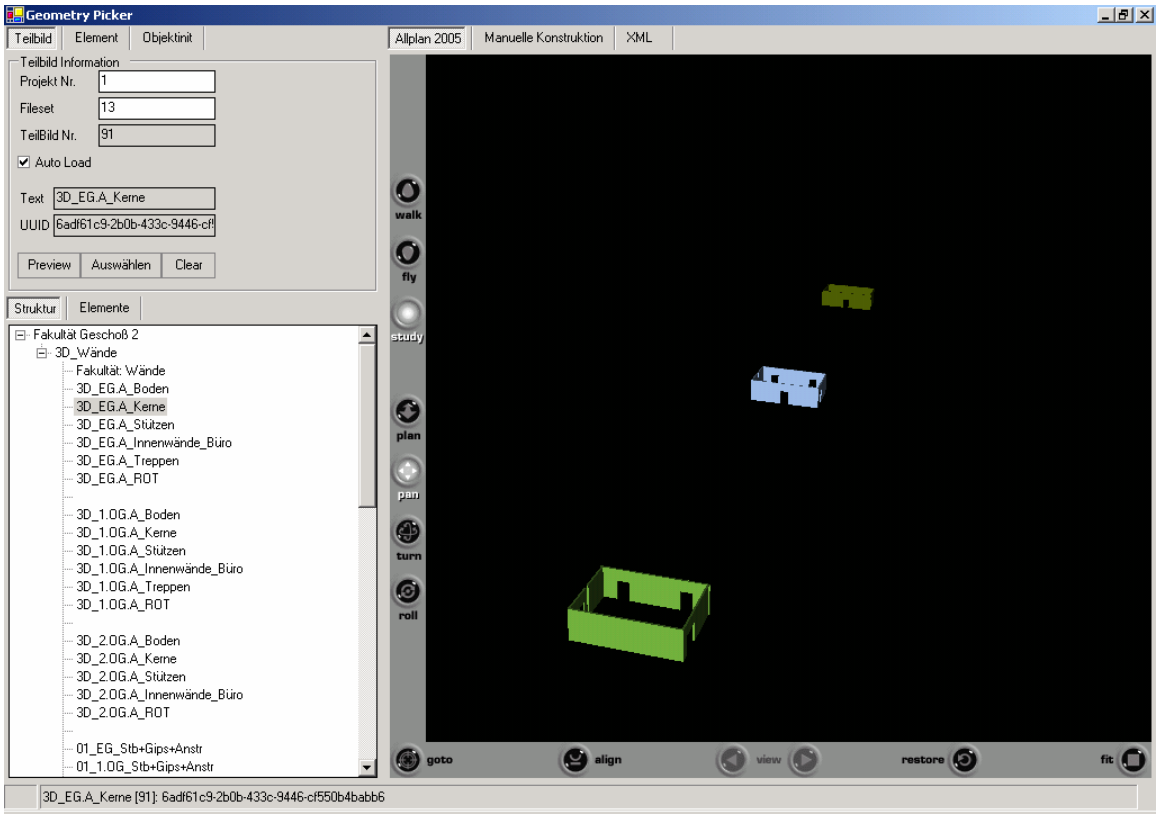


Abb. 30: Auflistung der Teilbilder in der Zeichnung 3D_Wände (UUID: 6adf61c9-2b0b-433c-9446-cf550b4babb6). Das Teilbild 3D_EG.A_Kerne ist ausgewählt worden.

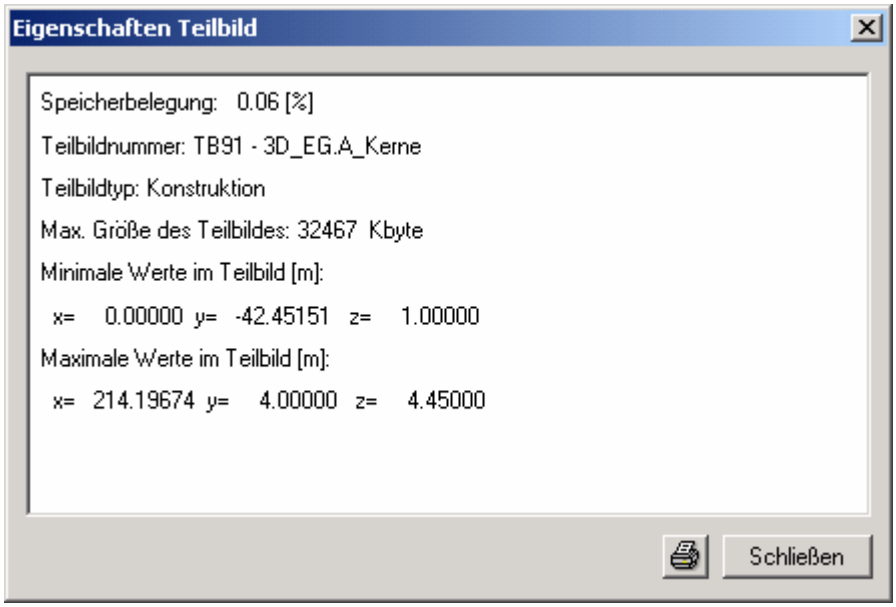


Abb. 31: Eigenschaften des Teilbilds 3D_EG.A_Kerne

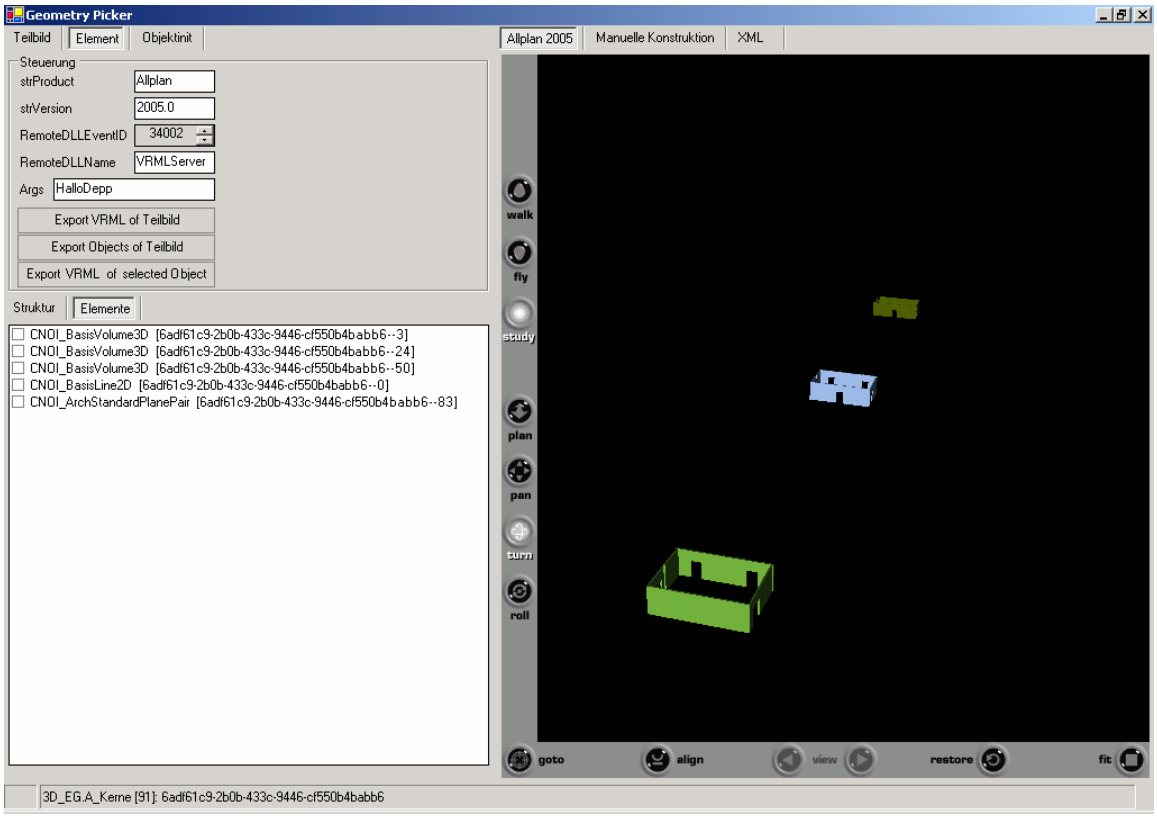


Abb. 32: Auflistung der Elemente in dem Teilbild 3D_EG.A_Kerne.

Nach Auswahl eines oder mehrerer Elemente in dem Teilbild werden diese dem aktuellen Objekt zugeordnet (Abb. 33). Die Eigenschaft *NodeRep* des Objekts enthält alle relevanten Daten für den Zugriff auf Allplan2005.

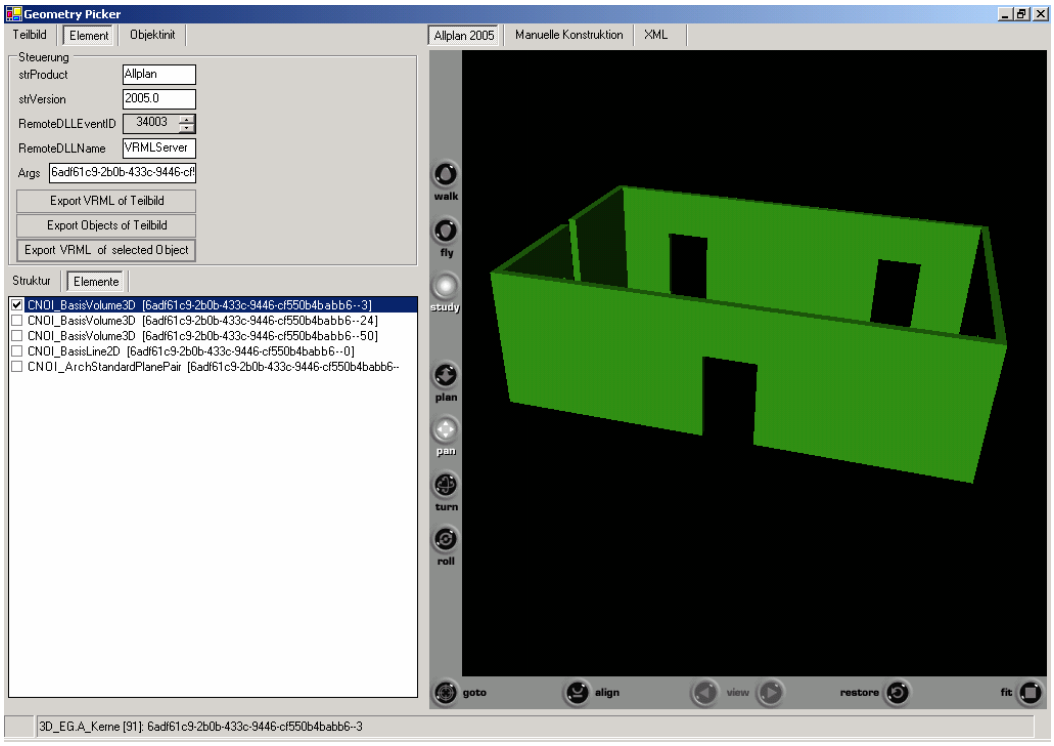


Abb. 33: Ein Element (NOIID: **6adf61c9-2b0b-433c-9446-cf550b4babb6—3**) im Teilbild 3D_EG.A_Kerne wurde ausgewählt. [8]

Weder die NAllCOM Schnittstelle noch das NOI unterstützen Ereignisse in ihren Objektmodellen. Daher ist es nicht möglich, Ereignisse in Allplan abzufragen. Ereignisse aus Allplan würden Informationen darüber ausgeben, welche Objekte verändert wurden und welche Änderungen in den Geometrien der ausgewählten Objekte stattgefunden haben. Informationen über solche Änderungen wären wichtig, um die Eigenschaft Geometry des Objektes zu aktualisieren.

Stattdessen wird in Intervallen nach dem aktuellen Wert abgefragt. Das *conobj* Objekt startet bei der Instanzierung mittels der `init()` Methode im Konstrukt einen serverseitigen Timer. Dieser Timer startet einen MTA Thread, der in vorgegebenen Zeitintervallen das Ereignis

```
private void tmrTimersTimer_Elapsed(object sender,
System.Timers.ElapsedEventArgs e)
```

aufruft.

---

8 Dieses Element wird den Kontextdaten (Projekt, Zeichnung, Teilbild, NOIID) der Objekteigenschaft NodeRep zugeordnet. Diese Informationen werden vom Objekttimer verwendet, um mittels der Remoting Infrastruktur die Eigenschaft Geometrie mit der neuesten Darstellung des Elements zu versehen.

Dieses Ereignis steuert die Aktualisierung der Eigenschaften des Objektes und bedient sich bei Eigenschaften, die aus anderen Applikationen stammen, der nötigen Remoting oder Services Objekte. Im Fall der Geometrie wird dazu das Remoting Objekt `RemoteAllplanObject.RemotableType,` was von `MarshalByRefObject` abgeleitet wurde, eingesetzt.
Der Aufruf eines Remoting Objektes, welches wiederum ein COM Objekt aufruft, ist durch die Änderungen in der Architektur von .NET gegenüber COM problematisch. Das Objekt `RemoteAllplanObject.RemotableType,` was wiederum über das MC++ Modul *cplus02* auf die COM Schnittstelle NAllCOMServer zugreift, ruft das ältere COM Objekt in C++ mit der Methode

```
HRESULT hr = ::CoInitializeEx(NULL, COINIT_APARTMENTTHREADED );
```

(STA – Single Threaded Apartment) auf. Da aber der Serverseitige Timer grundsätzlich ein MTA (Multi-Threaded Apartment) Thread startet und ein Wechsel zwischen STA und MTA während der Laufzeit sowohl von .NET[9] wie Win32[10] untersagt ist, müssen alle Steuerungen, die auf COM zugreifen, außerhalb dieses Threads und dem MC++ Modul erfolgen.

Das Ergebnis der timergesteuerten Aktualisierung wird in (Abb. 34) dargestellt. Dort hat der Timer 23 Intervalle durchlaufen. Zu Testzwecken wurde die Periode des Intervals auf 5 000 Millisekunden gesetzt. In diesem Fall gab das Modul *VRMLServer* eine beliebige Farbe für das Objekt aus, was während der Entwicklung dazu genutzt wurde, um festzustellen, ob das Modul aktiv ist.

---

9 Im .NET Framework wird der Aufruf

`Thread.CurrentThread.ApartmentState = ApartmentState.STA`

ignoriert, wenn der Thread bereits gestartet wurde und vom Typ MTA ist.

10 Ein Versuch das Apartmentmodel in Win32 während der Ausführung eines Threads zu verändern führt zur Rückgabe des HRESULT Wertes RPC_E_CHANGED_MODE. In der Win32 Dokumentation für CoInitializeEx wird dieser Rückgabewert wie folgt kommentiert: „Once the concurrency model for a thread is set, it cannot be changed. A call to **CoInitializeEx** on a thread that was previously initialized with a different concurrency model will fail and return RPC_E_CHANGED_MODE. „ [MSDN: Plattform SDK: COM **- COM Documentation Team Release: October 2002**]“.

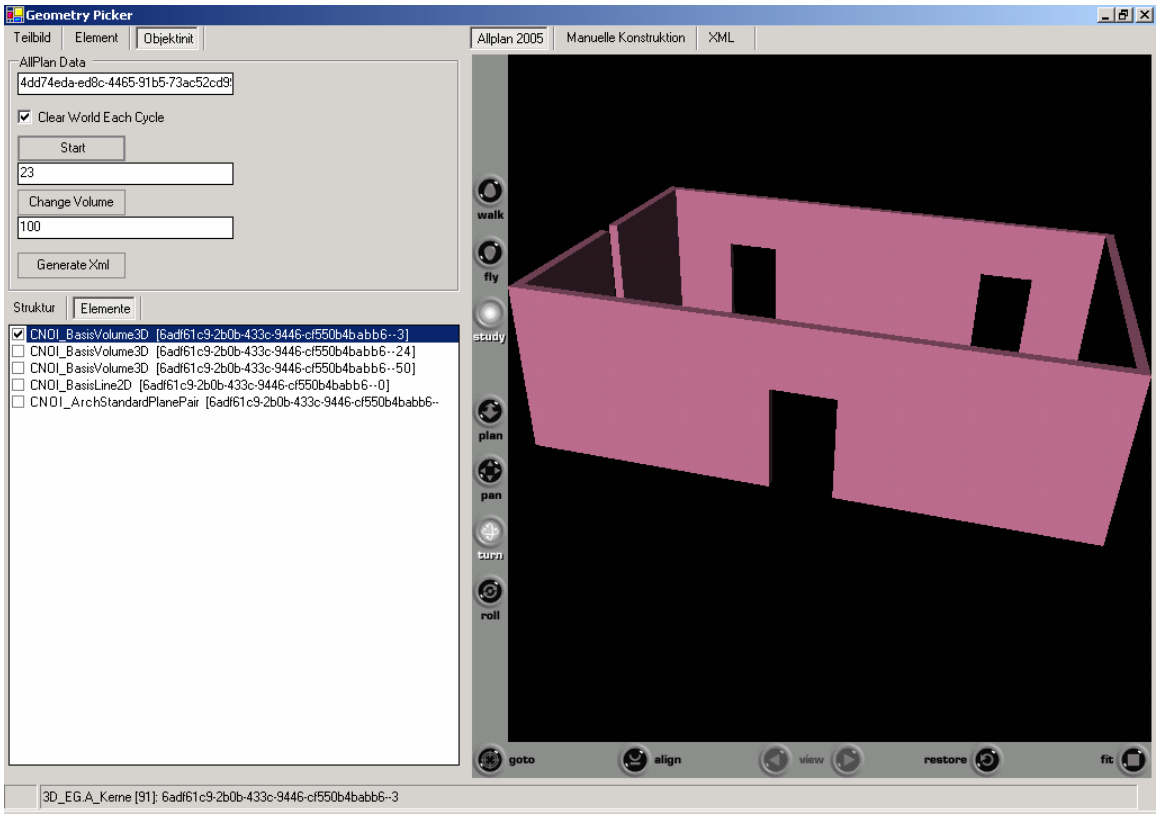


Abb. 34: Nach 23 Intervallen des Timers gibt das Modul VRMLServer eine rosa gefärbte Version des ursprünglich gewählten Elements aus Allplan 2005 aus. [11].

2.2.4 Vorteile der Auslagerung von 3D CAD Objekten

Die Kombination der NOI und des NallCOM servers ermöglicht einen semi-automatisierten Zugriff auf CAD Objekte. Das ist eine große Verbesserung gegenüber den Schnittstellen STEP oder IFC, da durch die COM Schnittstelle alle Informationen dieser Objekte und die Relationen zwischen den Objekten und anderen Applikationen zugänglich gemacht werden können. Das Argument, dass IFC oder STEP eine plattformunabhängige Verarbeitung der Objekte erlaubt, ist in Anbetracht der weit verbreiteten Standards XML, XML Schema und SOAP nicht mehr aktuell. Objekte können nun aus Allplan im XML Format mittels SOAP serialisiert werden und dann entweder in anderen Applikationen weiter verwendet werden, oder mit Technologien wie OpenGL, VRML oder SVG plattformunabhängig dargestellt werden. Der in dieser Arbeit verwendete Ansatz ist verbesserungsbedürftig, bietet aber eine erste Erfahrung mit der Trennung von 3D CAD Objekten aus der CAD Anwendung heraus.

---

[11] Dieses Element wird mittels Timer und Remoting Infrastruktur in Intervallen dem conobj Objekt zugeordnet

III Ergebnis der Trennung

1 Objektabstrahierung

Durch die Objektmodellierung wird versucht, Objekte im vier-dimensionalen Raum (3D + Zeit), mit Hilfe von objektorientierten Techniken (Vererbung, Attributen und Methoden) zu definieren und zu erfassen. Typischerweise wird z.B. eine Stütze durch deren Geometrie, räumliche Position (Koordinaten), Materialspezifikation, allgemeine Attribute (wie z.B. Eigentümer, ausführende Fa., Datum der Installation), beschrieben, womöglich verknüpft mit mehreren Dokumenten/Daten wie Statik, Gutachten, Plänen, Ausführungsbeschreibungen, Kalkulationen, juristischen Informationen usw.

Wie oben schon besprochen, ist die geometrische Abgrenzung der Objekte oft schwierig oder auch beliebig und vom Standpunkt des Betrachters abhängig. Hinzu kommt die große Anzahl an Attributen, welche ein Objekt ausmachen. Das sich stark verändernde Umfeld beeinflusst die Terminierung und die Kostenschätzung, die auf den Zustand eines Objektes während der Entwurfsphase entscheidend einwirken und somit auch dessen Darstellung in 4D CAD Zeichnungen, Terminplänen usw.

Durch die Abstrahierung aller Objekte in einem geometrisch genau definierten imaginären Projektraum, in welchem z.B.

- Geometrien in zwei Dimensionen erfasst werden (mit Hilfe von der Topologie),
- die Zeit als Dimension bleibt oder zu einem Attribut wird,

kann ein klarer Objektraum ohne unnötige Details ermittelt werden, der programmatisch wesentlich leichter zu erfassen ist. Dieser kann durch eine Transformation immer in die virtuelle multidimensionale (3D + Zeit + weitere Dimensionen) Welt verwandelt werden. Ähnliche Verfahren wurden trotz ihrer Komplexität erfolgreich mathematisch erfasst.

2 Mischformen von Gegenständen

Bei der Planung, insbesondere bei starker Veränderung ihrer Ergebnisse und bei komplexer qualitativer Gestaltung, treten oft geometrische Gebilde auf, die nur unter schwierigkeiten oder auch beliebig in funktional sinnvolle Einheiten einzuteilen sind. Hinzu kommt das Problem der Sichtweise. Zur Veranschaulichung das einfache Beispiel eines Hauses: Die inneren Trennwände können jeweils in einzelne Objekte aufgeteilt werden; es können aber auch die Zimmer als Objekte betrachtet werden. Von außen her werden die Außenwände als kontinuierliche Segmente (Objekte) gesehen.

Die Trennung nach Objekten von innen und außen führt zu unterschiedlichen Ergebnissen. Hinzu kommt das Problem der Trennungsfugen. Sollen rechtwinklig aneinander stehende Wände zu einem 45° Winkel oder mit einer Überlappung getrennt werden?

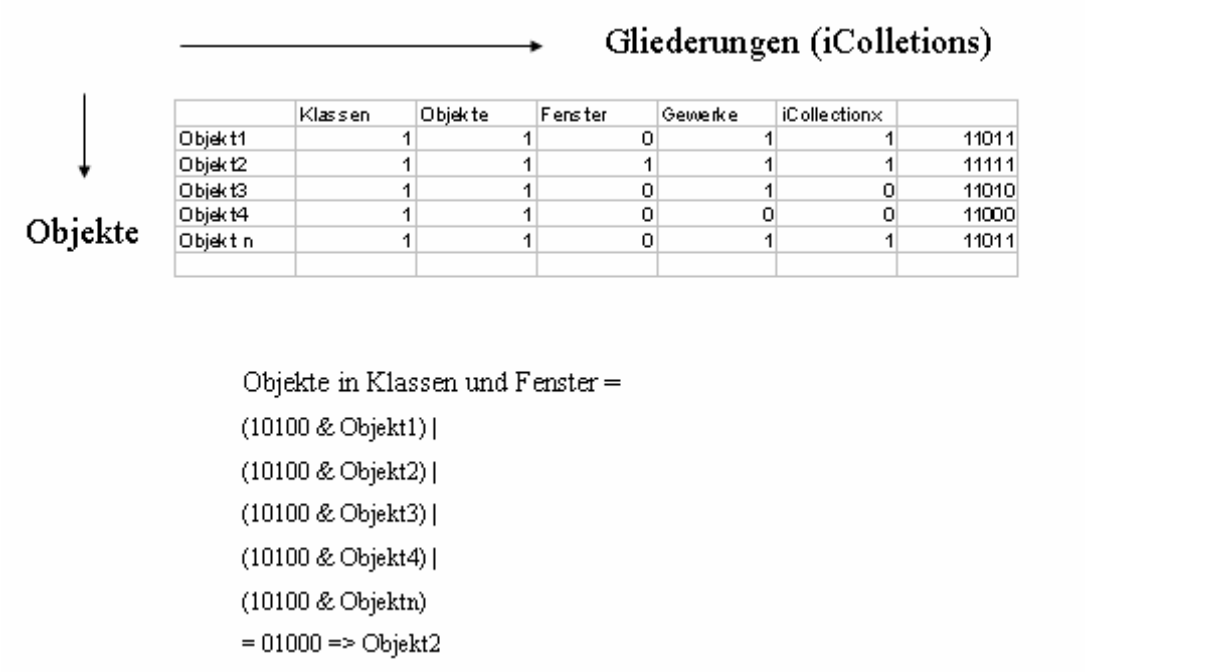
Eine doppelte Sichtweise ist auch erreichbar (Innen- und Außenansicht mit unterschiedlichem Kontext):

- Die Objekte der Innenansicht werden als Gruppe gegliedert. Bei der Auswahl der Gruppe erscheinen sie als kohärentes Gebilde, z.B. ein Raum. Die Methoden der Kostenverdichtung und Terminierung greifen auch auf diese Gruppe so über, dass zusätzliche Kostenelemente und Terminelemente entstehen.
- Die Objekte in der Außenansicht werden gruppiert. Die Auswahl dieser Gruppe hebt die Außenwände hervor. Die eingebauten Methoden der Kostenverdichtung und Terminierung ergeben Kosten und Termine für die Außenwände.

Bei architektonisch aufwändigen Formen und/oder technisch anspruchsvollen Gebilden (z.B. Tunnelbau oder im Anlagenbau) entsteht das Problem der Aufteilung. Oft ist eine Aufgliederung gar nicht nötig und kann das ganze Gebilde als eine funktionale Einheit fungieren. Das kommt vor, wenn das Gebilde Bestandteil der Leistung einiger Unternehmer wird, oder von einem Planungsbüro erarbeitet wird. Damit wird die komplexe Aufgabe der Unterteilung auf den Auftragnehmer verlegt. Von Ihm kann das Gebilde in kleinere Objekte zerlegt werden, und mit Hilfe einer Gliederung als Sammelobjekt im Projektraum zur weiteren Bearbeitung zur Verfügung gestellt werden.

2.1 Auswertung der Gruppen

Die beiden Gruppierungen überschneiden sich. Da beide Gruppierungen auf der Klasse *iCollection* basieren, die wiederum die Informationen der beinhalteten und überlappenden Knotenpunkte in Hashtables gespeichert halten, können in einer Tabelle die überlappenden Knotenpunkte isoliert werden, um den Grad der Überlappung zu identifizieren. Das geschieht mit Bit-Arithmetik und entspricht einem JOIN Ausdruck in SQL. Ein Beispiel wird in Abbildung 35 dargestellt. Eine andere Möglichkeit der Auswertung geschieht mittels XQuery. In Teil C wird der Einsatz von XQuery zur Auswertung von Objekten demonstriert. Bit-Arithmetik ist zwar leistungsfähiger als XQuery, muss aber in einer XQuery Abfrage eingebettet sein. Daher ist es sinnvoller trotz Leistungseinbüßen die XQuery Funktionalitäten für eine solche JOIN Abfrage zu benutzen.



	Klassen	Objekte	Fenster	Gewerke	iCollectionx	
Objekt1	1	1	0	1	1	11011
Objekt2	1	1	1	1	1	11111
Objekt3	1	1	0	1	0	11010
Objekt4	1	1	0	0	0	11000
Objekt n	1	1	0	1	1	11011

Abb. 35: Eine binäre Vorgehensweise bei einer kombinierten Abfrage von Gruppen oder Strukturen.

2.2 Übergang von abstrakter zu geometrischer Form

Das Anlegen eines Objektes ohne geometrische Abbildung ergibt ein geometrisch abstraktes Objekt; abstrakt deshalb, weil das Objekt bislang keine geometrische Form besitzt. Ansonsten kann das Objekt als ein funktionales Gebilde betrachtet werden. Es besitzt laut der Klasse *conObj* oder einer Ableitung das Attribut *Geometry*. In diesem Attribut, das als *uml:object* definiert wurde, kann eine geometrische Darstellung angegeben werden. Das X3D Pendant, das im abstrakten Fall eine graue Kugel darstellt, wird automatisch im entsprechenden Attribut aktualisiert. Dieses Attribut kann mit einer Applikation zur geometrischen Modellierung, wie AutoCAD oder Allplan, gefüllt werden. Eine solche Applikation sollte gleichzeitig eine X3D Repräsentation erstellen, da dieses Format als allgemeine geometrische Schnittstelle zwischen den vielfältigen CAD Formaten dienen kann. Es besitzt genügend Funktionalität im Bereich der Anschaulichkeit und Interaktion, um baubetriebliche Untersuchungen unterstützen zu können.

Die Modellierung abstrakter oder nicht im Detail definierter Gedanken sowie die Modellierung von Objekten bzw. stellvertreter Objekten gelingt trotz geometrischer Ungenauigkeit (in den Anfangs – oder auch zu allen aderen Phasen). Die eher intuitive Praxis, bei der Stellvertreterobjekte während der anfänglichen Phasen eines Projektes eingesetzt werden, findet sich auch bei der Erstellung eines Angebots, Terminplans, einer Zeichnung, der Projektdokumentation usw.

3 Auswirkungen einer objektorientierten Modellierung

3.1 Planung und Angebotsbearbeitung

Aus Sicht der Planungssteuerung, der Terminplanung, des Vertragsmanagements und der Ausführung, ist die Dokumentation der Planung und vor allem deren Änderungen ein wichtiger Bestandteil des gesamten Projektes. Das liegt daran, dass Änderungen in der Planung nichtlineare Folgen in der Ausführung haben. Die Planung ist meistens auch Bestandteil der Vertragsdokumentation und somit Grundlage für andere Planungsinstrumente wie Kostenschätzung und Terminplanung.

Änderungen in der Planung wirken sich oft in diesen Disziplinen aus. Somit ist die lückenlose Dokumentation der Änderungen von Wichtigkeit. Sie erfolgt dadurch, dass Änderungen in der geometrischen Planung (CAD) automatisch mittels der An-

bindung an das Objektmodell dokumentiert werden. Ein wichtiger Punkt ist die Granularität der Änderungen, die ein Ereignis auslösen. Dieses Kriterium muss so justiert werden, dass eine Flut von unwesentlichen Änderungsmeldungen vermieden wird. Da Änderungen in der CAD Planung sich auch auf die Terminplanung und Kalkulation auswirken, ist die Integration dieser Disziplinen während der Planungsphase wünschenswert. Die Applikationen für die Terminplanung und Kalkulation können Änderungen im Objektmodell automatisch über die Anbindung mittels Dienste wahrnehmen und darstellen. Letztlich müssen Regelwerke erstellt werden, die wiederum Ereignishändler definieren. Diese Ereignishändler reagieren auf Änderungen, die für ein Projekt als wichtig erachtet werden. Das Erstellen solcher Regeln ist ein komplexes Unterfangen und, abgesehen von der Demonstration der Machbarkeit, nicht Bestandteil dieser Arbeit.

3.2 Projektmanagement

Das Projektmanagement beschäftigt sich überwiegend mit der Planung und Verwaltung von Arbeitsabläufen, der Verwaltung von logistischen Prozessen, Kostenkontrolle, Dokumentation von Ereignissen und Steuerung menschlicher Interaktionen rund um ein Projekt. Diese Aktivitäten werden zunehmend mittels EDV-technischer Anwendungen durchgeführt, als unterstützende Medien und auch als Steuerungsinstrumente. Dadurch können Änderungen in den Modellen elektronisch erfasst werden, in soweit diese Applikationen genügend Funktionalitäten in ihren Schnittstellen (API's) bereitstellen.

Die Kopplung dieser Instrumente mit dem Objektmodell würde solche Änderungen automatisch dokumentieren und das integrierte Objektmodell entsprechend fortschreiben. Das Objektmodell wurde bewusst so konzipiert, dass aktuelle und zukünftige Anwendungen nicht verändert oder als integrierte Paketlösungen eingesetzt werden müssen, da ein solcher Ansatz unpraktikabel ist. Projektbeteiligte können gewohnte und effiziente Arbeitsweisen weiterhin verfolgen, das Objektmodell dokumentiert diese Informationen unabhängig von den einzelnen Applikationen. Irrelevante und nichtige Änderungen können herausgefiltert werden, indem bestimmt wird, welche Anwendungen mit dem Objektmodell verbunden werden und welche Teile der auf diesen Anwendungen befindlichen Daten übertragen werden. Auch wenn Ereignishändler schon definiert wurden und Ereignisse ausgelöst wurden, können diese nachträglich abgefangen, protokolliert und stillgelegt werden.

3.3 Nachtragsmanagement

Ein wichtiger Baustein für das Nachtragsmanagement ist die Identifizierung und Analyse baubetrieblicher Störungen. Störungen können wie folgt umschrieben werden [Mechnig, M., 1998. S. 104]:

> *„Störungen sind zeitlich befristete Zustände einer Wertschöpfungskette, in denen durch das Einwirken von Störgrößen auf die Produktionsfaktoren und deren Kombinationsprozess eine unmittelbar festgestellte Abweichung vom geplanten (optimalen) Prozessverlauf und/oder dessen Ergebnis entsteht."*

Eine Behinderung wird definiert als:

> *„Baubetrieblich sind Behinderungen ein Unterfall von Störungen."*

Für die Betrachtung einer ganzheitlichen Erfassung möglicher Störungen sind Behinderungen also als Untermengen von Störungen zu behandeln. Die Differenzierung erfolgt im Aufbau des Ereignismodells. Ursprung und Art der Störung sind nicht relevant für die technische Modellierung, da diese als Bestandteil des Ereignismodells, das nach fachlich sinnvollen Kriterien aufzubauen ist, eine Störung ausmachen. Wichtig ist lediglich, alle Objektänderungen, die im fachlichen Sinne eine Störung verursachen können, zu identifizieren und mit Ereignissen zu versehen. Die fachliche Differenzierung erfolgt dann in der Ereignisbehandlung oder nachgelagerten Filtern. Dort werden die entsprechenden Methoden mit einem verteilten Aufruf ausgeführt, die wiederum entsprechend den fachlichen Anforderungen Änderungen in dem Objektmodell bewirken.

Das Objektmodell sieht bei Objektänderungen ein Klonen des Objektes vor. Daher wird das Objekt exakt kopiert, das kopierte Objekt deaktiviert und in der Kopie die Änderung angebracht. Diese Änderung löst natürlich auch ein entsprechendes Ereignis aus. Somit ist der Stand vor der Änderung dokumentiert und eine Ereignisbehandlung zur Verarbeitung weiterer assoziierter Informationen gegeben.

Ein komplexes Ereignismodell kann zu vielen gleichzeitigen Aufrufen führen. Deren Abarbeitung setzt eine globale MessageMap voraus und stellt ein Leistungsproblem dar, da auch performante Server mit einem solchen kaskadierten Modell schnell überfordert sind noch. Problematischer wird das Auftreten einer Kette von Ereignissen durch den interaktiven Aufruf von Ereignissen untereinander. Diese Situation kann in zwei Fällen auftreten:

- Ereignisbehandlungen rufen weitere Methoden auf.
- Attribute kapseln Objekte, die bei Änderungen wiederum Ereignisse auslösen.

Damit kein geschlossener Kreislauf entsteht, ist eine Ereignisverfolgung nötig. Wenn bei einer solchen Ereignisanalyse alle Ereignisse nur einmal vorkommen, ist ein geschlossener Kreislauf ausgeschlossen.

Eine Ausnahme stellt die Berechnung von Optimierungen dar. Diese erfordern oft eine Annäherung mit Hilfe von iterativen Verfahren. Eine solche Annäherung würde unter normalen Umständen zu einem geschlossenen Ereigniskreislauf führen, somit sollte während einer Optimierung die Ereignisbehandlung ausgeschaltet oder modifiziert werden. In diesem Falle ist es sinnvoll, die Optimierung zu abstrahieren und nur das Ergebnis in das Objektmodell einzuarbeiten, da sonst durch Inanspruchnahme von Rechnerleistung unnötige temporäre Zwischenstände generiert würden.

IV Dokumentation der Ereignisse

1 Das Wesen der Ereignisse

Bei der Entwicklung eines Ereignismodells ist die Definition eines Ereignisses im IT-technischen Sinne hilfreich [.NET Framework Developers‘ Guide, 2002. WS]:

> *Ereignisse sind im Voraus definierte Signale, die von Objekten ausgegeben werden beim Eintreten vordefinierter Zustandsänderungen. Objekte selbst können Ereignisse ausgeben. Die Objekte selbst kapseln eine Ereignisbehandlung, die mit Hilfe von delegates den Ereignissen zugeordnet wurde. Diese delegates wiederum kapseln die Ereignisbehandlung.*

Ein Ereignis wird als *Event* definiert. Dazu gehört eine Behandlungsfunktion, die beim Auftreten des *Events* aufgerufen wird. Die Funktion kann innerhalb einer Methode mit dem Befehl RaiseEvent aktiviert werden.

Ereignisse können sowohl auf der Objektebene wie auch der Projektebene verarbeitet werden. Wenn Ereignisse innerhalb eines Objektes bearbeitet werden, haben diese, soweit keine expliziten Interaktionen zwischen Eigenschaften existieren, z.B. zwischen Volumen und Fläche, keine weiteren Auswirkungen.

Das .NET basierte Ereignismodell benötigt drei Komponenten:

- Einen delegate für den Handler
- Eine Klasse zur Handhabung der Ereignisargumente
- Einen Auslöser vom Typ OnEvent

Im Folgenden wird der Code in der Klasse constructionobject aufgeführt, der delegate sowie die Klasse zur Kapselung der Ereingisargumente:

```
public delegate void PropertyChangeEventHandler(object sender, Proper-
tyChangeEventArgs fe);
/// <summary>
///ObjectEventArgs: a custom event inherited from EventArgs.
/// </summary>
public class PropertyChangeEventArgs : EventArgs
{
      public PropertyChangeEventArgs(Guid objectId,
            string attribute, object OldValue, object NewValue)
      {
            this.objectId = objectId;
            this.attribute = attribute;
            this.OldValue = OldValue;
            this.NewValue = NewValue;
      }

      // The object change event will have 4 pieces of information--
      // 1) Which object caused the change
      // 2) Which attribute was changed

      public Guid objectId;
      public string attribute;
      public object OldValue;
      public object NewValue;

}    //end of class ObjectEventArgs
```

Innerhalb einer Eigenschaft kann geprüft werden, ob sich diese Eigenschaft im Vergleich zum vorherigen Zustand verändert hat. Dieser Vergleich kann wiederum einen Klon-Vorgang auslösen (Abb. 36):

```
[XmlIgnore]
public string _Geometry;
public string Geometry
{
      get
      {
            return this._Geometry;
      }
      set
      {
      //here we call the delegate if we detect a change in geometry
      //using string comparison
            this.OldProperty =
            this._Geometry==null?string.Empty:this._Geometry;
            this._Geometry = value;
            PropertyChangeEventArgs fe = new PropertyChangeEven-
            tArgs(this.UniqueId, "Geometry", this.OldProperty, value);
            PropertyChange(this.UniqueId, fe);
      }
}
```

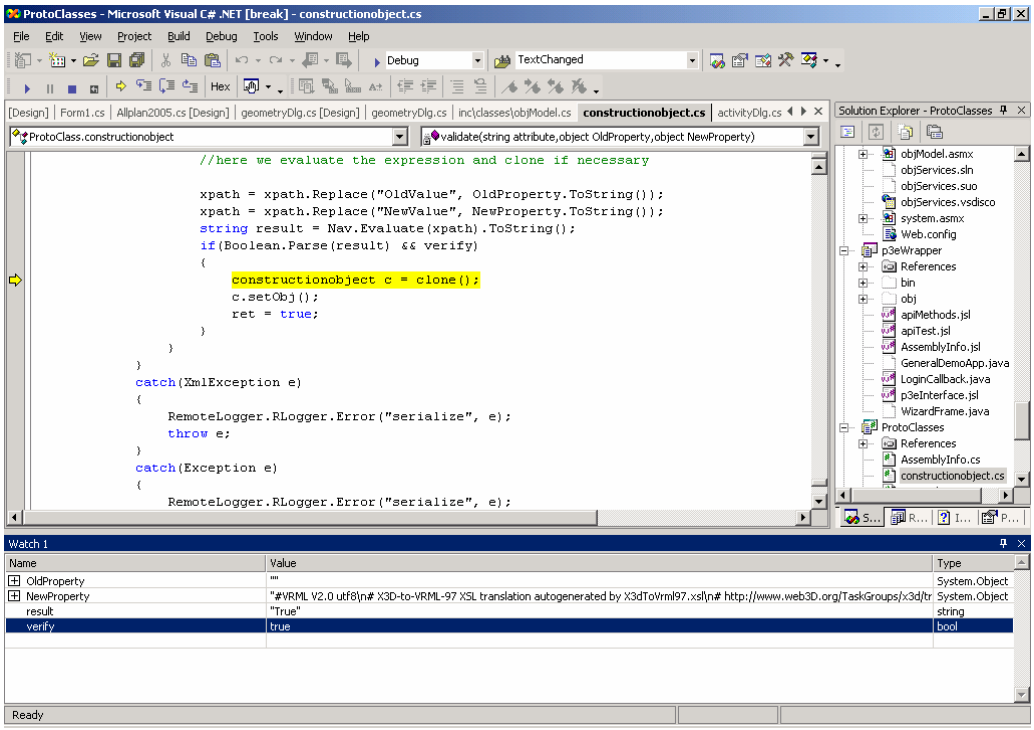


Abb. 36: Klon-Vorgang wird aufgrund der Änderung in der Eigenschaft Geometrie ausgelöst. Blick in den Debugger.

Dieser Klon-Vorgang bewirkt das Anlegen eines neuen Objektes. Abbildung 38 stellt die Situation in dem Terminplanungsprogramm unmittelbar vor dem Klon-Vorgang dar.

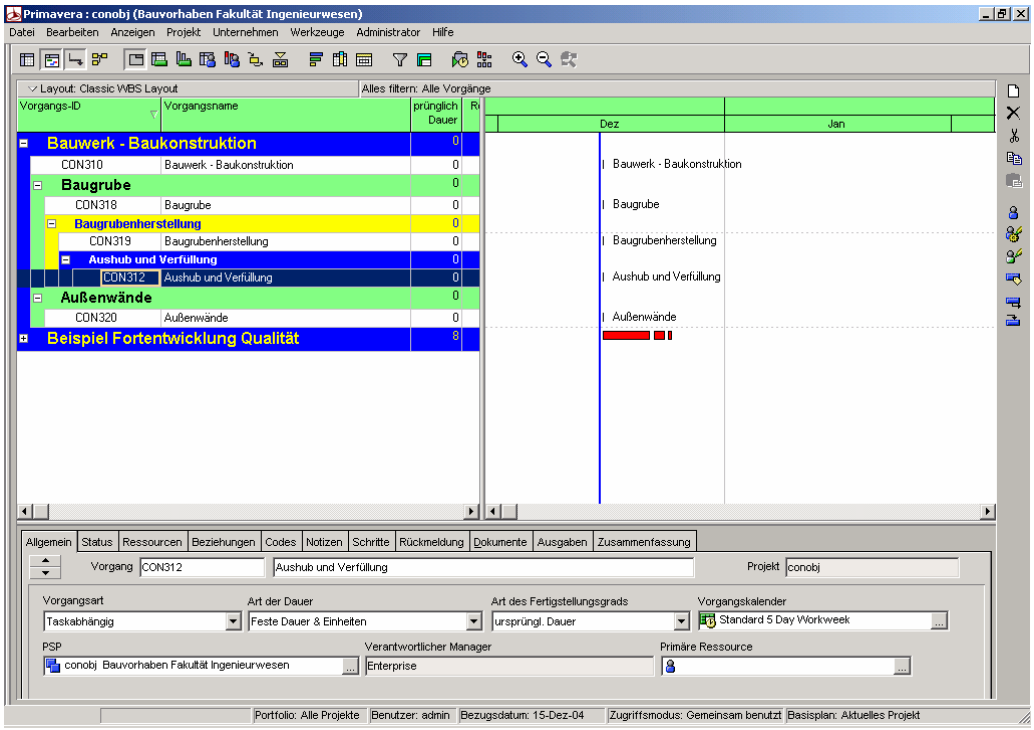


Abb. 37: Situation in der Terminplanung unmittelbar vor dem Ereignis PropertyChange.

Abbildung 38 stellt die Situation unmittelbar nach dem Klon-Vorgang dar. In der Kalkulation erfolgt ein ähnlicher Prozess.

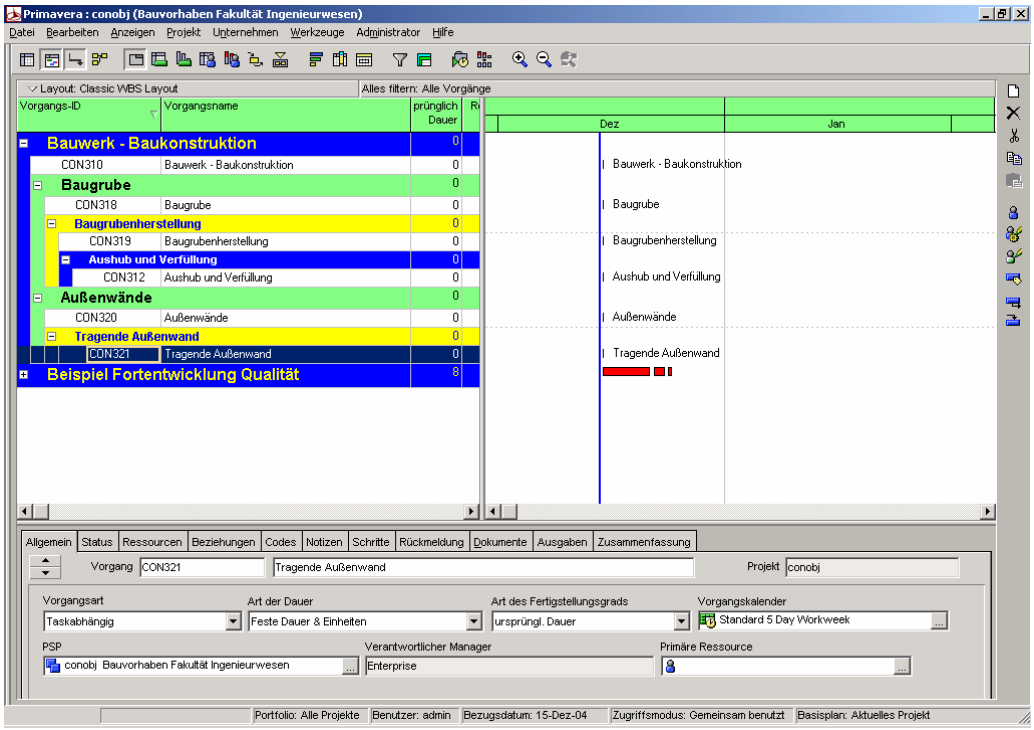


Abb. 38: Situation in der Terminplanung unmittelbar nach dem vom Ereignis PropertyChange ausgelösten Klon-Vorgang.

Eine typische aus dieser Mechanik entwickelte Vererbungshierarchie wird in Abbildung 39 dargestellt.

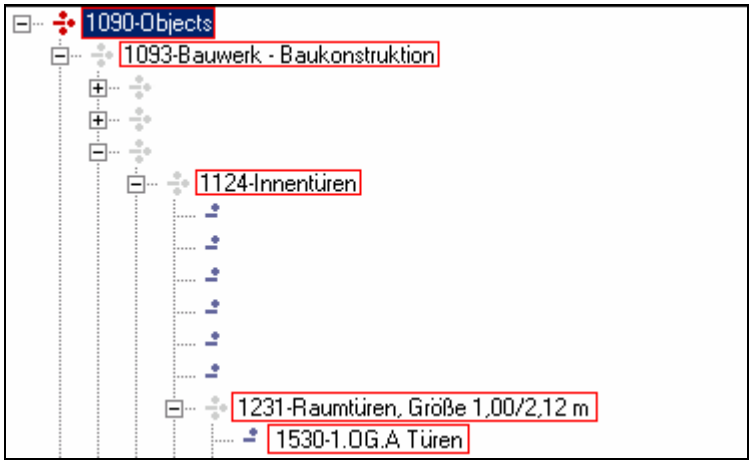


Abb. 39: Hierarchie der Änderungen eines Objektes der HOAI Leistungsphase 3 (1530 - 1.OG.A Türen)

Eine weitere Möglichkeit der Ereignisverarbeitung ist die der objektübergreifenden Ereignisbehandlung. Diese hat den Vorteil, dass Abhängigkeiten zwischen den Objekten modelliert werden können. Ein klassisches Beispiel ist die iterative Berechnung der Zuschläge, wenn diese wiederum bei der Gesamtsumme selbst einbezogen werden. Eine solche Lösung könnte wie folgt aussehen:

Die Klassenbeschreibung listet pro Eigenschaft eine Methode auf, die bei der Änderung dieser Eigenschaft vom System aufzurufen ist. Diese Methode kann wiederum andere Ereignisse auslösen, die auch mit Ereignisbehandlungen versehen wurden. Da eine Klasse keinen imperativen Code enthält, wird die Ereignisbehandlung durch einen Dienst vermittelt. Dieser Dienst empfängt das Ereignisobjekt samt Ereignisquelle als Argument, trägt das Objekt in den Ereignis Cache ein und startet einen Thread zur Behandlung des Ereignisses. Dieser Thread entnimmt die Informationen zum Ursprung (Quell Eigenschaft) und zur auszuführenden Methode (die Ereignisbehandlung) aus dem Cache. Während der Ausführung des Threads (d.h. der Ereignisbehandlung), wird das Ereignisobjekt im Cache sowie die Eigenschaft, deren Änderung das Ereignis ausgelöst hat, mit Hilfe einer Critical Section oder einem ähnlichen Synchronisierungsmechanismus unzugänglich gemacht. Dadurch wird die Eindeutigkeit der Eigenschaft in einem Multi-Threaded Betrieb gewährleistet. Nach erfolgreicher Abarbeitung der Behandlung werden beide Objekte befreit, und das Ereignisobjekt vom Cache gelöscht. Abbildung 40 stellt diesen Ereigniskreislauf dar.

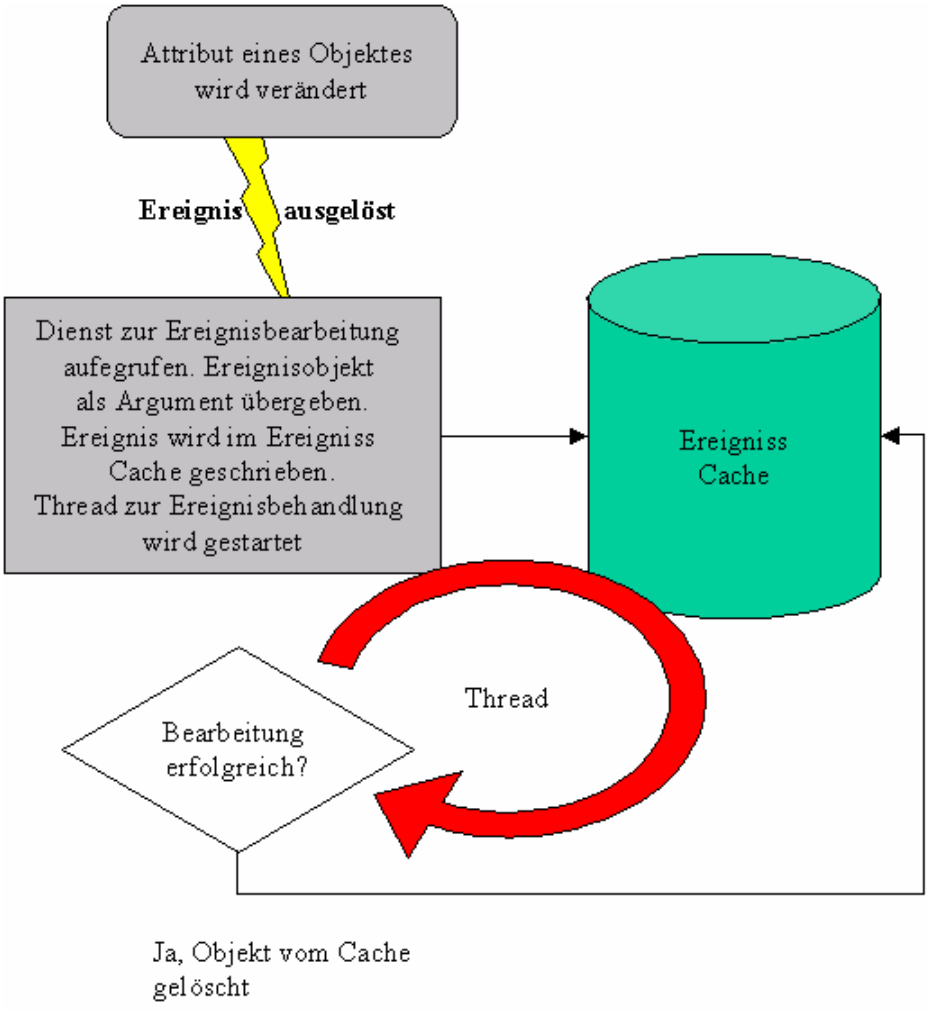


Abb. 40: Der Ereigniskreislauf des Objektmodells

Die Ereignisbehandlung erfolgt mit Hilfe von .NET Remoting, die in der Namespace `System.Runtime.Remoting` untergebracht ist und deren Objekte alle von `System.MarshalByRefObject` abgeleitet werden.

2 Entwicklung des Ereignismodells

Durch die Entkopplung von Objektdefinition und -erfassung vom geometrischen Gebilde (CAD) findet eine Abstraktion der Objekte hin zur funktionalen Betrachtung statt. Objekte können nun zusätzlich zu den anderen Informationen eine geometrische Beschreibung erhalten. Diese Beschreibung stellt aber nicht die Basis der Objekte dar, sondern ist nur eine gleichwertige Eigenschaft.

Die Basis der Objekte ist abstrahiert worden und leitet sich damit vom Urobjekt aller objektorientierten Sprachen ab, dem in C++, C# und Java oder in VB.NET definierten *Object* oder *object*.

Dieser Ansatz führt keinesfalls zu einer niedrigeren Programmierschicht, in der alle Arten der Objekte als Klassen *a priori* festgelegt werden. Durch den Einsatz moderner Serialisierungsmechanismen wird die dynamische Definition von Klassen und die Implementierung von beliebig vielen Objekten ermöglicht. Dadurch, dass die Grundklassen (abstrakt) baubetrieblich orientiert definiert werden, ist es möglich, eine für die Projektbeschreibung sinnvolle Klassenhierarchie anzulegen, die zur Laufzeit beliebig erweitert werden kann und alle wesentlichen Informationen für eine integrierte Projektsteuerung durch die Vererbung beinhaltet.

Es entsteht durch die Abstraktion eine dem Baubereich zugewandte „Virtuelle Maschine“. Diese Maschine stellt einen dynamischen Compiler auf der Basis existierender virtueller Maschinen dar, ist jedoch im Kern aufgrund der abstrakten Klassen baubetrieblich orientiert. Alle Grundklassen haben einen baubetrieblichen Bezug. Mit dieser Maschine gelingen die Verifizierung, das Speichern und das Verwalten von Objekten.
Der Aufbau dieser Maschine verlangt Auskunft über die Objektstruktur (Attribute, Eigenschaften), das Ereignismodell (Ereignisauslösung und -behandlung) und die Objektmanipulation (Methoden). Durch die Festlegung dieser Bestandteile der virtuellen Maschine werden die Anforderungen für die Mechanik der Maschine definiert. Diese Regeln werden dann in den Kern der Maschine integriert und sind ohne Programmierung veränderbar, anders als bei starren Objektmodellen.

Diese Regeln bestimmen, wie ein Objekt aussehen sollte, wie und wann Ereignisse ausgelöst werden, wie diese Ereignisse sich auf Methoden auswirken und wie Methoden die Objekte manipulieren.

Die Objektdefinition ist wichtig für die Interaktion mit anderen Programmen wie Terminplanung, Kalkulation, CAD usw.

Zur konkreten Umsetzung müssen folgende Begebenheiten ausgearbeitet werden:

- Welche Attribute und Eigenschaften muss ein Objekt mindestens erhalten?
- Welche Ereignisse treten ein, wenn diese Eigenschaften verändert werden?
- Welche Methoden sind hierfür erforderlich?
- Für welche Anwendungen müssen die Eigenschaften ausgewertet werden?

Eine beispielhafte Auflistung möglicher Ereignisse und der daraus aufgerufenen Methoden ist in Abbildung 41 aufgelistet.

Fachbereich	Attribute	Erläuterung	Data Type	Event	Method
IT	classGuid		cGuuid	onClassCreate	genSchema()
	objGuid		oGuuid	onObjCreate	isObject()
				onObjDel	checkIntegrity()
	objId		String		
	Type		String		
	classSchema		String		
	timeStamp		LongDateTime		
	Owner		String		
Geometrie	Geometry		Object		
	Positioning		Object		
	Volume		Float		
	Mass		Float		
	massCentroid		Array		
	boundingArea		Float		
	DIN276_2.Ebene				
	DIN276_3.Ebene				
Qualität	DIN18300				
Allgemein	Bauherr		String		
	Grundstueck		String		
	codeHOAI		String		
	descriptionHOAI		String		
	codeDIN182		String		
	descriptionDIN182		String		
Terminplanung	Dauer		Array of ShortDateTime		setCC()
	Fertigstellung		Boolean		setStart()
	Mengen		Array of Float		setEnd()
	Aufwandswerte		Array of Float		setOD()
	Ressource	Maschinen, Personal	Array of String		setRelation()
	Start	Start + Dauer = Ende	LongDateTime		
	Duration	Dauer = AW x Menge / AK x TA (Herstelldauer HD)	Float		
	End		LongDateTime		
	AK				
	TA				
	plannedEarlyStart	Soll-Start (3x,früh,spät,forc.))	LongDateTime		
	plannedEarlyEnd	Soll-Ende (abgel)(3x)	LongDateTime		
	plannedLateStart		LongDateTime		
	plannedLateEnd		LongDateTime		

Fachbereich	Attribute	Erläuterung	Data Type	Event	Method
	plannedForcedStart		LongDateTime		
	freeFloat		Boolean		
	plannedDuration	Soll-Dauer	Float		
	Calender	Kalender	Array		
	acutalStart	Ist - Start	LongDateTime		
	actualDuration	Ist - Dauer	Float		
	actualEnd	Ist - Ende	LongDateTime		
	performance	Aufwandswert	Float		
	relationSS	Beziehungen: SS, SF, FF	Array		
	relationSF		Array		
	relationFS		Array		
	relationFF		Array		
	Relation	Object	Object		
Mengen	**calcMass**		Float		getSurface()
	actualMass		Float		getVolume()
Kosten	calcBudget	Kostenrahmen = Budget des Bauherrn = Calc. Budget	Float		setbudget()
	calcDetailBudget	Kostenschätzung = BRI x Kostenkennwert = Calc. detail Budget	Float		setCalcBudget()
	Kostenberechnung	Kostenberechnung = DIN 276 - 2.Ebene (Grobelemente) oder 3.Ebene x Kostenkennwert	Float		setDetCalcBudget()
	Kostenanschlag	Kostenanschlag = Leistungsbeschreibung Preise	Float		setTenderValue()
	Kostenfeststellung	Kostenfeststellung = Abrechnung/Schlussrechnung	Float		setAccount()
Sonstiges	Costs		Float	onGeomChange	calcSchedule()
	Verguetungs–Änderung		Float	onSchedule	remFromIColl()
				onPosChange	rePosition()
				onQualChange	setCostsInsecure()
				onEreignis	setDurInsecure()
	DIN1833		Boolean	onQualChange	setCostsInsecure()
	Canceled		Boolean	onTradeChange	setPara2()
	Approved	genehmigt = aproved	Boolean		
	Released	freigegeben	Boolean		
	Tendered	ausgeschrieben	Boolean		
	Awarded	vergeben	Boolean		
	Created	erstellt	Boolean		
	Lag	Verzug	Boolean		
	Assured	abgenommen	Boolean		
	Surveyed	abgerechnet	Boolean		

Abb. 41: Ereignismodell

Zusätzlich zu den Systemereignissen besteht die Möglichkeit einer dynamisch veränderbaren Ereignisbehandlung. Diese Ereignisse – sowie deren behandelnden Methoden – werden durch die Klassendefinition dem System zur Verfügung gestellt und durch Objektmanipulationen ausgelöst. Dies geschieht, indem eine im XML Schema definierte Objektmanipulation vom Client an den Server gemeldet wird. Dieses Ereignis wird dann in einem Ereignis Cache gespeichert, das wiederum durch einen Listener überwacht wird. Der Listener aktiviert einen schlafenden Thread, der die dem Ereignis zugeordnete Methode ausführt. Nach dem Abarbeiten des Ereignisses wird dieses wiederum aus dem Cache gelöst.

Da Methoden der Art nach imperativ sind und nur mit größerem Aufwand dynamisch definiert werden können, bieten sich Remoting Methoden mit Hilfe von XML Diensten oder einem verteilten Protokoll wie .NET Remoting oder JINI an. Dadurch wird der einem Ereignis zugeordnete Methodenname automatisch zu einem Verweis auf einen Dienst oder eine verteilte Methode. Bei der Angabe von Ereignissen und Methodennamen müssen die Dienste im System verankert und registriert werden. Somit ist eine gewisse Flexibilität im System möglich.

3 Dynamische Ereignisverarbeitung

Dadurch, dass die jeweiligen Ereignisse dem Client mit Hilfe von Angaben im XML Schema bekannt sind, werden diese je nach Manipulation am Objekt ausgelöst. Der Client registriert diese Ereignisse im Ereignis Cache, wodurch ein Listener aktiviert wird. Dieser Listener kann wie ein Webserver fungieren, indem auf einem Port eine Nachricht abgewartet wird und dann auf eine Nachricht reagiert wird. Als Lösung bieten sich XML Dienste an, da diese auf dem Webserver, der auch als Listener fungiert, gestapelt sind.

Der Webdienst startet einen Thread, dieser schreibt das Ereignisobjekt in den Cache (eine Tabelle in der Datenbank) und ruft die Methode zur Ereignisbehandlung auf. Dieser Thread hat exklusiven Zugriff zu dem Ereignisobjekt und bleibt bestehen, solange die Ereignisbehandlung noch nicht beendet ist. Es ist entscheidend, dass der behandelnde Thread exklusiven Zugriff auf das Objekt hat, damit sichergestellt wird, dass die Ereignisbehandlung auch exklusiv auf das zu verändernde Objekt zugreifen kann. Das ist wichtig, damit keine Konflikte zwischen Methoden auftreten, da sonst durch das Multi-threading Verfahren Eigenschaften im teilverarbeitenden Zustand

wiederum durch andere Methoden aufgegriffen werden und somit die Rahmenbedingungen der Methoden nicht erfüllt werden. Diese Problematik, auch als Synchronisierung bekannt, ist ein wichtiger Bestandteil bei verteilten Architekturen mit mehreren Teilnehmern. Die Problematik wird in Abbildung 42 veranschaulicht.

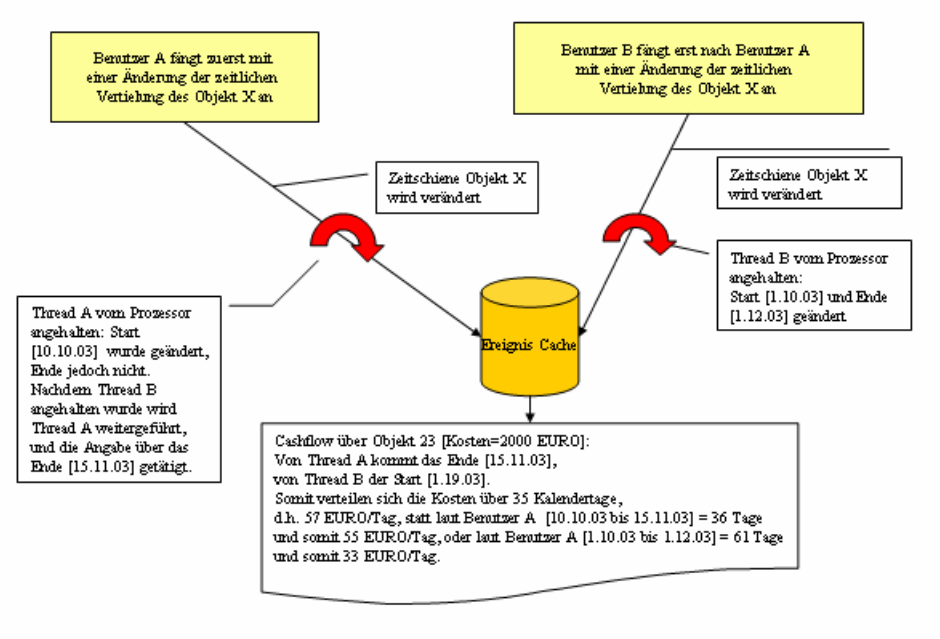


Abb. 42: Veranschaulichung der Problematik einer multi-Threaded Umgebung

Sobald die Ereignisbehandlung abgearbeitet wurde, wird das Ereignisobjekt aus dem Cache gelöscht und der Thread beendet.

4 Klonen als eindeutige Dokumentation der Ereignishistorie

Die progressive Entwicklung des einzelnen Bauwerkskörpers von der anfänglich abstrakten hin zu der endgültigen exakten Form über alle Merkmale hinweg, stellt zugleich mit den anderen Bauwerkskörpern die Entwicklung des Projektes über alle Phasen dar. Diese Entwicklung beeinflusst die Kostenentwicklung, die Planungsentwicklung, die Terminierung und das Projektmanagement. Daher ist es wesentlich, dass sämtliche instanzierte Objekte erhalten bleiben. Um die Progression der Klassen zu ermöglichen, ist die objektorientierte Vererbung zwingend.

Dazu gelten folgende einfache Regeln, die aufgrund ihrer Invarianz direkt in die Applikation implementiert wurden:

- Regel 1: Ein existierendes Objekt kann nicht vom Typ her verändert werden, es sei denn es findet eine explizite Typumwandlung statt.
- Regel 2: Wenn eine Typumwandlung stattfindet oder ein neuer Typ hinzugefügt wird, dann soll eine neue Klasse generiert werden. Wenn die Klasse nicht im Einsatz ist, können Änderungen ohne weiteres angebracht werden. Wenn die Klasse schon im Einsatz ist, wird das Objekt geklont und es findet eine explizite Typumwandlung hinauf zu der neuen Klasse, die von der existierenden Klasse ableitet, statt.

Die programmtechnische Umsetzung dieses Systems hat bisher gezeigt, dass eine einfache Vererbung ausreichend ist. Dieser Punkt ist entscheidend für die Wahl der Programmiersprache und damit der Entwicklungsplattform, weil in den Sprachen Java und C# nur eine Einfachvererbung vorgesehen wird, wobei C++ eine Mehrfachvererbung unterstützt.

Die Progression wird erreicht, indem bei jeder Änderung der in dem Objekt zugrunde liegenden Klasse eine Vererbung erzwungen wird. Bei Änderungen eines Objektes gleich bleibender Klasse wird das Objekt geklont (versioniert). Alle nicht-aktuellen Objekte werden gekennzeichnet, indem die Eigenschaft *Active* auf den Wert `false` gesetzt wird, bleiben aber dem Projekt insgesamt erhalten (für Auswertungen, für terminliche Soll-/Ist-Vergleiche etc.) und wirken sich daher nicht mehr auf den aktuellen Datenbestand aus. Abbildung 43 veranschaulicht das fortschreitende Klonen bei Modifikationen an Objekten.

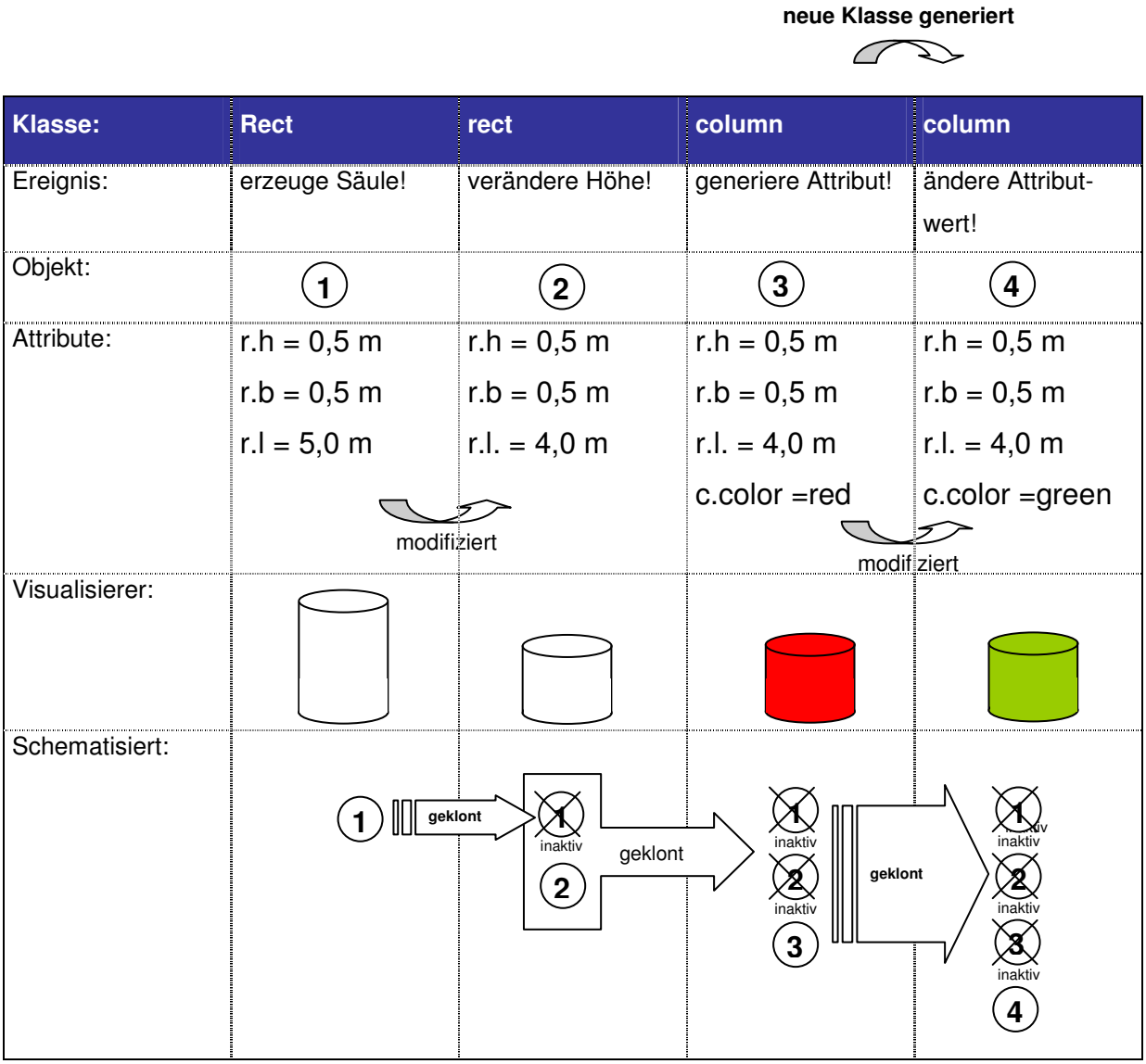


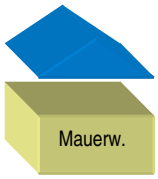
Abb. 43: Modifikation an Objekten

Ähnlich kann die Entwicklung eines einfachen Hauses mit Hilfe des Klonens dokumentiert werden (Abb. 44).

LP	Darstellung	Code	Kosten	Terminplanung
1				
2		Public class cFamilienhaus : cAssemblyPerm public cFamilienhaus (); this.Validate(); this.projectName = “Familienhaus”; this.projectOwner = “Schmidt”; ...	BRI evtl. funktionale Anforderung	BRI sonst. Anmerkungen

3		Public class cFamilienhausDach : cFamilienhaus
		public cFamilienhausDach();
		this.Validate();
		int[,] geomArr = new int[6,3];
		geomArr.setValue(0,0,0);
		geomArr.setValue(0,1,0);
		geomArr.setValue(0,2,0);
		geomArr.setValue(1,0,2); geomArr.setValue(1,1,0);
		geomArr.setValue(1,2,0);
		geomArr.setValue(2,0,1); geomArr.setValue(2,1,0.7);
		geomArr.setValue(2,2,0);
		...
		int[] posArr = new int[3];
		geomArr.setValue(0,1.5,0);
		this.setGeomerty(polygon, geomArr, posArr);
		public double vol = this.getVolume();
		public double mass = this.getMass();
		public double massCentroid = this.getMassCentroid();
		public double surface = this.getSurface();
		...
		gleiche für Grundbauwerk
			DIN 276, 2te Stelle Mengen Grobelemente	Mengen Grobelemente Objektgruppierung beliebig Zuweisung von Vorgangsdauern für Objektgruppen
				Definition von räumlichen Gruppierungen
				Vorgänger, Nachfolger etc. der Gruppen.

LP	Darstellung	Code	Kosten	Terminplanung
4				

5		cFamilienhausDach cDach = new cFamilienhausDach();
		ArrayList codeArr = new Array-List();
		codeArr.put(cDach.
		getHashCode(),"Hausdach") ;
		...
		cDach.setDinCode(18xxx, co-deArr);
		codeArr.put(cDach.
		getHashCode(),"Hausdach") ;
		...
		cDach.setDinCode(276, codeArr);
			DIN 1833
			Qualitäten
			DIN 276 3. Stelle
			gebäude- und gewerke-orientierte Bau-beschreibung
				DIN 1833
				Mengen der Leitpositionen
				Räumliche Gruppierung
				Zuweisung Dauer, Vor-gänger, Nachfolger etc. für Leitpositionen
				Zuweisung Soll-Termine und Soll-Zustand zu Ob-jekten und Gruppen

6			DIN 1833 Mengen geglie-dert nach DIN 276 Mengen geglie-dert, oder nach DIN18xxx Gruppierung zu Leitpositionen räumliche Grup-pierung	Wie vor.
7			Zuweisung Ver-tragsdaten zu Objekten und Gruppen	Zuweisung Vertragsdaten zu Objekten und Gruppen

LP	Darstellung	Code	Kosten	Terminplanung
8			Zuweisung Abweichungen zu Planung Zuweisung Ist-Mengen Zuweisung Mängelanmerkungen Anmerkungen bezogen auf einzelne Objekte, Gruppen, Leitpositionen etc.	Zuweisung Ist-Termine/ dauern und Ist-Zustand zu Objekten und Gruppen Soll/Ist-Vergleich
9			Gruppierung nach Firmen, Zuständen, Differenzen, Soll/Ist etc.	Gruppierung nach Firmen, Zuständen, Differenzen, Soll/Ist etc.

Abb. 44: Entwicklung der Klassenhierarchie für ein Familienhaus

Beim Klonen sind die Regeln, nach denen geklont werden soll, nicht eindeutig. Nicht jede Änderung sollte zum Anlass eines Klon-Vorgangs dienen, sonst werden die Änderungsgraphen schnell unübersichtlich. Das System identifiziert auf Binärebene, ob eine Änderung gegenüber dem letzten gültigen Stand vorkam. Damit dieser Vergleich differenzierter erfolgt, können binäre Vergleiche auf der Ebene der Attribute erfolgen. Dazu dienen Klassen als Schablone, da sie die Typinformationen der Objekte beinhalten. Weiterhin werden zusätzliche Kriterien bei positivem binärem Vergleichstest auf Attributebene wünschenswert sein. Z.B. wird das System eine Änderung in dem Volumen eines Baukörpers in CAD feststellen. Der Binärvergleich wirft eine Änderung auf. Diese kann zusätzlich mit einer Methode auf einem Dienst oder Remoting Objekt untersucht werden. Reicht die Änderung des Volumens nur bis hin zu einem gewissen Prozentsatz, wird nicht geklont. Darüber hinaus wird geklont. Das heißt: Bei Änderungen des Volumens unterhalb dieses Prozentsatzes wird das System diese Veränderungen zwar registrieren, die Objekte werden jedoch nicht geklont und diese Änderungen damit auch nicht im System festgehalten.

V Strukturierung der Objektwelt

Ein zentraler Aspekt eines Projektes ist die Strukturierung, auch Gliederung genannt. Die Gliederung dient der Aufteilung eines Projektes in kleinere Gruppen und Mengen zur Verbesserung der Anschaulichkeit und der Klassifizierung. Eine Gliederung dient ferner als Schnittstelle zwischen den am Bauprojekt beteiligten Parteien. Im Projektverlauf werden häufig mehrere Gliederungen aufgestellt. Diese müssen untereinander sinnvoll zugeordnet werden, um im Gesamtkontext die relevanten Informationen zu erhalten. Eine Leitstruktur (als Strukturbaum) dient zur Orientierung aller Projektbeteiligten. Diese wird für die wichtigsten Abschnitte und Gewerke von der Projektleitung erarbeitet (z.B. für Gebäude, Planbereich, Ebene usw.). Alle Projektbeteiligten richten sich danach aus.

Die jeweiligen Beteiligten haben jedoch oft interne, arbeitstechnische Gründe zur Etablierung einer eigenen Struktur, z.B. die Aufteilung der Planung nach Leistungsbildern und nach Gewerken oder die Aufteilung in Gruppen für die Kalkulation.

Somit ist die Zuordnung der Strukturen untereinander von entscheidender Bedeutung.

Beispiel

Leitstruktur (Gebäude, Planbereich, Ebene): ZGB, 101, E4
Planung (Disziplin, Leistungsphase, Freilaufsatz): Tragwerksplanung, Bewehrungsplanung
Kalkulation (Gebäude, Funktionalität, Gewerke, Kostengruppe DIN 276, Qualitätsgruppen): ZGB, Fundamente, Betonarbeiten.

Da bei einer umfassenden Dokumentation die Integration der Informationsbestände in einem fachlich sinnvollen Konstrukt erfolgen sollte, ist die Zuordnung der jeweiligen Gliederungen entscheidend. Bei der herkömmlichen Methodik der Arbeits- und Informationsteilung entsteht diese Problematik automatisch. Diese Inkongruenzen können mit Hilfe von Zuordnungen und einer sinnvollen Gliederung teilweise überwunden werden, jedoch mit einem erheblichen Aufwand.

Dadurch, dass bei dem hier beschriebenen Objektmodell die Objekte alle relevanten Informationen integrieren, und die Strukturierung der Objekte auch der Schlüssel zur Aufteilung objektübergreifender Leistungen ist, wird diese Art der Informationszuordnung grundsätzlich geleistet.

Diese Zuordnung erfolgt über die Aufteilung der zugeordneten Gliederungen oder Gruppierungen. Diese sind in der Klasse *iCollections* implementiert, die wiederum eine Sammlung von Knotenpunkten (Klasse *INode*) als Filter definiert. Die Klasse übernimmt die Serialisierung der Knoten und die Aktualisierung der Filter mit Hilfe von *Hashtables*. Das Objektmodell verfügt auch über zwei interne Gliederungen, jeweils für die Klassen- und Objektbäume. Diese werden automatisch entsprechend der Klassen- und Objektgenerierung vom Objektmodell erzeugt und den Objekten bzw. Klassen zugeordnet.

1 Generierung der Gliederungsstrukturen

Bei der Entwicklung einer Gliederungsstruktur werden die Knotenpunkte im jeweiligen Filter angelegt. Diese Filter kapseln Referenzen auf die Knotenobjekte vom Typ *INode*, die wiederum als selbständige Objekte angelegt werden. Filter sind vom Typ *iCollections*, die wie *INode* serialisierbar sind, da sie die Schnittstelle `ISerializable` implementieren.

Filter können über eine im System vorgesehene Maske neu angelegt oder konfiguriert werden. Dabei wird angegeben, ob Filter öffentlich (public) oder privat (private) sind. Diese Eigenschaft wirkt sich auf die Suchergebnisse aus, da private Filter nur innerhalb eines Projekts zu finden sind. Somit wirkt sich diese Eigenschaft auf Sicherheit und Scope aus; bei einer produktiven Version dieses Konzepts sind diese Überlegungen wichtig, im Rahmen dieser Arbeit sind Sie allerdings nicht von Belang.

In dieser Maske sind Schnittstellen zu den am System beteiligten Applikationen vorgesehen, entweder vom Typ XML Dienst oder .NET Remoting. In der hier implementierten Desktop Applikation ObjektViewer wurde für jede Schnittstelle ein Tabulator vorgesehen, um die jeweils benötigten Parameter einstellen zu können. Diese Schnittstellen verfügen häufig über grundsätzlich unterschiedliche Technologien, da auch in tieferen Schichten auf externe API's zugegriffen wird. Das führt dazu, dass die Konfiguration der jeweiligen Schnittstellen nicht verallgemeinert werden kann.

Ein Beispiel dieser Implementierung ist der Unterschied zwischen den Schnittstellen von der P3e Applikation, eine Java basierte API, die mit einem RMI Framework kommuniziert und der Allplan Schnittstelle, eine Kombination aus COM (`System.Runtime.InteropServices`) und in einem Remoting Dienst gekapselten Native Object Interface (C++).

Durch den Einsatz der Namespaces `System.CodeDOM` und `System.Reflection` können flexible Systeme geschaffen werden, um solche Schnittstellen dynamisch zu integrieren. Eine ähnliche Applikation wurde von Weyer [Weyer, C., 2004. S. 1] entwickelt, um Web Dienste dynamisch einzulesen und ihre Methoden zur Laufzeit zur Verfügung zu stellen (Abb. 45). Die Entwicklung einer solchen dynamischen Schnittstelle für die jeweiligen Dienste der zu integrierenden Applikationen ist aufwendig und nicht Bestandteil dieser Arbeit.

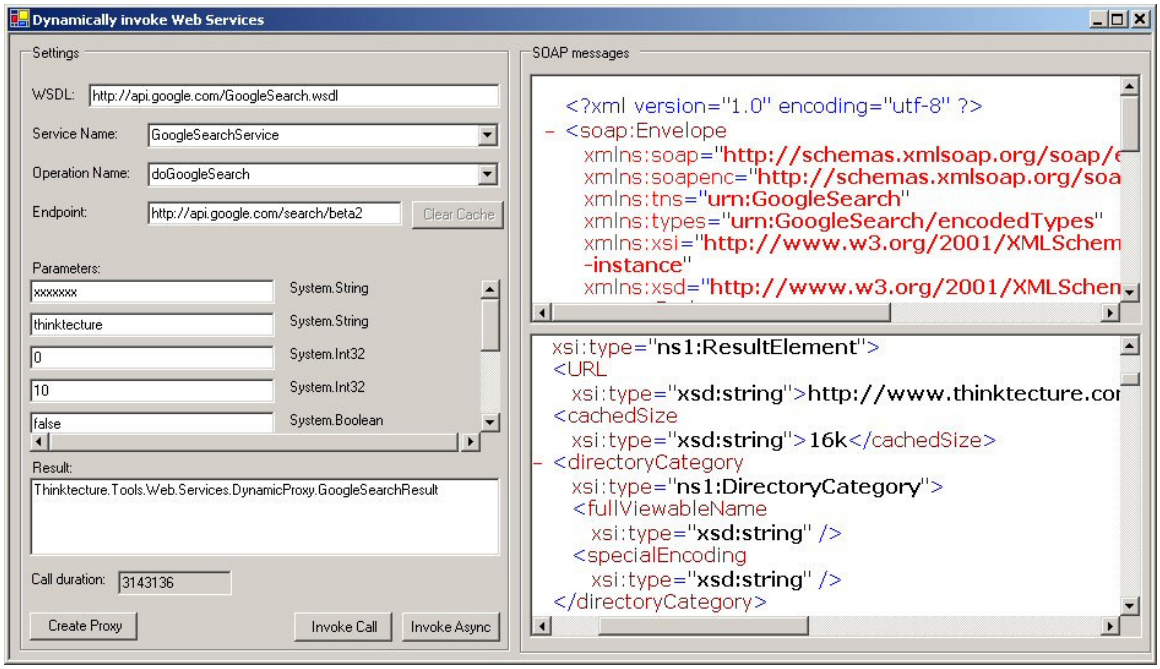


Abb. 45: Web Dienste dynamisch einlesen und instanzieren [Weyer, C., 2004. S. 1]

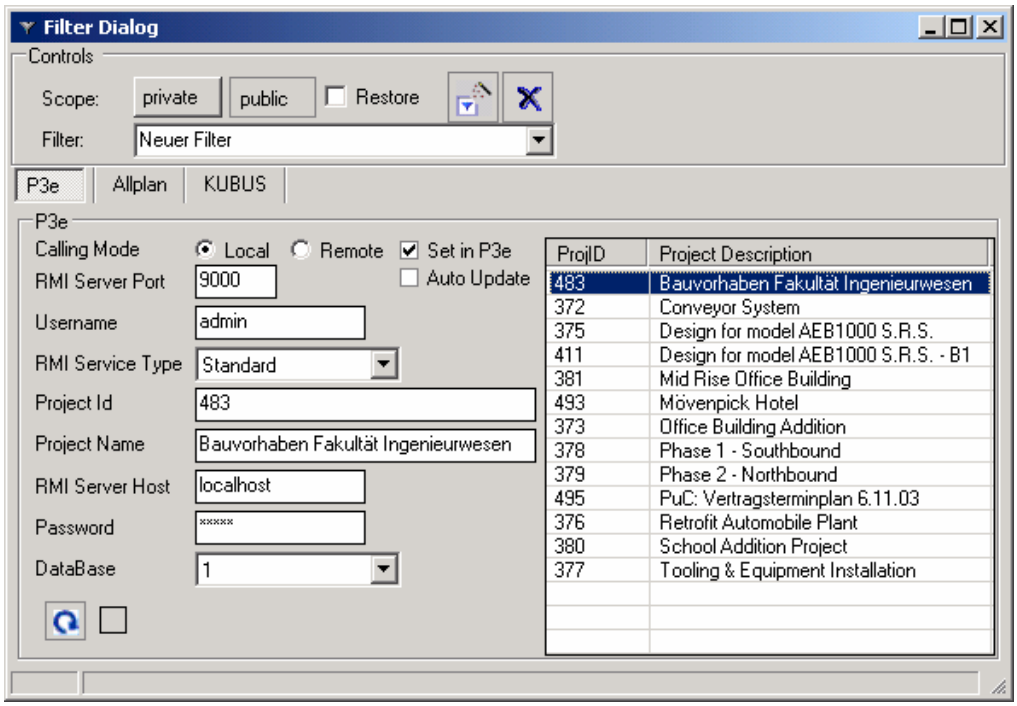


Abb. 46: Anlegen eines neuen Filters oder Konfiguration eines existierenden Filters. Tabs beinhalten Parameter für die jeweiligen Dienste oder Remoting Objekte. Aktueller Tab stellt Parameter für P3e Dienst dar. Dieser kapselt eine mittels RMI eingebundene Java API. Die Kapselung erfolgt über J#.

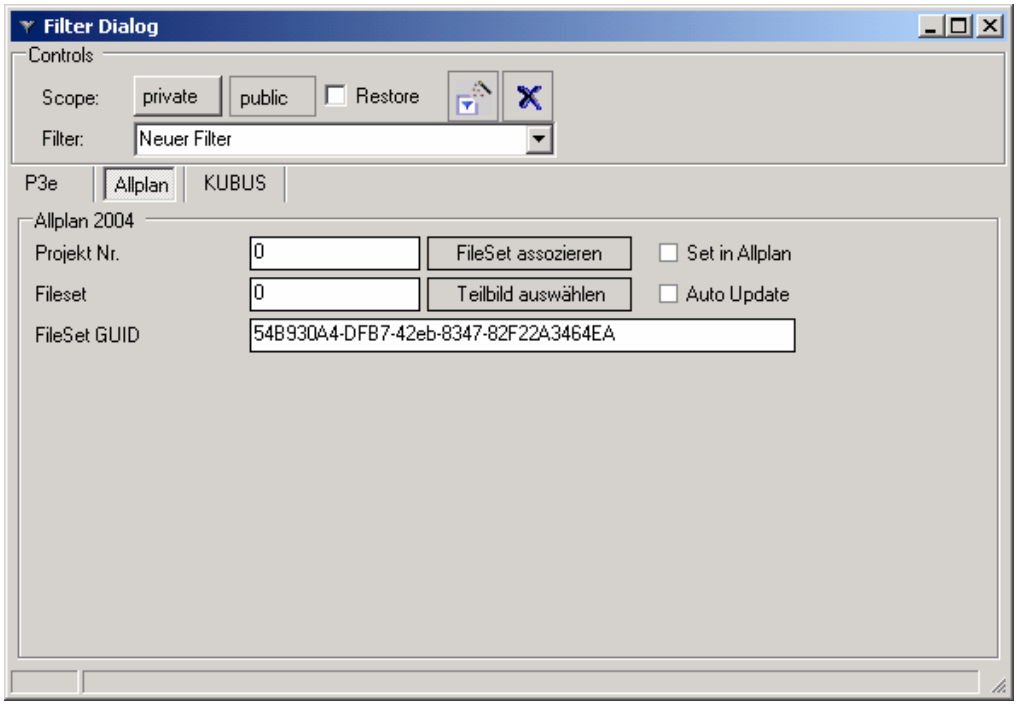


Abb. 47: Filter mit Tab für Allplan Schnittstelle. Da aus Allplan fertige Strukturen und Pläne bezogen werden, ist diese Schnittstelle nicht markiert zum Schreiben einer neuen Gliederungsstruktur in Allplan.

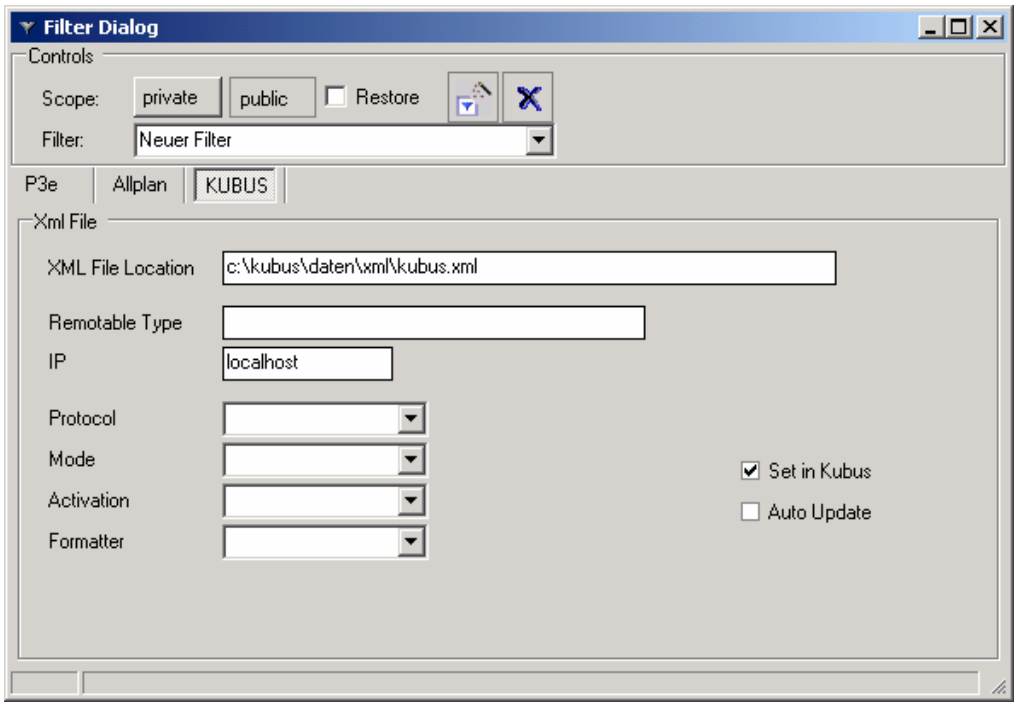


Abb. 48: Filter mit Tab für KUBUS Schnittstelle. Diese Schnittstelle ist nicht dynamisch und besteht aus einer XML Datei, die wiederum in KUBUS eingelesen wird, um die Änderungen anzuzeigen.
Diese Schnittstelle ist markiert zum Schreiben einer neuen Gliederungsstruktur in KUBUS.

Die Filter werden als Instanzen von iCollections für die jeweiligen Applikationen konfiguriert (P3e: Abb. 46; Allplan Nemetschek 2005: Abb. 47; KUBUS: Abb. 48). *INode* Objekte besitzen Eigenschaften zur hierarchischen Verknüpfung der Knotenpunkte, für Termine und für Kostenlaufzeitinformationen. Beim Anlegen dieser Objekte (Abb. 49) werden, falls gewünscht, analog der vorgesehenen Dienste oder Remoting Methoden (Schnittstellen) und der Vorgehensweise mit Filter entsprechende Strukturknoten in den jeweiligen Applikationen angelegt (P3e: Abb. 50; Allplan Nemetschek 2005: Abb. 51; KUBUS: Abb. 52) und mit der gleichen *nodeId* (eine *Guid*) versehen. Diese Vorgehensweise hat den Vorteil, dass alle an dem System beteiligten Applikationen die gleichen Strukturen besitzen, was die Zuordnung der Objekte oder der Eigenschaften zu den Objekten in einem applikationsübergreifenden Kontext vereinfacht, da Transformationen vorneweg ausgeschlossen sind.

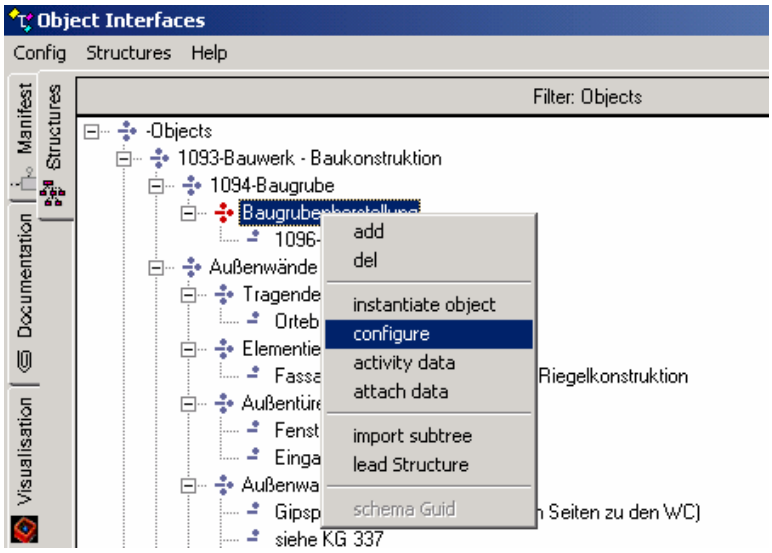


Abb. 49: Anlegen bzw. Aktualisierung eines Knotens in einem Filter (iCollection) bzw. Konfiguration eines vorhandenen Knotens.

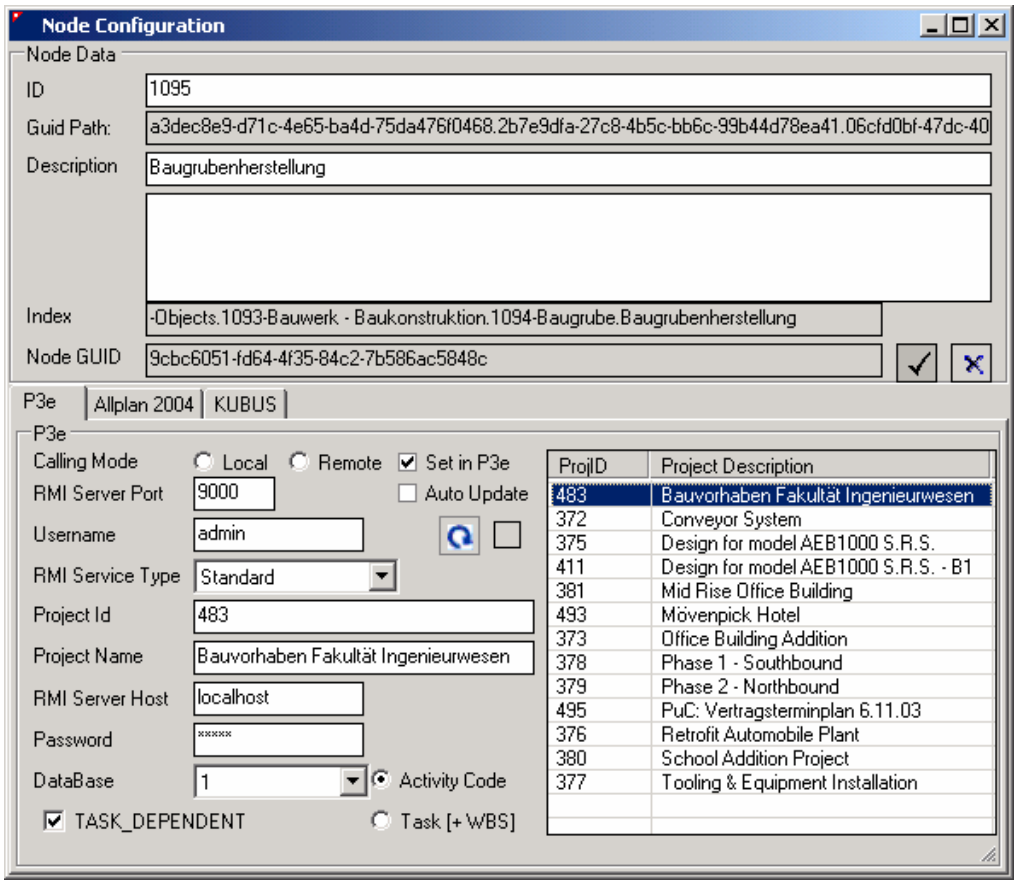


Abb. 50: Aktuelles Tab stellt Parameter für P3e Web-Dienste dar. Mittels dieses Dienstes wird ein Knoten analog zum conobj INode Knoten (als Activity Code) in dem aktuellen Gliederungssystem (Activity-Code Structure) in P3e eingefügt bzw. aktualisiert.

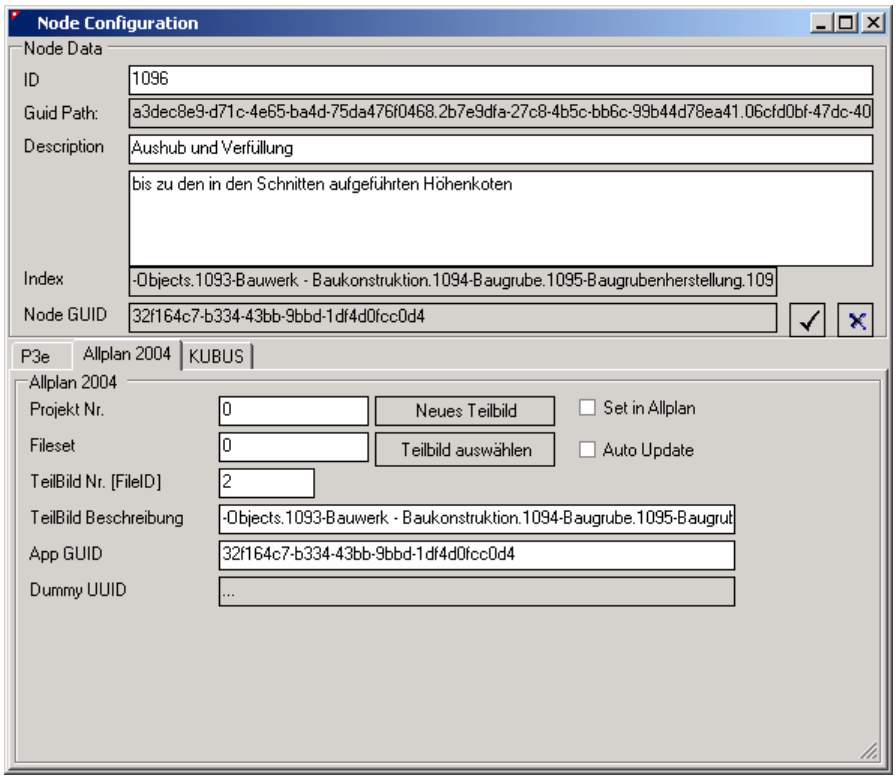


Abb. 51: Filter mit Tab für Allplan Schnittstelle. Diese Schnittstelle ist für das Schreiben analog Abb. 58 nicht aktiviert.

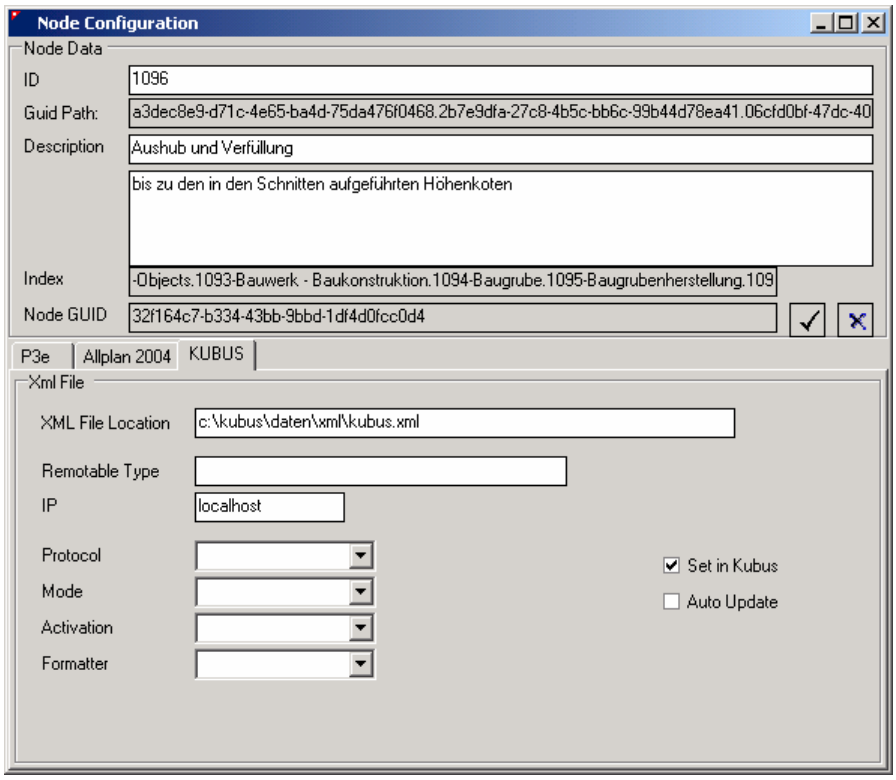


Abb. 52: Filter mit Tab für KUBUS Schnittstelle. Analog Abb. 59 wird ein Knoten in der alternativen Gliederung von KUBUS entsprechend dem aktuellen Filter eingefügt bzw. aktualisiert.

Die aktualisierten Gliederungssysteme, mit den entweder aktualisierten oder eingefügten Knoten vom Typ *INode* werden in Abbildung 50 für P3e und Abbildung 52 für KUBUS dargestellt.

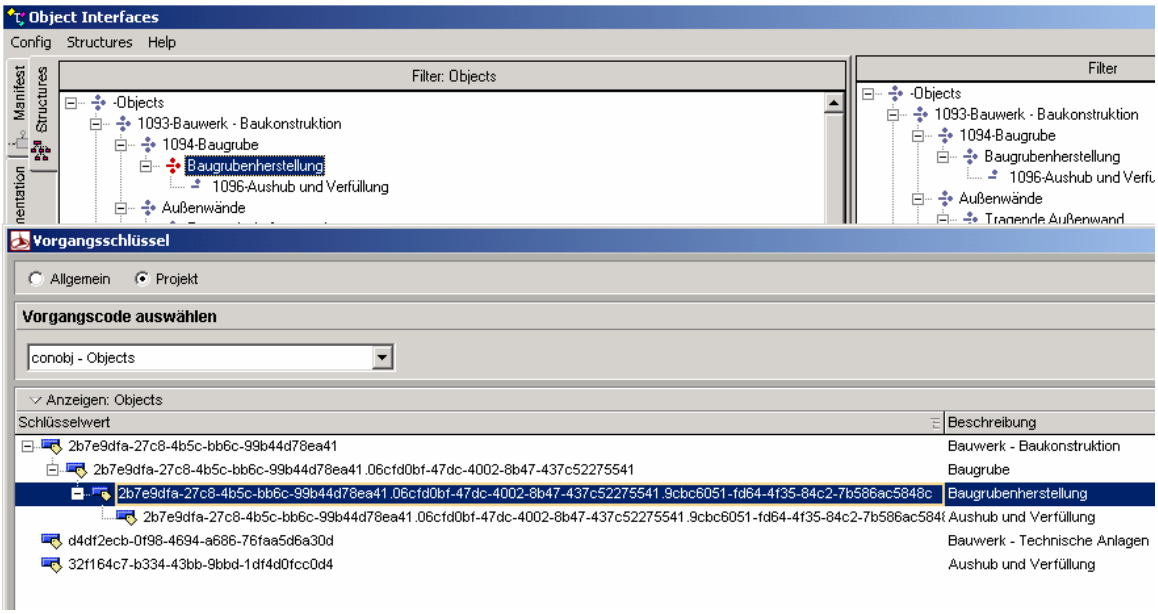


Abb. 53: Aktualisierte Knoten im Objektmodel und als Activity Code mit NodeId-FullPath in der entsprechenden Activity Code Structure in P3e.

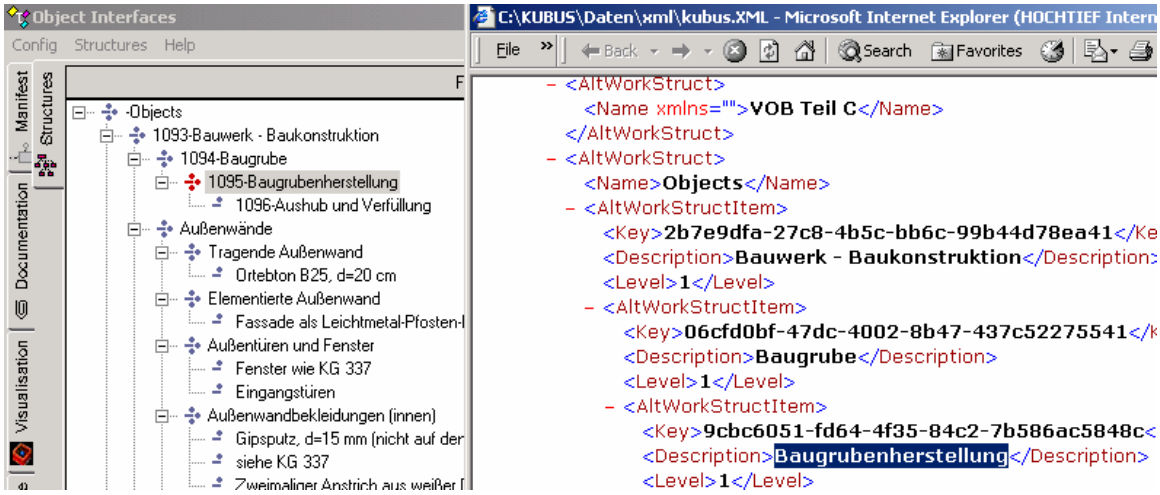


Abb. 54: Aktualisierte Knoten im Objektmodel und als alternativer Gliederungspunkt in der alternativen Gliederung von KUBUS.

Zwei Gliederungsschlüssel vom Typ *iCollections* werden automatisch im System angelegt:

- CLassStructure
- Objects

CLassStructure entsteht durch die Vererbung von Klassen. Diese Struktur muss vom System verwaltet werden, um Typsicherheit im Objektmodell zu garantieren.

Das Gliederungssystem **Objects** wird automatisch angelegt, jedoch nicht automatisch mit Knoten versehen. **Objects** bildet die Objektstruktur eines Bauobjektes, in diesem Fall eines Gebäudes ab. Es stellt eine 1:1 Darstellung der tatsächlichen Objekte dar. Dieses Shadowing Prinzip ist strikt genommen überflüssig, hat sich aber in der Entwicklung des Systems bewährt. Zur Verbesserung der Laufzeit werden die Objekte während der Entwicklung des Gliederungssystems **Objects** instanziert und in der Datenbank serialisiert. Somit brauchen sie nur beim nächsten Aufbau des Systems in einen Cache (DOMCache vom Typ *Hashtable*) geladen zu werden und stehen allen Zugriffen schnell zur Verfügung.

2 Analysen der Gliederungen

Objekte werden i.d.R. mehreren Gliederungspunkten zugeordnet. Dadurch entsteht eine gewisse Überlappung der Gliederungen (Abb. 35). Eine Analyse dieser Überlappungen kann mit Bytesummen erfolgen und würde Aufschluss geben über:

- Die Häufigkeit einer Objektreferenzierung, damit auch eine Indikation der Objektwichtigkeit.
- Ähnlichkeiten der Gliederungssysteme und damit ein Werkzeug zur Eliminierung von redundanten Gliederungen
- Zuständigkeiten, dass Gliederungen auch immer etwas mit Hierarchien zu tun haben
- Arbeitspakete, wobei deren Bündelung mit der Objectsmap ersichtlich wird

Diese Information wird dann zur Veranschaulichung mit einer SVG-basierten XSLT transformiert.

3 Analyse des Objektgraphs

Zwei Eigenschaften eines Objekts stellen dessen Veränderungen dar:

- Klassenhierarchie
- Klon-Historie

Durch die dynamische Vererbung und deren kontinuierliche Dokumentation sowie durch die Versionierung eines Objekts (Klonhistorie) entsteht die Möglichkeit, umfassende Baumstrukturen der Entwicklung des Objekts zu konstruieren. Dazu gibt es effiziente Methoden. Bei den in Teil C aufgeführten XQuery Abfragen werden komplette Baumstrukturen von den Filtern und den Objekten aufgebaut, um die Abfragen zu ermöglichen. Diese Strukturen werden in einem Zwischenspeicher (`XQueryNavigatorCollection`) geladen und dienen zugleich auch zur Feststellung der Änderungshierarchie, die entweder direkt aus dem Objekt entnommen, mittels einer XQuery Abfrage eruiert oder dem **Objects** Filter entnommen werden können. Ein solcher Baum kann die Änderungshistorie eines Objektes umfassend darstellen, soweit sie in dem System gepflegt worden ist.

4 Zuordnung von Dokumenten mit Objekt übergreifenden Inhalten

Das Problem der Abgrenzung entsteht dadurch, dass diskrete Bauwerksobjekte in verschiedenen Zusammensetzungen in diversen bauwerksübergreifenden Dokumenten und Plänen zusammengefasst werden, die jedoch oft bezugslos sind. Die Zusammenfassung solch inkohärenter Verweise ergibt sich stets durch die Gegebenheit, dass Dokumente und Pläne als Objekte modelliert werden (übliches Verfahren im modernen Dokumentenmanagement).

Dieses Problem kann elegant durch den Einsatz der objektorientierten XML-Modellierung dieser Dokumente gelöst werden. Die Dokumente, welche wie gewohnt eher prozessorientiert bleiben, bieten mit Hilfe ihrer XML-Struktur die Möglichkeit, über Verfahren wie XPath oder XPointer Dokumentabschnitte den jeweiligen Bauwerken zuzuordnen, bleiben jedoch als in sich geschlossenes Objekt in dem Projektraum bestehen.

4.1 Beispiel

Ein Mängelschreiben des Bauherrn weist auf zahlreiche objektübergreifende Mängel hin. Die einzelnen Punkte können den betroffenen Objekten zugewiesen werden, ohne dass das Schreiben dupliziert werden muss.

Der Verweis erfolgt nicht, wie herkömmlich, über die URL, sondern durch die Assoziierung der Klasse des Dokumentes mit der des Bauwerkes mittels der objektorientierten „`friend`“ Anweisung. Eine weitere Methode der Zuordnung ist der Einsatz von Beziehungen, die auch als Objekte modelliert werden können.

VI Vorgehensweise der Modellierung

1 Objektversionisierung

Die Objektversionierung ist ein wichtiger Aspekt der Objektverwaltung zur Sicherung der Aktualität von Daten und um den Zugriff auf vergangene Darstellungen zu ermöglichen.

Das Objektmodell garantiert die Objektversionierung durch das Klonen bei Objektänderungen. Das Klonen führt zu der Deaktivierung eines vorherigen Standes und zum Schaffen eines aktuellen Standes. Somit ist die Historie (Vorgänger) dokumentiert und greifbar, und der aktuelle Stand steht im Mittelpunkt. Die Versionierung verfolgt die Historie der Objektänderungen.

2 Vererbungshierarchie

Den Objekten werden bei der Erstellung des Objektbaums Klassen zugeordnet. Diese Klassen dienen als Schablone für den strukturierten Dateninhalt der Objekte. Diese Klassen vom Typ `objLib.cls.objSchema` bestehen aus einem objektorientiert abgebildeten Rumpf, sowie einer im XML Schema festgehaltenen Klassenbeschreibung. Jedem Objekt kann nur eine Klasse zugeordnet werden analog moderner Sprachen wie Java oder C#.

Bei Änderungen in der Klassenstruktur sowie deren Typdeklarationen kommt es analog objektorientierter Sprachen wie C++, Java oder C# unweigerlich zu einer Vererbung. Diese Vererbung wird analog eines Compilers oder einer IDE durch die Applikation geleistet, indem sie identifiziert wird und nach den erwünschten Änderungen an der `base` Klasse zur Speicherung unter einem neuen Klassennamen auffordert. Somit entsteht eine abgeleitete Klasse mit den Inhalten der `base` Klasse sowie alle neu hinzugefügten Typinformationen. Diese kann wiederum Objekten zugeordnet werden auch bereits instanzierten Objekten, da die Vererbung ein Kontinuum der Daten gewährleistet. Abbildung 55 veranschaulicht den Ausgangszustand kurz vor der Änderung einer Klasse, hier die Klasse *root*.

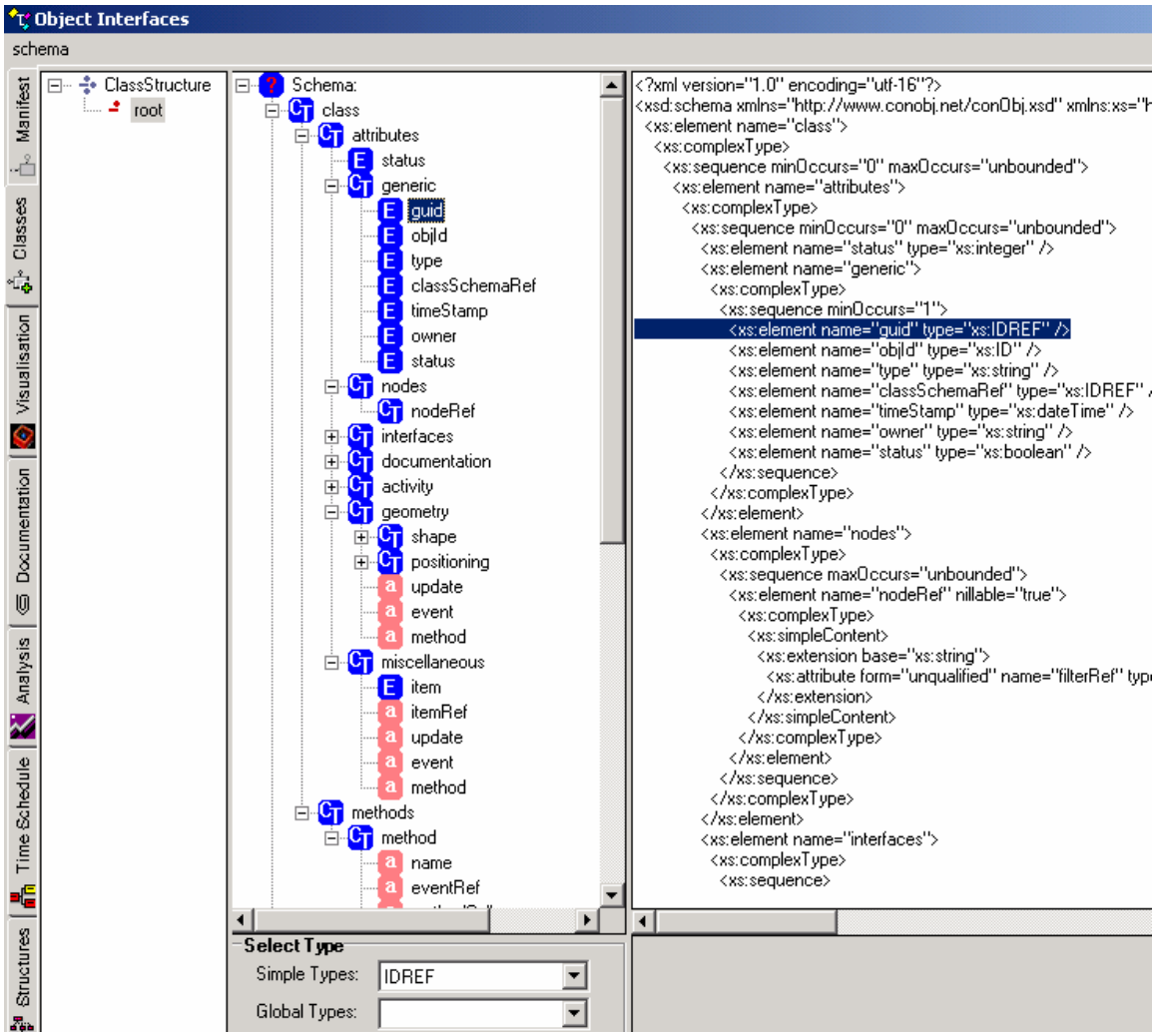


Abb. 55: Zustand der Klasse *root* bevor Änderungen an Inhalt und Typinformation die Klasse zu einer Vererbung zwingen.

Der Klasse *root* wurden folgende Informationen hinzugefügt:

- surfacearea (m²) – Oberfläche des CAD Objektes
- volume (m³) – Volumen des CAD Objektes
- material – Material das dem Objekt in CAD zugeordnet wurde
- objecttype – CAD Objekttyp

Nach dieser Zuordnung sieht die Klasse wie in Abbildung 56 aus.

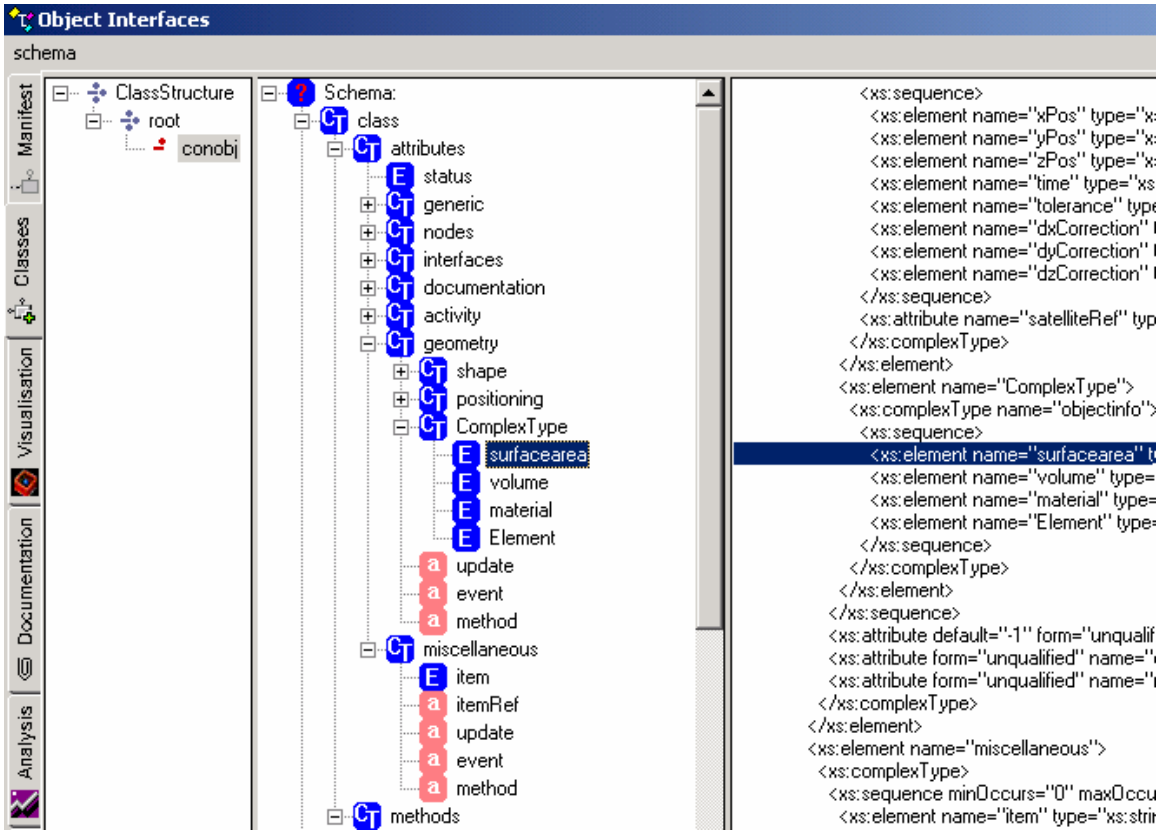


Abb. 56: Zustand der Klasse root nach den oben genannten Änderungen an Inhalt und Typinformation, die eine Vererbung verlangten.

Analog zu diesem Verfahren wurde für die Basisklasse constructionobject ein XML Schema, was in Anhang A.IV aufgeführt wird, entwickelt.

3 Schnittstellen und Transformationen

Moderne Applikationen werden zunehmend mit XML-Diensten versehen. Im Normalfall bieten diese jedoch lediglich Lesefunktionalität. Daten können mittels XML-Dienste im XML Format aus den Applikationen gelesen werden.

Zum Einsatz der hier entwickelten Methode bedarf es jedoch auch der Möglichkeit, Daten in den jeweiligen Applikationen einzulesen. Nur so ist es möglich, die Objektstruktur in allen Applikationen die am Objektmodell teilhaben sollen, auf dem gleichen Stand zu halten.

Die Objektstruktur wird mittels Dienste in die jeweilige Applikation hineingelesen, die Applikation versieht diese Struktur mit Daten, die Daten werden dann als Attribute aus der jeweiligen Applikation im JIT Verfahren ausgelesen. Diese Vorgehensweise wird in Abbildung 57 dargestellt.

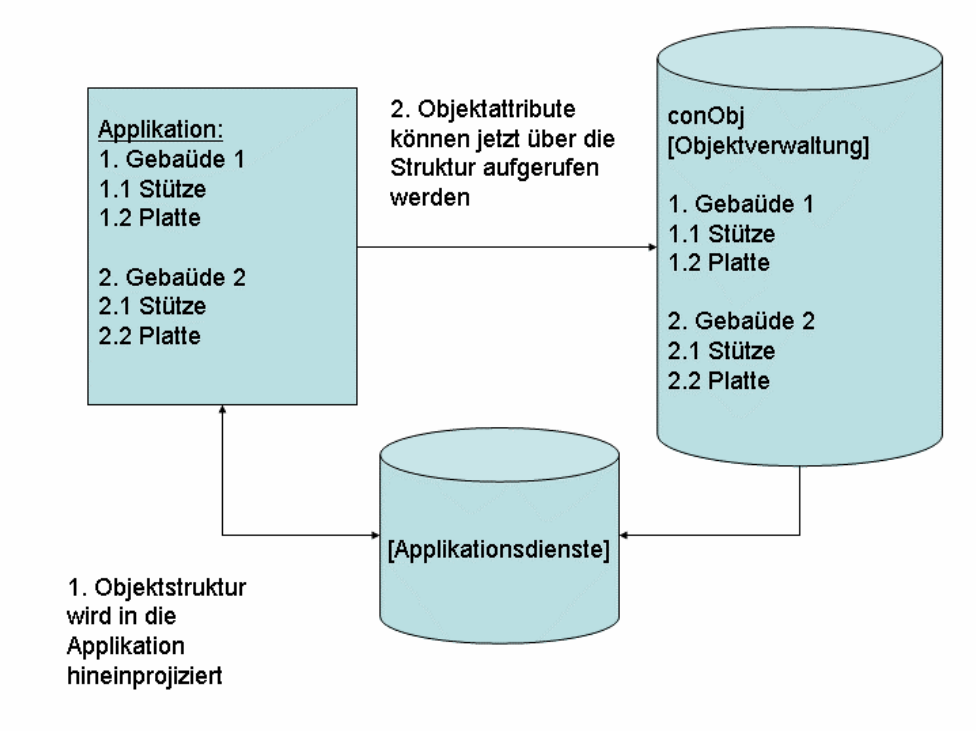


Abb. 57: Einfügen bzw. Extrahieren von Gliederungen in andere bzw. aus andere(n) Applikationen. Objektspace Administration

Zur Prüfung dieses Konzeptes kommen drei Applikationen in Betracht:

- Nemetschek Allplan 2005 – bedingt dienstfähiges 3D CAD
- Primavera P3e – moderne Terminplanungsapplikation
- KUBUS – Kalkulations- und Kostenkontrolle

Allplan 2005 ist bedingt dienstfähig. P3e bietet zwar eine umfangreiche API, diese ist jedoch nicht leistungsfähig und muss aufgrund der Java/.NET Inkompatibilität erst mit einer Dienst-Schnittstelle versehen werden, um die Integration in der .NET Framework zu ermöglichen. Diese Dienste sind direkt über die Datenbankschicht implementiert worden.

3.1 3D CAD

Die technologische Möglichkeit einer kompletten Trennung der 3D CAD Objekte aus Allplan heraus wurde in Kapitel 2.2 oben im Detail erläutert.

CAD Programme weisen generell zwei Arten von Körpern auf:

- Volumenkörper und
- Flächenkörper

Volumenkörper werden mit mathematischer Formel dargestellt und können auf Grund der geometrischen Informationen im mathematischen Modell sehr exakt dargestellt werden allerdings mit der Folge hoher Belastung des Speichers und der Rechenkapazität. Bei dem Export von Volumendaten ist wichtig, dass die Hostapplikation über einen Volumenkern verfügt, einen sog. ACIS-Kern oder Parasolid-Kern.

Flächenkörper weisen eine bestimmte Anzahl von Dreiecken (Polygonen) auf, so dass immer ein Verlust der Darstellungsschärfe entsteht jedoch bei einer verbesserten Rechenleistung. Der Detaillierungsgrad der Polygonmodellierung kann auch eingesetzt werden, um LOD (Level Of Detail) Effekte zur Steigerung der Rechenleistung zu erzielen (siehe Anhang A.II.3.1.5), indem entfernte oder weniger wichtige Objekte grundsätzlich mit wenigen Polygonen modelliert werden. Im Gegensatz dazu können nahe oder wichtige Objekte mit vielen Polygonen detailgetreu modelliert werden, um einen hohen Realismusgrad zu erreichen.

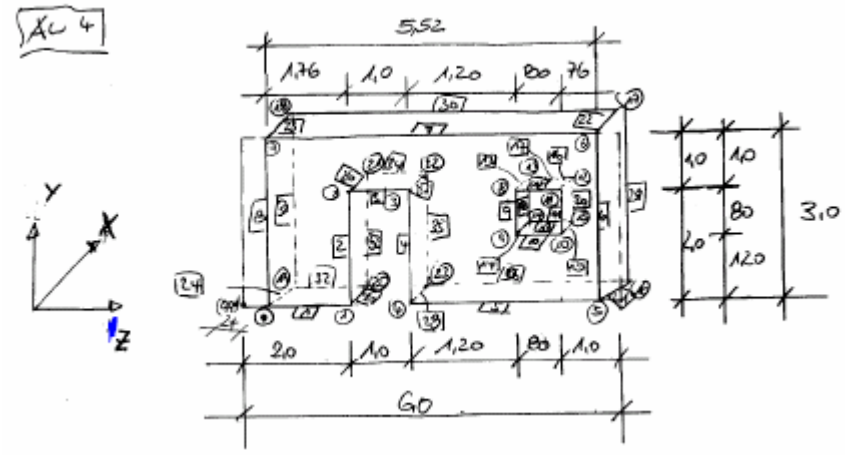


Abb. 58: Eine einfache Wand mit Tür´- und Fensteröffnung als Skizze

Die automatisierte Berechnung von Volumina und Flächen erfolgt beim NOI über Volumenkörper. Allerdings wird seitens der Fakultät Bauwesen (Universität Dortmund) eher in architektonischen Elementen gearbeitet, die solche Berechnungen nicht unterstützen. Mit dieser Vorgehensweise können Elemente in Punkten und Linien aufgeteilt (Abb. 58) und anschließend in einer Matrix konvertiert werden, um die Darstellung in einem 3D Format zu ermöglichen (Abb. 59).

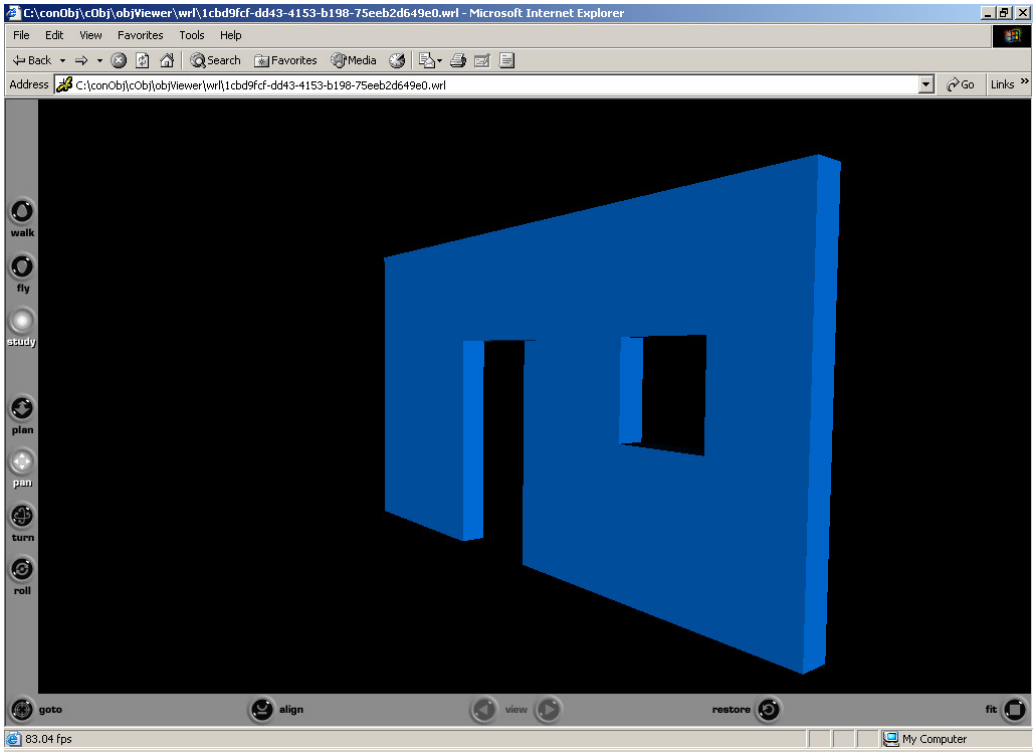


Abb. 59: Eine einfache Wand mit Tür- und Fensteröffnung als Skizze (oben) und als VRML Element eines im Objektmodell enthaltenen Objekts dargestellt. Dieses geometrische Objekt wurde als 3D Objekt mit den Klassen im Namespace objLib.geometry konstruiert.

Die Zuordnung der CAD Objekte zu den Knotenpunkten in einer Objekthierarchie findet analog zum Kapitel 2.3 statt. Diese Vorgehensweise ist noch nicht ausgereift und könnte den Anforderungen der Praxis nicht genügen. Sie reicht jedoch zur Demonstration des Systems.

3.2 Terminplanung

Im europäischen Raum werden folgende Terminplanungsprogramme verstärkt eingesetzt:

- Primavera und deren Derivate,
- MS Project und
- Power Project.

Alle 3 Programme unterstützen Schnittstellen, die durch .NET Wrapper Klassen zugänglich gemacht werden können. Die aktuelle Version von Primavera, P3e, unterstützt eine Java basierte API. Für das hier beschriebene Objektmodell wurde die P3 Schnittstelle als verteiltes System mit .NET Remoting implementiert (siehe Abb. 65 und Listing unter der Namespace *rem.p3e* des Projekts *remObj* in Anhang E).

Die API für P3e wirde mit Java implementiert. Dieses Objektmodel greift wiederum auf die darunter liegende Datenbank (Oracle oder SQL Server) zu. Da die Möglichkeit eines direkten Zugriffs auf das Java Objektmodel wegen der Inkompatibilität von Java und .NET nicht existiert, muss eine sogenannte Bridge (Brücke) zwischen dem P3e API und dem Objektmodell geschaffen werden. Hier gibt es mehrere Möglichkeiten, die alle jedoch auf verteilte Technologien wie SOAP, IIOP oder Socket basierte Methoden zugreifen. Das Produkt JNBridge bietet mittels einer technologischen Brücke zwischen .NET und Java (Abb. 60) die Möglichkeit, sogenannte. Wrapper Klassen aus dem Java Objektmodel zu generieren, die dann in einem .NET Projekt angebunden werden und von den dort benutzten Sprachen wie C#, VB.NET, MC ++ oder J#, eingesetzt werden können (siehe Abb. 61).

Es stellt sich jedoch heraus, dass die von JNBridge generierten Klassen sich allein für die Ansprache aus J# eignen. Das liegt an den in J# benutzten Konventionen, die der Java Sprache ähnlich sind. Somit wird eine Wrapper Klasse in .NET auf Basis J# geschrieben, die wiederum die Proxy Klassen, die mittels JNBridge [JNBridge, 2004. WS] aus der P3e Integration API generiert werden, ansprechen. Diese Wrapper Klassen werden dann in einer .NET Remoting Infrastruktur (Abb. 62) eingebettet, um den Aufruf auf das Terminplanungsprogramm als Windows Dienst zu ermöglichen (Abb. 65).

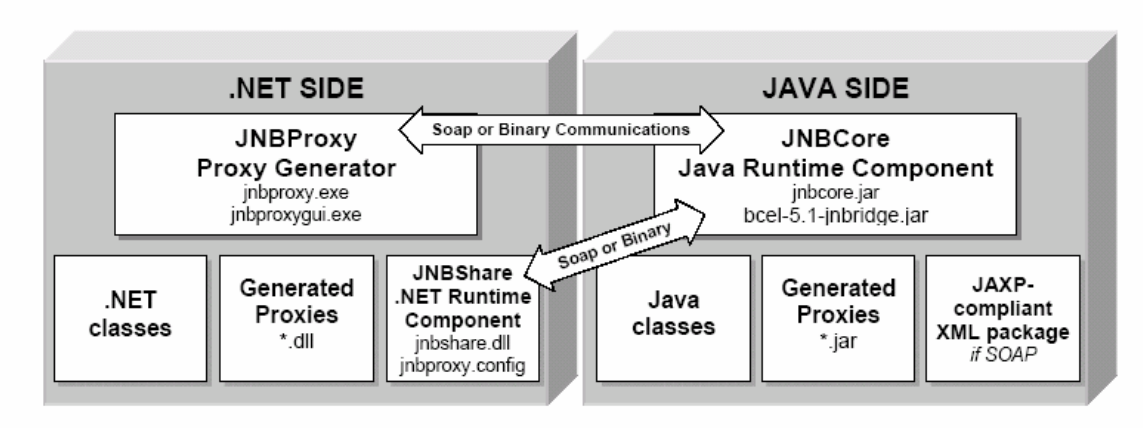


Abb. 60: Architektur von JNBridge [JNBridge, 2004. WS]

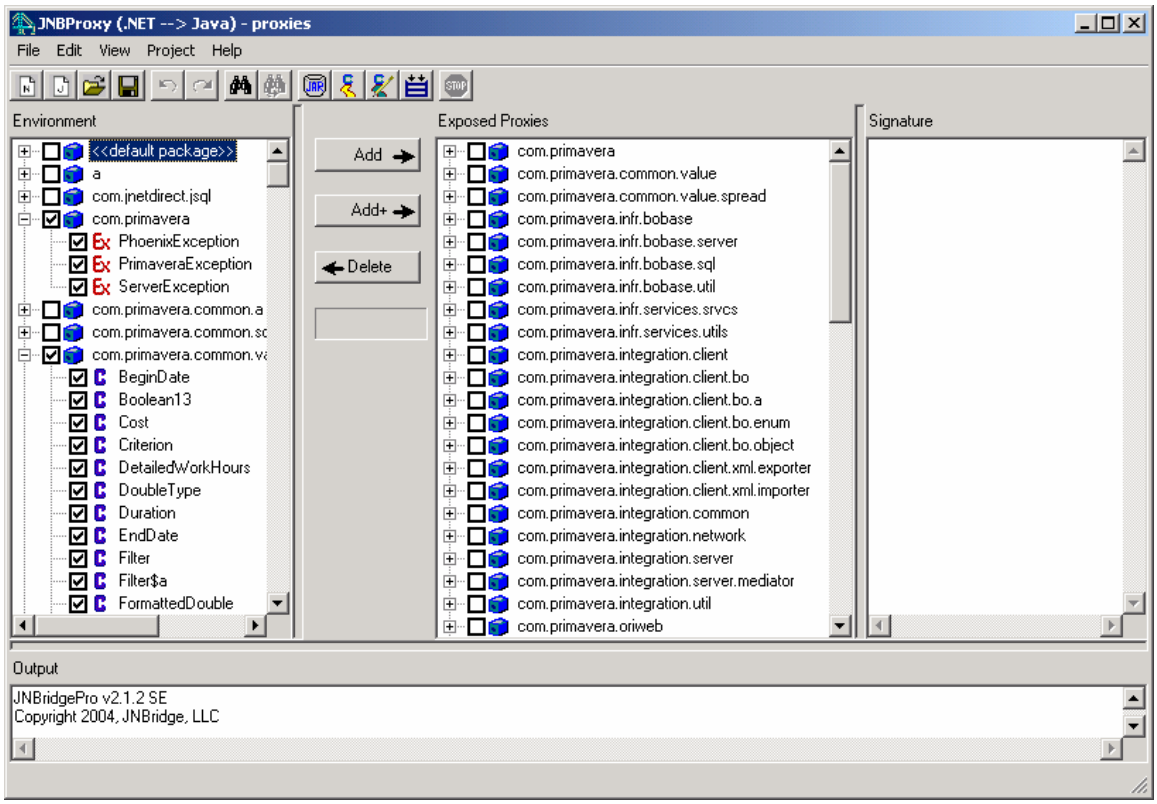


Abb. 61: JNBridge Proxy Class Generatorion Tool zur Generierung von Proxy Klassen für den Zugriff auf die P3e Integration API [JNBridge, 2004. WS].

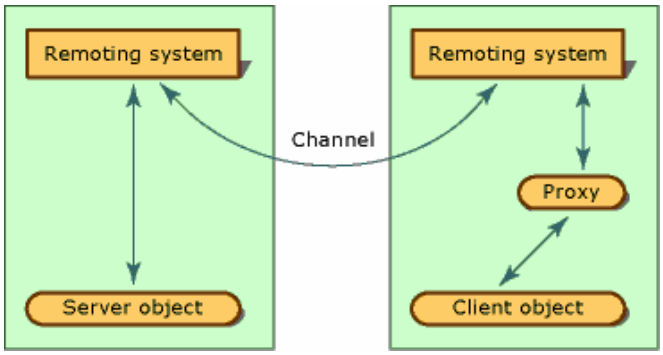


Abb. 62: Die .NET Remoting Infrastruktur [.NET Framework Reference, Microsoft Corporation, 2003. WS]

Abbildung 63 stellt die Konfigurationsdatei des Servers dar. Das verteilte Objekt wird als „well-known“ klassifiziert. Durch die feste URL ist sie dem Client bekannt (Abb. 64). Well-known Objekte besitzen zwei mögliche Modi:

- SingleCall – das Objekt wird bei jedem Aufruf instanziert.
- Singleton – Das Objekt wird nur einmal instanziert, steht allen Clients zur Verfügung.

Weiterhin wird im *ref* Attribut der Wert *tcp* gesetzt. Diese Angabe zeigt dem Remoting Framework an, den *binary formatter channel*, der auf TCP beruht, einzusetzen. Dieser *channel* ist zwar im Gegensatz zum HTTP (SOAP) *channel* nicht für den Zugriff über eine Firewall geeignet, es sei denn, der Port (hier 9002) wird freigegeben. Er zeichnet sich jedoch durch erhöhte Leistung vor allem bei Marshall-By-Value Remoting aus. Die Möglichkeit der Verschlüsselung ist auch hier gegeben, verbunden allerdings mit einem Leistungsabfall.

```
<configuration>
   <system.runtime.remoting>
      <application>
         <service>
            <wellknown
               mode="Singleton"
               type="remObj, remObj"
               objectUri="remObj.rem"
            />
         </service>
         <channels>
            <channel ref="tcp" port="9002"/>
         </channels>
      </application>
   </system.runtime.remoting>
</configuration>
```

Abb. 63: Die Konfigurationsdatei des remote Servers bestimmt das Übertragungsprotokoll (Channel) und das verteilte Objekt remObj.

```
<configuration>
   <system.runtime.remoting>
      <application>
         <client>
            <wellknown
               type="remObj, remObj"
               url="http://localhost:9002/remObj.rem"
            />
         </client>
      </application>
   </system.runtime.remoting>
</configuration>
```

Abb. 64: Konfigurationsdatei des Clients

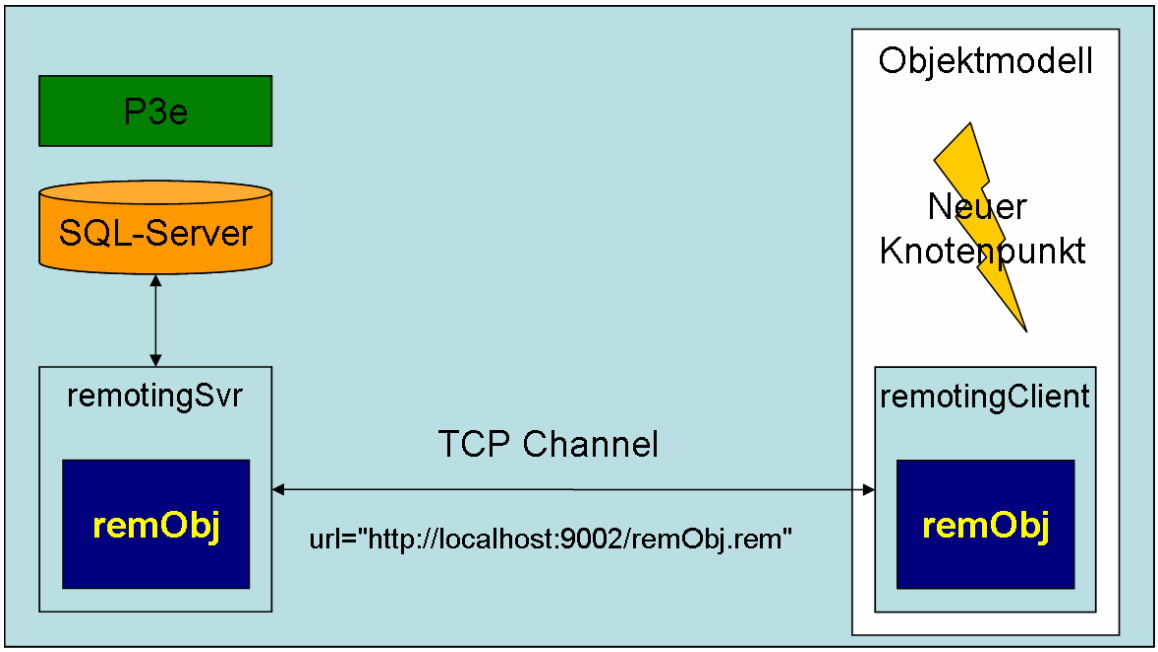


Abb. 65: Realisierung der Anbindung P3e-Objektmodell mittels .NET Remoting; Vorgehen der verteilten Infrastruktur beim Einfügen eines neuen Knotens.

Theoretisch sind gute Reaktionszeiten mit .NET Remoting und dem TCP-Channel Formatter zu erreichen. Auf die in der Praxis bessere Perfomance des HTTP-Channels wurde früher hingewiesen. Dabei ist zu beachten, dass das Hosten von Objekten auf dem IIS eine starke Konformität der WSDL Definition erfordert.

3.2.1 Einschränkungen des P3 Objektmodells

P3 hat ein umfangreiches Objektmodell für die Versionen 2.0b und 3.0 realisiert. Dieses Objektmodell, auch als RA bekannt, wird inzwischen nicht mehr von Primavera unterstützt. Da dieses Objektmodell auf einer 16-bit Architektur basiert, ist es nicht tauglich für den Einsatz in der .NET Umgebung. RA ermöglicht den Zugriff auf jegliches Objekt und dessen Methoden, auch auf das Projekt Objekt, und somit die Möglichkeit, das Projekt zu berechnen und Nivellierungen durchzuführen.

Die aktuelle Version der Primavera API, die Primavera Integration API 5.0, ist Java basiert und daher technologisch komplett anders aufgebaut, als die Vorgänger Version. Die neue API ist technologisch ausgereift und einer der wenig professionellen Schnittstellen im baubetrieblichen Bereich. Die Java Basis der Schnittstelle verursacht Schwierigkeiten bei der Integration in .NET. Die in diesem System genutzte Lösung, JNBridge, wurde oben beschrieben.

Es gibt jedoch drei Begrenzungen hinsichtlich eines integrierten Objektmodells:

- Die Zeitschiene wird einer Aktivität zugeordnet, dieser Aktivität können Gliederungsknoten zugeordnet werden. Dadurch, dass die Aktivität und nicht der Gliederungsknoten die Basis der Zeitschiene darstellt, kann nur jeweils ein Gliederungsknoten pro Gliederung und Aktivität zugeordnet werden. Diese Beschränkung gilt sowohl in den Versionen P3 2.0b und 3.0 wie auch P3e.
- Die Zeitschiene hat als kleinste Einheit jeweils entweder Minuten, Stunden oder Tage. Diese Entitäten sind nicht in sich verschachtelt, wie in der Kalkulation üblich.
- P3 speichert in den Versionen 2.0b und 3.0 alle Daten in einer Btreive Datenbank und kennt nur dieses Datenbankformat. Es besitzt auch nicht ein extern zugängliches Ereignismodell. Daher muss eine Schnittstelle das COM Interface implementieren.
- Änderungen in P3 oder dem integrierten Objektmodell werden mit Hilfe der Schnittstelle im jeweils anderen System aktualisiert. Aufgrund der geschlossenen Architektur von P3 kann sich die Applikation keine Änderungen von Daten außerhalb der Datenbank merken.
- P3e ist mit dem SQL Server oder Oracle als Basis wesentlich leistungsfähiger; die oben beschriebene Problematik, der nicht nach außen weitergegebenen Ereignisse, trifft allerdings auch hier zu.

3.2.2 Datenaktualisierung

Das Objektmodell erstellt ein P3e-Projekt, in dem alle Knotenpunkte gespeichert werden, die mit einer Zeitschiene versehen worden sind. Beim Laden des Objektmodells werden die neuesten Informationen aus dem P3e-Projekt gezogen. Ein Benutzer der P3e-Applikation kann jedoch wie gewohnt die Termine der Aktivitäten, die im Objektmodell durch Gliederungsknoten dargestellt werden, ändern. Das Objektmodell wiederum aktualisiert die Objekte durch Abgleichen mit den Vorgängen in der P3e-Datenbank mit Hilfe der P3e API Schnittstelle.
Problematisch an diesem Modell ist die Regelung der Zugriffsrechte und die Handhabung von Lösch- und Einfügungsvorgängen:

- Zugriffsrechte verhindern, dass in P3e und dem Objektmodell Daten gleichzeitig geändert werden (Synchronisierung).
- Die Datenbasis sichert, dass in beiden Systemen stets die gleichen Gliederungsknoten bzw. Vorgänge vorhanden sind.

3.3 Kostenschätzung

Die Schnittstelle zur Kostenschätzung wurde exemplarisch als Beweis der beliebigen Erweiterbarkeit des Systems für den Umfang dieser Arbeit bedacht. Ein objektorientiertes Programm zur Kostenschätzung kann die Objekte des Objektmodells benutzen, um eine Kostenschätzung für das Projekt zu berechnen. Dazu werden Objekte entweder als Hilfskonstrukte herangezogen (für Ausführungskosten), oder bestimmen direkt das endgültige Resultat. Da alle Objekte eine Eigenschaft vom Typ (Klasse) Kosten besitzen, was wiederum ein Kostenmodell modelliert, kann also das Kostenmodell als Eintrittspunkt für Programme zur Kostenschätzung dienen.

Grundsätzlich ist jedes Kostenmodell mit nur zwei Klassen modellierbar[12]:

- einer Ressource – entspricht einem Vektor - KAS, Kosten, Ressourcen und
- einer Komponente – entspricht einer Matrix – Bearbeitungsobjekte wie Vorwerte, Elemente, direkte Kosten und indirekte Kosten.

---

[12] Eine intensive UML Analyse eines ausgereiften Kalkulationsprogramms sowie eines baubetrieblichen Kalkulationsprozesses ergab, dass die Grundstruktur einer Kalkulation nur aus zwei Klassen besteht, wovon die zweite von der ersten erbt. Da ein Modell zur Kostenschätzung im Prinzip einfacher in der Konstruktion als ein Kalkulationsmodell ist, kann davon ausgegangen werden, dass diese Aussage auch für ein Modell zur Kostenschätzung zutrifft.

Indem in der ersten Ebene Ressourcen definiert und diese wiederum in Komponenten aufgenommen werden, verschachteln sich die Komponenten wiederum ineinander, wodurch eine Kostenschätzung als Hierarchie verschachtelter Komponenten entsteht. Somit ist das Objektmodell einfach und lehnt sich an eine objektorientierte Betrachtung an.

In diesem Objektmodell wurde die Klasse *cost* in der namespace *objLib.costs* etabliert, um ein minimales Kostenobjekt darzustellen. Hiermit ist die Schnittstelle zu einem Programm zur Kostenschätzung gegeben.

Durch das dynamische Ereignismodell kann diesem Attribut auch eine Ereignisbehandlung zugewiesen, bzw. kann das Objekt ausgebaut werden. Veränderungen in der Geometrie oder der Positionierung würden Methoden aufrufen, die diese Veränderungen aus Kostengesichtspunkten berücksichtigen.

Zeitliche Veränderungen würden Auswirkungen in der Finanzierung eines Objektes verursachen. Da die Projektfinanzierung die Summe der Finanzierung der einzelnen Objekte ist, würde eine objektspezifische Finanzierungsberechnung in Summe die Projektfinanzierung darstellen. Die Finanzierungsberechnung würde als Methode auf Änderungen in dem Attribut „timeLine“ mittels eines ausgelösten Ereignisses reagieren und somit die Projektfinanzierung aktualisieren. Allerdings bedarf eine solche Betrachtung eines globalen Messagemap, was nicht Bestandteil dieser Arbeit ist und auch zu Problemen mit unendlichen Ereignisschleifen führen kann.

4 Architektur der Applikation

Die Gesamtarchitektur folgt den im vorangegangenen Text fachlich definierten Erfordernissen(). Insbesondere wurden folgende grundlegende Anforderungen an die Softwarearchitektur umgesetzt:

- Zentraler Datenspeicher
- Verteilte Dienste für Anbindung an Fremdsysteme
- Late-Binding mittels Reflection
- Dynamische Erweiterung des Objektmodells mittels `System.CodeDom`
- Abfangen und Verarbeiten von Ereignissen auf Objektebene

Ein wichtiges Merkmal des Systems war die uneingeschränkte Einbindung von externen Programmen durch Dienste (siehe Abb. Arch.1, Arch.2 und Arch.3). Zurzeit

wurden nur drei Dienste integriert, aber das System kann um beliebig viele Dienste erweitert werden. Allerdings sind die auf dem Markt befindliche Dienste in sehr unterschiedlichen technologischen und qualitativen Varianten verfügbar. Die Bandbreite reicht von vollwertigen Schnittstellen wie der P3e Integration Service bis zu rudimentären Einbindungen wie XML Dateien die von anderen Applikationen generiert wurden. Daher kann ein solches System nur den Grad der Automatisierung erreichen, der von den angebundenen Diensten ermöglicht wird. Das Grundsystem wurde jedoch für einen wesentlich höheren Grad der Automatisierung ausgelegt um weiterentwickelte Systeme anbinden zu können.

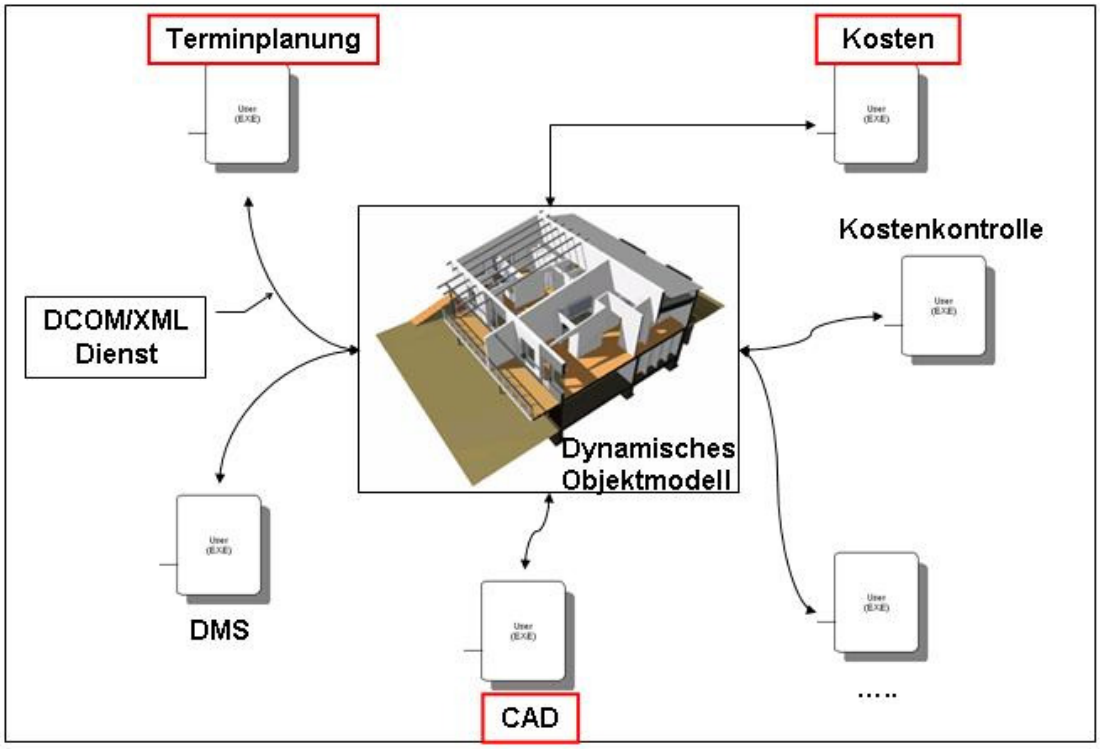


Abb. Arch.1. Externe Dienste können uneingeschränkt hinzugefügt werden.

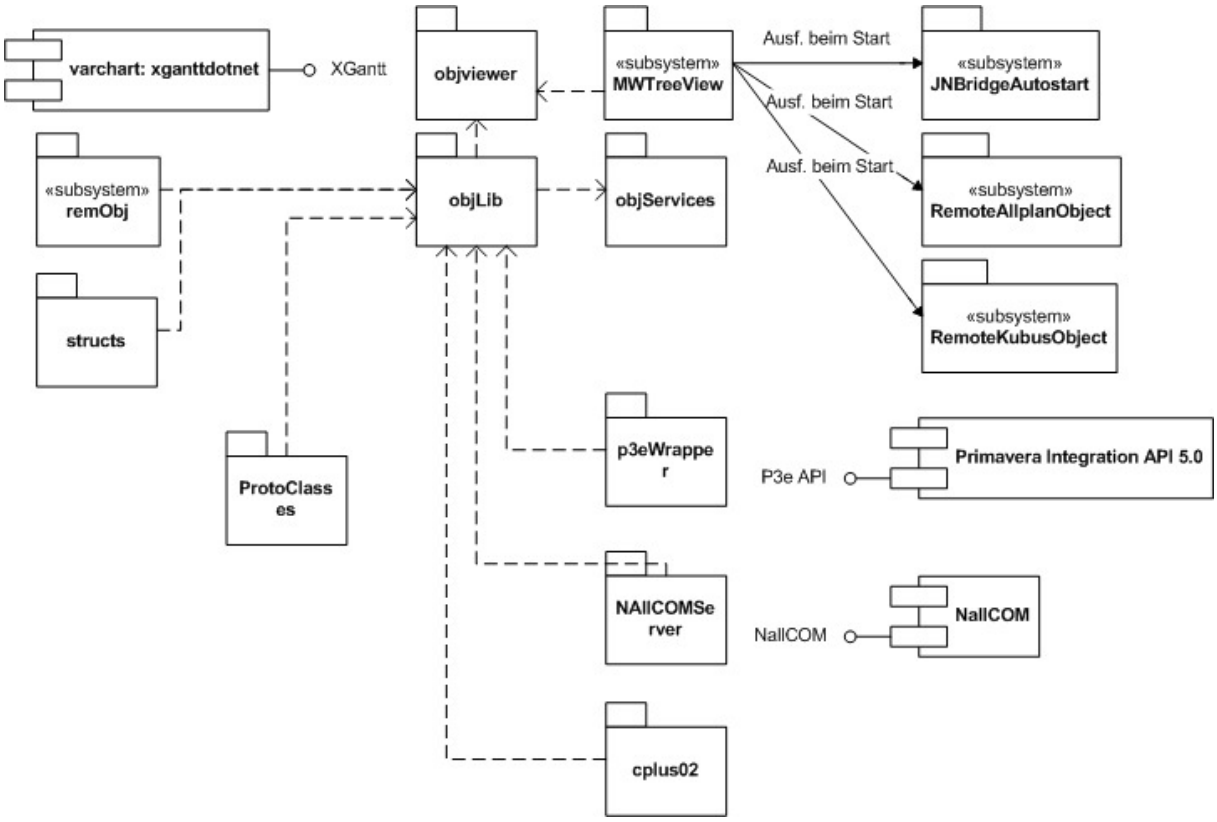


Abb. Arch.2. Architektur der Pakete des Gesamtsystems

Modul	**Funktion**
Varchart:xganttdotnet	Steuerelement zur Anzeige von Terminen der einzelnen Objekte.
ObjectViewer	Applikation zur Darstellung der Gebäudestruktur und aller dynamisch generierten Objekte.
MWTreeView	Erweiterte Baumstruktur zur Auswahl von einigen Objekten oder eine Gruppe von Objekten.
JNBridgeAutostart	Modul zur automatischen Instanzierung des Proxy Servers für die Interkommunikation zwischen Primavera und dem System.
remObj	Verteiltes Objekt zur Interaktion mit p3eWrapper.
objLib	Bibliothek für alle Hilfsklassen zur Steuerung der Benutzeroberflächen, Datenbank zugriffe und dynamische Objektgenerierung.
objServices	Web Dienst für Interaktion mit Datenbank.
RemoteAllplanObject	Verteiltes Objekt zur Interaktion mit NallCOMServer.
structs	Hilfsklasse zur Zwischenspeicherung von Informationen für

	verteilte Zugriffe (Logins), Aufrufe von Methoden und anderen Situationen.
RemoteKubusObject	Verteiltes Objekt zur Interaktion mit Kubus.
ProtoClasses	Namespace zur Verwaltung aller dynamisch generierten Klassen. Auch die Basisklasse constructionobject ist hier untergebracht.
p3eWrapper	Klassen zum Aufruf von P3e API Methoden mittels Proxy Klassen die mit JNBridge generiert wurden. Die P3e API wird über ein Socket auf TCP Ebene angesprochen.
Primavera Integration API 5.0	Programmierschnitstelle zu Primavera.
NAllCOMServer	Proxy für einen .NET Remoting Zugriff auf die Klasse NallCOM
NallCOM	Allplan NAllCOM Schnittstelle (COM)
cplus02	Methoden zur Steuerung der NOI aus .NET (MC++)

Abb. Arch.3 Funktion der einzelnen Module im Gesamtsystem

Zur Laufzeit werden Objekte ja nach Anforderung dynamisch generiert (Abb. Arch.4). Hier wird nur die dynamische Generation von Bauobjekten angezeigt da die in der Applikation generierten Objekte nur als Hilfskonstrukt für die Ausführung der Methoden in System.CodeDOM und System.Reflection dienen und somit weniger interessant sind.

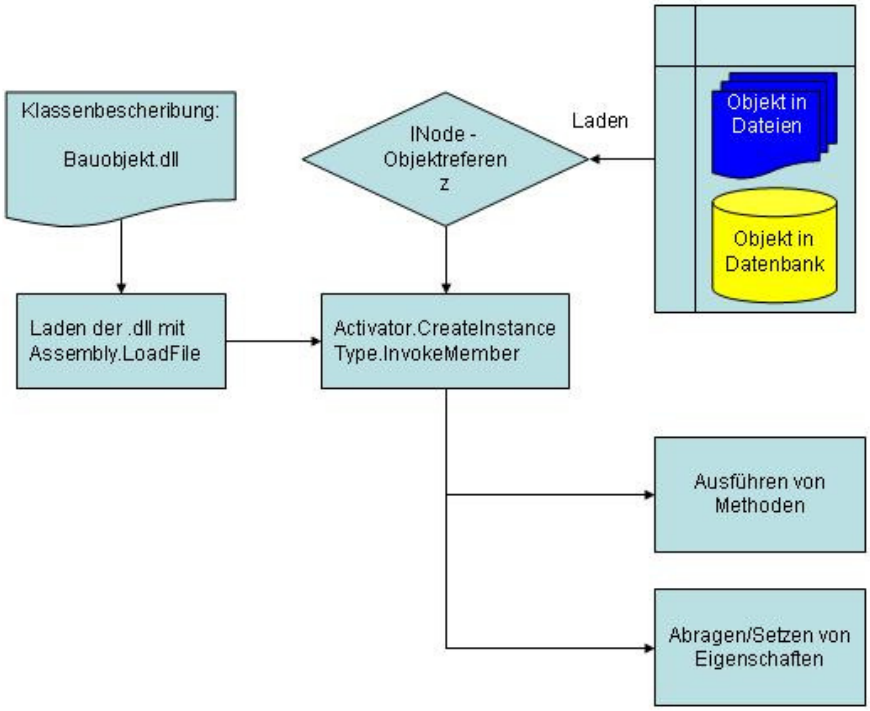


Abb. Arch.4 Dynamische Objekt-Generierung

4.1 Datenbank

Die persistente Ablage der Objekte in der Datenbank erfolgt über Hilfsklassen. Es werden sowohl die eindeutigen Kennungen der Objekte als auch eine Serialisierte Version der Objekte in der Datenbank gespeichert. Die Serialisierung erfolgt mittels der `System.ISerializable` Schnittstelle. Objekte werden zusätzlich auf der Festplatte in XML Dateien gespeichert. Die generierte Klassenbeschreibungen zur Generierung des Quellcodes werden auch in der Datenbank und auf der Festplatte gespeichert. Das Format der Klassen ist Text/XML da die Klasse `XmlSchema` nicht serialiserbar ist.

4.2 Dienste

Die externen Dienste werden mittels `Reflection` eingebunden. Dadurch erhöhen sich zwar der Speicherbedarf und die Rechenzeit des Systems, dies hat jedoch den Vorteil, dass die vollständige Beschreibung der Dienste zur Kompilierzeit nicht bekannt sein muss, sonder mittels Late Binding zur Laufzeit ermittelt wird. Dies ermöglicht den Einsatz von neuen Softwareversionen der externen Systeme ohne eine Neukompilierung des Systems.

Im Wesentlichen werden zwei grundlegende Technologien verteilter Dienste unter-

schieden: Web Dienste und objektorientierte Middleware wie DCOM, CORBA, RMI. Letztere sind plattformspezifisch was eine plattformneutrale dynamische Einbindung nahezu unmöglich macht. XML-Dienste können mittels `Reflection` und `CodeDOM` (Abb. Arch.5) eingebunden werden.

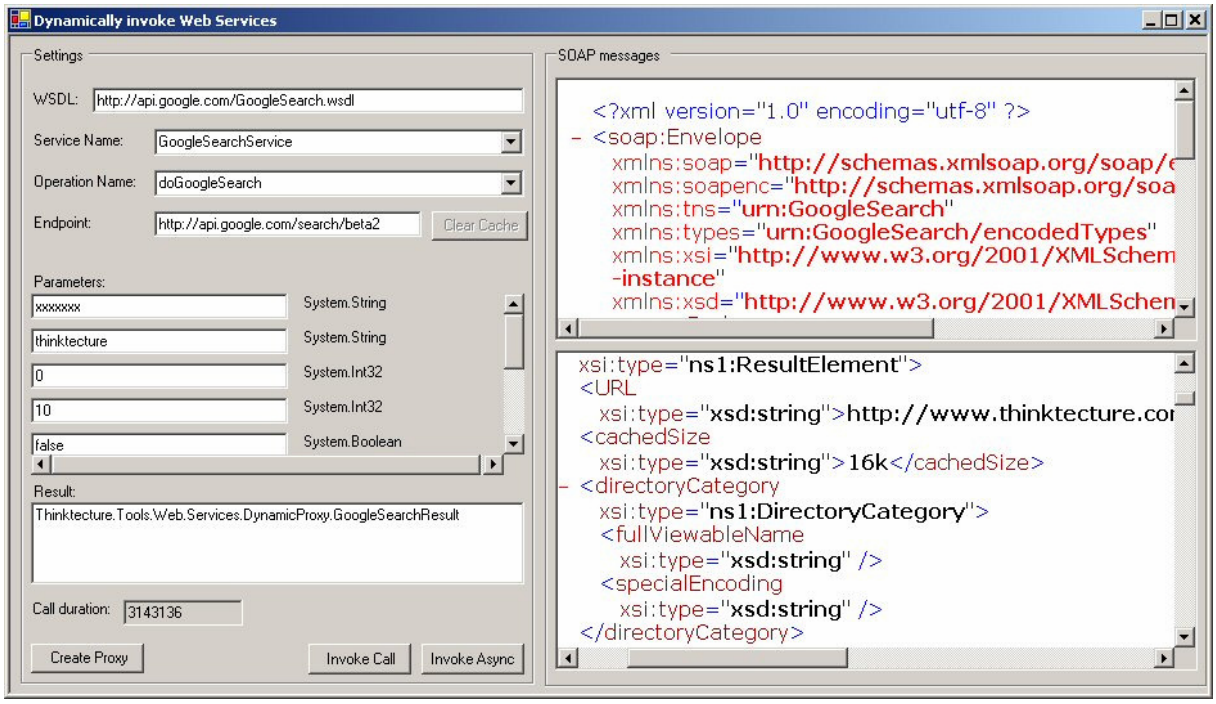


Abb. Arch.5. Aufrufen von Xml Dienste ohne vorherige Informationen zu den Klassen (Late-Binding). [Weyer, C., 2004. S. 1]

Aufgrund der großen Unterschiede der drei in dieser Arbeit integrierten Dienste (Allplan, P3e, KUBUS) ist eine durchgängige Integration mittels `Reflection` nicht möglich.
Lediglich die Einbindung des P3e Dienstes wurde jedoch mittels `Reflection` realisiert, da P3e eine sehr gut entwickelte Schnittstelle aufweist (P3e Integration API).

Die Dienste von Anwendungen die komplexe Objektmodelle benötigen, bieten in der Regel auch nur komplexe Schnittstellen an. Im Falle Allplan trifft diese Beobachtung besonders zu, da hier ein COM-Server sowie wie noch zusätzlich das NOI angesprochen werden müssen damit die Schnittstelle funktioniert (Abb. Arch.6).

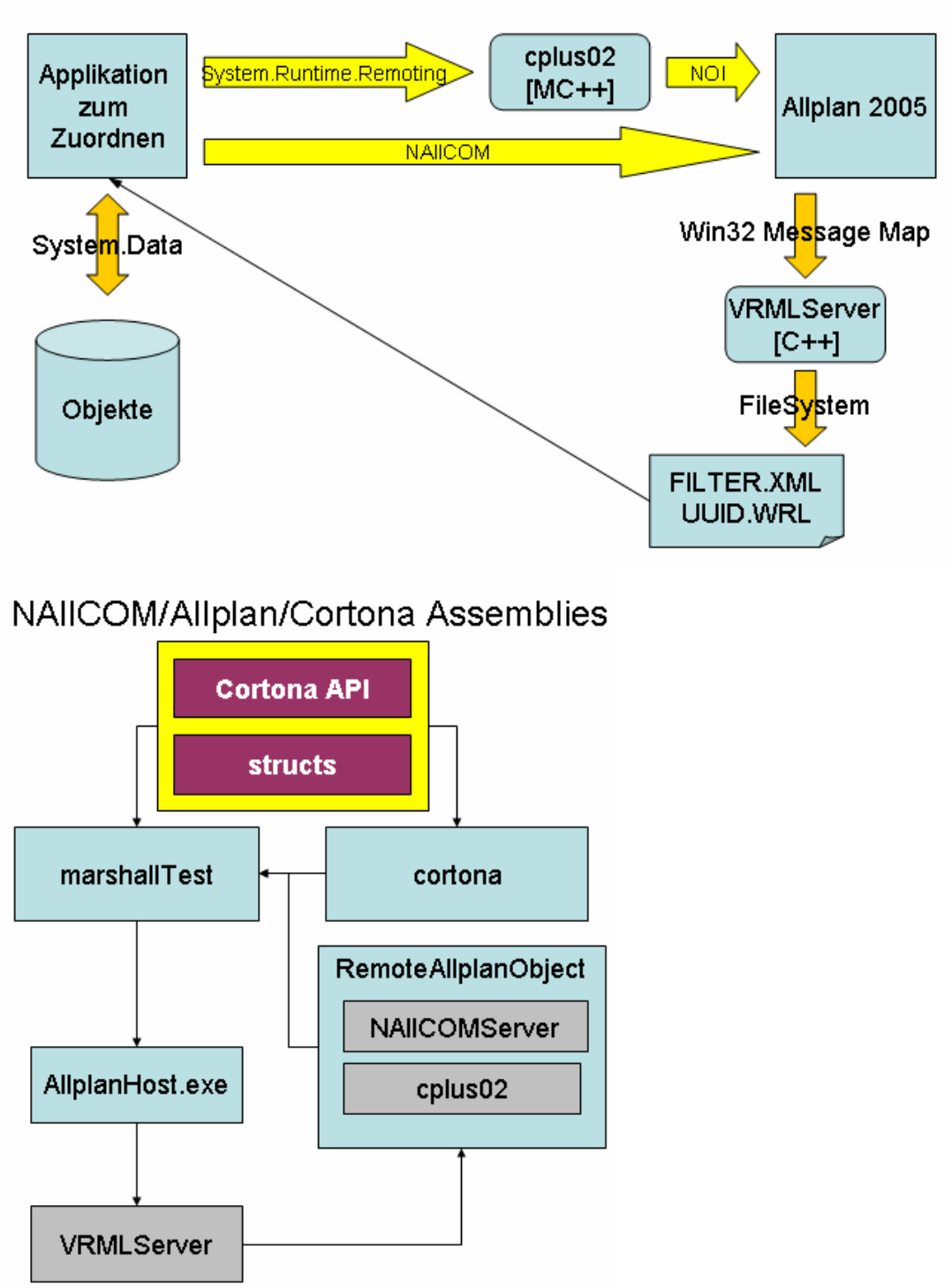


Abb. Arch.6. API's bzw. Module für den pseudo-automatischen Remote Zugriff auf 3D CAD Objekte in Allplan 2005

4.3 Dynamische Erweiterung

Die dynamische Erweiterbarkeit des Systems ist ein zentrales Merkmal dieser Arbeit. Dadurch wird sichergestellt, dass der Modellierung eines Bauprojektes keine Grenzen gesetzt werden. Die Grenzen des Modells erweitern sich entsprechend den fach-

lichen Anforderungen des Projektes und sind nicht durch eine Vorüberlegung festgelegt worden. Das Erhöht zwar die Vielfalt der Klassen ermöglicht aber optionale Erweiterungen des Systems und somit die Akzeptanz des Systems.

Die Vorgehensweise bei der Erstellung einer neuen Klasse wurde anhand eines Beispiels in Kap 3.4.2 erläutert, und wird in Abb. Arch.7 und Arch.8 dargestellt. Diese dynamische Vorgehensweise unterstützt die fachliche Arbeit gemäß Abb. 20.

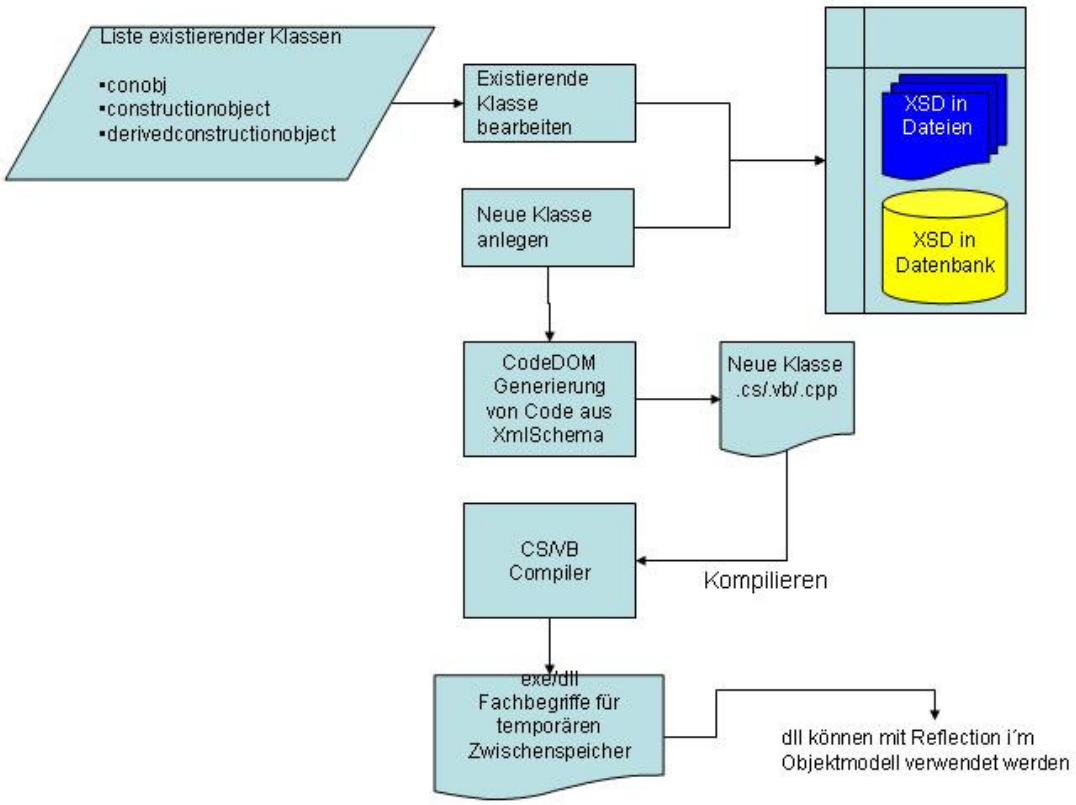


Abb. Arch.7. Vorgehensweise bei der dynamischen Erstellung von neuen Klassen.

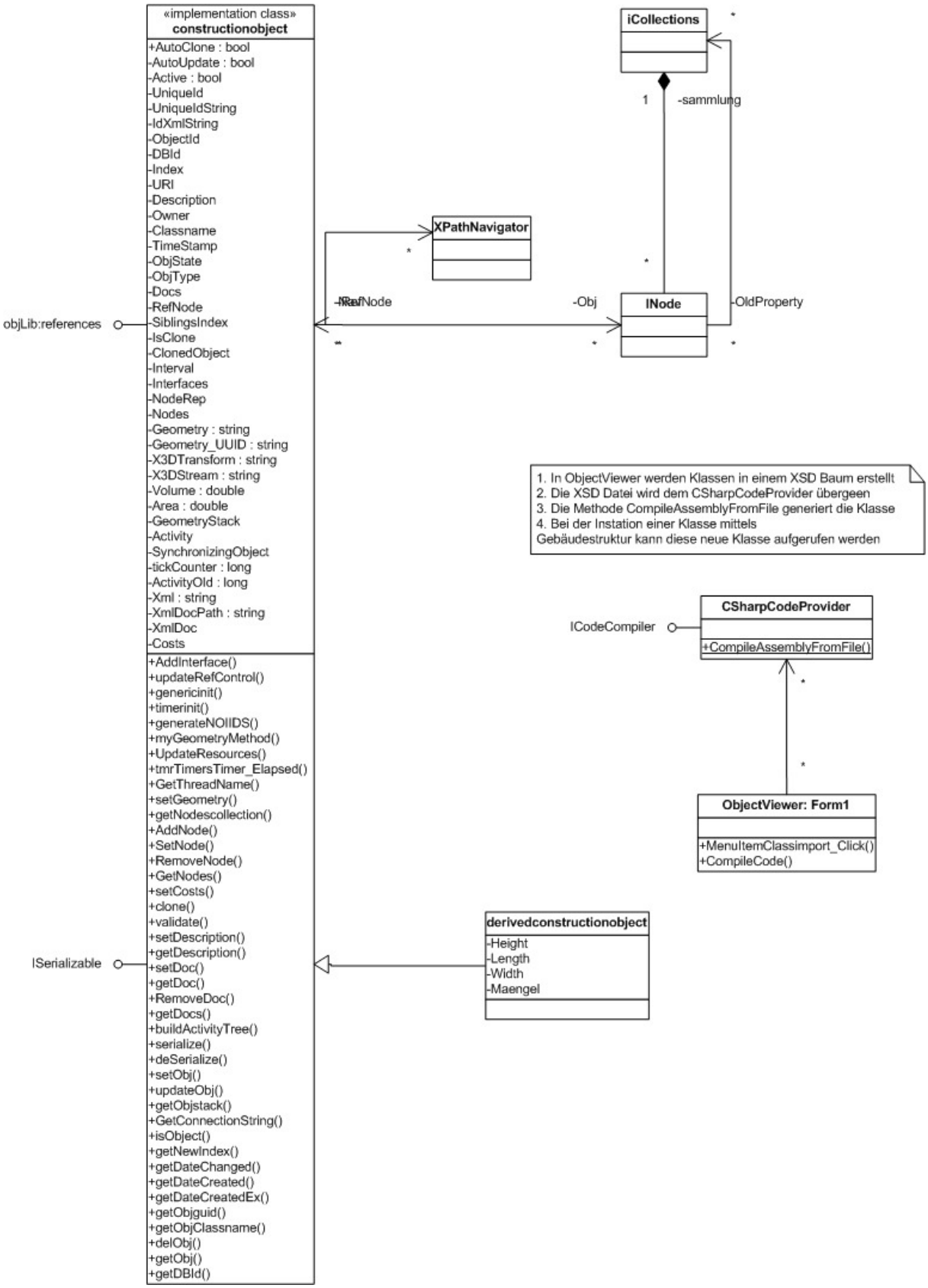


Abb. Arch.8. UML der im dynamischen Erzeugungsprozess verwendeten Klassen

Mittels `Reflection` ist es möglich Instanzen aus existierenden Klassen zu generieren. Das geschieht indem die dll (die die Klasseninformation im Binärformat vorliegt) ge-

laden und mittels Reflection angesprochen wird.
Durch die dynamische Vererbung ist es möglich während der Laufzeit neue Klassendefinitionen zu erstellen, die Klassendefinition zu kompilieren (im dll Format gespeichert) und wiederum daraus neue Objekte zu generieren. Somit können alle Elemente der objektorientierten Programmierung wie z.B. Klassen, Methoden, Eigenschaften und Attribute während der Laufzeit angesprochen und genutzt werden (Abb. Arch.9).

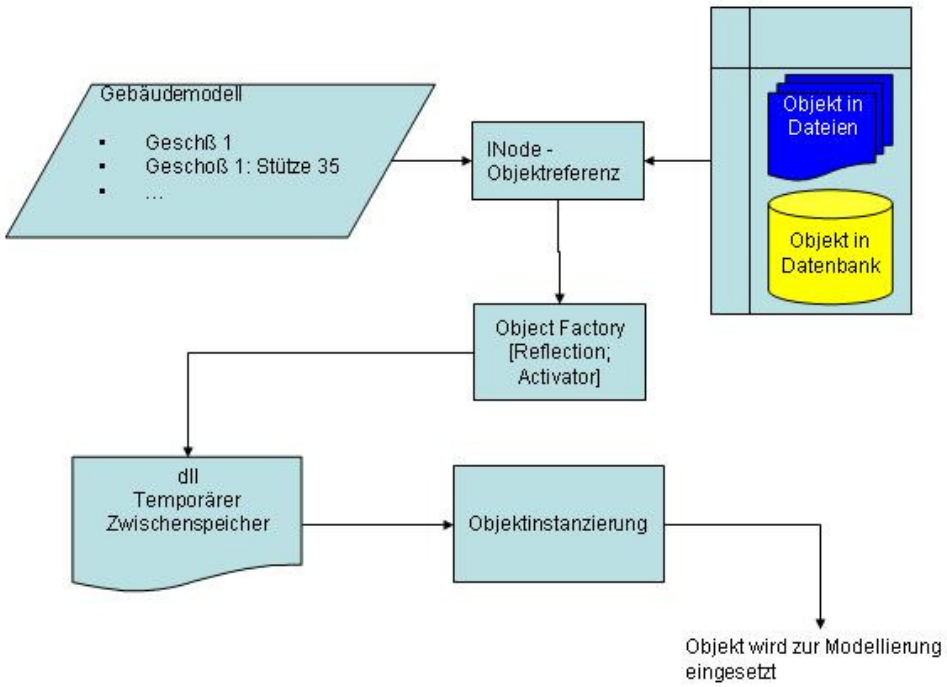


Abb. Arch.9. Neue Objektinstanzen werden zur Laufzeit aus automatisch generierten dll-Dateien erzeugt.

Anhand vom Beispiel in Kapitel **1.2 Das Objektmodell im verteilten Einsatz** soll die Instanzierung von Objekten im dynamischen System erläutert werden (siehe Abb. Arch.10):

Die Klasse Stütze ist vorhanden und in einer dll namens *„bauobjekte.dll“* gespeichert.
„*bauobjekte.dll*“ wird mittels der Methode `Assembly.LoadFile` geladen
Das erzeugte `Assembly` Objekt wird mit den Klassen und Methoden des `System.Reflection` Namespace verwendet um Objekte zu instanzieren, deren Methoden auszuführen und deren Eigenschaften zu setzen oder auszulesen.
Diese Vorgehensweise ermöglicht den Umgang mit Objekten als Erweiterung eines kompilierten Objektmodells.

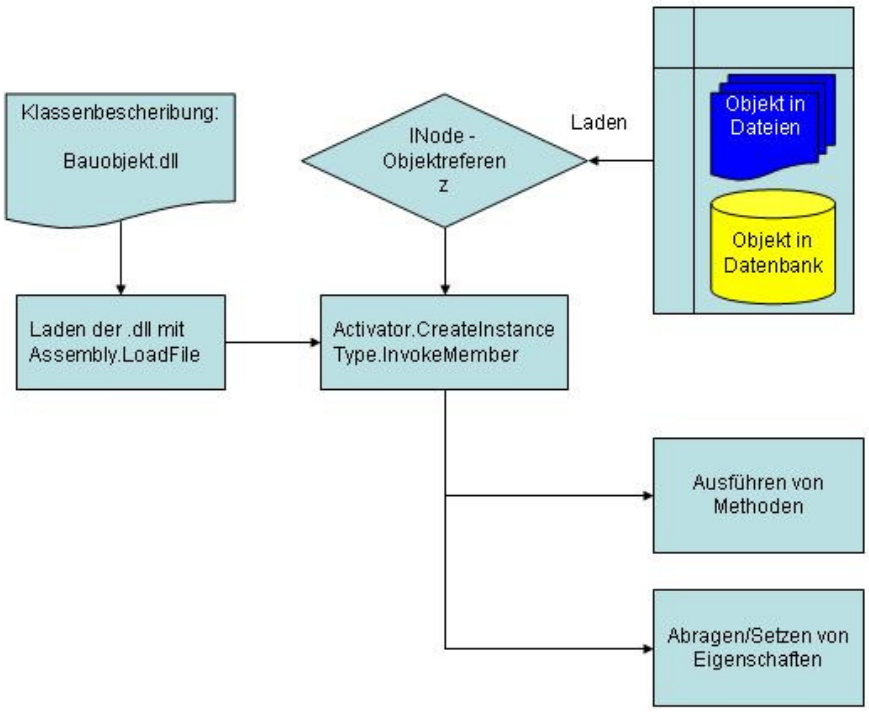


Abb. Arch.10. Dynamische Objektinstantiierung mittels Reflection

4.4 XQuery Abfragen

Abfragen werden mittels `XQuery` getätigt. Dadurch, dass das alle Klassen von der Basisklasse constructionobject erben, und diese mit der `ISerializable` Schnittstelle versehen wurde, können Objekte zu beliebiger Zeit in eine XML Darstellung umgewandelt werden. Dieses Objektformat wird für `XQuery` Abfragen benötigt. Die zu durchsuchenden Objekte werden als Xml Dokumente geladen und mit der Methode `CreateNavigator` in die Form von `XPathNavigator` umgewandelt. Dieser `XPathNavigator` ist eine Zeiger auf das XML Dokument was die Klasse beschreibt. Es werden mehrere solche Zeiger in einer Sammelklasse zusammengefasst (`XQueryNavigatorCollection`). Diese Sammelklasse wiederum wird einen sog. `XQueryNavigator` übergeben. Mit diesem Zeiger kann über die Liste der Zeiger iteriert werden. Anschließend können `XQuery` Abfragen (`XQueryExpression`) mit dem Execute Befehl über die Sammelklasse abgesetzt werden und geben dann einen Xml-basierten Abfragewert zurück. Das Ergebnis als `XQueryNavigator` kann nachfolgend in Text umgewandelt werden `(.ToXml())` und die zurückgegebenen Objektzeiger können verwendet werden um die Objekte in die Applikation zu laden.

Mit dieser Vorgehensweise (Abb. Arch.11) können alle Eigenschaften im Objekt-

graph abgefragt werden.

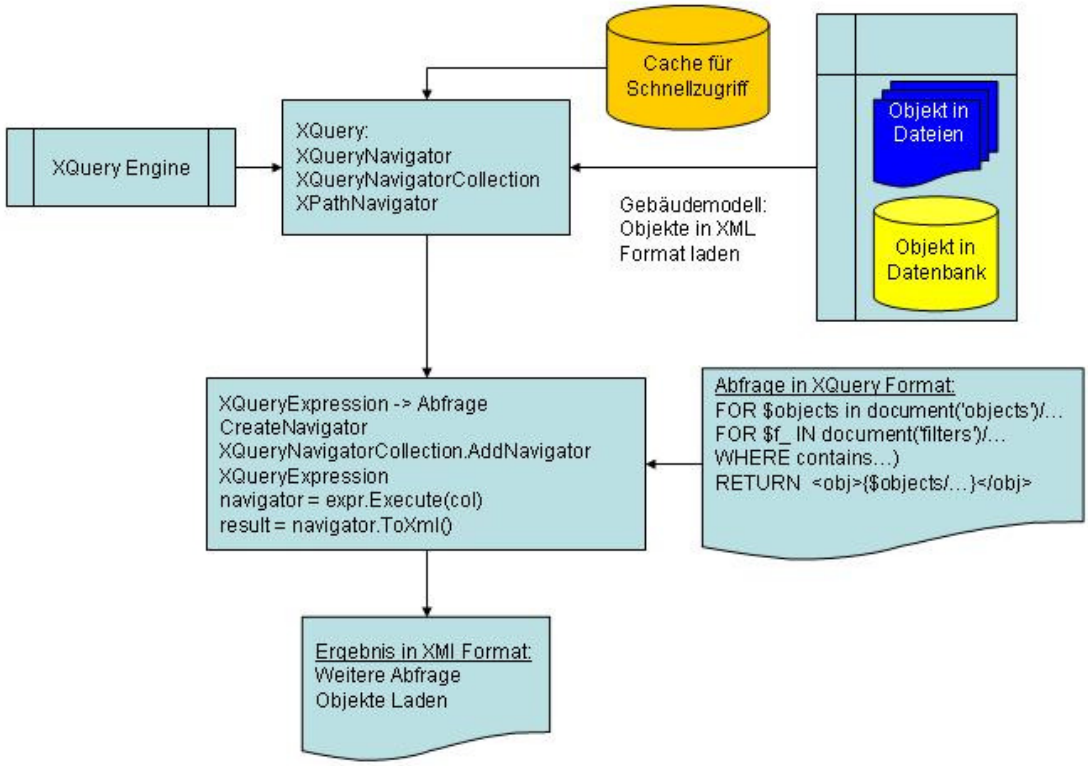


Abb. Arch.11. Schematische Vorgenesweise bei XQuery Abfragen

C Anwendung des Systems

1 Anmerkungen zur Umsetzung des Objektmodells in der Praxis

Der Einsatz des Objektmodells in der Baupraxis kann nur durch die Einbeziehung von etablierten Gliederungs- und Ordnungsstrukturen sinnvoll erfolgen. Solche Strukturen sind im Bauwesen wesentlich weniger international vereinheitlicht als es in der Elektronik oder Informatik der Fall ist. Sie sind auch weniger strukturiert, was u.a; auf die lange Tradition des Bauwesens und den Einfluss der Rechtsprechung zurückzuführen ist.

Zum Beispiel regelt die Norm ISO 11898/11519 im Maschinenbau die Konstruktion und Kodierung des CAN BUS (Controller Area Network) (Abb. 66). Ein weiteres Beispiel: Ein Protokoll zur Verschlüsselung von IP Paketen in Netzwerken, das IPSec, wird durch folgende RFC's (Request For Comment) verkörpert:

- RFC 2401: Security architecture for the Internet Protocol
- RFC 2402: Authentication Header
- RFC 2406: Encapsulating Security Payload
- RFC 2407: IPsec Domain of Interpretation (IPsec DoI)
- RFC 2408: Internet Security Association and Key Management Protocol (ISAKMP)
- RFC 2409: Internet Key Exchange (IKE)

Abb. 66: Can Bus Message Frame [Davis, L., 1998-2007. WS]

Diese Normen sind in allen Ländern gültig. Beim CAN BUS werden allerdings zusätzlich zu der Norm herstellerspezifische Implementierungen vorgenommen, was zu Irritationen führt. Jedoch kann bei der Implementierung des CAN BUS von einer minimalen Funktionalität ausgegangen werden. Bei den Normen für Netzwerkprotokolle würde die Nicht-Einhaltung einer Norm dazu führen, dass Produkte in Netzwerken nicht eingesetzt werden können. In der Informatik ist dieser Trend der Zentralisierung von Normen noch stärker vorhanden als es sich in der W3C verkörpert (Auszug in Abb. 67), die im Wesentlichen die technischen Strukturen des Internets vorgeben.

- Accessibility
- CSS
- CSS Validator
- Databinding
- Device Independence
- DOM
- HTML
- HTML Tidy
- HTML Validator
- HTTP
- Internationalization
- MathML
- Mobile Web Initiative (W3C-MWI)
- PNG
- Privacy and P3P
- RDF
- SMIL
- SOAP/XMLP
- SVG
- URI/URL
- Validators
- Web Services
- XForms
- XHTML
- XLink
- XML
- XML Base
- XML Encryption
- XML Key Management
- XML Processing
- XML Query
- XML Schema
- XML Signature
- XPath
- XPointer
- XSL and XSLT

Abb. 67: Wichtige Normen der W3C

Zwar gibt es ähnliche Ansätze in der Bauindustrie, z.B. die SIC-Codes, diese haben sich aber in der Regel in den einzelnen Ländern nicht durchsetzen können, sind in ihrer Tiefe unzureichend und werden nur zur Referenzierung eingesetzt (vgl. Watson, W., 1998. S. 3; Bakkmoen, K., 1996. S. I). Interessant ist in diesem Zusammenhang der Vorschlag von Bakkmoen zur Gliederung der Projektdokumentation, die durchaus eine Anlehnung an die HOAI ermöglicht (Abb. 68). Die SIO hat laut Watson [Watson, W., 1998. S. 10] drei Ansätze zur Klassifikation von Projektinformationen vorgeschlagen:

- Klassifikation nach Funktion (Leistungskriterien und Spezifikationen),
- Klassifikation nach Objekt (System, Bauteil, Teil, Komponente, Produkt, Material und Anteil),
- Klassifikation nach Ort (Grundstück, Gebäude, Teil einer Struktur). Diese Einteilung kann wie folgt noch feiner gegliedert werden.

HOAI Leistungsphase	Klassifikation nach Bakkmoen [Watson, 1998. S. 5]
1. Grundlagenermittlung	▪ An idea combined with an outline or broad financial analysis ▪ Notional illustration and diagrams
2. Vorplanung	▪ Outline (perception) drawings and criteria ▪ Preliminary (facility function) descriptions
3. Entwurfsplanung	▪ Material and systems integration ▪ Unique (project specific) plans, elevations, and detail drawings
4. Genehmigungsplanung	
5. Ausführungsplanung	▪ Developed specifications and schedules ▪ Shop (fabrication) drawings
6. Vorbereitung der Vergabe	Bidding and contractual documents
7. Mitwirkung bei der Vergabe	
8. Objektüberwachung	Record documents of actual constructed facility
9. Objektbetreuung und Dokumentation	▪ As-constructed facility drawings ▪ Operational documentation ▪ Continually or periodically altered facility records ▪ Demolition criteria and materials for recycling

Abb. 68: Vergleich HOAI mit Projektdokumentation nach Watson [Watson, 1998. S. 5].

In Deutschland haben sich umfassende Verordnungen, Verdingungen, Gesetze und Normen, die jeweils wechselseitig und im Zusammenspiel zu betrachten sind, durchgesetzt:

- BGB
- HOAI
- VOB/A
- VOB/B
- VOB/C
- DIN276
- DIN18
- DIN1356-1
- DIN69901
- DIN 1356-2

Eine erfolgreiche Projektabwicklung kann nur durch die integrierte Betrachtung dieser Verordnungen und Gesetzte erfolgen. Dadurch, dass Facetten dieses komplexen Bestandes rechtliche Fragestellungen sind, sind diese auch Änderungen ausgesetzt, die u.U. zu schwierigen Fragestellungen führen (vgl. Langen und Schiffers, 2005. S. 89, Rdn. 232).

Da eine Modellierung der rechtlichen Prozesse aufgrund ihrer Komplexität entfällt, konzentriert sich eine praxistaugliche Betrachtung der Anwendung des Objektmodells auf Aspekte der Verordnungen, welche für die Strukturierung und Gliederung des Bauobjekts relevant sind.

Die Strukturaspekte der HOAI wurden entwickelt, um eine phasenbezogene Entlohnung der Leistungen während eines Bauobjektes zu ermöglichen. Allerdings können Elemente dieser Struktur auch zur Gliederung des Informationsbestands eines Projektes eingesetzt werden. Tatsächlich ist eine solche Gliederung eine Voraussetzung für eine Überschneidung eines Objektmodells mit Berechnungen nach der HOAI falls diese Funktionalität erforderlich wäre. Es sollte bemerkt werden, dass der Trend zur Integration von Tätigkeiten in den Frühphasen der HOAI und die Hervorhebung der Betrachtung des gesamten Gebäudelebenszyklus zu einer Skepsis gegenüber der Nützlichkeit der HOAI geführt hat [Henckels, D., 2004. S. 12].

Bei der Identifizierung der Strukturaspekte der HOAI ist darauf zu achten, ob Elemente statischer Natur sind oder verschiedenen Bedingungen unterliegen. Letztere sind ein Hinweis darauf, dass es sich eher um ein dynamisches Netzwerk handelt, was evtl. nicht als ein gerichteter Graph modelliert werden kann, sondern eher als ein gerichteter azyklischer Graph [Wagner, C., 2003. S. 46]. Dieser Aspekt bestimmt die Modellierung, da ein azyklischer Graph mit einer klassischen, relationalen Struktur nicht zu modellieren ist.

Um einen Konflikt bei der Modellierung zu vermeiden, sollten die statischen Elemente der HOAI isoliert werden. Diese können dann als Ordnungsstruktur etabliert werden. So beschreiben die Leistungsbilder Bereiche der Planungsleistungen von Architekten (Abb. 69). Diese Bilder stellen ein Strukturelement dar und können somit unproblematisch als Strukturierungselemente genutzt werden.

Teil	§§	Leistungsbild
II	10 ff	Objektplanung Gebäude, Freianlagen und Raumbildende Ausbauten
VII	51 ff	Objektplanung Ingenieurbauwerke und Verkehrsanlagen
VIII	62 ff	Tragwerksplanung
IX	68 ff	Technische Ausrüstung
X	77 ff	Thermische Bauphysik
XI	80 ff	Schallschutz und Raumakustik
XII	91 ff	Bodenmechanik, Erd- und Grundbau
XIII	96 ff	Vermessungstechnische Leistungen

Abb. 69: Häufig vorkommende Leistungsbilder der HOAI [Langen und Schiffers, 2005. S. 110]

So können auch die Leistungsphasen der HOAI als statische Elemente betrachtet werden (Abb. 70).

HOAI Leistungsphase	Beschreibung
1. Grundlagenermittlung	Ermitteln der Voraussetzungen zur Lösung der Bauaufgabe durch die Planung
2. Vorplanung	(Projekt- und Planungsvorbereitung) Erarbeiten der wesentlichen Teile einer Lösung der Planungsaufgabe
3. Entwurfsplanung	(System- und Integrationsplanung) Erarbeiten der endgültigen Lösung der Planungsaufgabe
4. Genehmigungsplanung	Erarbeiten und Einreichen der Vorlagen für die erforderlichen Genehmigungen und Zustimmungen
5. Ausführungsplanung	Erarbeiten und Darstellen der ausführungsreifen Planungslösungen
6. Vorbereitung der Vergabe	Ermitteln der Mengen und Aufstellen von Leistungsverzeichnissen
7. Mitwirkung bei der Vergabe	Ermitteln der Kosten und Mitwirkung bei der Auftragsvergabe
8. Objektüberwachung	(Bauüberwachung) Überwachung der Ausführung des Objekts
9. Objektbetreuung und Dokumentation	Überwachen der Beseitigung von Mängeln und Dokumentation der Gesamtergebnisse

Abb. 70: Leistungsphasen der HOAI

Ein Beispiel der Entzerrung der umfassenden Verordnungen ist die sechste Grundleistung der Leistungsphase 3, Grundleistungen. Diese Leistungsbeschreibung sieht eine „Kostenberechnung nach DIN 276 oder nach dem wohnungsrechtlichen Berechnungsrecht“ vor. Sie dient zur „Ermittlung der in diesem Planungsstadium absehbaren, angenäherten Gesamtkosten und kann Voraussetzung für die Entscheidung des Bauherrn über die Durchführung der Baumaßnahme wie geplant oder aber in geänderter Form sowie Grundlage einer erforderlichen Finanzierung sein;“. Die Gliederung der DIN 276 wiederum stellt eine Struktur zur Identifikation der Kostenelemente dar, die dann entsprechend den Aufwandswerten des Ermittlers bestimmt werden können (Abb. 71). Diese Struktur kann als eigenständige Gliederung in dem Objektmodell angelegt werden (Abb. 72).

Zusammenhängende Kosten werden in Kostengruppen gegliedert. In der 1. Ebene der Kostengliederung werden die Gesamtkosten in folgende sieben Kostengruppen gegliedert: 100 Grundstück 200 Herrichten und Erschließen 300 Bauwerk / Baukonstruktionen 400 Bauwerk / Technische Anlagen 500 Außenanlagen 600 Ausstattung und Kunstwerke 700 Baunebenkosten
Für die genauere Kostenermittlung werden Kosten in die 2. Ebene, noch genauer in die 3. Ebene aufgeschlüsselt. Auszug 300er Kosten:
360 Dächer (2.Ebene) 361 Dachkonstruktionen (3.Ebene) 362 Dachfenster, Dachöffnungen 363 Dachbeläge 364 Dachbekleidungen 369 Dächer, sonstiges

Abb. 71: Auszug aus der Gliederung der DIN 276

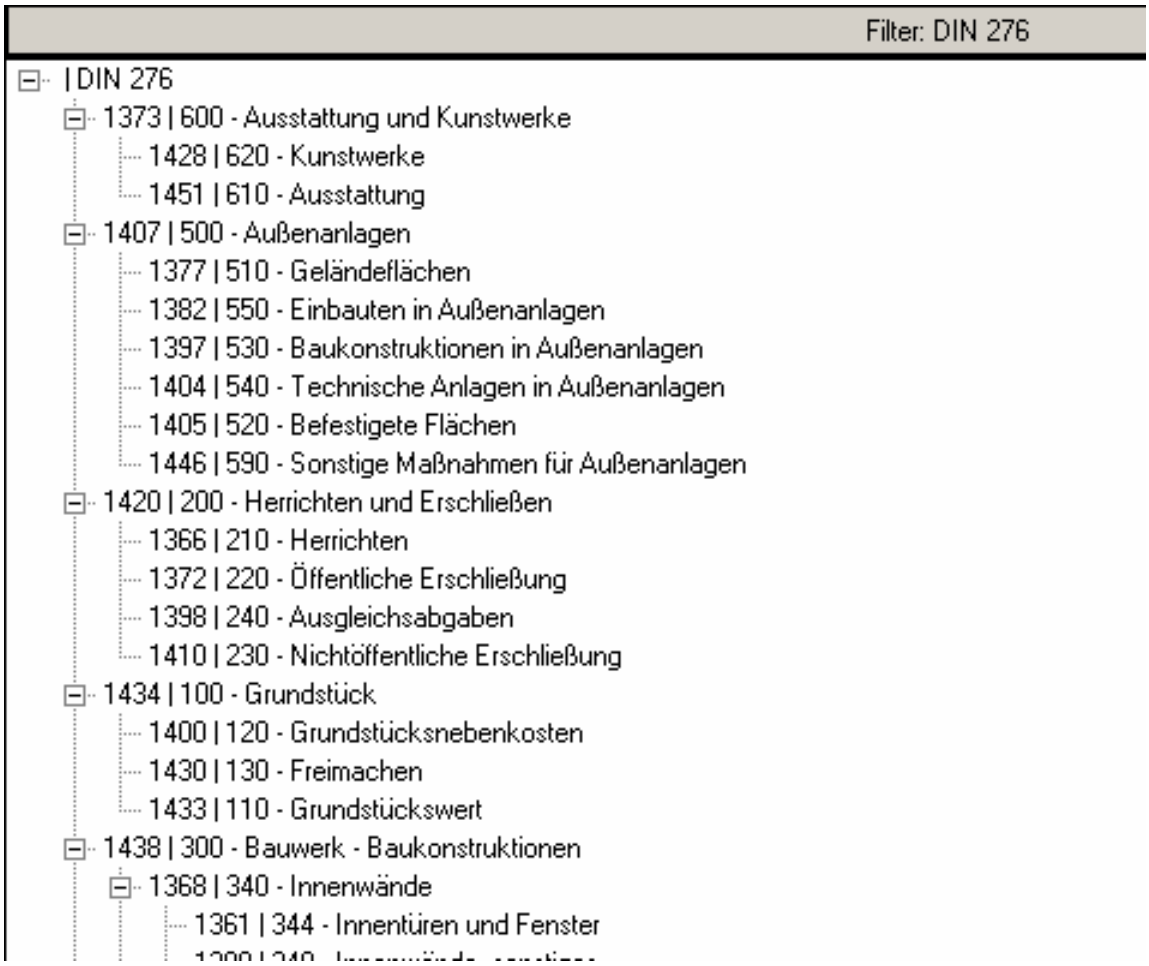


Abb. 72: Gliederung der DIN 276 als Gliederungsstruktur im Objektmodell.

Die HOAI sieht eine stufenweise Festlegung der Qualitäten eines Bauvorhabens analog zu den Planungsphasen vor [Langen und Schiffers, 2005. S. 47]. In Abbildung 73 werden Instrumente der Qualitätsfestlegung dargestellt und aufgeführt, in welcher Form diese Instrumente womöglich in das Objektmodell einfließen können.

Die Spalten stellen auch den gedanklichen Übergang in das Objektmodell dar.

Instrument	Norm	Art
Raum- und Funktionsprogramm	-	Gliederung
▪ Raumbuch ▪ Anforderungsraumbuch ▪ Planungsraumbuch ▪ Bestandsraumbuch	-	Gliederung
Baubeschreibung	DIN 276	▪ Dokumentation ▪ 2D CAD ▪ 3D CAD
Leistungsbeschreibung (inkl. Leistungsverzeichnis)	▪ §633 Abs. 2 BGB ▪ §13 Nr. 1 VOB/B	▪ Gliederung ▪ Teilleistung ▪ Ordnungszahl
Ausführungsplanung	DIN1356-1	▪ 2D CAD ▪ Technische Beschreibungen ▪ Schalpläne ▪ Rohbauzeichnungen ▪ Bewehrungszeichnungen ▪ Fertigteilzeichnungen ▪ Verlegezeichnungen
Bemusterung	▪ DIN18338-DIN18339 ▪ DIN18330-DIN18331 ▪ DIN18379-DIN18380 ▪ Weitere DIN 18…	Gliederungspunkte

Abb. 73: Tabellarische Zusammenfassung der Elemente der Qualitätsfestlegung entsprechend der Auflistung nach Langen und Schiffers [Langen und Schiffers, 2005. S. 47-49], Gegenüberstellung der Systemgliederungen.

Eine Schwäche eines gegliederten Objektmodells ist die Annahme, dass Gliederungspunkte dediziert zugeordnet werden können. Diese Voraussetzung kann in den Frühphasen eines Bauvorhabens entweder nicht erfüllt oder nur in groben Zügen determiniert werden. Das liegt daran, dass Gliederungsstrukturen von vornherein eindeutige Knotenpunkte haben müssen, damit eine Genauigkeit bei dem Aufbau einer Gliederung vorhanden sein kann, die in der Praxis noch gar nicht möglich ist. Diese Genauigkeit ergibt sich nur bei genormten Strukturen wie DIN 276, deren Zuordnung zu Bauteilen u.U. zu Kompatibilitätsproblemen führen kann.

Das Objektmodell bietet jedoch die Möglichkeit höhere Knotenpunkte auszuwählen, die weniger spezifisch sind, und daher bei einer Qualitätsentwicklung samt Objekt-Klonung auf einen untergeordneten und daher aussagekräftigeren Wert setzen. XQuery kann auch hier hilfreich sein, da in Kombination mit XPath spätere und genauere Gliederungen über deren Wurzelknoten abgefragt werden können.

Der Auflistung der Qualitätsfestlegungen in Abbildung 73 ist zu entnehmen, dass eine Modellierung der Qualitätsentwicklung eines Bauvorhabens bei zunehmender Planungsphase besser gelingt. In der Phase 5, Ausführungsplanung, wird typischerweise eine Bemusterung anhand von DIN 18338 – 18380 und weiteren Normen vorgenommen. In dieser Phase kann eine Qualitätsangabe im Objektmodell problemlos erfolgen.

Dieser Zustand tritt auch ein, wenn einem Bauvertrag die Bestimmungen der VOB/B zugrunde liegen, weil dann automatisch die Bestimmungen der ATV-DIN 18299 ff. Vertragsbestandteil sind (vgl. Langen und Schiffers, 2005. S. 91, Rdn. 242). Hier kann aufgrund der relativen Genauigkeit der DIN 18299 ff. Normen eine hohe Genauigkeit bei der Zuordnung erreicht werden, was die Akzeptanz der Auswertungen erhöht.

2 Strukturierte Erfassung und Verarbeitung von Planungsdaten

2.1 Grundsätzliches

Als Start für eine strukturierte Erfassung von Planungsdaten wird die Leistungsphase 3 – Entwurfsplanung – herangezogen. Ziel ist es, eine Struktur vorzugeben, welche die ganzheitliche Erfassung von Planungs- und Ausführungsdaten in einem Projektmodell ermöglicht. Dabei wird gebäudegeometrieorientiert und somit projektorientiert geplant.

Grundlagen sind folgende Unterlagen der Leistungsphase 3 – Entwurfsplanung:

a) die zeichnerischen Darstellungen,
b) die gebäudeorientierte Baubeschreibung,
c) die Berechnungen und Ermittlungen.

2.1.1 Zeichnerische Darstellungen

Die Entwurfsplanung ist die planerische Fortsetzung des nach Abschluss der Leistungsphase 2 verabschiedeten Planungskonzeptes. Die Aufgabe der Leistungsphase 3 besteht aus der Konkretisierung aller bisherigen Planungsinhalte zu einer öffentlich-rechtlichen Genehmigungsreife.

Diese Genehmigungsreife wird durch die Integration der Planungsbeiträge der Fachingenieure und -planer erzielt. Das Ergebnis aus allen Fachbeiträgen ist der vom Objektplaner dargestellte Gesamtentwurf der Bauaufgabe. Die zum Gesamtentwurf gehörigen Zeichnungen geben dabei einen Überblick über die Bauaufgabe und werden bei Hochbauten in der Regel in Maßstäben von 1:100 bis 1:200 angelegt.[13] Diese Zeichnungen der Entwurfsplanung müssen alle zur Prüfung auf öffentlich-rechtliche Zulässigkeit notwendigen Informationen enthalten.

2.1.2 Gebäudeorientierte Baubeschreibung

Die zeichnerischen Darstellungen werden durch eine Objektbeschreibung ergänzt. Die Anfertigung der Objektbeschreibung sollte in Form einer gebäudeorientierten Baubeschreibung aufgebaut sein und alle kostenrelevanten Sachverhalte enthalten. Dazu gehören

- die zum Grundstück und Baugrund notwendigen Informationen,
- die Angaben zu den gewählten Baukonstruktionen,
- die einzusetzenden Materialien,
- die Daten zur technischen Gebäudeausrüstung,
- die geplanten Leistungen des raumbildenden Ausbaus sowie
- die Erfassung sämtlicher Baunebenleistungen.

Die gebäudeorientierte Betrachtung bedeutet, ein Gebäude in seine einzelnen Bauelemente zu unterteilen. Diese Betrachtungsweise ist der Schlüssel für die ganzheitliche Erfassung und ermöglicht

- das Bauwerk in seiner Gesamtheit, in allen Belangen zu prüfen und zu überdenken,
- mit hoher Kostensicherheit die Bauelemente zu bewerten,

[13] Die Maßstabsanforderungen werden durch die Baubehörden auch individuell zum Bauvorhaben festgelegt.

- bewertungssichere Mengen zu ermitteln,
- projektorientierte Bauabläufe zu erstellen,
- alternative Ausführungsvarianten zu untersuchen und
- die Fortsetzung der Planungsleistungen von definierten Inhalten.

2.1.3 Berechnungen und Ermittlungen

Neben der zeichnerischen Darstellung des Gesamtentwurfes und der gebäudeorientierten Baubeschreibung sind Berechnungen und Ermittlungen für eine öffentlich rechtliche Genehmigungsfähigkeit erforderlich. Dazu dienen unter anderem die Ermittlungen von

- Flächen und Volumina nach DIN 277,
- Vorbemessungen der Tragwerkplanungen,
- Ermittlungen zur technischen Gebäudeausrüstung,
- Angaben zum Brandschutz,
- Ermittlungen zum Baugrund.

Bevor die Unterlagen zum Bauantragsverfahren eingereicht werden, ist vom Objektplaner eine Kostenberechnung gemäß DIN 276 – Kosten im Hochbau[14] – durchzuführen. Diese Kostenberechnung stellt die Bewertung des zeichnerischen Gesamtentwurfes und der gebäudeorientierten Baubeschreibung dar. Das Ergebnis sind die angenäherten Gesamtkosten des Bauwerkes, wobei dieses nicht isoliert ermittelt und betrachtet werden darf. Sondern erst der Abgleich mit der zum Abschluss der in Leistungsphase 2 aufgestellten Kostenschätzung die wesentliche Aussage zulässt, ob der Gesamtentwurf in das auftraggeberseitig festgelegte Kostenbudget passt.
Somit sind drei zentrale Ziele in der Leistungsphase 3 zu verfolgen:

- Genehmigungsreife, Gesamtlösung durch Integration aller Fachbeteiligten,
- Freigabe (seitens Auftraggebers) eines funktionalen Gesamtentwurfs,
- Freigabe (seitens Auftraggebers) der wirtschaftlichen Bewertung des Gesamtentwurfes und der gebäudeorientierten Baubeschreibung.

[14] DIN 276 – Kosten im Hochbau.

Diese Unterlagen dienen dazu, die ausführungsorientierte Betrachtungsweise eines Projektes aufzunehmen und SOLL-Ausführungsdaten (wie Pläne), SOLL-Ausführungsvorgaben (wie Terminpläne), SOLL-Vergütungsregelungen (wie Verträge) zu erstellen und IST-Daten (wie Mängel, Leistungs- und Abrechnungsstände usw.) festzuhalten.

2.2 Gegenwärtige Vorgehensweise bei der manuellen Erfassung von Planungs- und Ausführungsdaten

Die gegenwärtigen, zweidimensionalen, zeichnerischen Darstellungen im Sinne von Planungsunterlagen der Leistungsphase 3 (siehe Abb. 74) ermöglichen bereits,

- die quantitativen Ansätze zu berechnen,
- eine Funktionsüberprüfung des Gebäudes vornehmen zu können und
- die gebäudeorientierte Baubeschreibung vollständig aufzustellen.

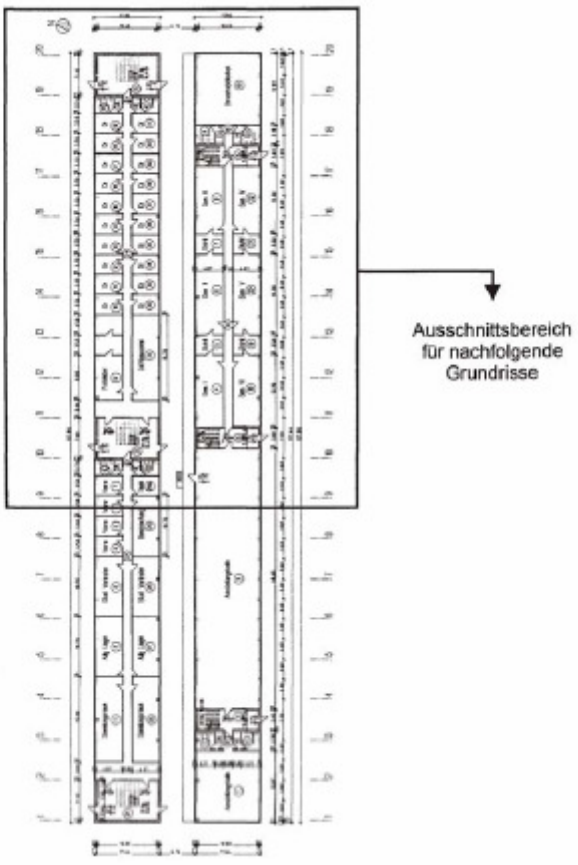


Abb. 74: Zweidimensionaler Grundriss - Erdgeschoss Leistungsphase 3 - Objektplanung [Kapellmann und Schiffers, 2000. S. 719]

Die Planungsunterlagen bei der manuellen Erfassung bestehen aus zweidimensionalen Grundrissen und Schnittdarstellungen (Abb. 75).

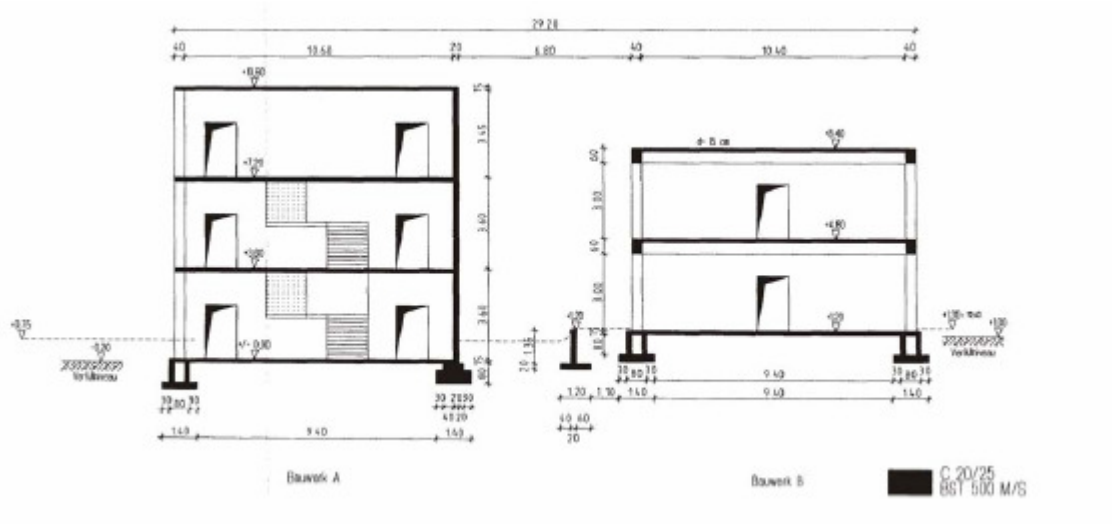


Abb. 75: Zweidimensionaler Schnitt - Leistungsphase 3 - Tragwerksplanung
[Kapellmann und Schiffers, 2000. S.576]

Zur Erfassung von Planungs- und Ausführungsdaten werden die Zeichnungen nutzungsorientiert aufgebaut. Sie werden herangezogen, um manuell die Grundsatzqualitäten des Ausbaues[15] zuzuweisen und deren Mengen zu ermitteln (Abb. 76). Dabei stehen die Qualitätszuweisungen im direkten Dialog zur Baubeschreibung. Zur Kennzeichnung der Qualitäten werden in die Grundrisspläne Schraffuren oder farbig angelegte Flächen der jeweiligen Oberflächenqualitäten eingetragen.

Diese manuelle Art der Qualitätskennzeichnung verschafft dem Anwender eine Übersicht und verhindert gezeichnete und farblich gekennzeichnete Flächen als zu erstellende Leistungen zu erfassen.

15 Als Ausbau werden alle nicht konstruktiven und nicht haustechnischen Elemente innerhalb des Gebäudes bezeichnet.

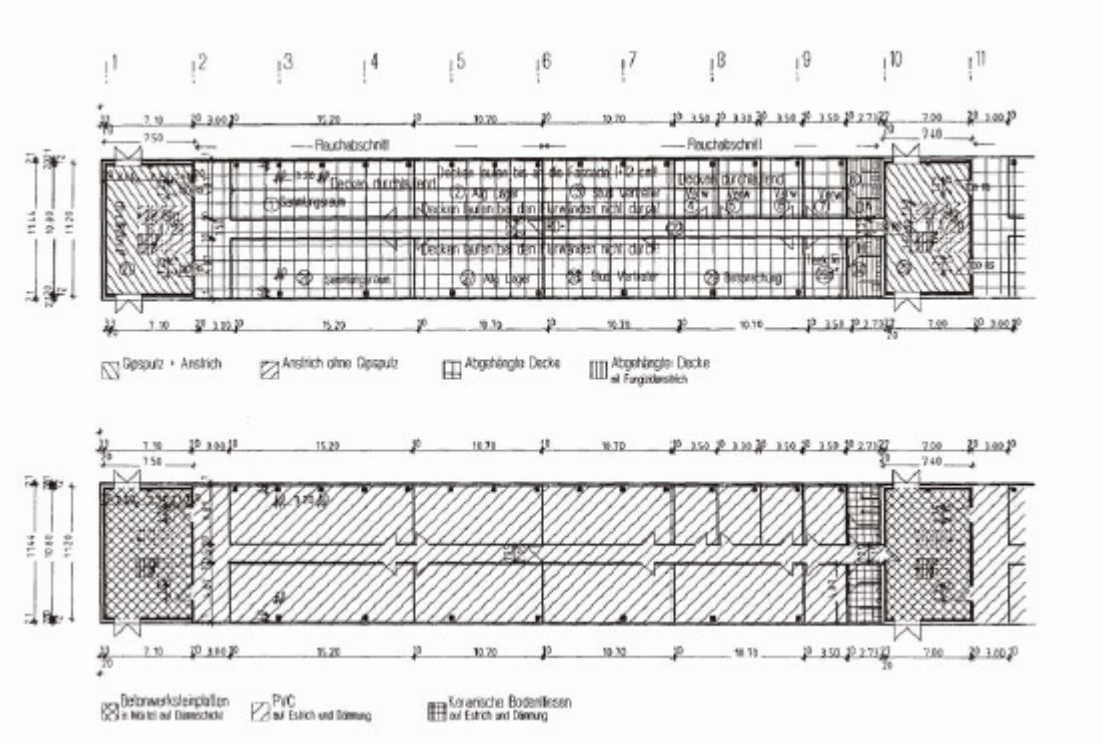


Abb. 76: Mengenermittlungspläne für Decken und Bodenqualitäten [Kapellmann und Schiffers, 2000. S. 774]

Zur Kennzeichnung der Wandqualitäten wird ebenfalls der Grundriss herangezogen. Die laufenden Meter der verschiedenen Wandqualitäten werden ermittelt (Abb. 77). Die zugehörigen Höhen sind aus den Schnittplänen abzugreifen.

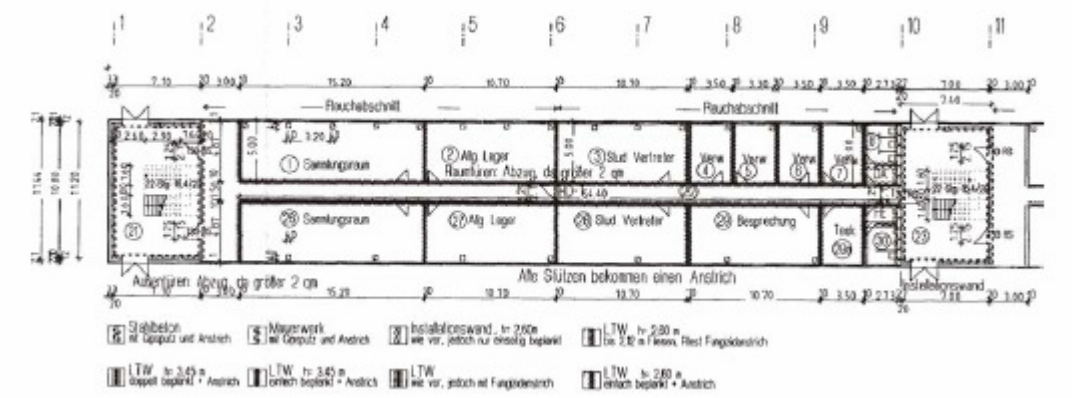


Abb. 77: Mengenermittlungsplan für Wandqualitäten.

Bei der Kennzeichnung der haustechnischen Leistungsbereiche sind die Grundriss- und Schnittdarstellungen der Leistungsphase 3 nicht ausreichend. Zwar sind beispielhaft beim Leistungsbereich der Sanitäranlagen (Abb. 78) die nutzungsorientier-

ten Inhalte wie Einrichtungsgegenstände[16] darstellbar und die zugehörige Anzahl ermittelbar, jedoch zur Berechnung der Ver- und Entsorgungsleitungen sind gesonderte Strangschemata erforderlich. Diese Zusammenhänge sind in den zweidimensionalen Grundriss- und Schnittplänen des Objektplaners nicht übersichtlich darstellbar.

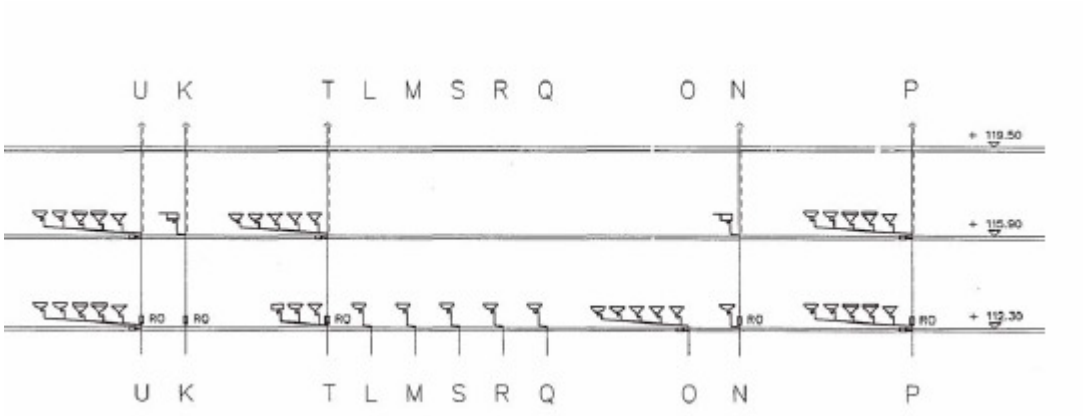


Abb. 78: Strangschemata für die Sanitärinstallationen.

Mit der manuellen Qualitätskennzeichung und der Erfassung der zugehörigen Mengen ist die Verknüpfung geschaffen, zwischen einer gebäudegeometrieorientierten Darstellung und dem späteren, ausführungsorientierten Baumanagement[17].

Die gewonnenen Mengenwerte und die Aufstellung der Qualitäten werden im Rahmen der Erstellung der produktionsorientierten Terminplanung erneut herangezogen.

Die zweidimensionalen Darstellungen und die fehleranfällige, manuelle Erfassung bei fehlenden Strukturen führen schließlich zu einer digitalen, ganzheitlichen Erfassung von Ausführungsdaten innerhalb eines Systems mit einer festen Struktur.

[16] Mit Einrichtungsgegenständen sind WC-Töpfe, Urinal- und Handwaschbecken gemeint

[17] Mit dem Begriff Baumanagement sind die Aufgaben zur Abrechnung, Vergabe, Terminplanung etc. gemeint

2.3 Digitale Erfassung von Ausführungsdaten

Gegenüber der manuellen Erfassung von Ausführungsdaten bietet die digitale Erfassung der Ausführungsdaten den Vorteil einer zentralen Datenhaltung,
d.h., dass hier eine ganzheitliche Erfassung der Daten innerhalb eines Systems vor liegt und die Gefahr einer redundanten Erfassung und/oder Fehlerfassung minimiert wird.

Die Erfassung der Daten erfolgt mittels einer dreidimensionalen (3D) Zeichnungstechnologie. Diese Technologie ermöglicht, die traditionellen Grundriss- und Schnittdarstellungen dem Anwender zur Verfügung zu stellen, aber auch die für das ganzheitliche Verständnis erforderlichen dreidimensionalen Visualisierungen.

Die Zeichnungen werden elementorientiert aufgebaut. Somit wird dem Anwender ermöglicht, aus den 2D oder 3D-Zeichnungen heraus eine Auswahl einzelner Elemente vorzunehmen, um diese planerisch und zeichnerisch fortzuentwickeln, oder mit neuen Daten zu ergänzen (Abb. 79).

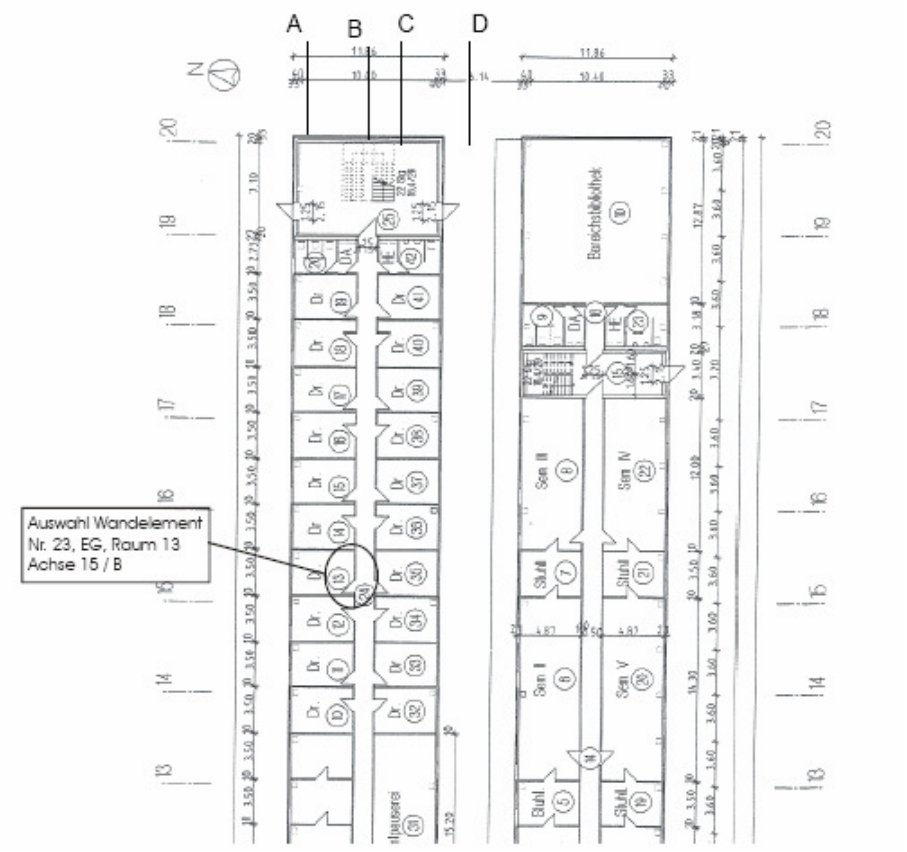


Abb. 79: Ausschnitt Entwurfsplanung - Leistungsphase 3 –
Auswahl Wandelement [Kapellmann und Schiffers, 2000. S. 719]

Das Auswahlobjekt besitzt bereits einige der Leistungsphase 3 entsprechenden Eigenschaften. Dazu gehören die geometrischen Eigenschaften mit systemintern ermittelten Mengenwerten und die Zuordnung einer Kostengruppe der DIN 276. Isoliert von der Gesamtplanung kann jedes Element dreidimensional visualisiert werden.

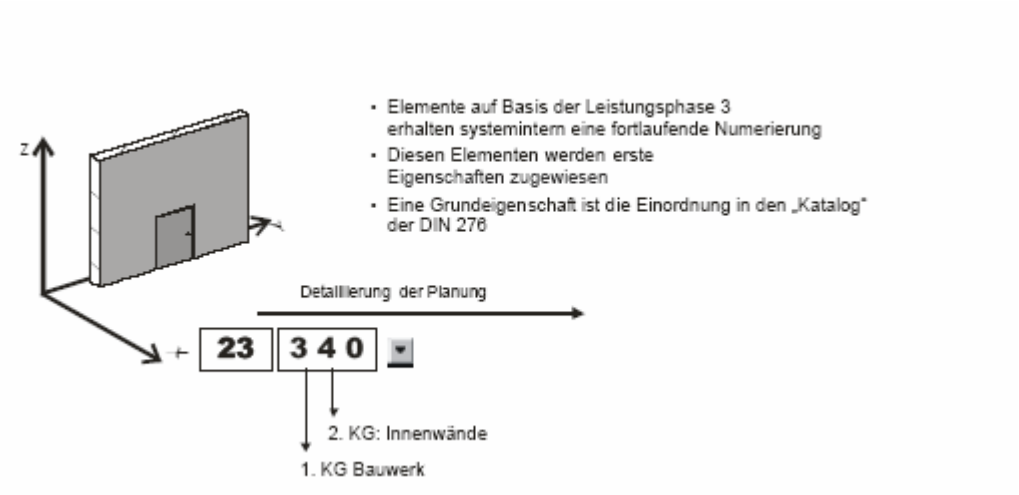


Abb. 80: Vorhandene 3D Elemente werden weiter kategorisiert.

Ungeachtet einer zeichnerischen Fortentwicklung kann das Element ausführungsrelevante Informationen enthalten. Dazu wird das Element in der dritten Ebene der DIN 276 gegliedert. Der Entwicklungsprozess des Elementes (Abb. 80) wird systemintern für die spätere Nachvollziehbarkeit festgehalten.

Das geschieht über den Mechanismus des Vererbens. Die Versionierung der Objekteigenschaften ist auch ohne das Klonen möglich. Jedoch wird dabei meist von einem statischen Klassenmodel ausgegangen. Durch den Entwicklungsprozess (Abb. 81 - 88) können nämlich mehrere Objekte unterhalb eines Knotenpunktes entstehen bzw. kann sich die Klasse eines Objektes durch die Vererbung verändern. Im ersten Fall würde bei einer klassischen Versionierung das Problem auftreten, dass die versionierten Eigenschaften in mehreren Objekten vorkommen. Im zweiten Fall würden neue Eigenschaften hinzukommen. Die Vererbung lässt grundsätzlich nicht zu, dass Eigenschaften aus Klassen entfernt werden. Aus diesem Grund werden Eigenschaften bei der Vererbung weiter geführt, auch wenn sie nicht mehr verwendet werden. In diesem Fall und bei der Erweiterung einer Klasse durch die Vererbung, müsste sich eine klassische Versionierung automatisch verändern.

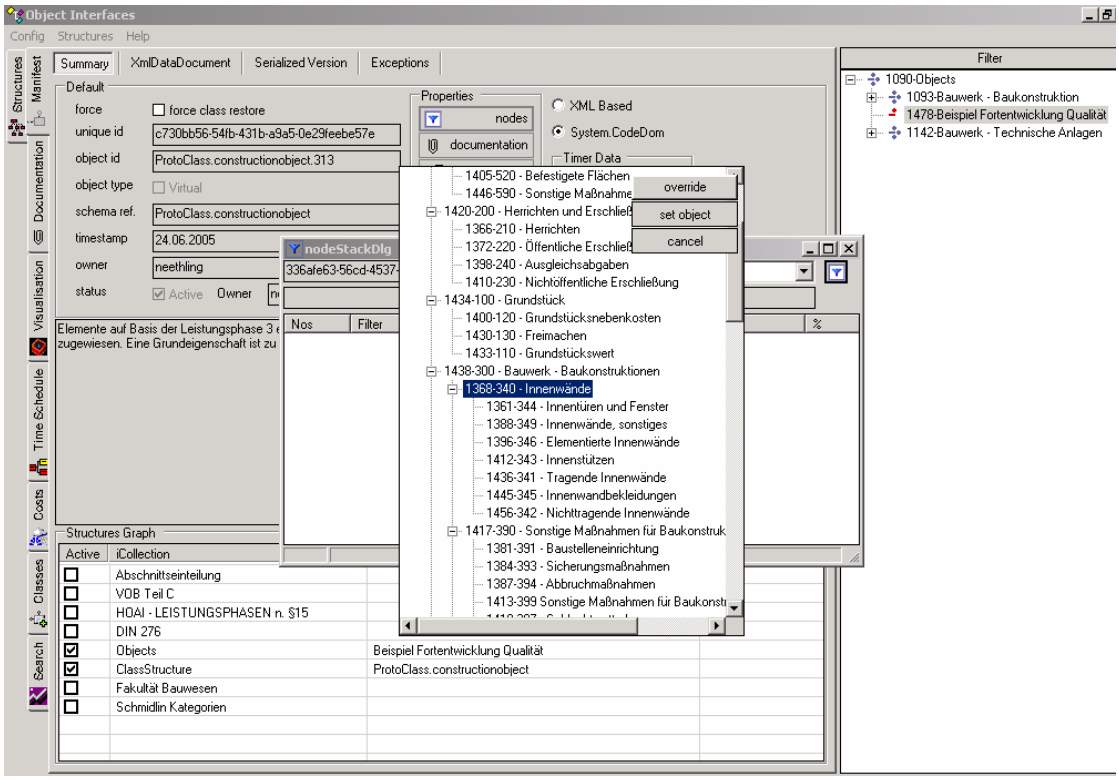


Abb. 81: Darstellung eines Wandelements im System. Detaillierung der Planung in DIN 276 bis zu 2.KG., Analog Abbild 93: Anlegen des Knotenpunks 340 im Filter DIN 276.

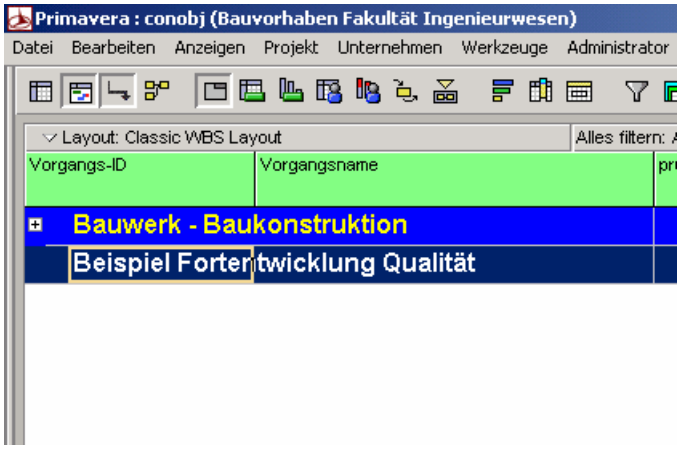


Abb. 82: Darstellung eines Wandelements im System. Detaillierung der Planung in DIN 276 bis zu 2.KG., Analog Abbild 93: Der verteilte Mechanismus legt automatisch einen Gliederungspunkt für das Innenwand–Element in der Terminplanung an.

Abb. 83: Darstellung eines Wandelements im System. Detaillierung der Planung in DIN 276 bis zu 2.KG., Analog Abbild 93: Der verteilte Mechanismus legt automatisch einen Gliederungspunkt für das Innenwand-Element (der Kosten) in der Kalkulation an.

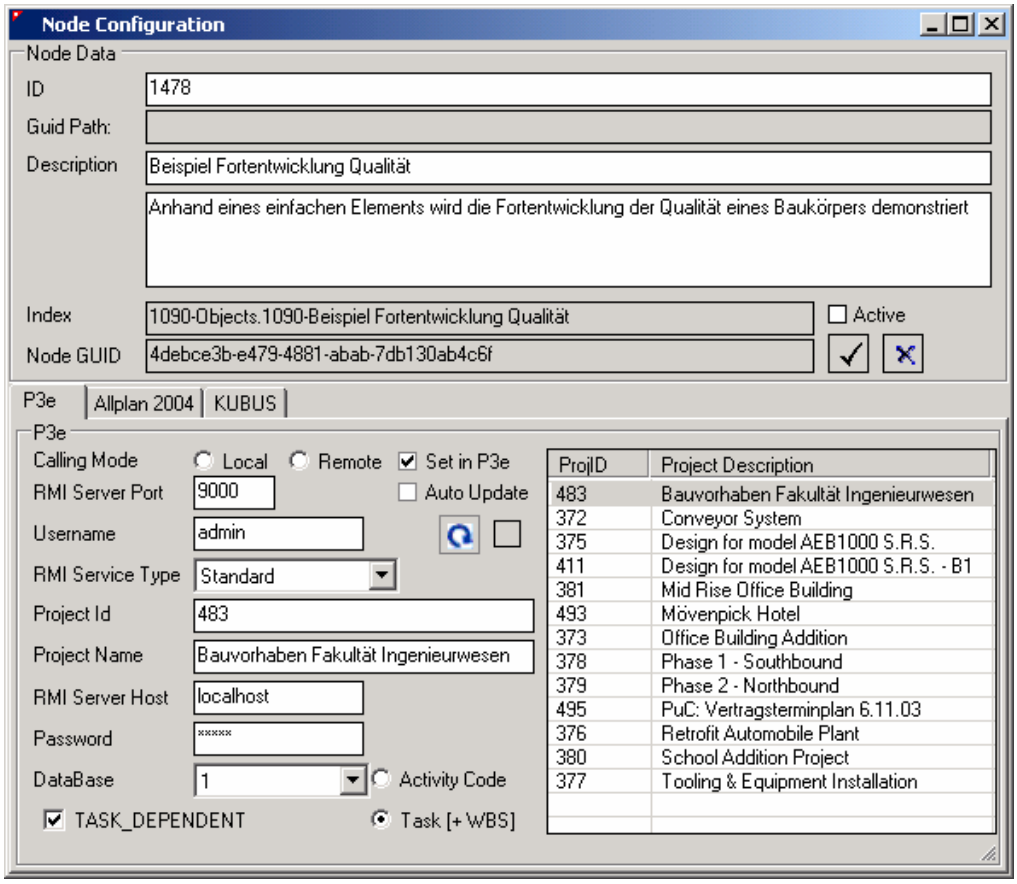


Abb. 84: Darstellung eines Wandelements im System. Detaillierung der Planung in DIN 276 bis zu 2.KG., Analog Abbild 93: Anlegen des Objekts Innenwand-Element im System.

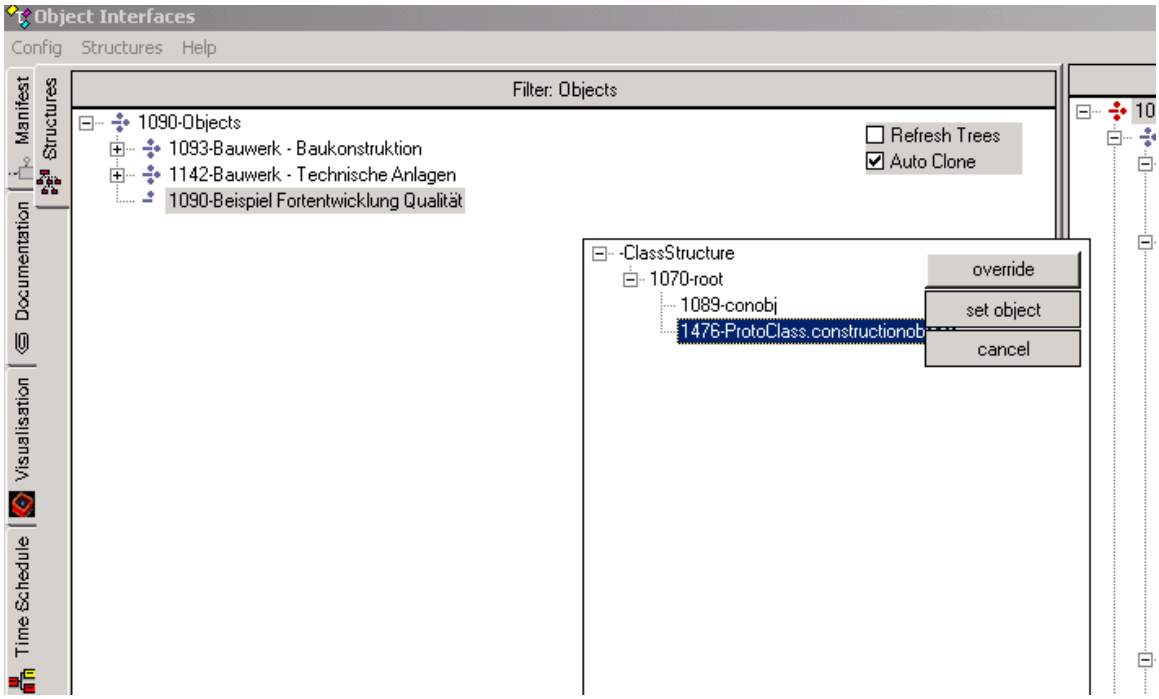


Abb. 85: Darstellung eines Wandelements im System. Detaillierung der Planung in DIN 276 bis zu 2.KG., Analog Abbild 93: Anlegen des Innenwand-Elements, Auswahl der dem zu Grunde liegenden Klasse.

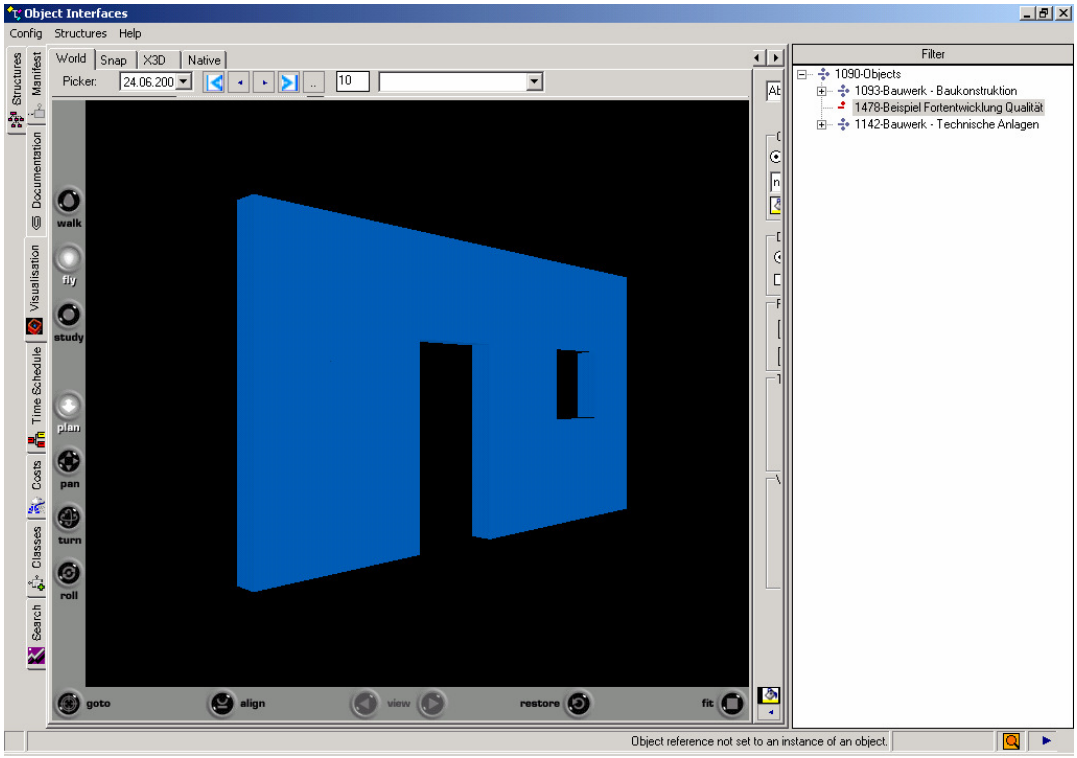


Abb. 86: Darstellung eines Wandelements im System. Detaillierung der Planung in DIN 276 bis zu 2.KG., Analog Abbild 93: Anlegen des Innenwand-Elements, Zuordnung der Geometrie.

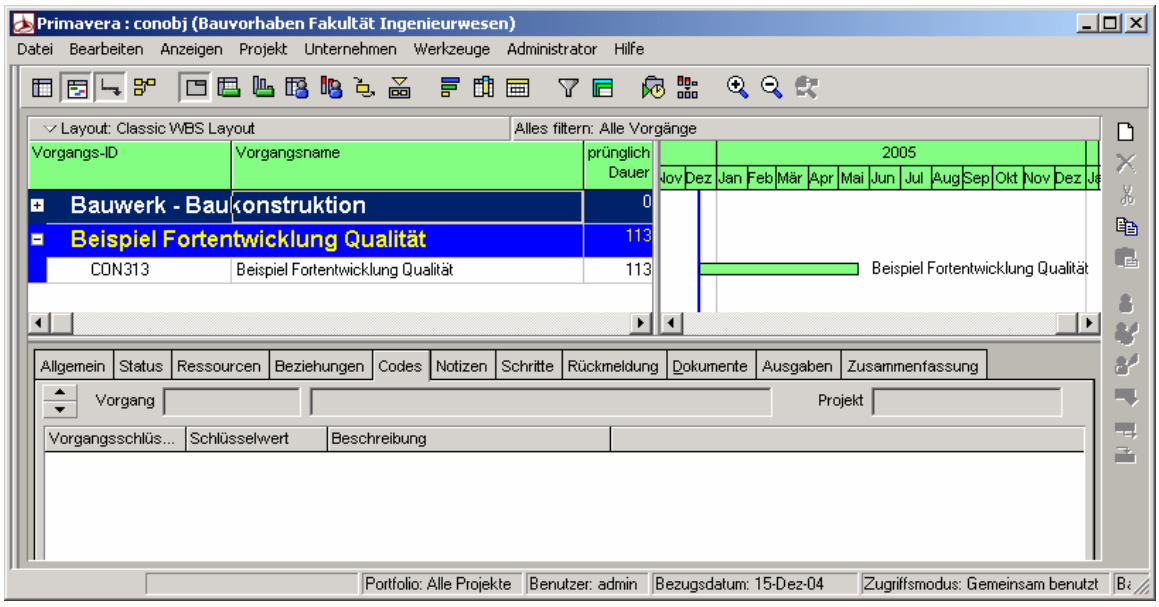


Abb. 87: Darstellung eines Wandelements im System. Detaillierung der Planung in DIN 276 bis zu 2.KG., Analog Abbild 93: Anlegen des Innenwand-Elements, verteilter Mechanismus legt automatisch einen entsprechenden Vorgang in der Terminplanung an.

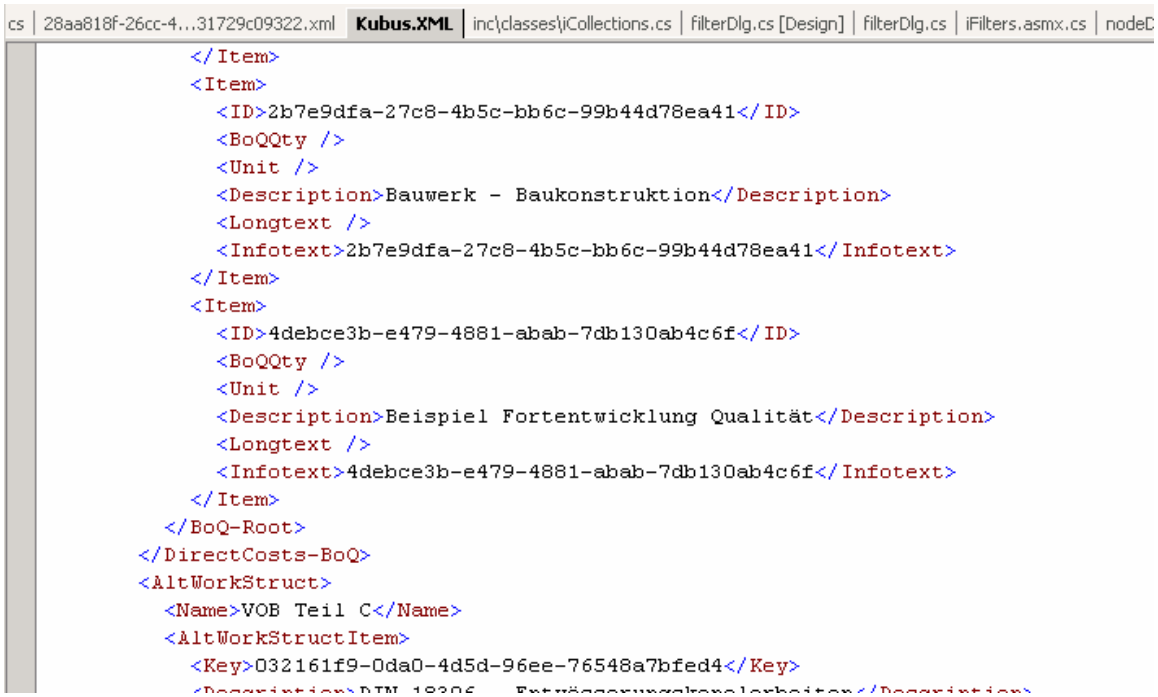


Abb. 88: Darstellung eines Wandelements im System. Detaillierung der Planung in DIN 276 bis zu 2.KG., Analog Abbild 93: Anlegen des Innenwand-Elements, verteilter Mechanismus legt automatisch einen entsprechenden Vorgang in der Kalkulation an.

Die Gliederung erfolgt über eine Kombination der Filter (vom Typ: *iCollections*) Abschnittseinteilung und DIN 276.

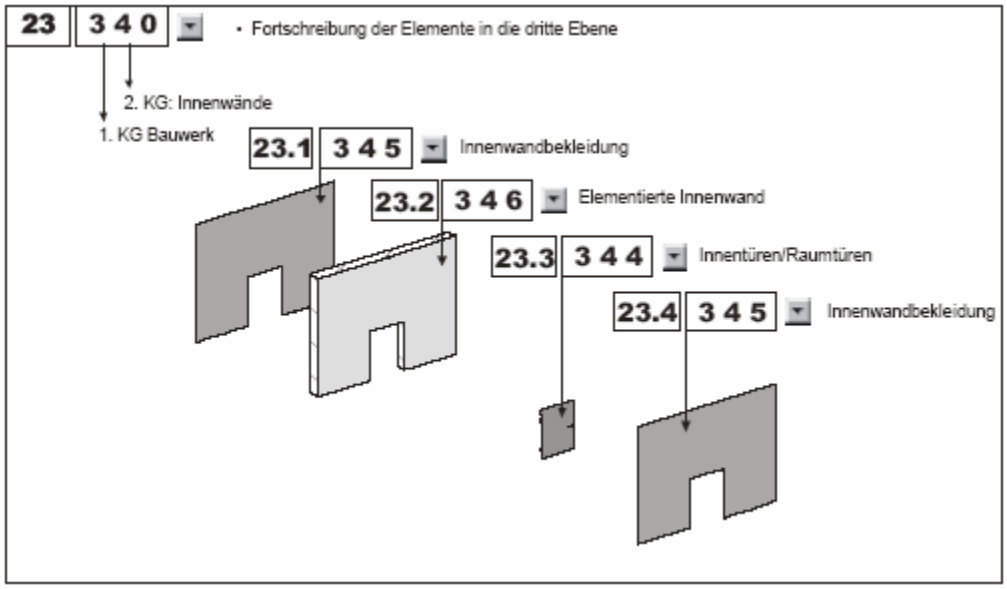


Abb. 89: Fortsetzung der Planung mit Festsetzung nach DIN 276

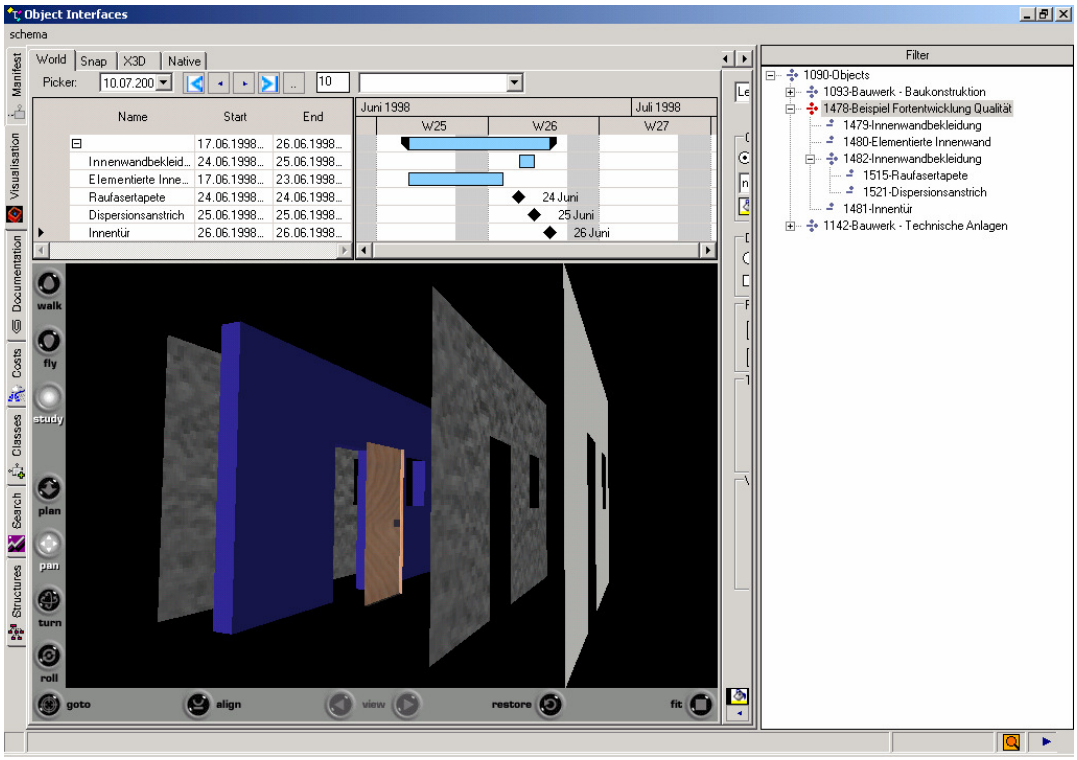


Abb. 90: Entwicklung der Planung im System nach schrittweiser Detaillierung der Elemente. Geometrische Darstellung in Kombination mit Terminierung.

Der Objektplaner weist jedem Element eine Qualität zu. Das Element soll differenziert werden. Im System führt diese Aussage, unabhängig von ihrem Ausgang zu einem Klon-Prozess. Aus dem ursprünglichen Element werden vier Elemente mit erhöhter qualitativer Information gebildet (Abb. 89 bzw. 90).

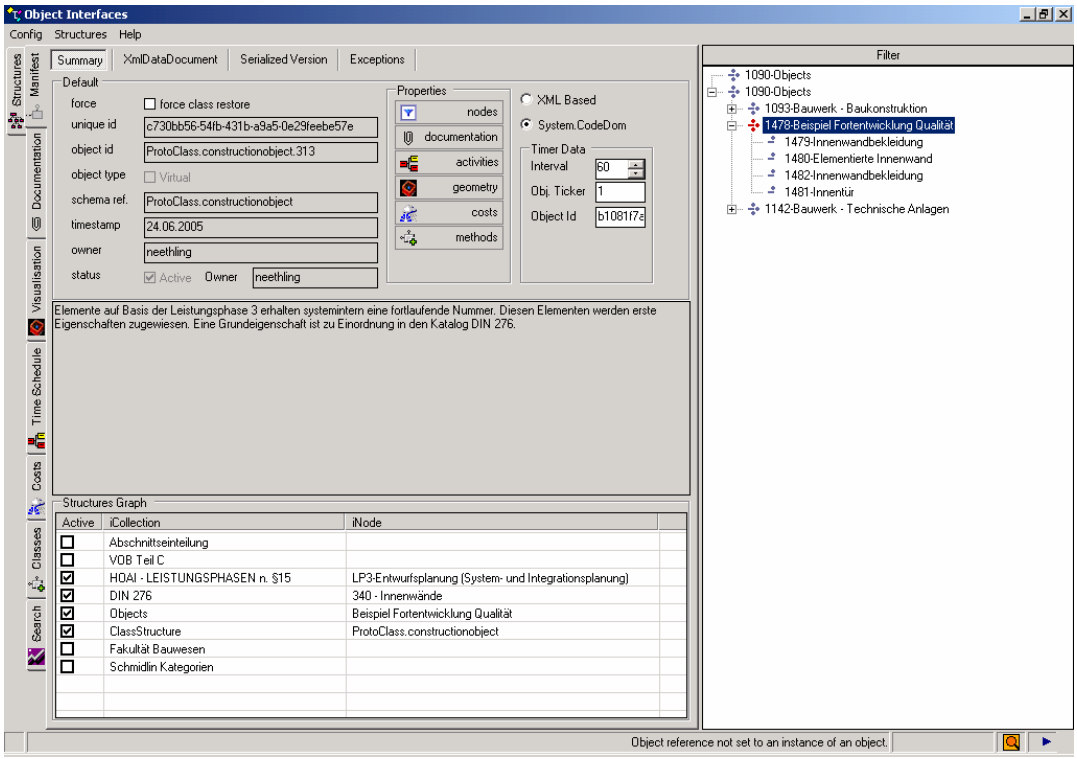


Abb. 91: Aufteilung des Innenwand-Elements durch zunehmende Detaillierung. Das Element wird in vier Unterelemente aufgegliedert.

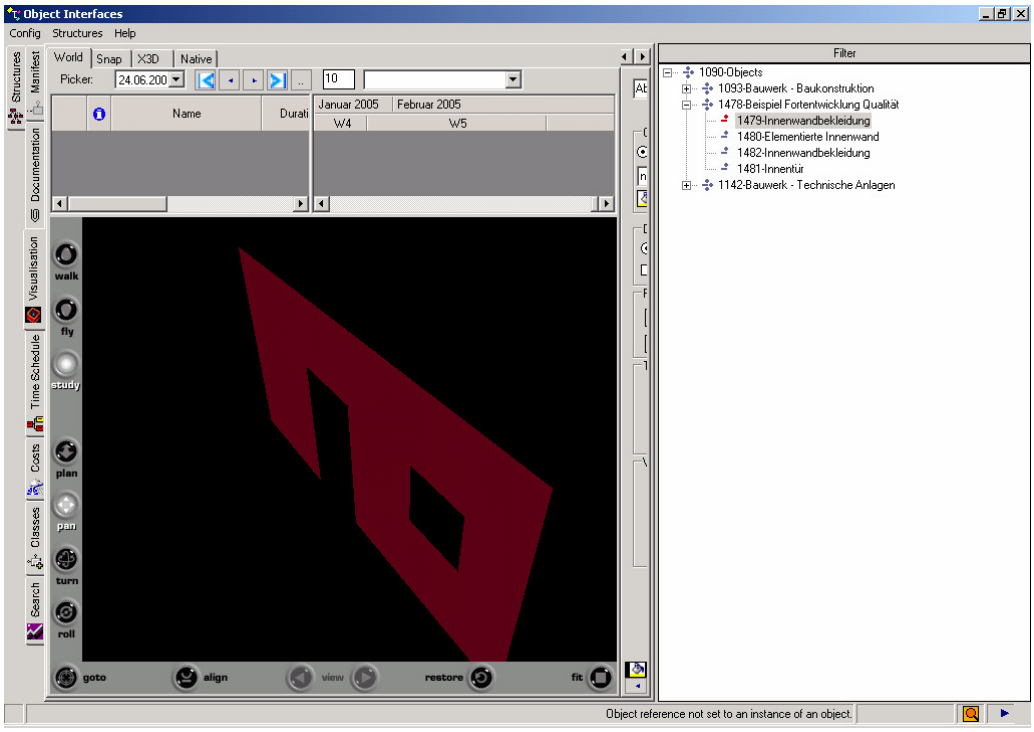


Abb. 92: Aufteilung der Innenwand durch zunehmende Detaillierung. Zuordnung der Geometrie, Innenputz.

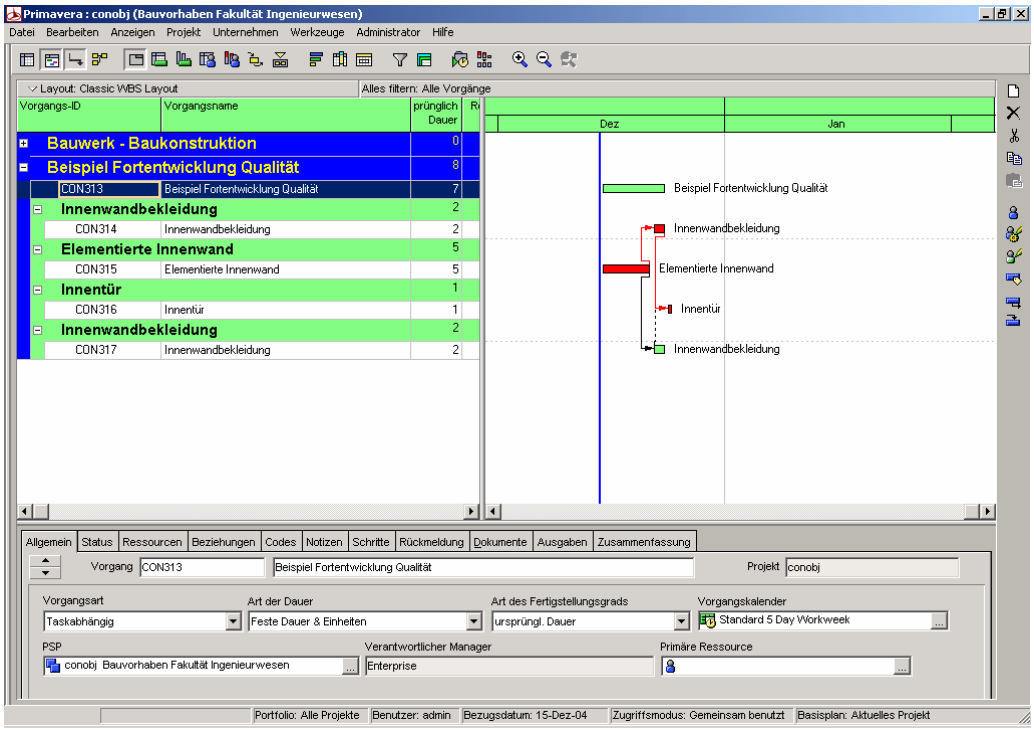


Abb. 93: Aufteilung der Innenwand durch zunehmende Detaillierung. Durch den verteilten Mechanismus wird entsprechend zu jedem Element ein Vorgang in der Terminplanung automatisch angelegt.

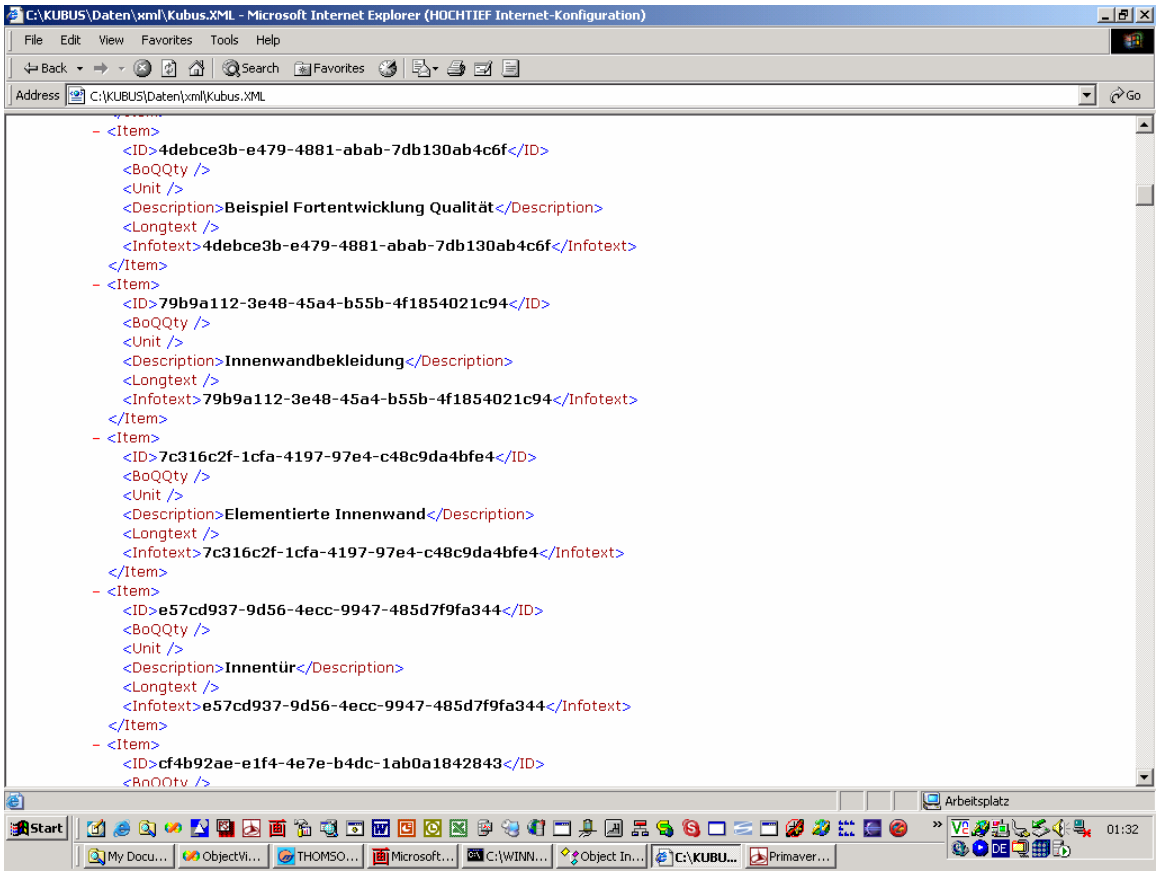


Abb. 94: Aufteilung des Innenwand-Elements durch zunehmende Detaillierung. Durch den verteilten Mechanismus wird entsprechend zu jedem Element ein Vorgang in der Kalkulation automatisch angelegt.

Die jeweilige Qualität wird entweder als Vorgabe aus der Baubeschreibung abgeleitet oder aus technologischen Gesichtspunkten planerisch ausgewählt (Abb. 91 - 94). Mit der ausgewählten Qualität ist auch der Leistungsbereich nach der DIN 18300ff. festgeschrieben (Abb. 95).

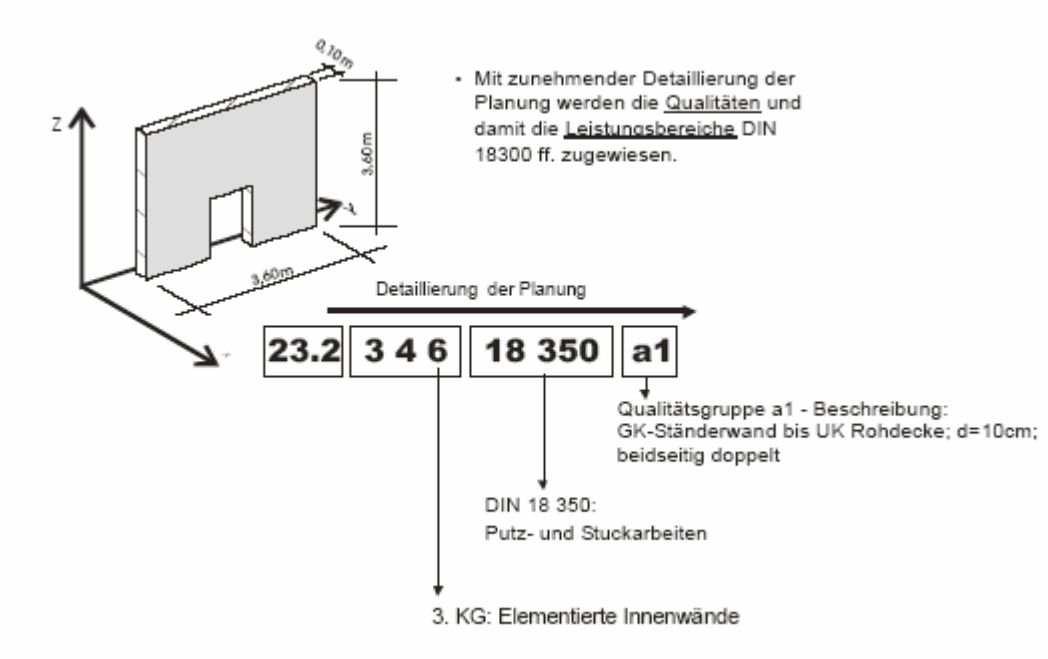


Abb. 95: Den Elementen werden Leistungsbereiche und Qualitäten zugewiesen

Diese weitere qualitative Differenzierung führt entweder zu einem neuen Klon–Prozess, oder einer feineren Qualifizierung des Elements, je nach dem ob die Regeln für den Klon-Prozess angesprochen werden oder nicht.

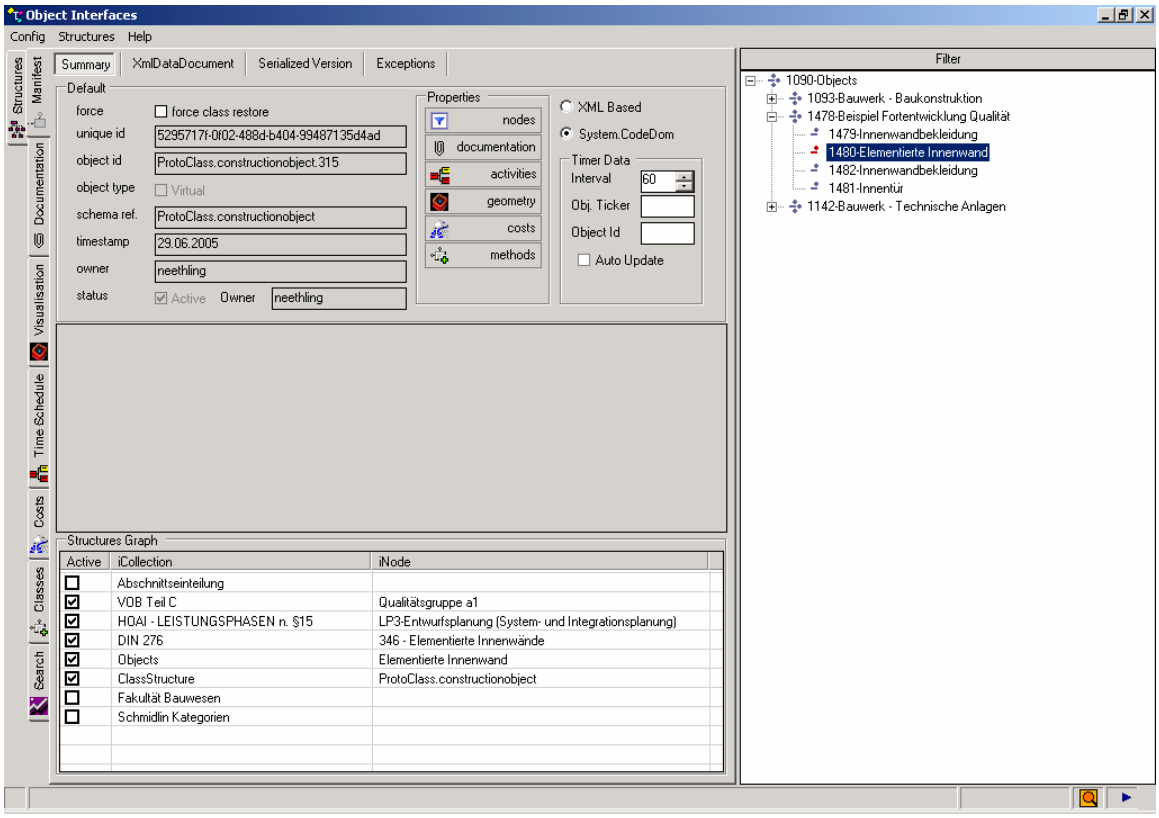


Abb. 96: Weitere Detaillierung der Innenwand ohne weitere Klon-Vorgänge.
Die Filter DIN 276 und DIN 18350 sind aktiviert.

Wenn die Klon-Regeln sich ausschließlich auf die Geometrie beziehen, werden weitere detaillierte Qualitätsangaben keine weiteren Klon-Vorgänge auslösen (Abb. 96).

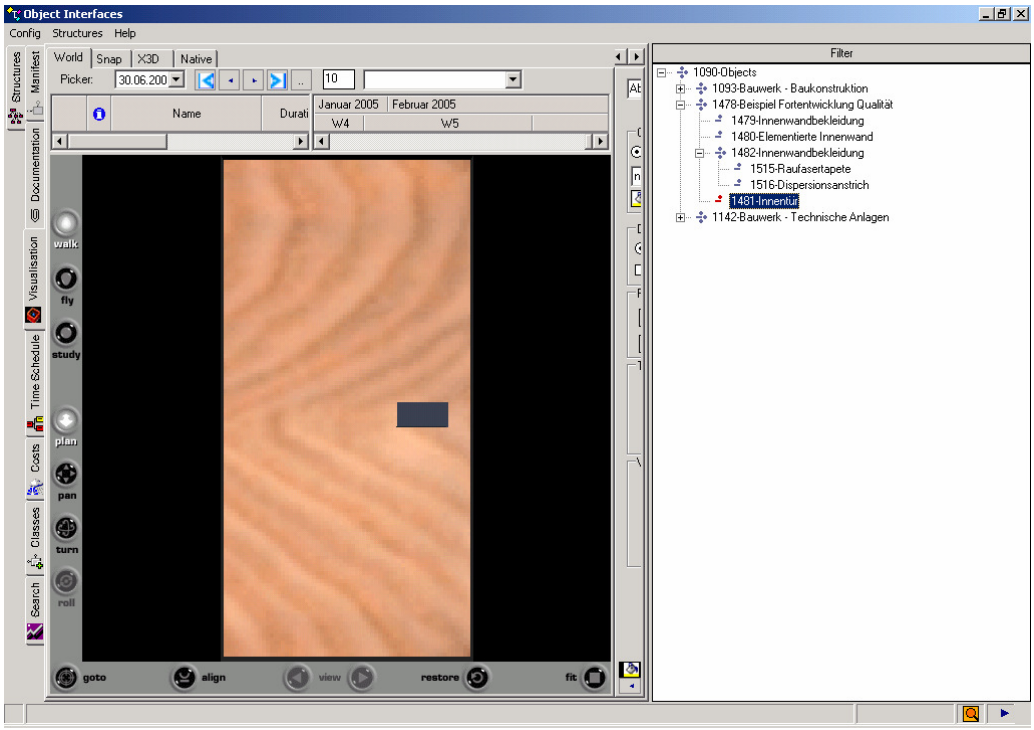


Abb. 97: Weitere Detaillierung der Innenwand: Innentür.

Die meisten Leistungsbereiche weisen verschiedene Qualitäten mit hohen Kostenspannen auf. Diese Leistungsbereiche sind weiter zu differenzieren und erhalten Qualitätsgruppen (A1, A2, A3, etc.) (Abb. 97).

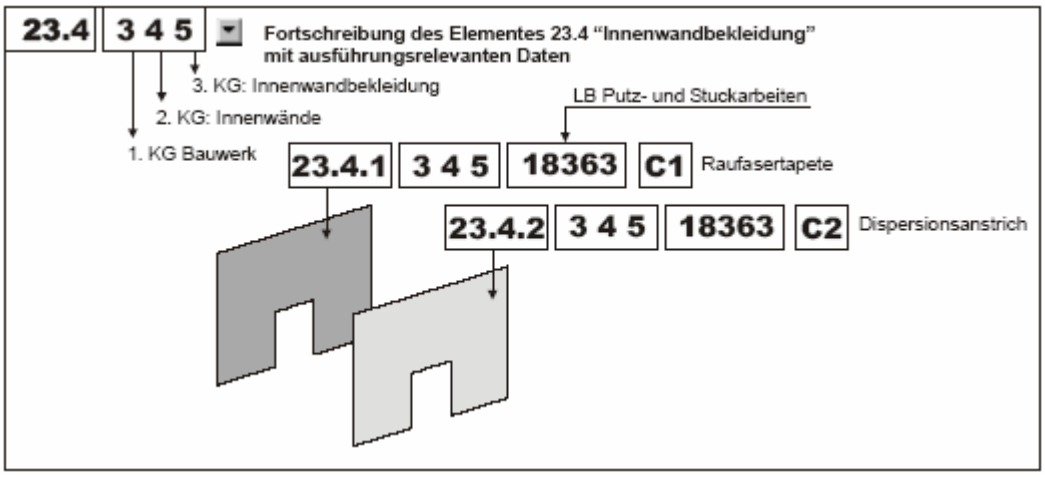


Abb. 98: Differenzierung eines Elementes

Analog zu vorgehender Betrachtung (Abb. 98) werden durch die weitere Differenzierung des Elementes die entsprechenden Filter für die Wandbekleidung zugeordnet (Abb. 99 - 104).

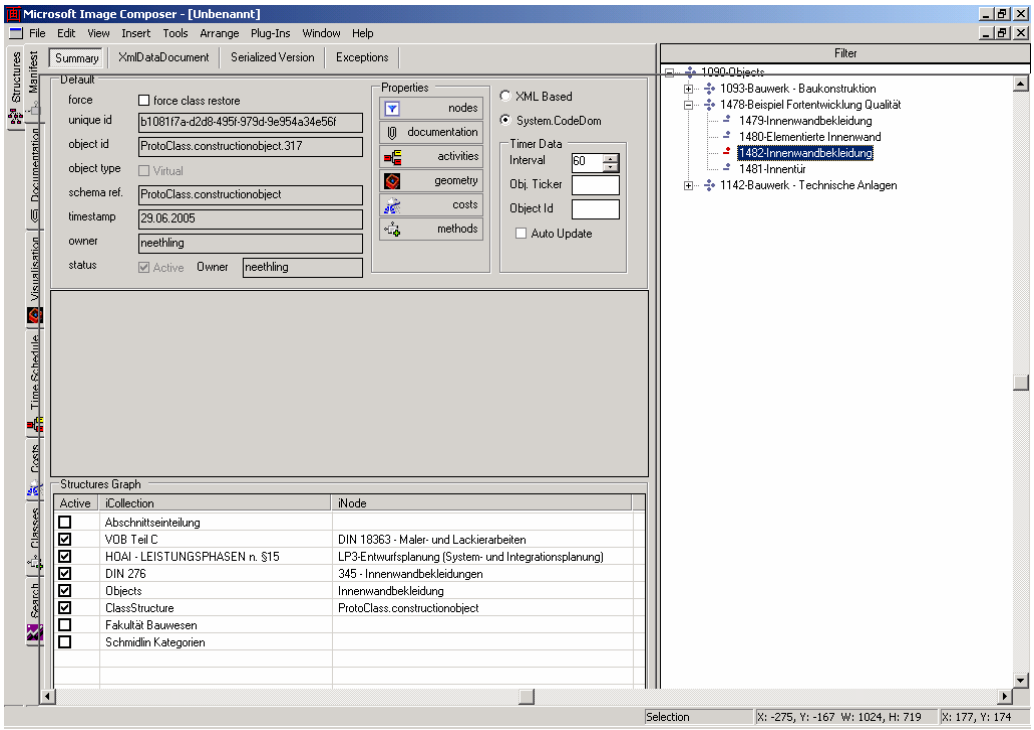


Abb. 99: Differenzierung der Wandbekleidung: Gliederungen des Objekts 23.4.1

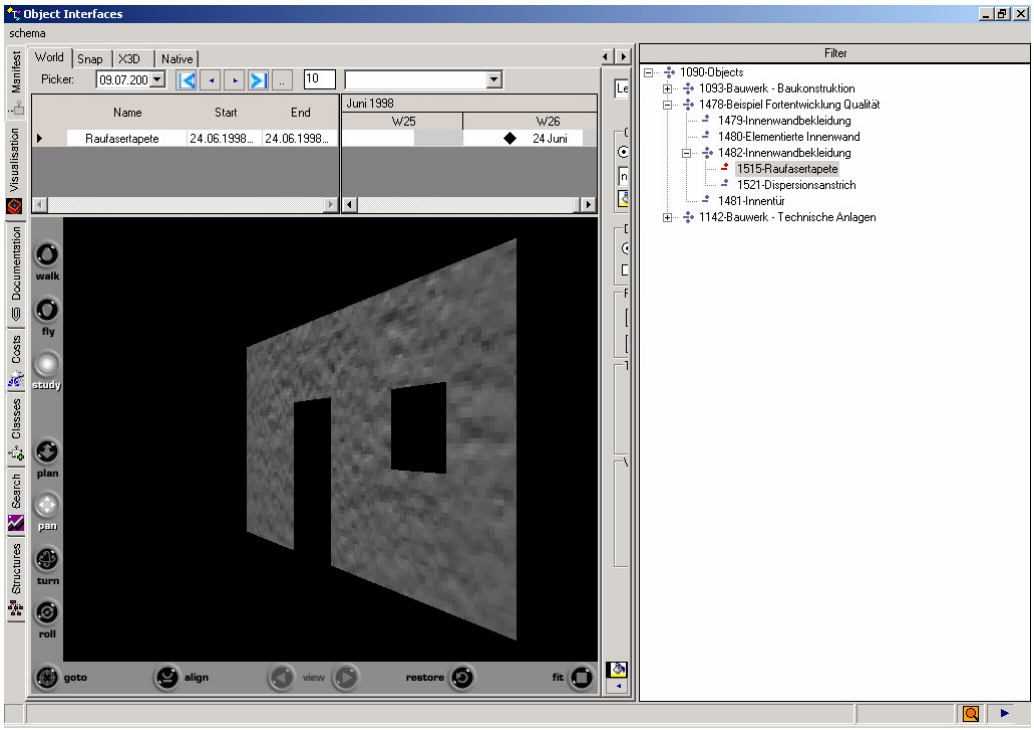


Abb. 100: Neues Objekt: Raufasertapete. Terminierung der Raufasertapete in Kombination mit der geometrischen Darstellung

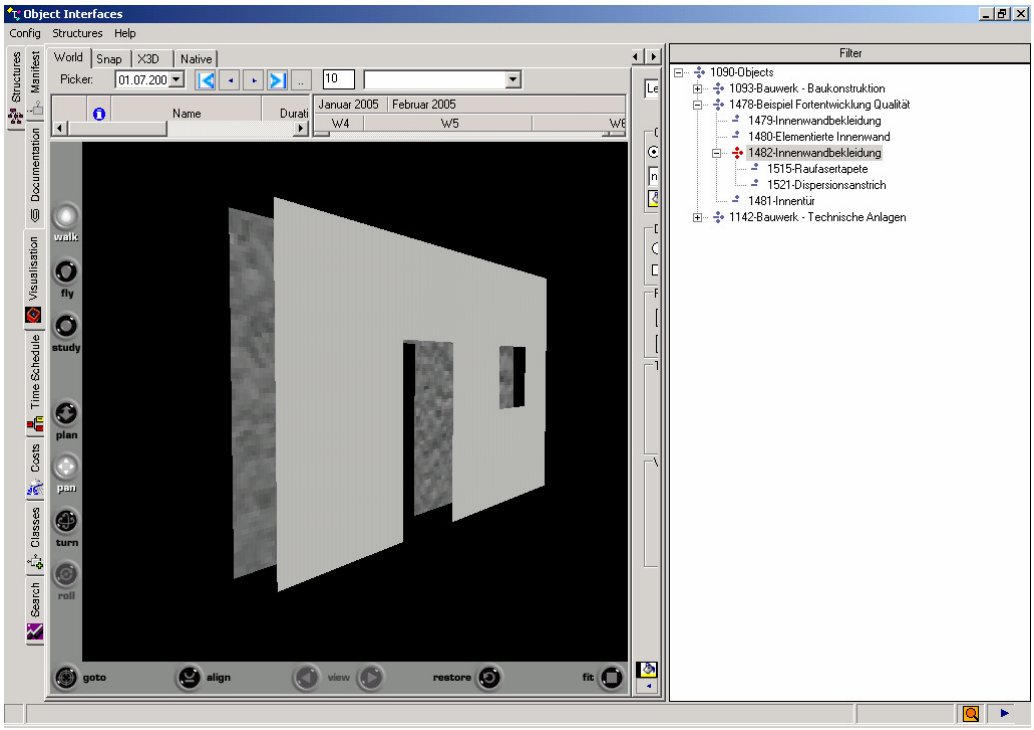


Abb. 101: Kombinierte Ansicht : 18363 C1-Rauhfasertapete und C2-Dispersionsanstrich

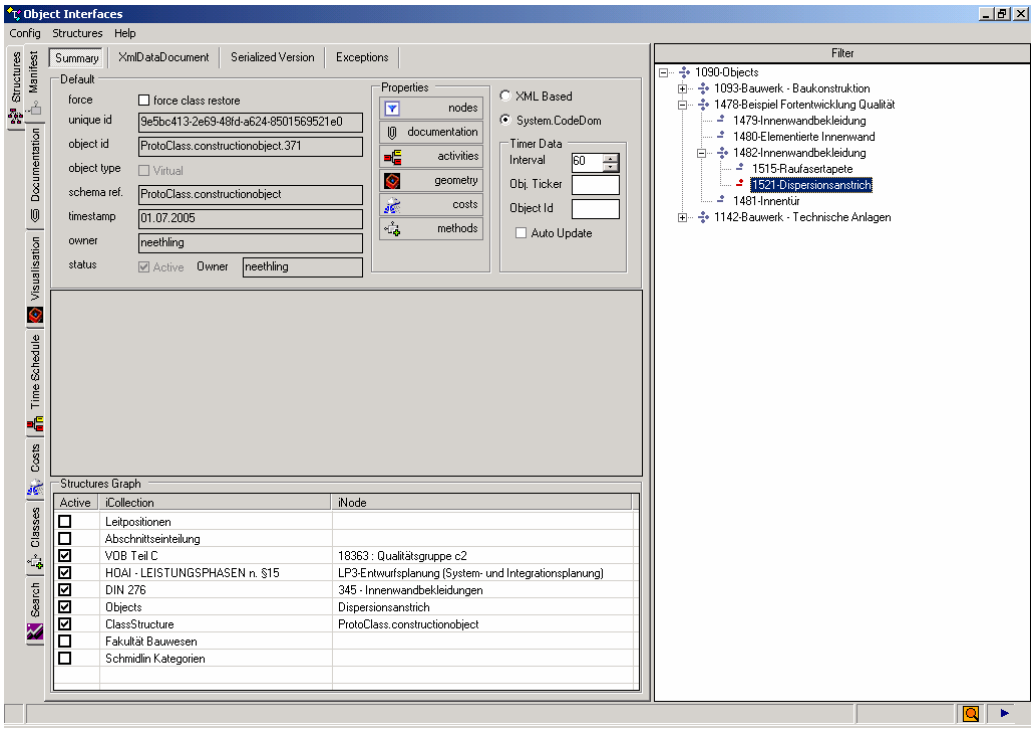


Abb. 102: Weitere Detaillierung der Wandbekleidung: 18363 C2 - Dispersionsanstrich

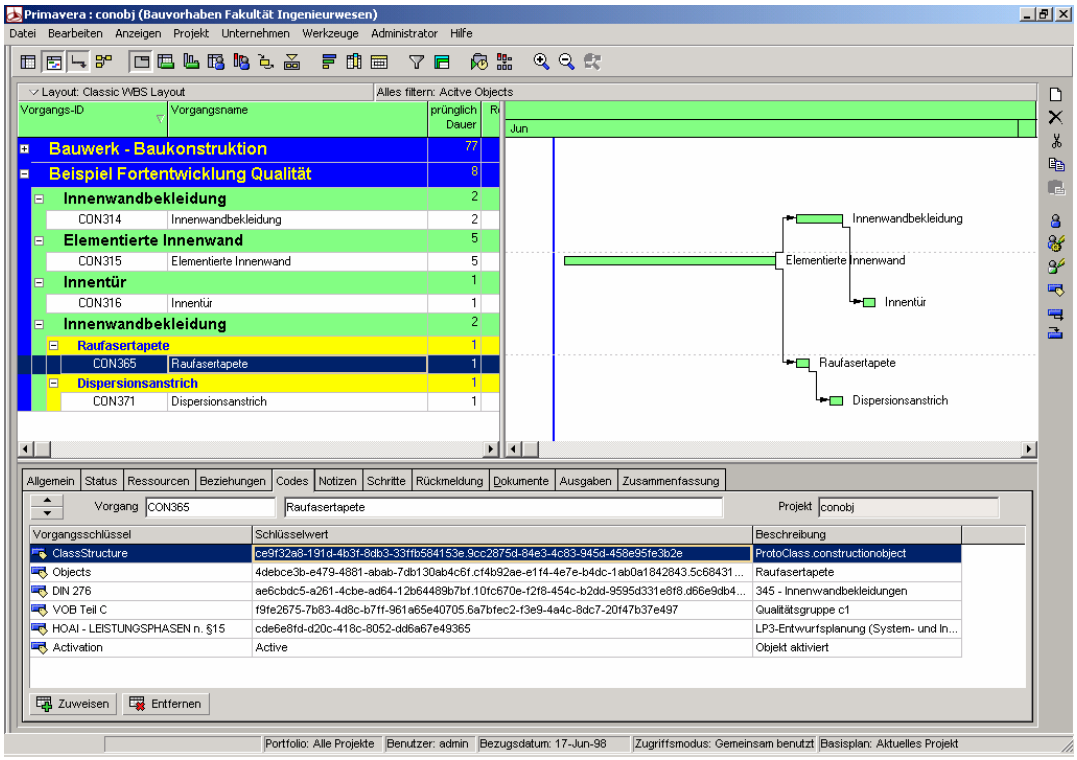


Abb. 103: Ein erhöhter Detaillierungsgrad, der zu neuen Objekten führt, verändert automatisch den Terminplan.

Durch die Fortschreibung der Gliederung der Elemente bei jeder Konkretisierung, bleiben die Entwicklungsstufen vom Ursprung des Elementes bis zum aktuellen Bearbeitungsstand erkennbar.

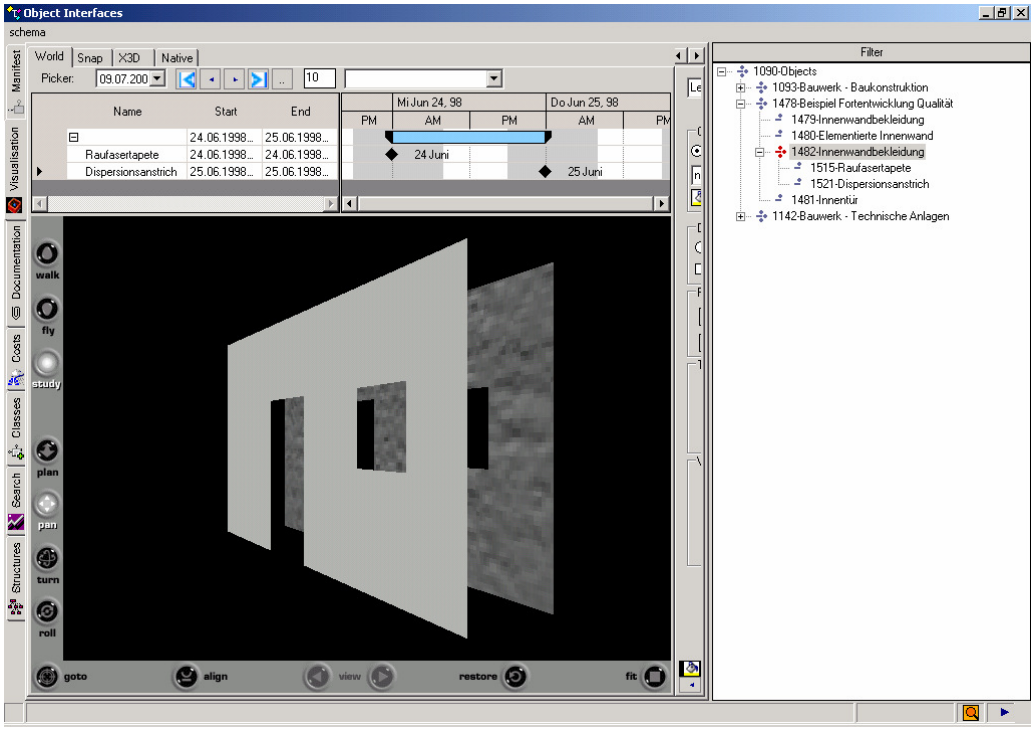


Abb. 104: Kombinierte Darstellung, Detaillierungselemente C1 und C2: Detaillierungen von Innenwandbekleidung inkl. Terminierung.

Alle Verknüpfungen zwischen Element und Baubeschreibungen, Zeichnungen oder anderen Informationen bleiben bei der planerischen Konkretisierung erhalten. Durch den objektinternen Verteilungsmechanismus werden diese Strukturen den Filterknoten und Objekten der Terminplanung und Kostenschätzung vorgeschrieben.

2.3.1 Elementorientierte Baubeschreibung

Die gebäudeorientierte Betrachtungsweise über Bauelemente führt bei der digitalen Erfassung der Elemente und ihrer Fortschreibung zu einer elementorientierten Baubeschreibung. Der Planer setzt Qualitätsstandards ein und weist diese den Elementen zu. Gleichzeitig ermöglicht ihm die detaillierte Behandlung, den Elementen konkrete Inhalte zuzuweisen. Der aktuelle Planungsstand ist in der elementorientierten Baubeschreibung somit permanent abgebildet, und wird als Abfallprodukt durch die im System verankerte Vorgehensweise generiert (Abb. 105).

(1)	(2)	(3)	(4)	(5)	(6)	(7)
Element	Kosten-gruppe	Beschreibung der Elemente	Qualitäts-gruppe	Planungs-stand	DIN	Info
	DIN 276					
...						
23.1	345	Innenwandbekleidung		LP 2		
23.2	346	Elementierte Innenwand: GK-Ständerwand bis UK Rohdecke; d=10cm, beidseitig doppeltbeplankt nach DIN 4102 (Brandschutz) und einer Kerndämmung aus MF, d=60mm nach DIN 18165 T1	A1	LP 3	18350	
23.3	344	Innentüren/Raumtüren		LP 3		
~~23.4~~						
23.4.1	345	Innenwandbekleidung: Raufasertapete auf GK-Wand	C1	LP 3	18363	
23.4.2	345	Innenwandbekleidung: Dispersionsanstrich auf Tapete	C2	LP 3	18363	
...						

Abb. 105: Auszug aus einer elementorientierten Baubeschreibung.

Sofern vertragliche Qualitäten für bestimmte Bauteile festgeschrieben sind, werden diese in der Baubeschreibung den Elementen zugewiesen. Darüber hinaus werden die allgemeingültigen Qualitäten durch die Übernahme in die elementorientierte Baubeschreibung, projektspezifisch angepasst.

2.3.2 Leitpositionen

Es reicht nicht aus, lediglich Leistungsbereiche und Qualitäten einem Element zuzuweisen. Gerade für den Übergang zur ausführungsorientierten Betrachtungsweise ist es erforderlich, die Leistungen innerhalb des Leistungsbereiches durch Leitpositionen zu kategorisieren (Abb. 106).

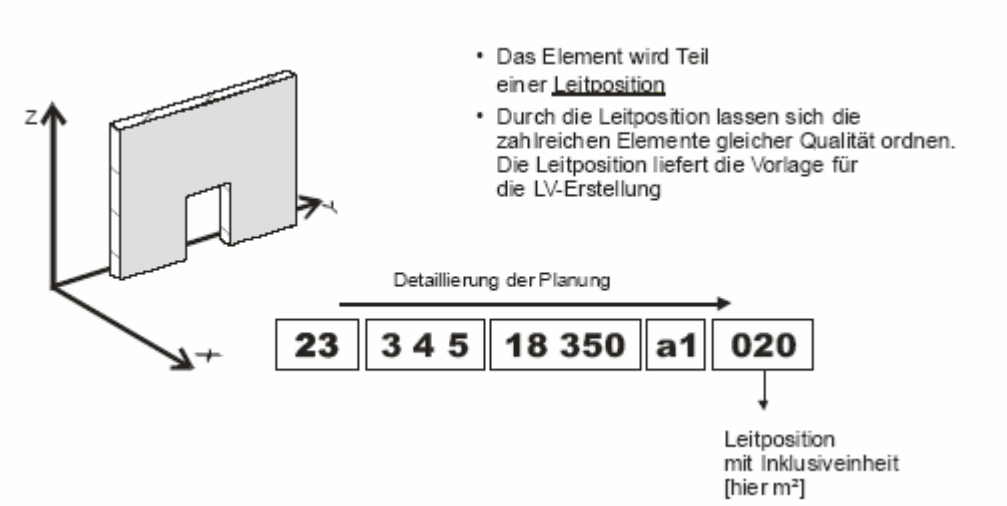


Abb. 106: Elemente werden einer Leitposition zugewiesen

Leitpositionen dienen

- der Umsetzung von der gebäudegeometrieorientierten Betrachtung in die ausführungsorientierte Betrachtung (durch iFilter),
- der kostenmäßigen Bewertung von Leistungen (Kubus durch RemoteKubusObj)
- der Generierung von Leitvorgängen (P3e mittels P3eServices und JNBridge),
- der Umschreibung in Einzelpositionen eines Leistungsverzeichnisses.

Die gebäudegeometrieorientierte Betrachtung der DIN 276 ist Grundlage der ganzheitlichen Betrachtung eines Bauwerkes. Der Betrachtungswechsel in die ausführungsorientierten Belange erfolgt mit der Qualitätsfestlegung. Für jede Qualitätsfestlegung wird eine Leitposition geschaffen. Diesen Leitpositionen werden die Elemente zugeordnet. Dadurch bleiben die ganzheitliche Erfassung und die Übersichtlichkeit erhalten.

Diese Entwicklung der Leitposition kann über einen zusätzlichen Filter erfolgen. Das bietet den Vorteil, dass der existierende Filter nicht überfrachtet wird mit Detaillierungen. Somit können auch alternative Gliederungen eingesetzt werden. (Abb. 107).

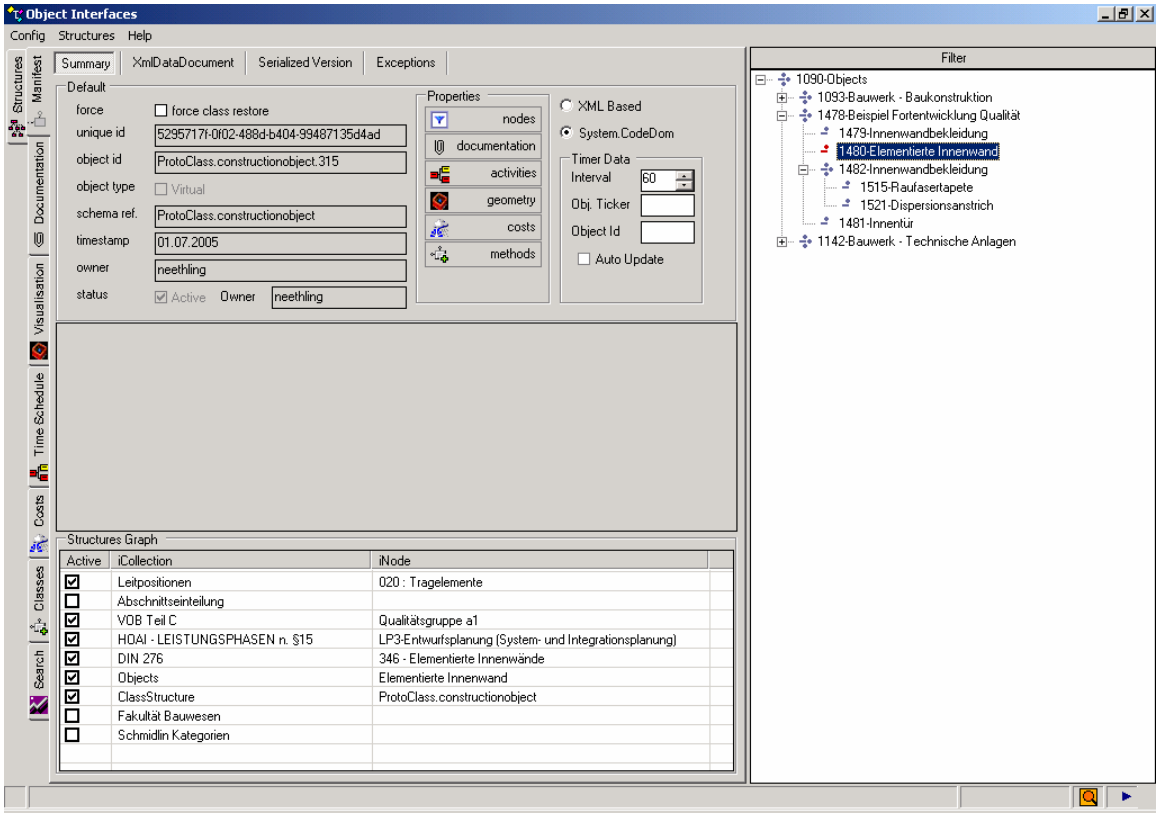


Abb. 107: Leitpositionen werden über einen zusätzlichen Filter zugeordnet.

Diese unterschiedlichen Betrachtungsweisen ermöglichen den fließenden Übergang von der gebäudegeometrieorientierten zur ausführungsorientierten Denkart durch eine Umsortierung der Daten [Langen und Schiffers, 2005. S. 216].

Die Erstellung einer hinreichenden Kostenberechnung des Bauwerks erfolgt durch die monetäre Bewertung der Leitpositionen. Eine Bewertung nach Bauteilen, also nach der 2. Ebene der Kostengliederung der DIN 276 vornehmen zu wollen, ist unseriös, weil keine brauchbaren Bewertungsmöglichkeiten vorliegen und es diese auch nicht geben kann. Eine korrekte Bewertung setzt die 3. Ebene der Kostengliederung voraus, also eine Betrachtung nach Bauelementen. Die z.T. hohen Kostenunterschiede der verschiedenen Qualitäten eines Leistungsbereiches verlangen eine Qualitätsfestlegung für die Elemente. Erst durch diese Festlegungen sind Leitpositionen geschaffen und eine hinreichende Kostenberechnung erzielbar (vgl. Kapellmann und Schiffers).

Beginnt nach der monetären Betrachtung die produktionsorientierte Sichtweise, sind aus den vorhandenen Leitpositionen Leitvorgänge zu entwickeln. Dazu werden i.d.R. Leitpositionen abschnittsbezogen zu einzelnen Leitvorgängen zusammengefasst. Dabei kann eine einzige Leitposition zu einem Leitvorgang führen, i.d.R. aber werden mehrere Leitpositionen zu einem Leitvorgang zusammengefasst (vgl. Kapellmann und Schiffers).

2.3.3 Geometrische Daten

Durch die dreidimensionale zeichnerische Erfassung der Elemente liegen die geometrischen Daten im System vor. Die Mengenermittlung der Elemente ist das rechnerische Ergebnis aus der polygonalen Knotenerfassung. Sie erfolgt aus den geometrischen Daten des an das Objektmodell angebundenen CAD Programmes. Die Mengen werden über folgendes Attribut aufgerufen:

```
CNOI_FreeAttributePtr pattrVolume =
CNOI_Service_FreeAttributes::GetAttributeByNumber(pDB, vec[i],223);
```

Die Volumen und Flächenmodelle ermöglichen, aus dem System die verschiedenen produktionsorientierten Mengen zu erhalten, soweit die dafür benötigten Klassen im CAD System gewählt wurden. Durch die Verbindung der Elemente mit einer Leitposition werden die einzelnen, elementorientierten Mengenwerte zusammengefasst.

Dabei richten sich die Mengenwerte nach der festgelegten Leiteinheit, der Leitposition, aus (Abb. 108).

(1)	(2)	(3)	(4)	(5)	(6)	(7)
Element	Breite	Länge	Höhe	Fläche	Volumen	Abzug
	[m]	[m]	[m]	[m²]	[m³]	n. DIN 277
				[(3)x(4)]	[(2)x(3)x(4)]	
23.2	0,10	3,60	3,60	12,96	1,296	Türöffnung <2,5 m²

Abb. 108: Tableau mit geometrischen Daten

Die hier erfassten Mengen stellen SOLL-Mengen dar, die

- für die Kostenberechnung in Kubus,
- für die Terminplanung (produktionsorientiert als SOLL-Vorgabe) in P3e,
- als SOLL-Menge für die Ausschreibungsvorbereitung und Vergaben in P3e

Verwendung finden.

Die Oberflächen und Volumenattribute der in Allplan eingesetzten geometrischen Klassen des Typs CNOI_Arch_Objects können nicht mit der NOI Schnittstelle automatisch ausgelesen werden, da sie Architekturobjekte und keine konstruktiven Elemente darstellen.

2.3.4 Grundsätzliche Einteilung

2.3.4.1 Unterteilung des Gesamtbauwerkes

Die Elemente sind erfasste, dreidimensionale, geometrische Figuren im Raum. Die Koordinaten der Polygonpunkte sind systemintern ablesbar. Neben der analytischen Ortsangabe im dreidimensionalen Koordinatensystem stellen die projekt- und produktionsspezifischen Gebäudegliederungen notwendige Unterteilungen dar.

Die räumlichen Gliederungskriterien eines Projektes können ab Leistungsphase 3 vorgenommen werden. Hinreichende Einteilungen sind:

- Bauwerk
- Abschnitt
- Geschoss
- Raum

Die ganzheitliche Erfassung hat hierbei den ergänzenden Vorteil, dass eine Kontrolle zwischen SOLL- und IST-Mengen stattfindet und somit ein Abgleich zwischen den angesetzten Mengen in der Kostenberechnung und den tatsächlichen Mengen erfolgt.

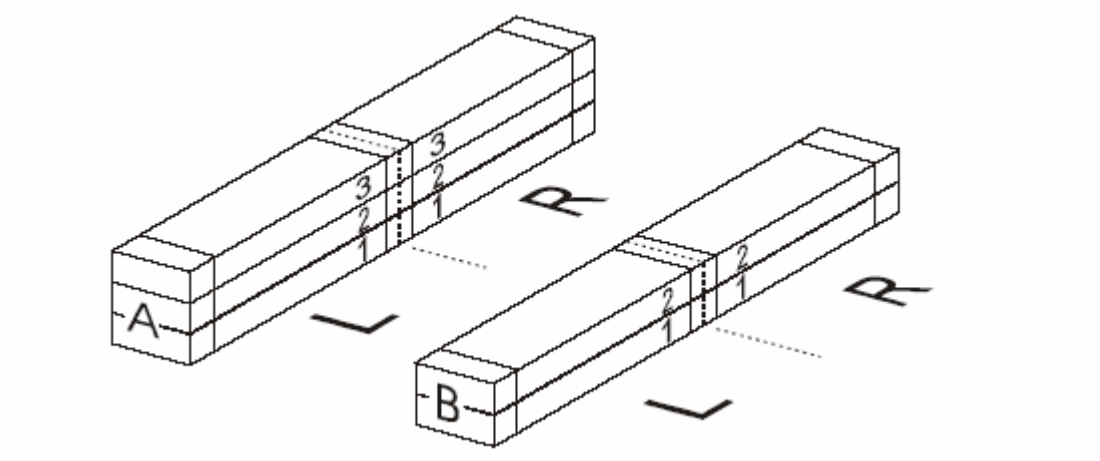


Abb. 109: Beispielhafte Abschnittseinteilung bei zwei Gebäuden eines Bauprojektes [Kapellmann und Schiffers, Bd. 2, 2000. S. 589].

Sofern es sich bei einem Bauprojekt um mehrere Bauwerke handelt, sind diese alphanumerisch zu kennzeichnen (Abb. 109). Die Abschnittsunterteilung des Bauwerkes steht eng zu einer nutzungsspezifischen Unterteilung und/oder zur produktionsorientierten Bauausführung. Im Hochbau ist die ebenenweise Betrachtung durch „Geschosse“ üblich. Als kleinste Untergliederung eines Bauwerkes wird der Raum betrachtet.

Eine zusätzliche Rasterung der Planungen erfolgt durch eine flächenmäßige Betrachtung, z.B. der Bekleidungsflächen, übereinander hängenden Bekleidungsflächen, Belagsflächen oder übereinander liegenden Belagsflächen. Dabei kann es sein, dass nicht alle Nutzflächen mit der Belagsart, z.B. „Estrich“ und/oder einer anderen Belagsart versehen werden.

2.3.4.2 Abschnittsbezogene Bauwerksgliederung

Bauabschnitte gliedern das Bauwerk meist ebenenweise, dies können sowohl horizontal unterteilte Bauabschnitte, als auch vertikal unterteilte Bauabschnitte sein.
Bauabschnitte werden dabei entweder auf die Geometrie oder auf den Leistungsbereich bezogen.
Die geometrisch orientierten Bauabschnitte untergliedern das Bauwerk als Ganzes und somit vollständig. Die leistungsbereichsbezogene Abschnittsuntergliederung ist zweckgebunden und orientiert sich an den produktionstechnischen Abläufen. Ein Estrich wird beispielsweise geschossweise eingebracht und orientiert sich nicht an einer späteren Raumaufteilung, sondern lediglich an technologischen und produktionstechnischen Randbedingungen innerhalb des Leistungsbereiches. Daher können sich die leistungsbereichsbezogenen Abschnittsunterteilungen des Bauwerks überlagern.

2.3.4.3 Untergliederung durch Geschosse

Im Hochbau werden Bauwerke in Geschosse untergliedert. Die technologischen Abläufe richten sich nach diesen nutzungsspezifischen untergliederung. Üblicherweise erfolgt die Fertigung ebenenweise. Für die Gliederung wurde Raumzuordnung als Filter etabliert, der jedem neu instanzierten Objekt zugeordnet wurde.

2.3.4.4 Räume

Räume sind nutzungsspezifische Einheiten. Sie stellen dabei die kleinste Bauwerksuntergliederung dar.
Die Untergliederung in Räume hat von der Phase des fortgeschrittenen Innenausbaues her den Vorteil „abschließbarer“ Einheiten. Die ausführenden Fachunternehmen können, ohne sich gegenseitig zu stören, in abschließbaren Einheiten ihre Ausbauleistungen erbringen.

Die Unterteilung des Bauwerkes in Räume liefert den fließenden Übergang von der ausführungsorientierten Denkweise in ein nutzungsspezifisches Gebäudemanagement. Die ganzheitliche Erfassung der Planungs- und Ausführungsdaten liefert dabei die für die Nutzungsphase wertvollen raumbezogenen Daten.

2.3.4.5 Elementorientierte Betrachtung

Die Elemente haben ihre fixe Positionierung im dreidimensionalen Koordinatensystem. Die Gesamtheit aller Elemente bildet das dreidimensionale Gebäudemodell, das Bauprojekt.

Jede Untergliederung dient der besseren Orientierung und als Kommunikationsmittel. Eine Untergliederung des Bauwerkes bedeutet, ausgewählten Elementen eine Untergliederungsbezeichnung zuzuweisen. Zahlreiche Elemente bilden einen Raum. Die Untergliederung in Raum, Geschoss, Bauabschnitt und Bauwerk wird als Information mit den Elementen verknüpft. Die ganzheitliche Erfassung in einem dreidimensionalen Gebäudemodell verhindert daher das redundante Erfassen von Informationen. Türen werden dem Raum in Aufschlagrichtung zugeschlagen. Eine Tür in der Flurwand eines Büros taucht mit ihren Elementeigenschaften innerhalb des Raumes „Büro“ auf. Innerhalb des Raumes „Flur“ ist lediglich die Wandöffnung erfasst. Innerhalb der Türliste bleibt die Tür einmalig.

Eine Trennwand zwischen zwei Räumen - oder der Estrich als Element unter diesen zwei Räumen - wird innerhalb des Raumbuches in beiden Räumen als Element aufgeführt jeweils mit dem, dem Raum zugewandten Wand- und Fußbodenaufbau. In Summe handelt es sich jeweils um ein Wand- und ein Estrichelement.

Die konkrete Ausbildung eines Bauelementes gemäß der DIN 276 kann durch mehrere Schichten erfolgen. Das Bauelement Bodenbelag kann z.B. aus den drei Schichten Dämmung, Estrich und PVC-Oberbodenbelag bestehen. In einem solchen Fall wird unterhalb des Elementes die Gruppe der Elementschichten behandelt. Die korrekte Ausbildung der Schicht in qualitativer Hinsicht führt unausgesprochen, tatsächlich und praktisch zu einer zusätzlichen Kategorie. So gelangt man zu der ersten ausführungsorientierten Kategorie des Leistungsbereiches. Jede Schicht steht dabei für einen Leistungsbereich (Abb. 110).

Räumliche und geometrische Angaben

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
Element	Ort	Bauwerk	Ebene	Achsen	Raum	Abschnitt	Geometrische Angaben		
							x	y	z
23.2	Flurwände außerhalb TH und Archive	B	EG	15/B	13				

Abb. 110: Tableau mit räumlichen und geometrischen Elementangaben []

3 Umsetzung des Fakultätsgebäudes im System

3.1 Auflistung der Objekte

Zur Evaluierung des OO-Konzeptes wurde das in Kapellmann und Schiffers aufgeführte Fakultätsgebäude gewählt [Kapellmann und Schiffers, 2000, Band II, Anhang A.2.4.3.3.1, Blatt 1]. Dieses Modell soll auch fortgeschrittene Konzepte wie Zeit, die Vererbung oder das Klonen von Objekten bei Änderungen prüfen.

Als erstes wurde das Fakultätsgebäude als OO-Model abgebildet. Die dadurch entstandene Objektlandschaft stellt den Zustand des Hauses in einer Zeitscheibe innerhalb der HOAI Phase 3 dar. Der Entwicklungsweg des Gebäudes, auf dem Objektveränderungen durch geklonte Objekte dargestellt werden, wurde in Kapellmann und Schiffers, Band II, nicht beschritten. Die Vererbungstechnik wurde in einem früheren Kapitel aufgeführt. Die wesentlichen Facetten des Systems sind die geometrische Darstellung, die Terminplanung und die Kostenermittlung.

3.1.1 Dokumentation des Gebäudes in der Phase 1

Die Phase 1 enthält in der Regel keine Zeichnungen, sondern eher eine funktionale Beschreibung des Gebäudes. Zur Visualisierung des Gebäudes wurde im Objektmodell allerdings eine Gesamtansicht dem einzigen Objekt dieser Phase zugeordnet.

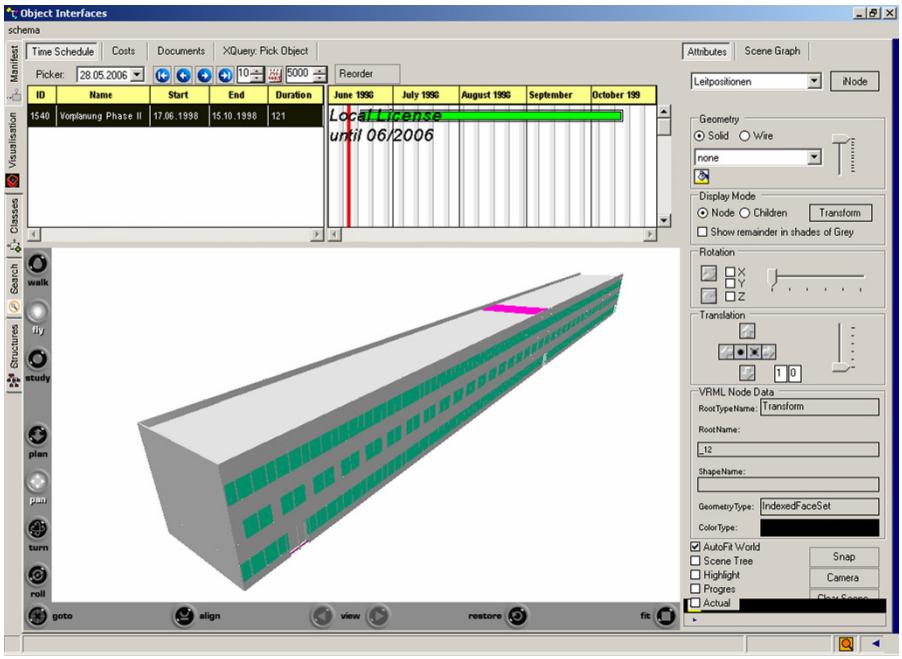


Abb. 111: Gesamtansicht Gebäude in der HOAI Leistungsphase 1. Nur ein Objekt wurde angelegt.

3.1.2 Dokumentation des Gebäudes in der Phase 2

Für die Phase 2 wurde das Gebäude wie folgt in fünf Objekte gegliedert wie folgt:

- EG
- 1.OG
- 2.OG (Abb. 113)
- Dach
- Fassade (Abb. 114)

Diesen Objekten wurde jeweils eine Gesamtgeometrie zugeordnet, ein beispielhafter Terminplan wird in Abbildung 112 aufgeführt.

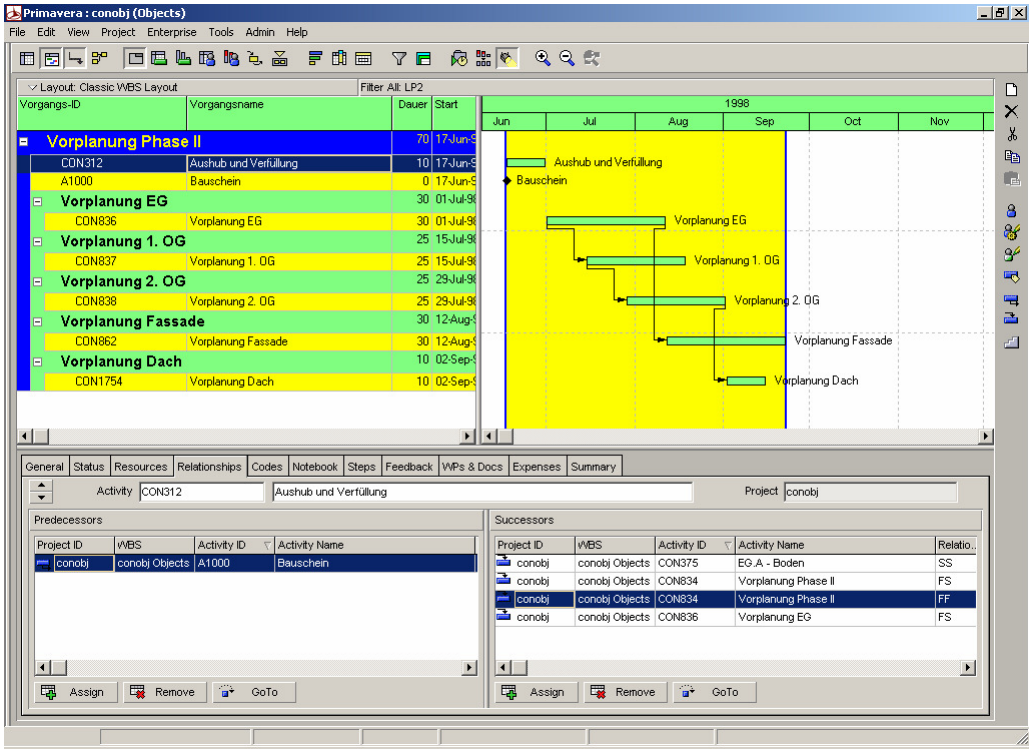


Abb. 112: Beispielhafter Terminplan HOAI Leistungsphase 2

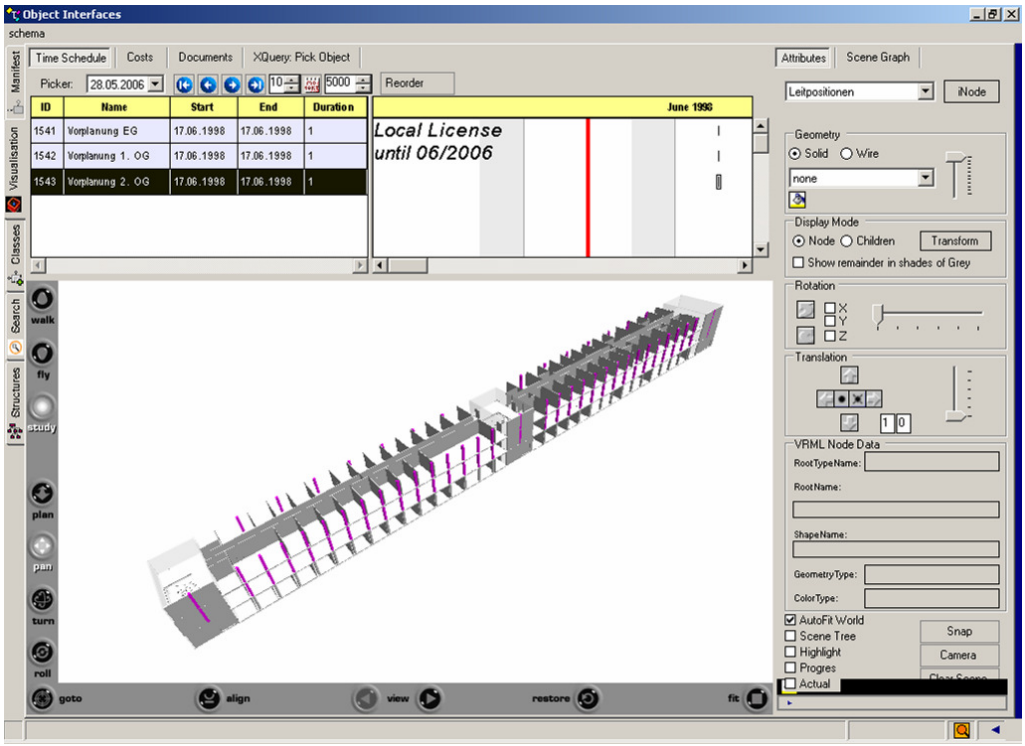


Abb. 113: Ansicht Gebäude in der HOAI Leistungsphase 2. Geschoße EG-2.OG ohne Dach und Fassade.

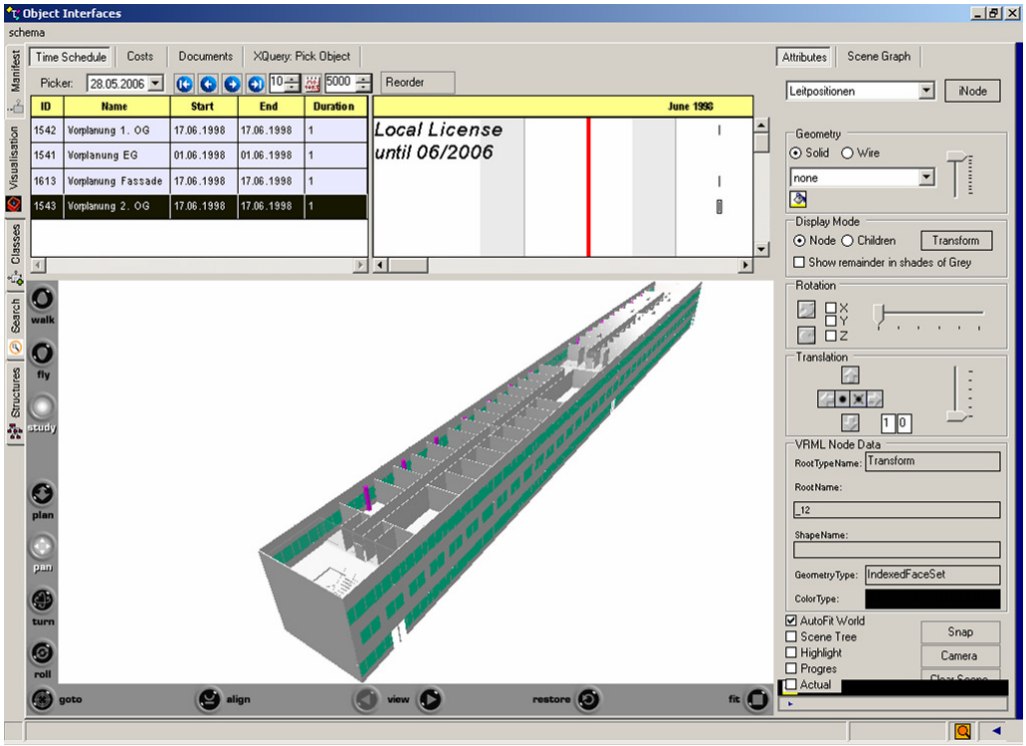


Abb. 114: Gesamtansicht: Gebäude in der HOAI Leistungsphase 2 ohne Dach.

3.1.3 Dokumentation des Gebäudes in der Phase 3

Abbildungen 115 bis 127 stellen Teilaspekte und Gesamtansichten des Gebäudes dar. Abbildung 116 stellt einen Ausschnitt aus dem Terminplan dar (P3e).

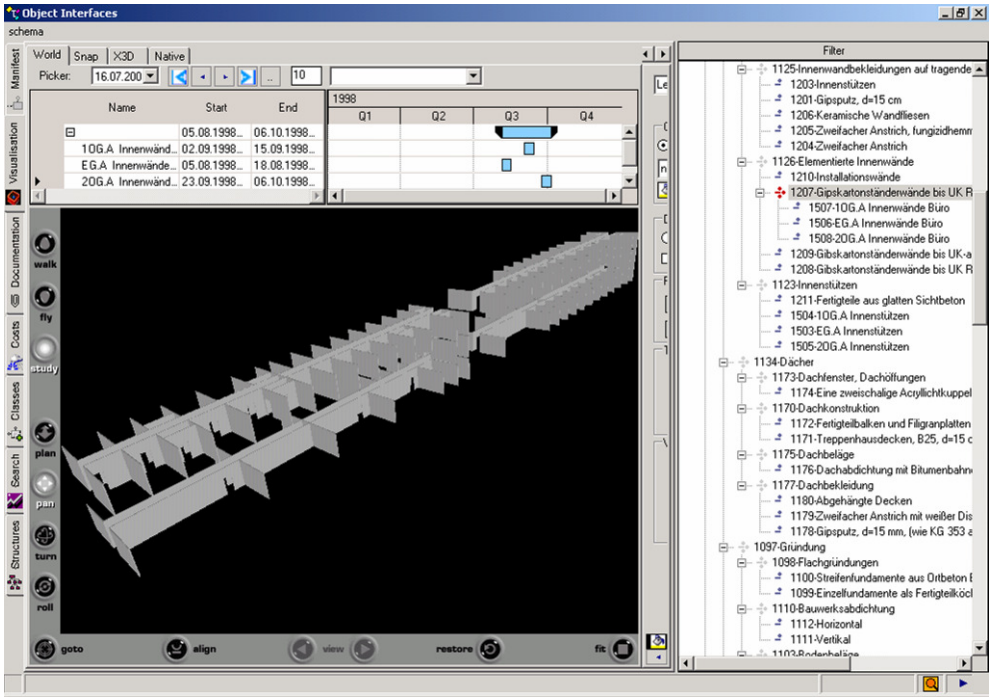


Abb. 115: Geometrische Darstellung der Objekte: Innenwände. Durch Auswahl eines Objekts werden dessen Kinder ermittelt. Deren Geometrien werden wiederum in den VRML Viewer geladen und dargestellt.

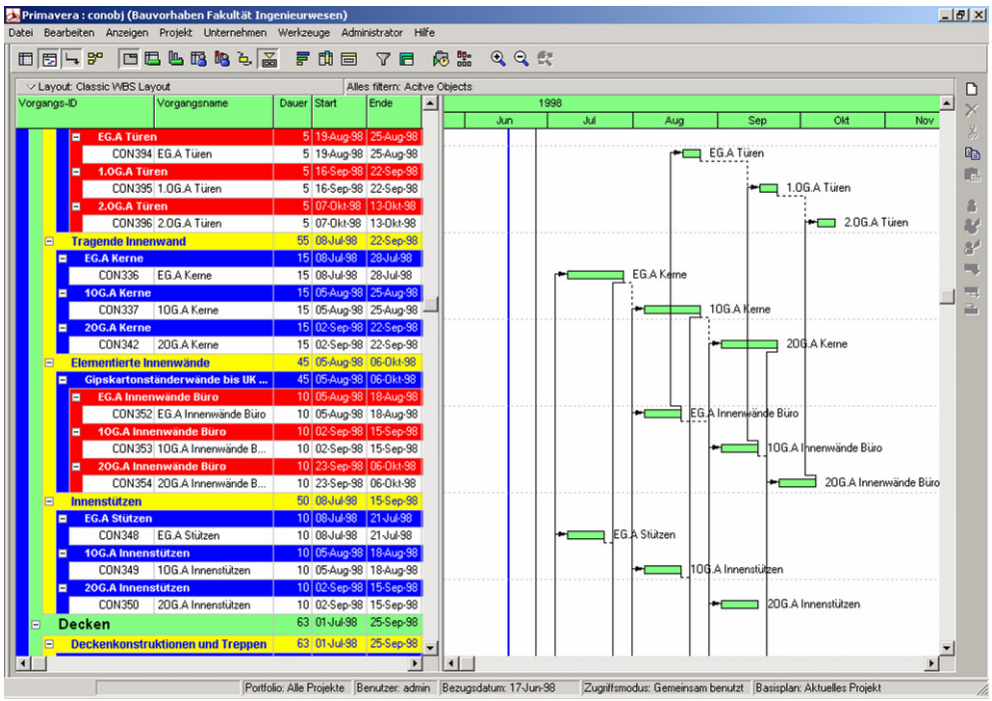


Abb. 116: Terminplanerische Darstellung der Objekte in dem Terminplanungsprogramm P3e. Eine ähnliche Darstellung ist in dem Systembrowser dadurch möglich, dass die Activity Objekte alle Termininformationen enthalten.

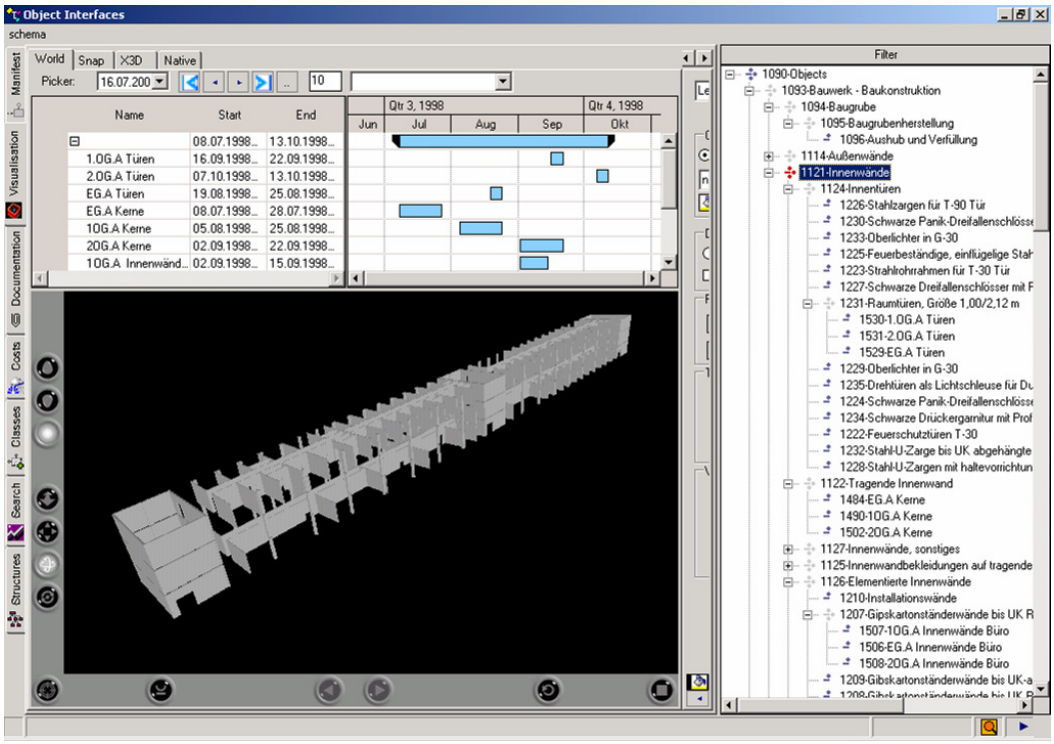


Abb. 117: Geometrische Darstellung der Objekte: Innenwände aller Art nach DIN 276.

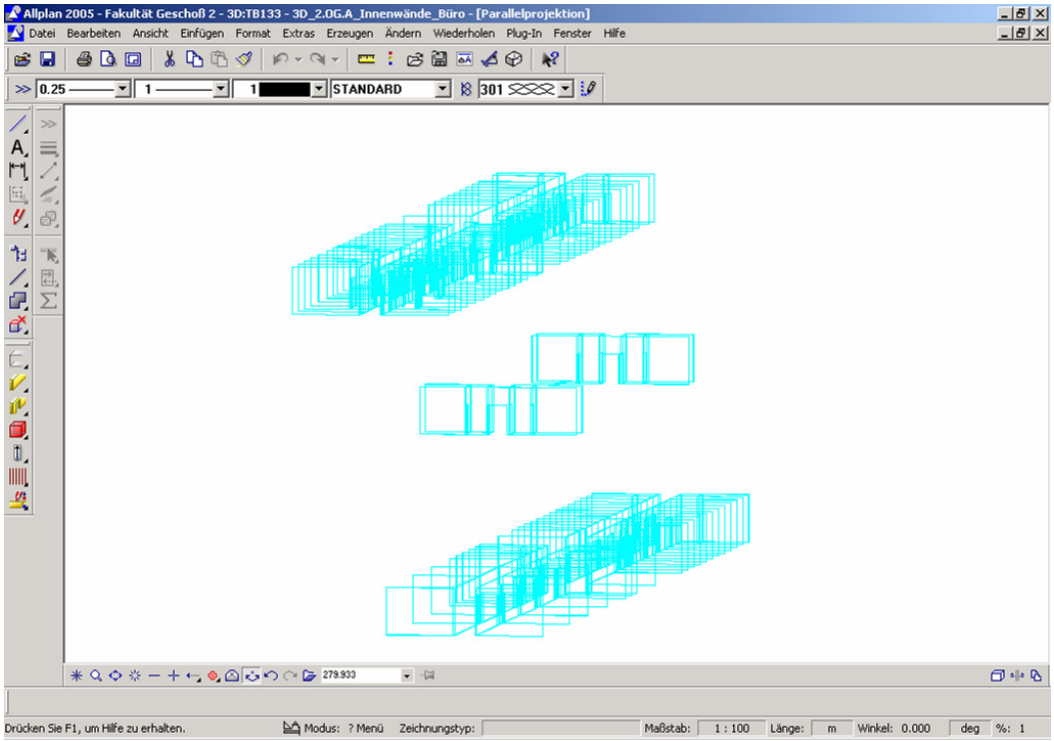


Abb. 118: Geometrische Darstellung der Objekte: Innenwände sind in Allplan2005 im Versatz gezeichnet worden.

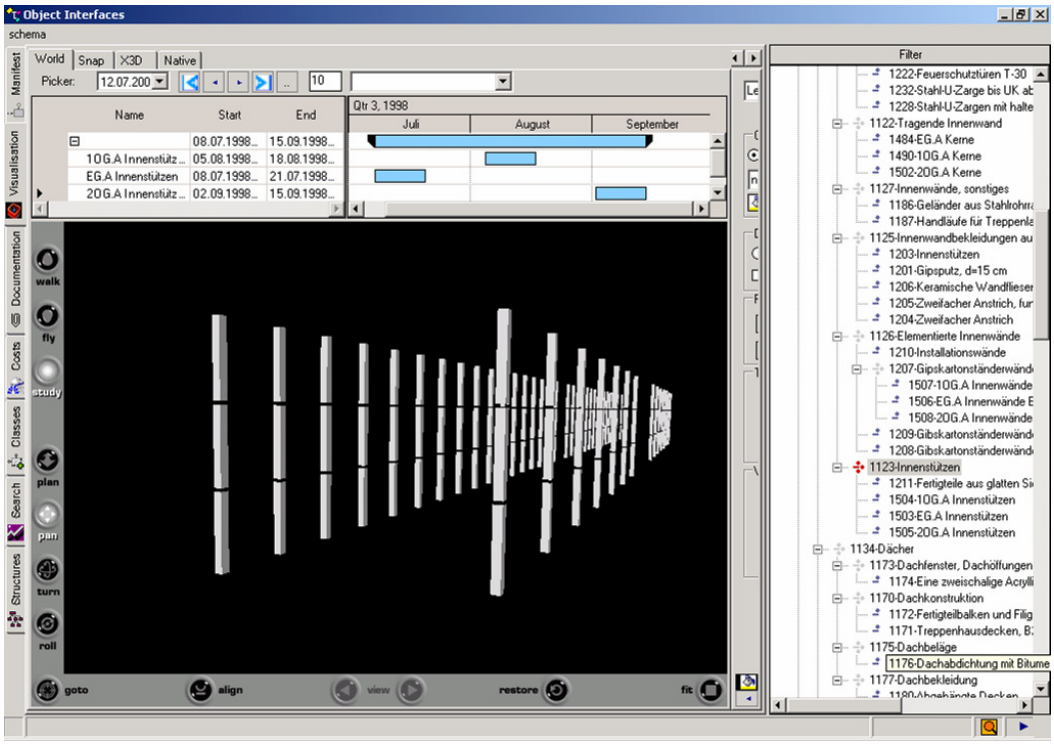


Abb. 119: Geometrische Darstellung der Objekte: Innenstützen.

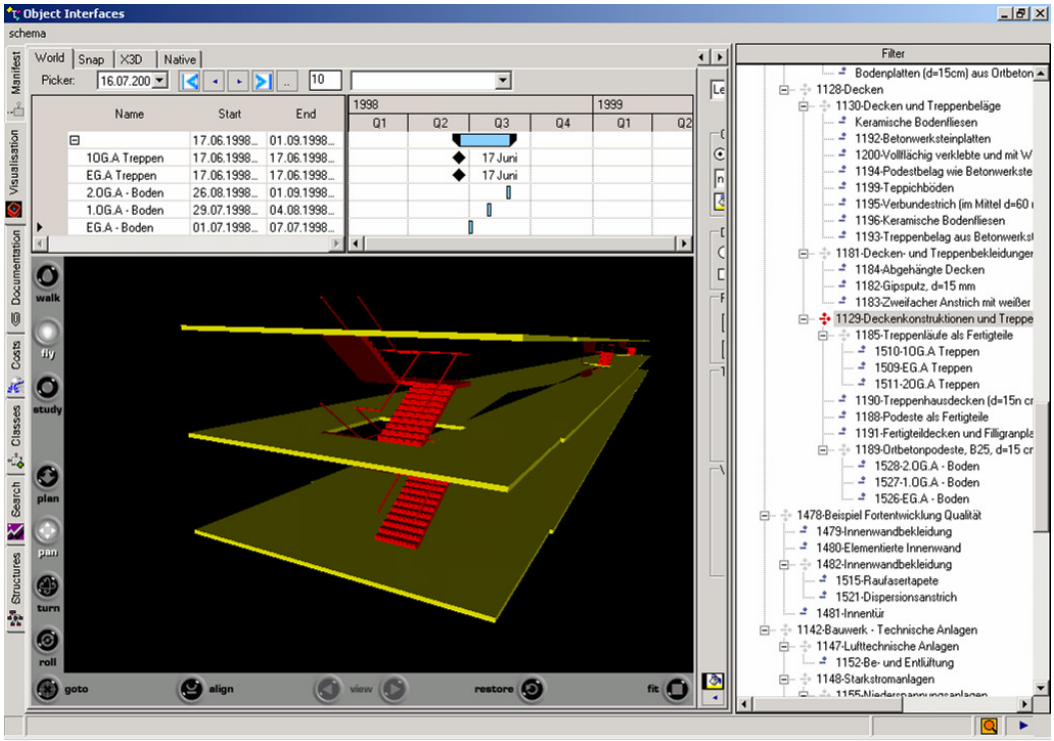


Abb. 120: Geometrische Darstellung der Objekte: Decken und Treppen, Detailansicht.

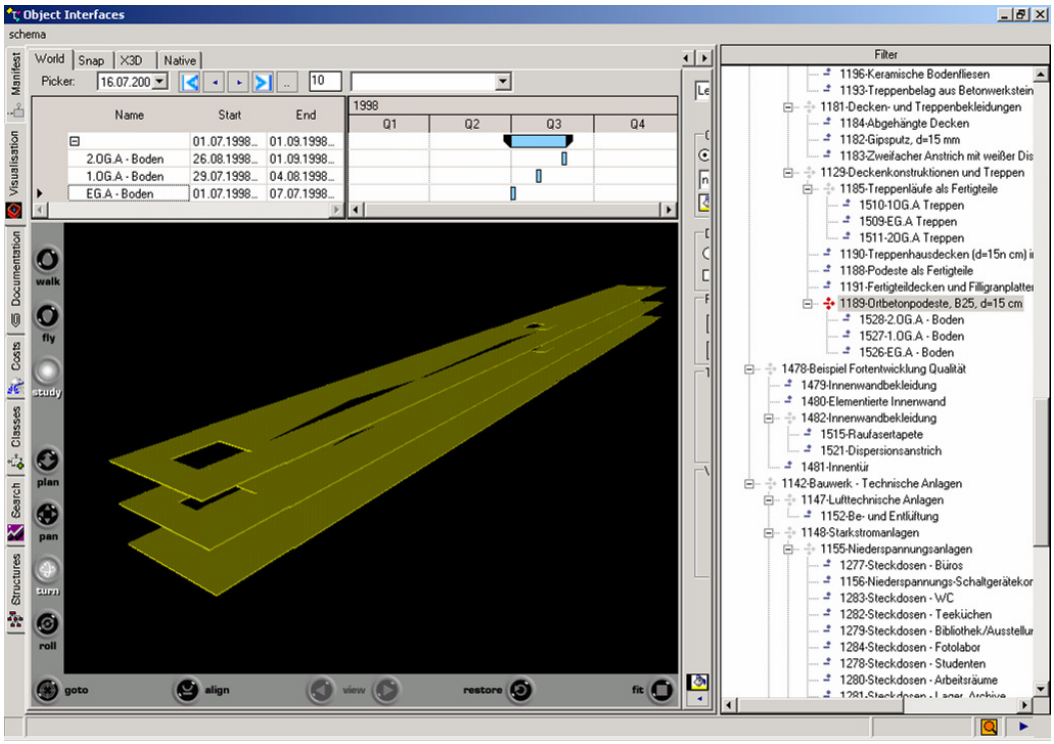


Abb. 121: Geometrische Darstellung der Objekte: Decken.

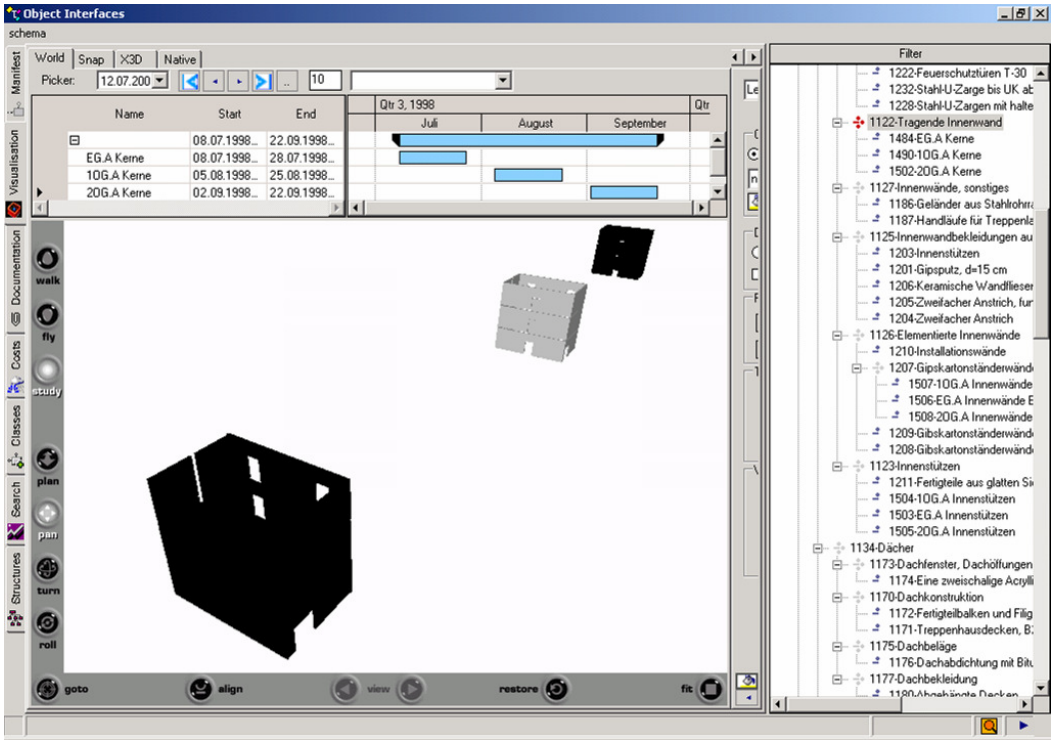


Abb. 122: Geometrische Darstellung der Objekte: Tragwerk, Kerne.

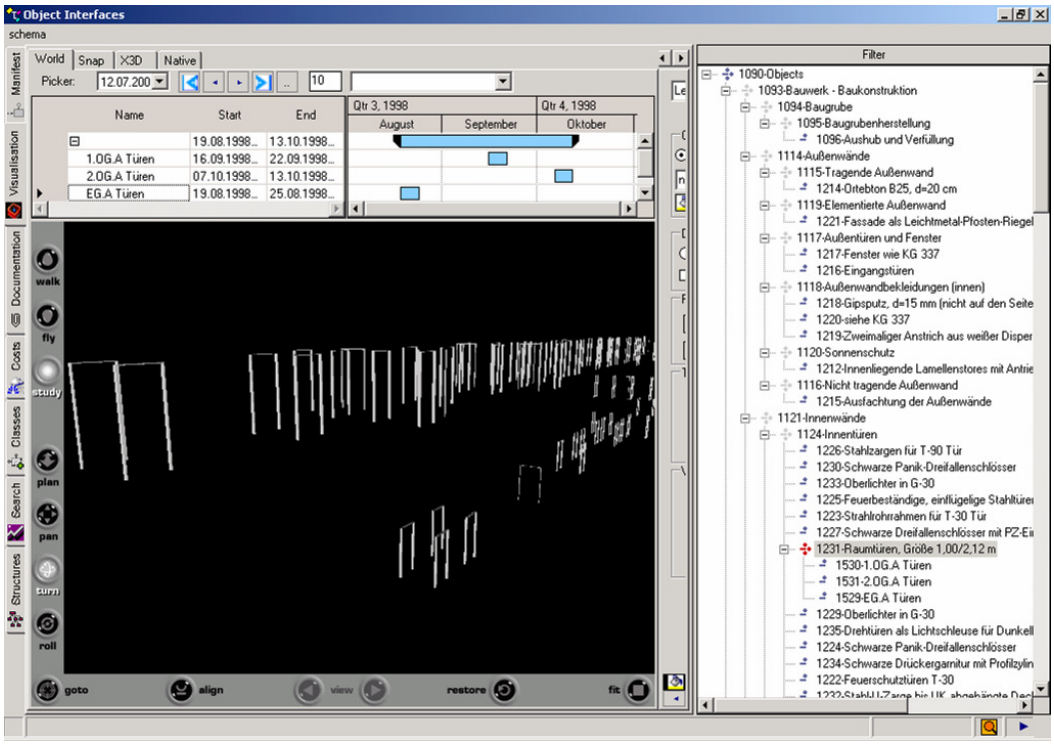


Abb. 123: Geometrische Darstellung der Objekte: Türrahmen.

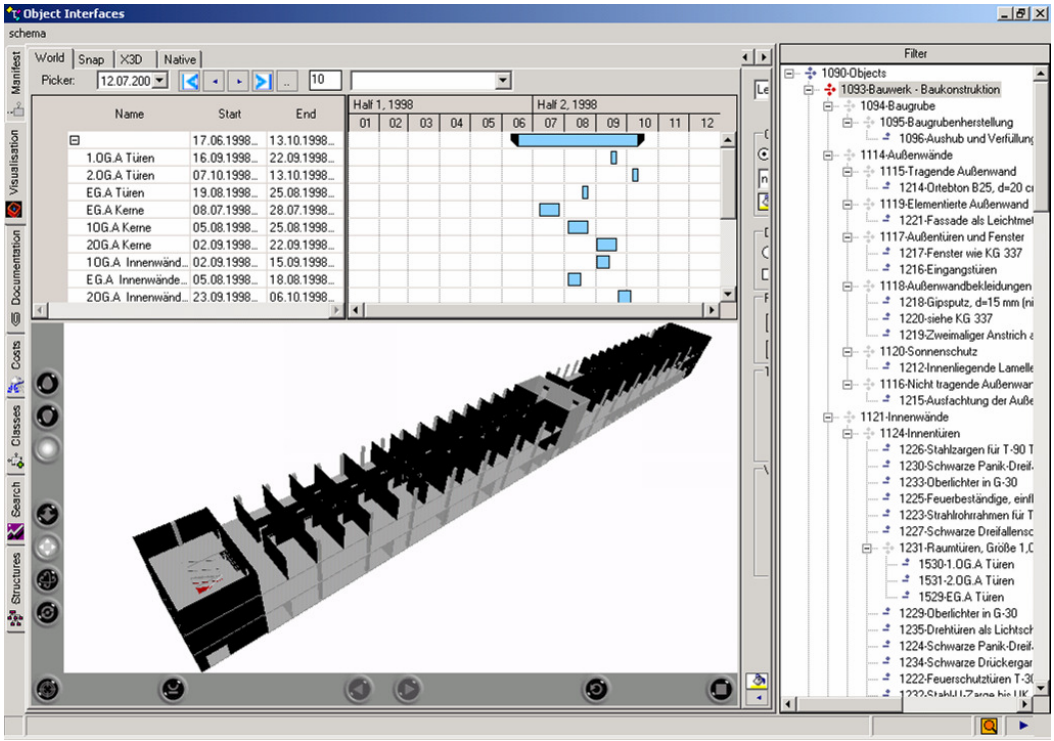


Abb. 124: Geometrische Darstellung der Objekte: Gesamtansicht, Herausstellung der Trennwände.

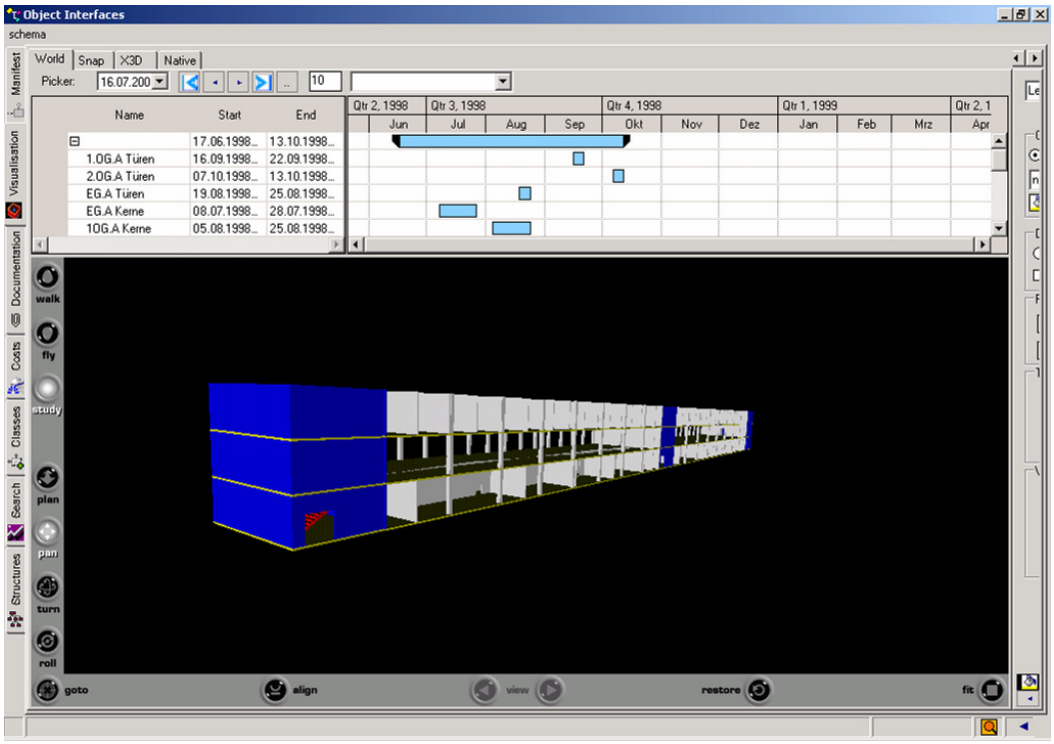


Abb. 125: Geometrische Darstellung der Objekte: Gesamtansicht, Herausstellung der Kerne und Decken.

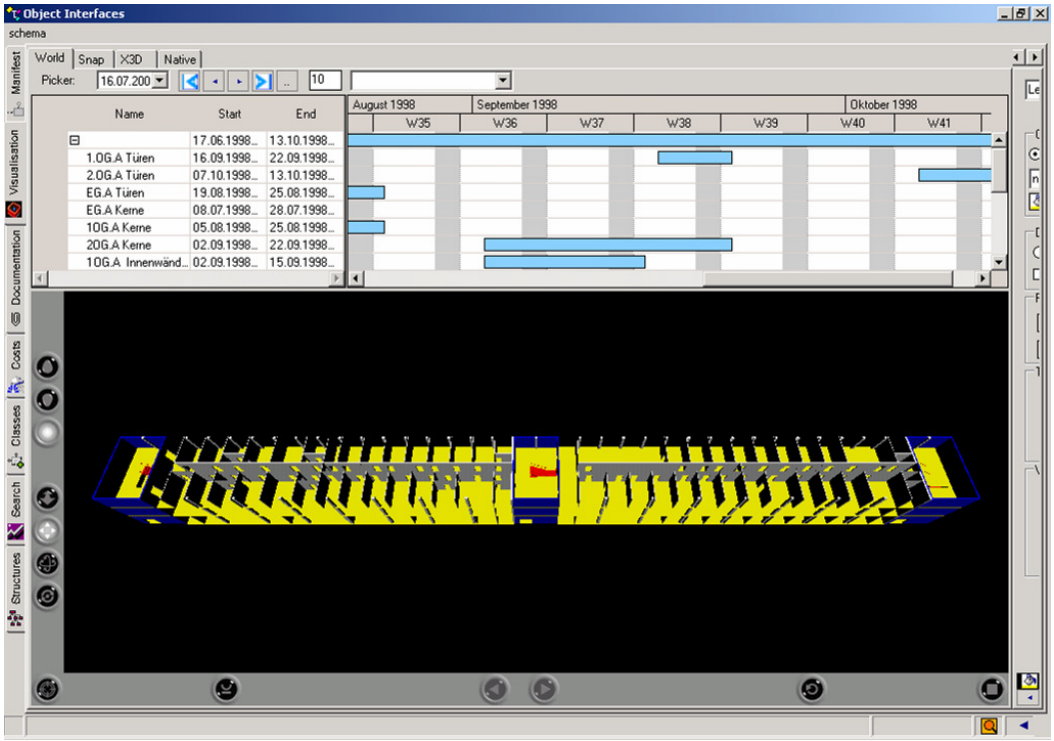


Abb. 126: Geometrische Darstellung der Objekte: Gesamtansicht aus der Perspektive.

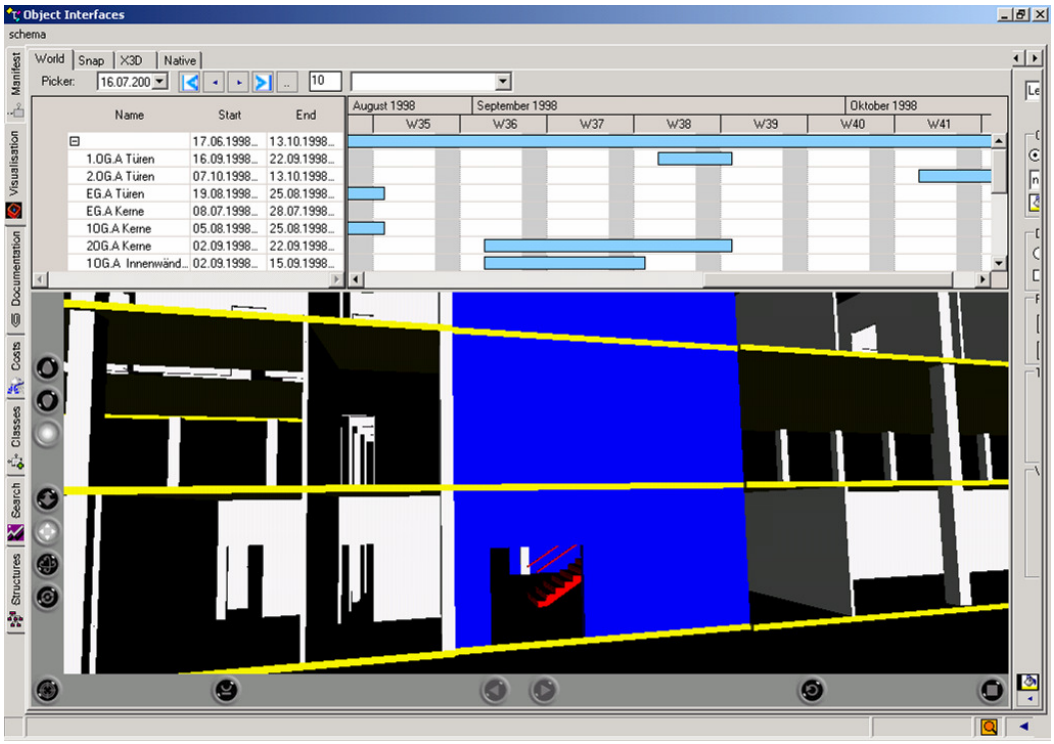


Abb. 127: Geometrische Darstellung der Objekte: Gesamtansicht aus der Perspektive.

3.1.4 Dokumentation des Gebäudes in der Phase 4

In der HOAI Phase 4, Genehmigung der Planungsunterlagen, liegt der Schwerpunkt bei der Überarbeitung von vorhandenen Unterlagen auf der Erlangung einer Baugenehmigung. Diese Unterlagen können einzelnen Objekten, soweit das sinnvoll ist, oder dem gesamten Gebäude zugeordnet werden.

3.1.5 Dokumentation des Gebäudes in der Phase 5

In der Phase 5, Ausführungsphase, wird das Gebäude elementweise gegliedert. Typischerweise sind Ausführungszeichnungen im 2D Format von größerer Bedeutung, jedoch veranschaulicht die elementweise Gliederung die größere Detaillierung dieser Phase (Abb. 128 - 129). Grundsätzlich können durch die erhöhte Atomisierung des Gebäudes bei steigender Phase Objekte gezielter angesprochen und abgefragt werden. Bei der Phasenzuordnung ist darauf zu achten, dass die HOAI bisher keine verbindlichen Aussagen zu 3D geometrischen Modellen macht. Der Rechenaufwand bei Simulationen und Abfragen steigert sich extrem in dieser Phase, da sich die Anzahl der Objekte von der Phase 3 bis Phase 5 von 20 auf 442 erhöht (siehe Abb. 152).

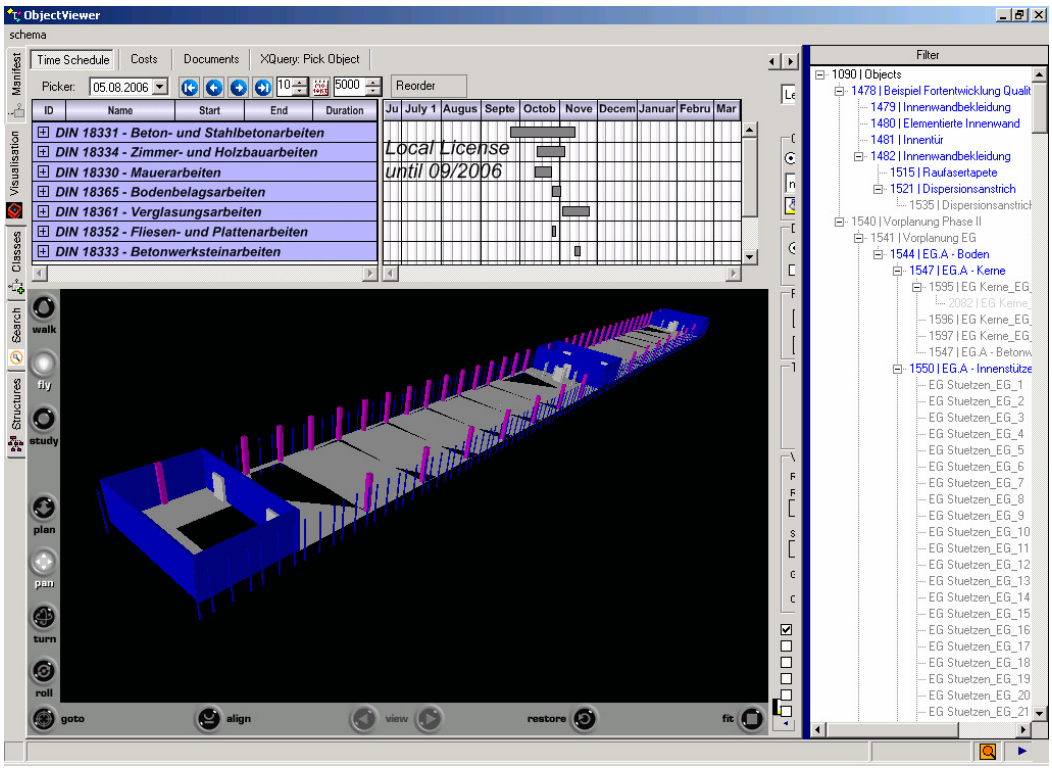


Abb. 128: Ansicht: HOAI Leistungsphase 5, Abschnittseinteilung 1.OG.

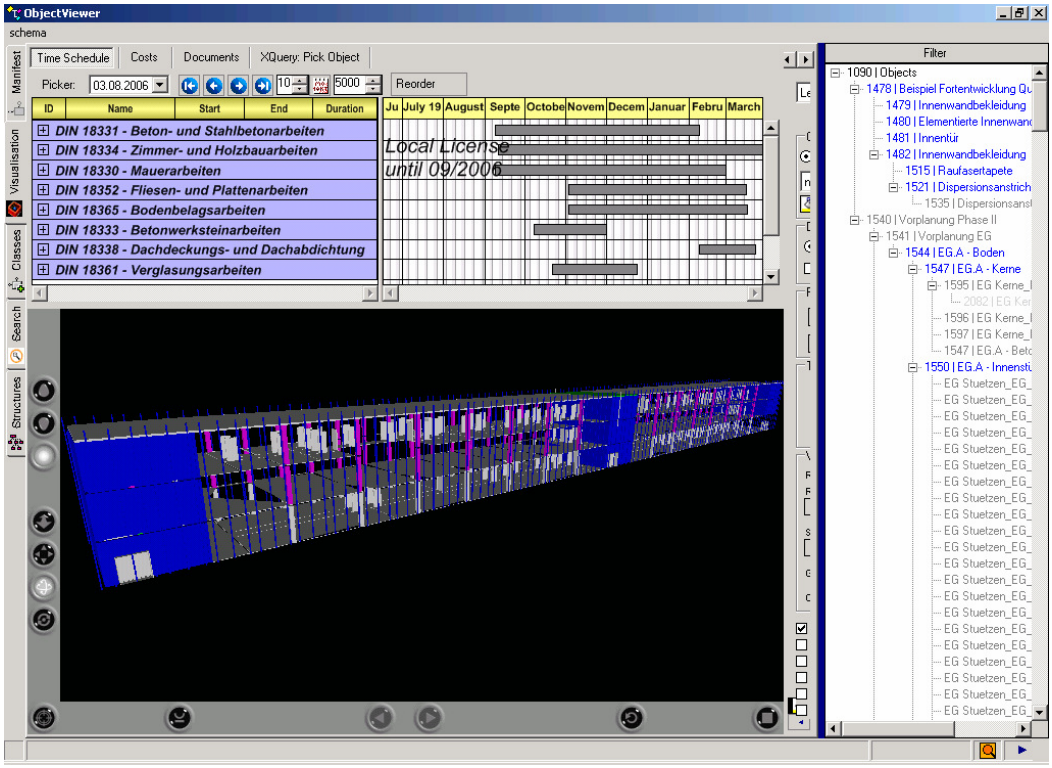


Abb. 129: Ansicht: HOAI Leistungsphase 5, Gesamtansicht. Aufgrund einer Änderung in der NOI-Schnittstelle von der Version Allplan 2005ß auf Allplan 2005.0a können Wandelemente ohne Achsen und Treppen nicht mittels der Schnittstelle ausgelesen werden. Die Wandelemente müssten neu konstruiert werden und Treppen könnten nur rudimentär in der Version Allplan 2006 ausgelesen werden. Die Wandelemente der von der Fakultät Bauwesen bereitgestellten Zeichnungen weisen keine Achsen auf, weshalb diese Elemente zusätzlich zu den Treppen nicht in den Auswertungen der HOAI Phase 5 vorkommen.

3.2 Auswertungen

3.2.1 Interaktionen

Grafische Interaktionen finden statt, indem Objekte mittels Kriterien gefiltert und Eigenschaften der Objekte in verschiedenen grafischen Modulen dargestellt werden:

- Geometrische Darstellung: Cortona VRML Control
- Terminplanung: ILOG Control
- Kosten: DataGrid
- Klassen: Treeview
- Manifest: Primitive Controls und DataGrid

Durch den objektorientierten Ansatz sind solche Darstellungen fachlich übersichtlich. Allerdings gestaltet sich die Steuerung der oben genannten Steuerelemente als programmtechnisch aufwendig, da viele Ereignishandler ausprogrammiert werden müssen und die Rahmenbedingungen der Applikation mittels geeigneten Fehlerbehandlungen eingehalten werden müssen. Falls diese Ereignisbehandlung im Rahmen eines dynamischen Objektmodells in der Applikation *ObjectViewer* auch dynamisch zu gestalten ist, führt dies zu einem erheblichen Aufwand.

Die grafische Darstellung wird dadurch erreicht, dass die *UniqueId* des Objektes abgefragt wird. Dieser Parameter wird dann in einer für Cortona geschriebenen Methode durchgereicht, um eine Referenz auf das entsprechende VRML Objekt im Cortona Steuerelement zu erhalten, wodurch zum Beispiel die Farbe verändert werden kann, um die Selektion des Objekts zu erreichen:

```
Public Sub changeColor(ByRef engine As VRMLAutomation.IEngine, ByVal os As
objStack)
     cubeColor = engine.Nodes("MAT" + os.Name).Fields("diffuseColor")
     cubeColor.let_value(os.cColor)
End Sub
```

Durch eine Referenz auf das geometrische Objekt kann z.B. die Rotation programmatisch gesteuert werden:

```
Public Sub set_rotation(ByRef engine As VRMLAutomation.IEngine, ByVal os As
objStack)
    Dim trn As VRMLAutomation.SFRotation = engine.Nodes.Item("TR" +
      os.Name).Fields("rotation")

    trn.x = CSng(os.rotAxis.rx)
    trn.y = CSng(os.rotAxis.ry)
    trn.z = CSng(os.rotAxis.rz)

    Select Case (os.dir)
        Case 1 'clockwise
            trn.Angle = trn.Angle - os.angle
        Case -1 'anti-clockwise
            trn.Angle = trn.Angle + os.angle
    End Select
End Sub
```

Eine andere Methode der Interaktion mit dem Objekt ist die Auswahl des Objekts im Cortona Steuerelement. Da eigentlich die geometrische Darstellung des Objektes selektiert wird und diese Darstellung in Wirklichkeit eine Vortäuschung der dritten Dimension ist, muss der Code mittels einer Rekursion der verschachtelten VRML Objekte feststellen, welches Objekt gemeint ist:

```
try
{
      vm = new VRMLMethods();
      MFNode chain;
      engine = (IEngine) axCortona1.Engine;
      chain = engine.Pick(e.x, e.y); //pickObj(chain)

      if(chain.Count != 0 )
      {
            if(infoStack == null)
                  infoStack = new geomInfo();

      //restore previous pick if not null
            if(currentPick != null) {
                  m.resetColor(ref infoStack, ref engine); }

      //now handle current pick
            SFColor color = vm.AxCortona1_MouseUp(ref infoStack, os,
                  engine, chain, getHighlight());
            GeometryType.Text = infoStack.GeometryType;
            RootName.Text = infoStack.RootName;
            ShapeName.Text = infoStack.ShapeName;
            RootTypeName.Text = infoStack.RootTypeName;
            currentPick = infoStack.VRMLNode;
            Color backColor = m.getRGBColor(infoStack.color);
            if(backColor == Color.Black)
                  ColorType.ForeColor = Color.White ;
            ColorType.Text = infoStack.color.red.ToString() + " | " +
                  ____infoStack.color.green.ToString() + " | " +
                  infoStack.color.blue.ToString(); //infoStack.ColorType;
            ColorType.BackColor = invertColor(backColor);
            if(e.shift == 1) {
                  activityDlg form = new activityDlg();
                  form.fGuid = os.Id;
                  form.ShowDialog(); }
            setTransparency();
      //move on
            VRMLFieldsCollectionClass routes = vm.moveOn(engine, chain);
      } else {
            ShapeName.Text = string.Empty;
            GeometryType.Text = string.Empty;
      }
} catch(COMException exc) {
      RemoteLogger.RLogger.Error("axCortona1_MouseUpEvent", exc);
      trace(statusBar1, 1, exc);
} catch(Exception exc) {
      RemoteLogger.RLogger.Error("axCortona1_MouseUpEvent", exc);
      trace(statusBar1, 1, exc);
}
```

Mit dieser Referenz können nun z.B. folgende Interaktionen durchgeführt werden:

- Time-Picker, inkl. farblicher Darstellung von gefertigten und nicht-gefertigten Elementen zu einem Zeitpunkt, wird verwendet um den Fortschritt darzustellen.
- Auswahl von einem Elemente durch Selektion eines Vorgangs im NETRONIC Steuerelement.
- Darstellung aller Elemente, deren spezifische Eigenschaften einen gewissen Wert haben.

Die Filterung des NETRONIC Steuerelements auf gewisse Vorgänge erfolgt über die Anweisung:

```
filter = new DefaultRowFilter("(StartTime > #" + startdates[0] + "#) &&
(EndTime < #" +  findates[0] + "#)");
ganttChart1.SetRowFilter(filter);
```

Solche Interaktionen sind sehr herstellerspezifisch und nur für relativ leistungsfähige Rechner geeignet. Hinzu kommt, dass das Medium Web diese Art der Darstellung signifikant einschränkt.

3.2.2 Abfragen

Auswertungen sind aufgrund des objektorientierten Modells grundsätzlich möglich und werden im Umfang nur durch die Fähigkeiten der Abfragesprache und die Rechnerleistung begrenzt. Für ein Berichtswesen bieten sich viele Technologien an. Entsprechend des modernen Ansatzes in der Applikationsentwicklung, Funktionalitäten nach Möglichkeit modular entwickeln zu lassen, wäre der Einsatz eines kommerziellen Reporting Tools wie Crystal Reports oder Intelliview, welches wiederum auf das Objektmodel zugreifen kann, eine angebrachte Lösung. Eine bessere Lösung bietet die Open Source Applikation und Entwicklungsumgebung BIRT. Der Zugang zum Source Code ermöglicht Erweiterungen für spezielle Szenarien.

Abfragen können mit verschiedenen Technologien realisiert werden. Objektorientierte Abfragesprachen wie OQL sind in einigen Reporting Tools implementiert worden. OQL weißt Ähnlichkeiten mit SQL auf, indem grundlegende Anweisungen identisch sind, jedoch findet die Abfrage von verschachtelten Eigenschaften der Objekte in der Punktnotation statt [Yang, 1999. S. 2]:

SELECT b.Material.DIN276 FROM BuildingObjects b WHERE b.ID=2256

Weiter ist zu beachten, dass der SQL 2.0 Standard viele objektorientierte Elemente aufführt. Hybride Datenbanken wie Intersystems Cache ermöglichen die Abfrage von Objekten mit einer ähnlichen Semantik, wobei der Pointer Zugriff mit dem Symbol -> erfolgt:

```
SELECT AwardType->Description, PMS->Market->Description
        FROM sp2005_adv.Project
        where ID=13
```

Die enge Verzahnung von XML und Objekten in der .NET Framework ermöglicht den Einsatz von XPath in der Abfrage nach dem Zustand von verschachtelten Objekteigenschaften [Saxon, 2003. S. 5].

Folgender Code führt eine Abfrage aus, die nach beliebigen Eigenschaften in beliebigen Klassen eine Vergleichsoperation ausführt:

```
string className = (comboBox_XPathClasses.SelectedItem as  i-
tem).Description;
string attribute = (comboBox_XPathProperties.SelectedItem as     i-
tem).Description;
string searchStr = textBox_XPathPropertyValue.Text;
string qry = "//" + className + "[@" + attribute + "='" + searchStr + "']";
ArrayList msg = objXPathnavigator.XPathTest(objnav.basket, qry);
if(msg.Count>0)
{
      iitem it = msg[0] as iitem;
      if(it.Error)
            statusBar1.Panels[0].Text = it.Description + "; " + it.Value;
      else
            dataGrid_XPathResults.DataSource = msg;
}
```

Das grundlegende Prinzip besteht darin, dass Objektgraphen in XML mit der System.Xml.XmlSerialization Namespace umgewandelt und Verschachtelungsebenen mit der XPath Syntax „/*/“ übersprungen werden können, z.B.:

```
/*/String[@oxp:key='TX']
```

Diese einfache Abfrage kann dazu benutzt werden, um die durch Änderungen deaktivierte Objekte zu identifizieren. Diese Objekte könnten z.B. als Basis für eine Nachtragsbetrachtung gelten. Die Abfrage ist deswegen einfach, weil das Objektmodell hier seine Stärken der Kapselung und Gliederung ausspielen kann. Die Abfrage sieht wie folgt aus:

```
//*/.[@oxp:Active='false']
```

Der Vergleich mit typischen Prozessen bei einer kombinierten elektronischen Auswertung von Terminplänen, CAD Daten, Kosten und anderen Quellen zur Aufbereitung von Nachträgen, veranschaulicht die Stärke des objektorientierten Ansatzes in der Projektmodellierung.

Allerdings genügt zu Demonstrationszwecken der Einsatz einer XML-tauglichen Abfragesprache wie XQuery (eine XML-basierte Abfragesprache die auch XPath beinhaltet und weiter unten behandelt wird), die zunehmend zum Repertoire der namhaften Datenbankanbieter gehört, um die Auswertungsmöglichkeiten des Objektmodells zu demonstrieren. XQuery ermöglicht auch JOINS über mehrere XML Ströme auszuführen [Cagel et. al., 2002. S. 124]. XQuery kann auch in Kombination mit XPath verwendet werden, indem z.B. in einer WHERE Klausel einer XQuery Abfrage ein XPath Statement eingesetzt werden kann [Cagel et. al., 2002. S. 142]. Allerdings ist die Implementierung von XQuery in der .NET Framework und den meisten XML Parsern begrenzt. Tatsächlich wurde die Unterstützung dieser Technologie seitens Microsoft in .NET 2.0 aufgegeben und stattdessen in SQL Server integriert. Als Alternative, kann diese Funktionalität mittels Drittanbietern z.B. Saxon integriert werden.

3.2.3 Abfragen des Objektmodells mittels XQuery

XQuery bietet mehrere Sprachelemente, die zusammengefasst als FLWR bekannt sind: FOR, LET, WHERE, RETURN [Cagel et al., 2002. S. 30]. Diese Elemente ermöglichen die Abfrage von XML Dokumenten analog SQL in relationalen Datenbanken. Zusätzlich können XPath Abfragen und Elemente aus XSLT integriert werden, um Strukturabfragen möglich zu machen. Diese Funktionalität ermöglicht weitaus komplexere Abfragen als SQL.

SQL basiert auf relationaler Algebra, und in den SQL Spezifikationen wurden systematisch viele Operatoren dieser Theorie umgesetzt (Selektion, Projektion, Union, Differenz, Kartesisches Produkt) [Cagle et al., 2002. S. 104]. XQuery wurde entwickelt, um in Kombination mit XPath XML Baumstrukturen zu Durchwandern und abzufragen und kann daher auch in heterogenen Netzen mit unterschiedlichen Datenstrukturen verwendet werden. XML Strukturen können über die Knotenpunkte in diesen Strukturen abgefragt werden. Ein weiterer Vorteil der XML-basierten Abfragesprachen ist, dass Ergebnisse den Ursprungsdokumenten weitaus ähnlicher sind. Wegen dieser Ähnlichkeit kann mittels XSLT das Ausgabeformat in der Abfrageapplikation direkt gesteuert werden.

Aktuelle Versionen von Datenbanken wie SQL Server und Oracle haben XQuery Funktionalität integriert, um die zunehmenden Mengen an gespeicherten XML Daten, die als Baumstrukturen gespeichert werden (meistens als BLOBs oder NTEXT), zu analysieren und abzufragen, ein Unterfangen wofür SQL nicht geeignet ist. Dieser Trend hat natürlich auch einen marktwirtschaftlichen Aspekt, da XQuery samt XML Technologien als direkte Konkurrenz von SQL-Datenbanken zu verstehen sind.

Ein weiterer Aspekt ist die Abfragetiefe. SQL wird dadurch begrenzt, dass Datenbanksysteme die Verknüpfung von Tabellen (JOINs) nur bis zu einer gewissen Tiefe unterstützen können. So erlaubt ACCESS eine maximale Anzahl JOIN Verknüpfung die weit unter der effektiven Anzahl von SQL Server oder Oracle liegt. Allerdings ist im Normalfall ein starker Abfall an Leistungsfähigkeit, auch bei professionellen Datenbanken wie SQL Server zu beobachten. XQuery hingegen erlaubt theoretisch unbegrenzte Verknüpfungen in Form von Expression-Chaining. In der Kombination unendlicher paralleler XML Dokumente und Expression Chaining stellt lediglich die Leistung der Abfrageapplikation (XQuery Engine) eine Grenze dar. Wie es bei SQL-basierten Systemen zu erwarten wäre, ist ein Abfall der Leistung bei zunehmender Anzahl von Dokumenten und Expression-Chains zu beobachten. Allerdings ist der Abfall eher linear und nicht so abrupt wie bei SQL Datenbanksystemen (Abb. 152).

Die XQuery Spezifikation sieht eine große Anzahl von Funktionen zur Manipulation von Zeichenfolgen vor, z.B. Datumszahlen und die Ermittlung von Positionsinformationen von Knotenpunkten in XML Strukturen. Im Vergleich zu SQL, - Tabellen (Tuples) dienen als Informationsgrundlage - liegt die Stärke von XQuery in der Verknüpfung von relationalen Informationen und Strukturinformationen.

Zur Ausführung einer XQuery Abfrage wird zuerst eine Sammlung von XML Dokumenten, die die Objekte und Strukturbäume repräsentieren, aufgebaut. Diese werden dann in einem Cache, der sog. `XQueryNavigatorCollection` aufgenommen. Auf diesem Cache wird dann die Abfrage mit dem `XqueryNavigator` ausgeführt, dessen Ergebnis als Text oder XML ausgegeben werden kann.

Analog SQL JOIN Abfragen können mehrere XML Dokumente auch im Verbund abgefragt werden. Im Objektmodell kann eine solche Abfrage eingesetzt werden, um die Integrität der Objekte im Bezug zu den Strukturbäumen wie DIN 276 zu prüfen. Typischerweise werden Abfragen zu Kombinationen von Strukturbäumen gemacht, z.B.

Abfrage: Alle Objekte die der Leistungsphase 3 und dem Projekt Fakultät zugeordnet sind

```
<results>{
FOR $objects in document('objects')/objectgraph//constructionobject
FOR $f_2650c076-ea0d-4680-8a9a-92ff3afad543 IN
      document('filters')/filters/iCollections
      [@filterId='2650c076-ea0d-4680-8a9a-92ff3afad543']
FOR $f_b591679d-8ac2-417b-bc56-7d6ef11dc265 IN
      document('filters')/filters/iCollections
      [@filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265']
WHERE contains($objects/Nodes/anyType/nodeId,
      '712cae69-76e9-4e78-a223-a1cd6d46d414')
      AND contains($objects/Nodes/anyType/nodeId,
      'cde6e8fd-d20c-418c-8052-dd6a67e49365')
RETURN
IF( $f_b591679d-8ac2-417b-bc56-7d6ef11dc265/
      @filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265'
AND $f_2650c076-ea0d-4680-8a9a-92ff3afad543/
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543'
)
THEN
<objects>
{$objects/UniqueId}{$objects/Index}{$objects/Description}
</objects>
ELSE
<objects></objects>
}</results>
```

Die geometrische Darstellung dieser Abfrage wird in Abbildung 130 dargestellt. Weitere Abfragen und Ergebnisse folgen in Abbildungen 130 – 134.

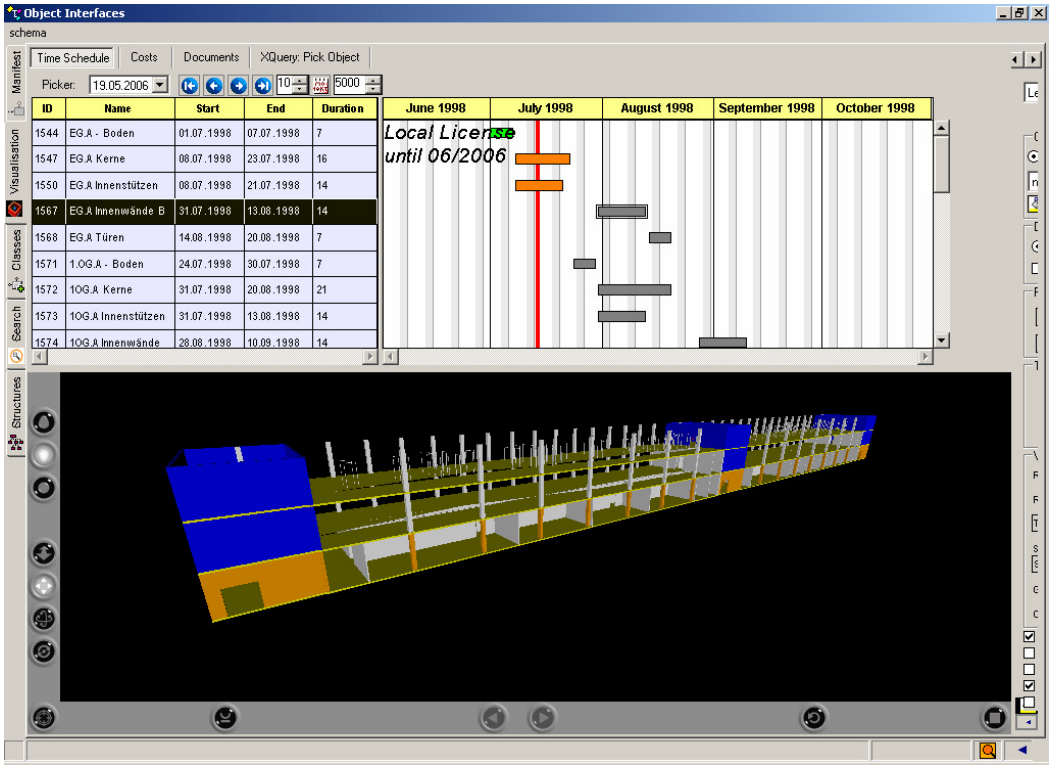


Abb. 130: Abfrage: Alle Objekte des Projekts „Fakultät“, die der HOAI Leistungsphase 3 zugeordnet wurden.

Weil die Objekte durch die *ISerializable* Schnittstelle beliebig in XML transformiert werden können und dasselbe für all Strukturbäume gilt, bietet XQuery die Möglichkeit, alle erdenklichen Abfragen dieser XML Strukturen zu ermöglichen. In Kombination mit XPath und einer Reihe von Funktionen, können einzelne Knotenpunkte abgefragt werden, auch in Kombination mit Strukturinformationen (z.B. alle Kinder in der Baumstruktur Abschnittseinteilung, die sich auf der Ebene 1 befinden). Somit sind die strukturellen Grenzen von SQL überwunden und außerdem schränkt nicht die Abfragesprache, sondern allein das Objektmodell das Ergebnis ein. Obgleich XQuery Applikationen (Query Engines) erst in einer ersten Generation zur Verfügung stehen, ist dies eine entscheidende Verbesserung gegenüber herkömmlichen Produktmodellen, die auf SQL basieren.

```
<results>{
LET $subtree := document('filters')/filters/iCollections[
      @filterId='28aa818f-26cc-43d5-b10e-231729c09322']//INode[
      @nodeId='1f81d4a5-dcf9-4212-8a20-7c8a53b51c49']
FOR $objects in document('objects')/objectgraph//constructionobject
FOR $f_2650c076-ea0d-4680-8a9a-92ff3afad543
IN document('filters')/filters/iCollections[
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543']
FOR $f_b591679d-8ac2-417b-bc56-7d6ef11dc265
IN document('filters')/filters/iCollections[
      @filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265']
FOR $f_28aa818f-26cc-43d5-b10e-231729c09322
IN document('filters')/filters/iCollections[
      @filterId='28aa818f-26cc-43d5-b10e-231729c09322']
FOR $f_10e6d318-ded9-413e-8241-3ec29e74dee8
IN document('filters')/filters/iCollections[
      @filterId='10e6d318-ded9-413e-8241-3ec29e74dee8']

WHERE contains($objects/Nodes/anyType/nodeId, '712cae69-76e9-4e78-a223-
      a1cd6d46d414')
AND contains($objects/Nodes/anyType/nodeId, '6d91abed-d811-4448-b3cc-
      e464f1870944')
AND contains($objects/Nodes/anyType/nodeId, '4cbbf401-0c5a-4e8f-b85f-
      368323d35468')
AND count($objects/Nodes//anyType[filterId='28aa818f-26cc-43d5-b10e-
      231729c09322'
AND contains($subtree, nodeId)])>0

RETURN
IF( $f_28aa818f-26cc-43d5-b10e-231729c09322/
      @filterId='28aa818f-26cc-43d5-b10e-231729c09322'
AND $f_b591679d-8ac2-417b-bc56-7d6ef11dc265/
      @filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265'
AND $f_2650c076-ea0d-4680-8a9a-92ff3afad543/
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543'
AND $f_10e6d318-ded9-413e-8241-3ec29e74dee8/
      @filterId='10e6d318-ded9-413e-8241-3ec29e74dee8')

THEN
<objects>
{$objects/UniqueId}{$objects/Index}{$objects/Description}
      {$objects/Nodes//anyType[filterId='28aa818f-26cc-43d5-b10e-
            231729c09322'
AND contains($subtree, nodeId)]}
</objects>
ELSE <objects></objects>
}</results>
```

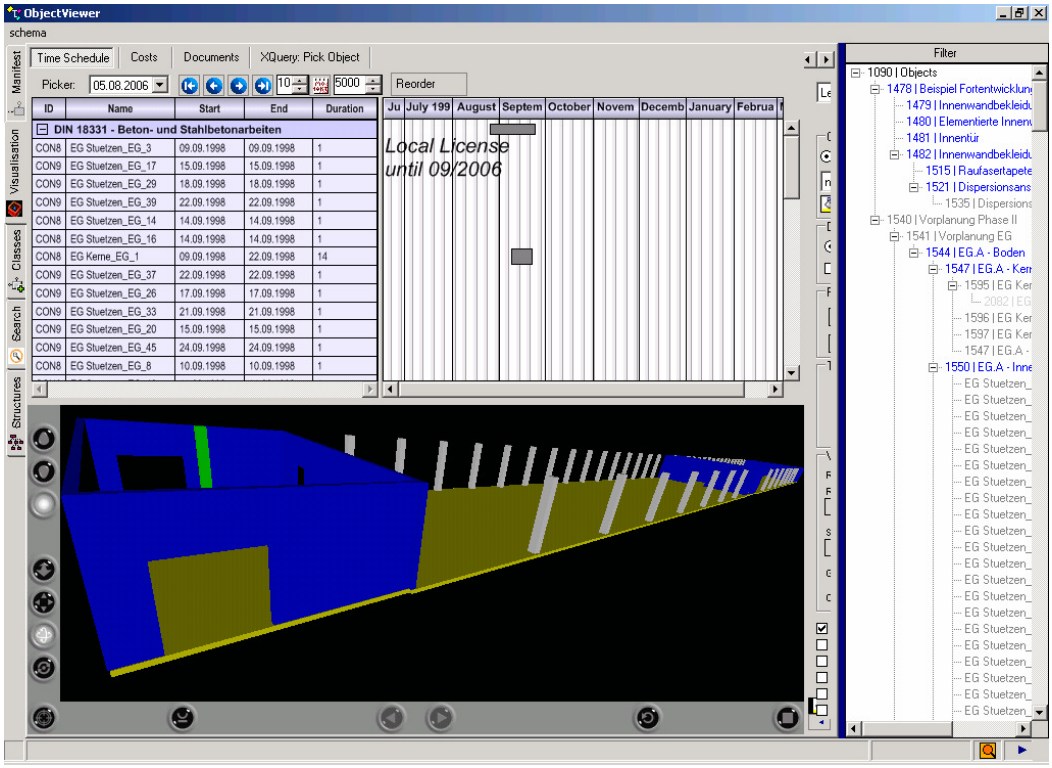


Abb. 131: Abfrage: Objekte des Projekts „Fakultät“ in der HOAI Leistungsphase 5 mit Elementen (auch untergeordnet) des Ordnungssystems Ebene EG und der Zuordnung Qualität VOB Teil C = {DIN 18331 - Beton- und Stahlbetonarbeiten [4cbbf401-0c5a-4e8f-b85f-368323d35468]}

```
<results>{
FOR $objects in document('objects')/objectgraph//constructionobject
FOR $f_2650c076-ea0d-4680-8a9a-92ff3afad543 IN
document('filters')/filters/iCollections[
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543']
FOR $f_b591679d-8ac2-417b-bc56-7d6ef11dc265
IN document('filters')/filters/iCollections[
      @filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265']
FOR $f_28aa818f-26cc-43d5-b10e-231729c09322
IN document('filters')/filters/iCollections[
      @filterId='28aa818f-26cc-43d5-b10e-231729c09322']
FOR $f_10e6d318-ded9-413e-8241-3ec29e74dee8
IN document('filters')/filters/iCollections[
      @filterId='10e6d318-ded9-413e-8241-3ec29e74dee8']

WHERE contains($objects/Nodes/anyType/nodeId,
      '712cae69-76e9-4e78-a223-a1cd6d46d414')
AND contains($objects/Nodes/anyType/nodeId,
      'cde6e8fd-d20c-418c-8052-dd6a67e49365')
AND contains($objects/Nodes/anyType/nodeId,
      '1508758b-a761-4e7f-9c58-b2039e575d40')
AND contains($objects/Nodes/anyType/nodeId,
      '4cbbf401-0c5a-4e8f-b85f-368323d35468')

RETURN
IF( $f_28aa818f-26cc-43d5-b10e-231729c09322/
      @filterId='28aa818f-26cc-43d5-b10e-231729c09322'
AND $f_10e6d318-ded9-413e-8241-3ec29e74dee8/
      @filterId='10e6d318-ded9-413e-8241-3ec29e74dee8'
AND $f_b591679d-8ac2-417b-bc56-7d6ef11dc265/
      @filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265'
AND $f_2650c076-ea0d-4680-8a9a-92ff3afad543/
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543'
 )
THEN
<objects>
{$objects/UniqueId}{$objects/Index}{$objects/Description}</objects>
ELSE
<objects></objects>
}</results>
```

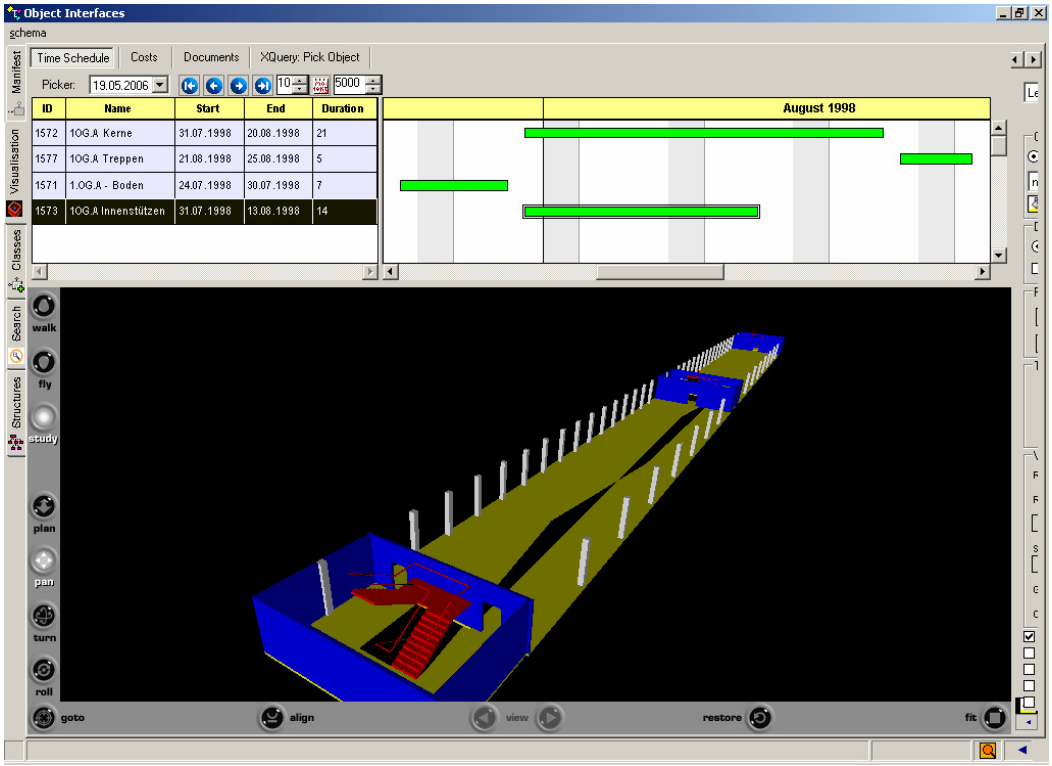


Abb. 132: Abfrage: Objekte des Projekts „Fakultät“, die der HOAI Leistungsphase 3, dem Ordnungssystem Ebene 2 und der Qualität VOB Teil C = {DIN 18331 - Beton- und Stahlbetonarbeiten [4cbbf401-0c5a-4e8f-b85f-368323d35468]} entsprechen.

```
<results>{
FOR $objects in document('objects')/objectgraph//constructionobject
FOR $f_2650c076-ea0d-4680-8a9a-92ff3afad543 IN docu-
ment('filters')/filters/iCollections[
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543']
FOR $f_b591679d-8ac2-417b-bc56-7d6ef11dc265
IN document('filters')/filters/iCollections[
      @filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265']
FOR $f_10e6d318-ded9-413e-8241-3ec29e74dee8
IN document('filters')/filters/iCollections[
      @filterId='10e6d318-ded9-413e-8241-3ec29e74dee8']
FOR $f_336afe63-56cd-4537-b737-7ee3b5154993
IN document('filters')/filters/iCollections[
      @filterId='336afe63-56cd-4537-b737-7ee3b5154993']

WHERE contains($objects/Nodes/anyType/nodeId, '712cae69-76e9-4e78-a223-
      a1cd6d46d414')
AND contains($objects/Nodes/anyType/nodeId, '6d91abed-d811-4448-b3cc-
      e464f1870944')
AND contains($objects/Nodes/anyType/nodeId, '46276258-7470-4a70-ab5d-
      97cce0256232')
AND contains($objects/Nodes/anyType/nodeId, '890b3438-ed69-49a6-b8fe-
      ae4a8f4576b8')

RETURN
IF( $f_10e6d318-ded9-413e-8241-3ec29e74dee8/
      @filterId='10e6d318-ded9-413e-8241-3ec29e74dee8'
AND $f_b591679d-8ac2-417b-bc56-7d6ef11dc265/
      @filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265'
AND $f_2650c076-ea0d-4680-8a9a-92ff3afad543/
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543')
THEN
<objects>
      {$objects/UniqueId}{$objects/Index}{$objects/Description}</objects>
ELSE
<objects></objects>
}</results>
```

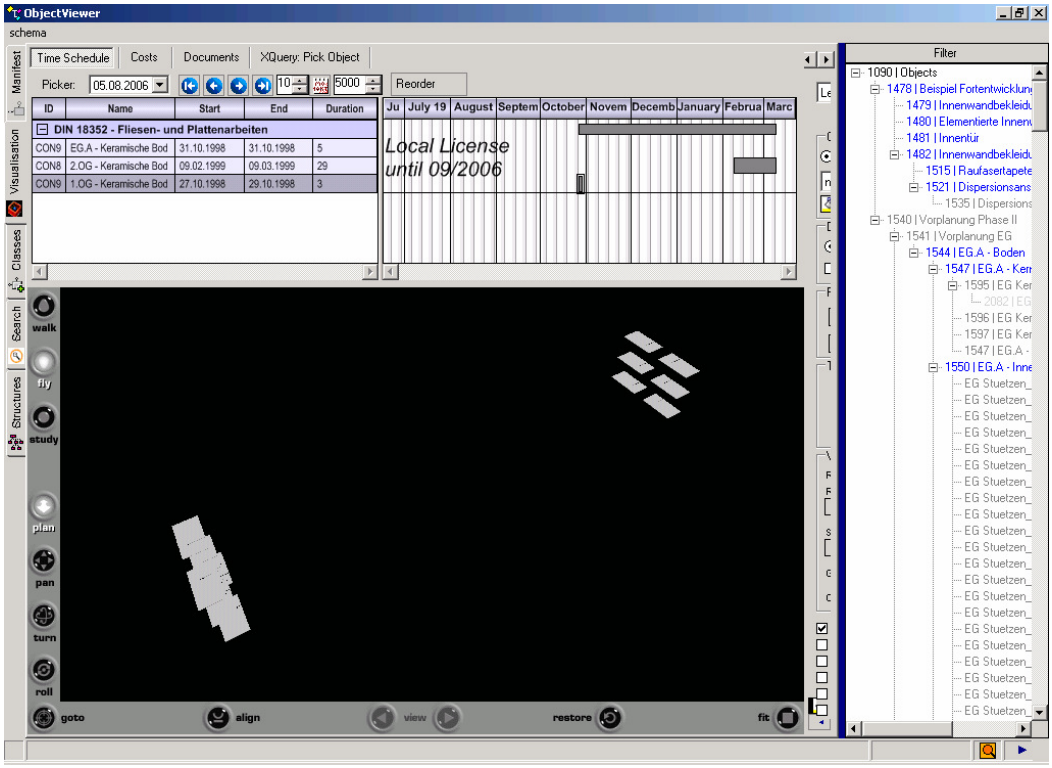


Abb. 133: Abfrage: Alle Objekte des Projekts „Fakultät“, die der HOAI Leistungsphase 5 und der Qualität VOB Teil C = {DIN 18352 - Fliesen- und Plattenarbeiten [46276258-7470-4a70-ab5d-97cce0256232]} und dem Knotenpunkt DIN 276 = {352 - Deckenbeläge [890b3438-ed69-49a6-b8fe-ae4a8f4576b8]} entsprechen.

```
<results>{
FOR $objects in document('objects')/objectgraph//constructionobject
FOR $f_2650c076-ea0d-4680-8a9a-92ff3afad543
IN document('filters')/filters/iCollections[
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543']
FOR $f_b591679d-8ac2-417b-bc56-7d6ef11dc265
IN document('filters')/filters/iCollections[
      @filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265']

WHERE contains($objects/RefNode/nArray/className,
      'ProtoClass.derivedconstructionobject')

RETURN
IF( $f_b591679d-8ac2-417b-bc56-7d6ef11dc265/
      @filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265'
AND $f_2650c076-ea0d-4680-8a9a-92ff3afad543/
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543')

THEN
<objects>
{$objects/UniqueId}{$objects/Index}
      {$objects/Description}{$objects/RefNode/nArray/className}
</objects>

ELSE
<objects></objects>}
</results>
```

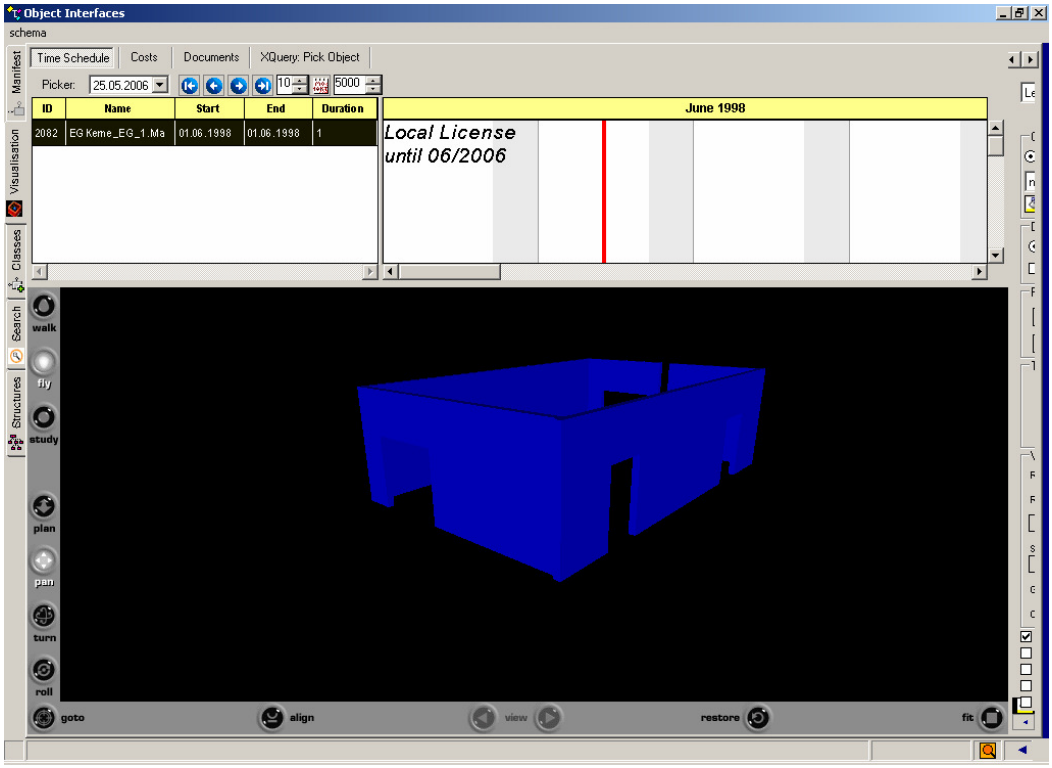


Abb. 134: Abfrage: Alle Objekte des Projekts „Fakultät“, die der HOAI Leistungsphase 5 entsprechen und Mängel aufweisen, d.h. vom Typ ProtoClass. derivedconstructionobject.

In den hier aufgeführten geometrischen Ansichten wurde die Option „Replace“ verwendet. Damit wird dem Cortona Steuerelement der Befehl erteilt, die aktuelle Ansicht zu löschen und durch die im Filter befindlichen VRML Nodes zu ersetzen. Eine andere Option besteht darin, die vorhandene Ansicht stehen zu lassen und die im Filter befindlichen VRML Nodes mit einer Farbe kenntlich zu machen. Eine solche Lösung würde analog dem im nächsten Kapitel beschriebenen Highlighting erfolgen.

Das Objektmodell ermöglicht aufgrund der Tatsache, dass alle Objekte in Kombination mit den Filtern abgefragt werden, die Aufstellung von unterschiedlich gegliederten Baubeschreibungen. Bei jeder Baubeschreibung handelt es sich um eine XQuery Abfrage, und bedingt durch die Tatsache, dass alle Objekte in Kombination abgefragt werden können, steht den Variationen der Baubeschreibungen nichts im Wege.

In Kapitel Teil B.IV.1 wurde die Hierarchie des Klonens der Objekte bei signifikanten Änderungen aufgeführt. Das Objekt in dem Ergebnis der Abfrage Abbildung 134 weist die in Abbildung 135 dargestellte Klon-Hierarchie auf.

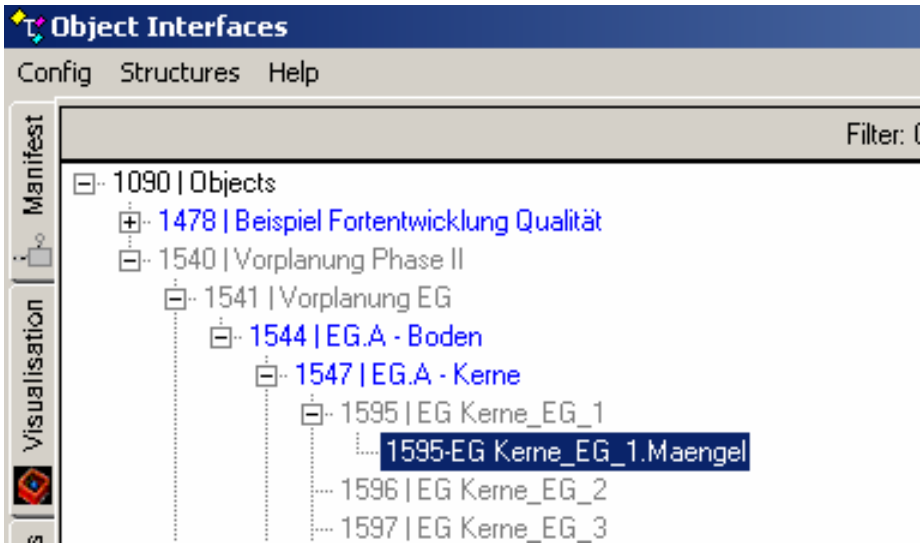


Abb. 135: Klon-Herarchie eines Objekts vom Typ `ProtoClass.derivedconstructionobject`

```
            <DBId>1090</DBId>
            <Description>Objects</Description>
            <Order>0</Order>
          </Parent>
          <ObjectId>0</ObjectId>
          <DBId>1540</DBId>
          <Description>Vorplanung Phase II</Descrip
          <Order>0</Order>
        </Parent>
        <ObjectId>0</ObjectId>
        <DBId>1541</DBId>
        <Description>Vorplanung EG</Description>
        <Order>0</Order>
      </Parent>
      <ObjectId>0</ObjectId>
      <DBId>1544</DBId>
      <Description>EG.A - Boden</Description>
      <Order>0</Order>
    </Parent>
    <ObjectId>0</ObjectId>
    <DBId>1547</DBId>
    <Description>EG.A - Kerne</Description>
    <Order>0</Order>
  </Parent>
  <ObjectId>0</ObjectId>
  <DBId>1595</DBId>
  <Description>EG Kerne_EG_1</Description>
  <Order>0</Order>
```

Abb. 136: Klon-Information ist auch in der Parent Eigenschaft als Hierarchie oder in der Nodes Sammlung eines Objekts als Sammlung erhältlich.

Diese Information kann auch aus der Nodes Sammlung der Objekte ausgelesen werden (Abb. 136). Dort werden alle aktuellen *INode* Knoten aus dem Objects Strukturbaum eingetragen. Diese bleiben bei einem Klon-Vorgang erhalten und verweisen daher auf das Elternobjekt. Falls ein Klon-Vorgang durch eine im Objekt verankerte Regel ausgelöst wurde, wird diese Information in dem `PropertyChangeEventArgs` des Ereignisbehandlers aufbewahrt. Bei manuell erfolgten Klon-Vorgängen würden solche Einträge typischerweise über eine Eingabe qualifiziert.

Langen und Schiffers [Langen und Schiffers, 2005. S. 214] beschreiben diese relativ leichte Neugliederung je nach Ausgangspunkt. Die in Abbildung 137 aufgeführte gebäudeorientierte Baubeschreibung (nach Bauelementen auf der Basis der DIN 276) entspricht der elementorientierten Tabelle in Abbildung 105 (außer den Angaben der Qualitätsgruppe, die ein zusätzlicher Filter darstellen würde und Info, die eine zusätzliche Eigenschaft eines Objekts darstellen würde). Sie lehnt sich an die gebäudeorientierte Aufgliederung, nach Langen und Schiffers [Langen und Schiffers, 2005. S. 214, Abbildung 30], wird mit einer XQuery Abfrage aus dem Objektmodell generiert und mit einem XSLT (Anhang E - elementorientierte_baubeschreibung.xslt) in das gewohnte Tabellenformat transformiert:

```
<results>{
FOR $objects in document('objects')/objectgraph//constructionobject
FOR $f_2650c076-ea0d-4680-8a9a-92ff3afad543
IN document('filters')/filters/iCollections[
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543']
FOR $f_b591679d-8ac2-417b-bc56-7d6ef11dc265
IN document('filters')/filters/iCollections[
      @filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265']
FOR $f_336afe63-56cd-4537-b737-7ee3b5154993
IN document('filters')/filters/iCollections[
      @filterId='336afe63-56cd-4537-b737-7ee3b5154993']

WHERE contains($objects/Nodes/anyType/nodeId, '712cae69-76e9-4e78-a223-
      a1cd6d46d414')
AND contains($objects/Nodes/anyType/nodeId, 'cde6e8fd-d20c-418c-8052-
      dd6a67e49365')

RETURN
{for $obj in $objects/Nodes/*/INode[
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543']]
IF( $f_b591679d-8ac2-417b-bc56-7d6ef11dc265/
      @filterId='b591679d-8ac2-417b-bc56-7d6ef11dc265'
AND $f_2650c076-ea0d-4680-8a9a-92ff3afad543/
      @filterId='2650c076-ea0d-4680-8a9a-92ff3afad543')

THEN <objects>{$objects/UniqueId}{$objects/Index}
{$objects/Description}</objects>

ELSE <objects></objects>
}</results>
```

```
- <results>
  - <objects>
      <_element>ProtoClass.constructionobject.337</_element>
    - <_kostengruppe>
        <description>341 - Tragende Innenwände</description>
      </_kostengruppe>
      <_beschreibung>1OG.A Kerne</_beschreibung>
      <_planungsstand>LP3-Entwurfsplanung (System- und Integrationsplanung)</_planungsstand>
    - <_din>
        <description>DIN 18331 - Beton- und Stahlbetonarbeiten</description>
      </_din>
    - <_ortsangabe>
        <description>Geschoß 2</description>
      </_ortsangabe>
    </objects>
  - <objects>
      <_element>ProtoClass.constructionobject.395</_element>
    - <_kostengruppe>
        <description>344 - Innentüren und Fenster</description>
      </_kostengruppe>
      <_beschreibung>1.OG.A Türen</_beschreibung>
      <_planungsstand>LP3-Entwurfsplanung (System- und Integrationsplanung)</_planungsstand>
    - <_din>
        <description>DIN 18334 - Zimmer- und Holzbauarbeiten</description>
      </_din>
    - <_ortsangabe>
        <description>Geschoß 2</description>
      </_ortsangabe>
    </objects>
```


	Element	Kostengruppe DIN 276	Beschreibung der Elemente	Planungs-stand	DIN
■	337	341	1OG.A Kerne	LP3	DIN 18331
■	395	344	1.OG.A Türen	LP3	DIN 18334
■	352	346 ; 342	EG.A Innenwände Büro	LP3	DIN 18331 ; DIN 18330
■	363	359	1OG.A Treppen	LP3	DIN 18331
■	350	343	2.OG - Innenstützen	LP3	DIN 18331
■	376	361 ; 351	1.OG.A - Boden	LP3	DIN 18331
■	353	346 ; 342	1OG.A Innenwände Büro	LP3	DIN 18350 ; DIN 18330
■	349	343	1OG.A Innenstützen	LP3	DIN 18331
■	394	344	EG.A Türen	LP3	DIN 18334
■	396	344	2.OG.A Türen	LP3	DIN 18334
■	336	341	EG.A Kerne	LP3	DIN 18331
■	375	351	EG.A - Boden	LP3	DIN 18331
■	348	343	EG.A Innenstützen	LP3	DIN 18331
■	354	346 ; 342	2.OG - Innenstützen	LP3	DIN 18350 ; DIN 18330
■	986	359	2.OG - Treppen	LP3	DIN 18331
■	342	341	2.OG - Kerne	LP3	DIN 18331
■	377	351	2.OG - Boden	LP3	DIN 18331

Abb. 137: XQuery Abfrage des Objektmodells analog gebäudeorientierter Baubeschreibung nach Bauelementen auf der Basis der DIN 276 [Langen und Schiffers, 2005. S. 214].

Analog Langen und Schiffers [Langen und Schiffers, 2005. S. 214] kann mit einer XQuery Abfrage demonstriert werden, wie „[eine] gebäudeorientierte Aufgliederung problemlos und ohne viel Aufwand – siehe dort die vorletzte Spalte – in eine ausführungsorientierte Baubeschreibung Abbildung 32 (S. 216) … überführt werden kann“. Diese Auswertung wird in der Abbildung 138 aufgeführt. Die XQuery Abfrage in Abbildung 137 beinhaltet bereits alle Informationen für die Darstellung in Abbildung 138. Daher braucht das XQuery in Abbildung 137 nicht geändert zu werden. Es bedarf lediglich einer geänderten XSLT (Anhang E - ausfuehrungsorientierte_ baubeschreibung.xslt) zur Erlangung der gewünschten tabellarischen Darstellung. Diese Situation demonstriert die Nützlichkeit des Verbunds von XQuery, XSLT und XML.

DIN 18...	Leit-Position	Beschreibung der Elemente	Kostengruppe DIN 276	Ortsangabe
330	**DIN 18330 - Mauerarbeiten**			
	1	[352] EG.A Innenwände Büro	346 ; 342	Geschoß 1
	2	[353] 1OG.A Innenwände Büro	346 ; 342	Geschoß 2
	3	[354] 2.OG - Innenstützen	346 ; 342	Geschoß 3
331	**DIN 18331 - Beton- und Stahlbetonarbeiten**			
	1	[337] 1OG.A Kerne	341	Geschoß 2
	2	[352] EG.A Innenwände Büro	346 ; 342	Geschoß 1
	3	[363] 1OG.A Treppen	359	Geschoß 2
	4	[350] 2.OG - Innenstützen	343	Geschoß 3
	5	[376] 1.OG.A - Boden	361 ; 351	Geschoß 2
	6	[349] 1OG.A Innenstützen	343	Geschoß 2
	7	[336] EG.A Kerne	341	Geschoß 1
	8	[375] EG.A - Boden	351	Geschoß 1
	9	[348] EG.A Innenstützen	343	Geschoß 1
	10	[986] 2.OG - Treppen	359	Geschoß 3
	11	[342] 2.OG - Kerne	341	Geschoß 3
	12	[377] 2.OG - Boden	351	Geschoß 3
334	**DIN 18334 - Zimmer- und Holzbauarbeiten**			
	1	[395] 1.OG.A Türen	344	Geschoß 2
	2	[394] EG.A Türen	344	Geschoß 1
	3	[396] 2.OG.A Türen	344	Geschoß 3
350	**DIN 18350 - Putz- und Stuckarbeiten**			
	1	[353] 1OG.A Innenwände Büro	346 ; 342	Geschoß 2
	2	[354] 2.OG - Innenstützen	346 ; 342	Geschoß 3

Abb. 138: Transformierte XQuery Abfrage des Objektmodells analog ausführungsorientierter Baubeschreibung nach Leistungsbereichen auf der Basis der DIN-Normen von VOB/C [Langen und Schiffers, 2005. S. 217].

3.2.4 Highlighting

Eine weitere Verwendung für XQuery Abfragen existiert bei dem sog. Highlighting oder Auswählen von Objekten über eine Desktop GUI. Objekte können somit aktiviert werden, indem das Objekt in einem Steuerelement zur Visualisierung mit einem Eingabegerät wie einer Maus aktiviert wird. Bei der VRML Darstellung (Cortona Client) eines Objekts muss nach der Registrierung des vom Benutzer ausgewählten Objekts festgestellt werden, welches Objekt im Objektmodell zu dem ausgewählten VRML Objekt (Node) gehört. Dann wiederum müssen alle VRML Objekte (Nodes), welche die geometrische Darstellung des Objekts ausmachen, farblich verändert werden (Abb. 139). XQuery wird verwendet, um diese Objekte ausfindig zu machen, die wiederum mit einer Methode farblich modifiziert werden:

```
//NOIID_VRML_PICK
string qry = @"<results>{FOR $conobj
IN document('objects')/objectgraph//constructionobject
      [NodeRep/__NOIIDS/objects/object='NOIID_VRML_PICK']
RETURN
<result>{$conobj/UniqueId}</result>}</results>";
```

Weiterhin wird die interne Verknüpfung der Disziplinen im Objekt des Objektmodells genutzt, um das Objekt in seiner Darstellung in anderen Steuerelementen, z.B. der Terminplanung, zu aktivieren.

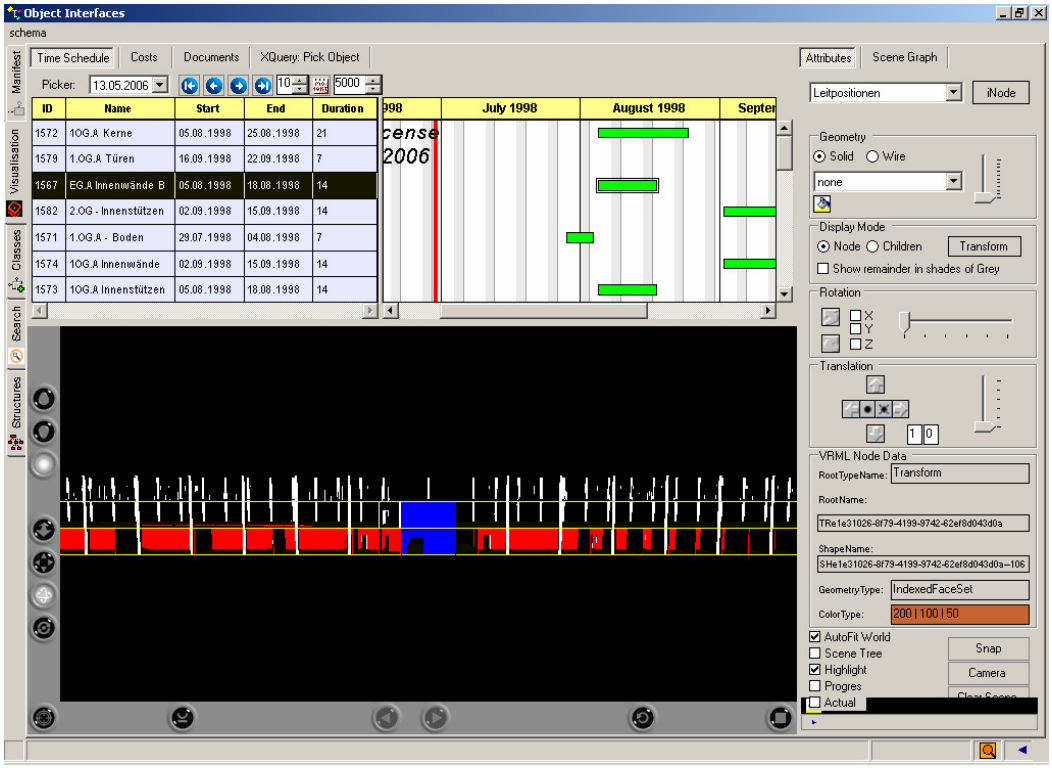


Abb. 139: Highlighting von allen VRML Nodes eines Objektes im Objektmodell, die einem vom Benutzer ausgewählten VRML Node im Cortona Client zuzuordnen sind. Das Objekt wurde auch im NETRONIC XGantt Steuerelement aktiviert.

Ein entsprechendes Highlighting findet statt, indem im NETRONIC Terminplanungssteuerelemenes ein Vorgang ausgewählt wird, welcher wiederum über die Verknüpfung mit der Terminplanung und geometrischen Darstellung im Objektmodell im Cortona Client farblich markiert werden kann.

3.2.5 Soll-Ist Vergleiche

Bei der Kopplung des Objektmodells mit einem Terminplanungsprogramm treten Fragen zur Darstellung des Objektfortschritts auf. Soll-Ist Vergleiche dienen der Überwachung des Bauprozesses und werden überwiegend im Controlling eingesetzt, z.B. während der Kostenkontrolle in der Produktion. Typischerweise werden die Soll-, Ausführungs- und Ist-Zustände terminlich und als Kostengegenüberstellung dargestellt. Diese Art der Darstellung kann um die Visualisierung der Geometrie erweitert werden (Abb. 140).

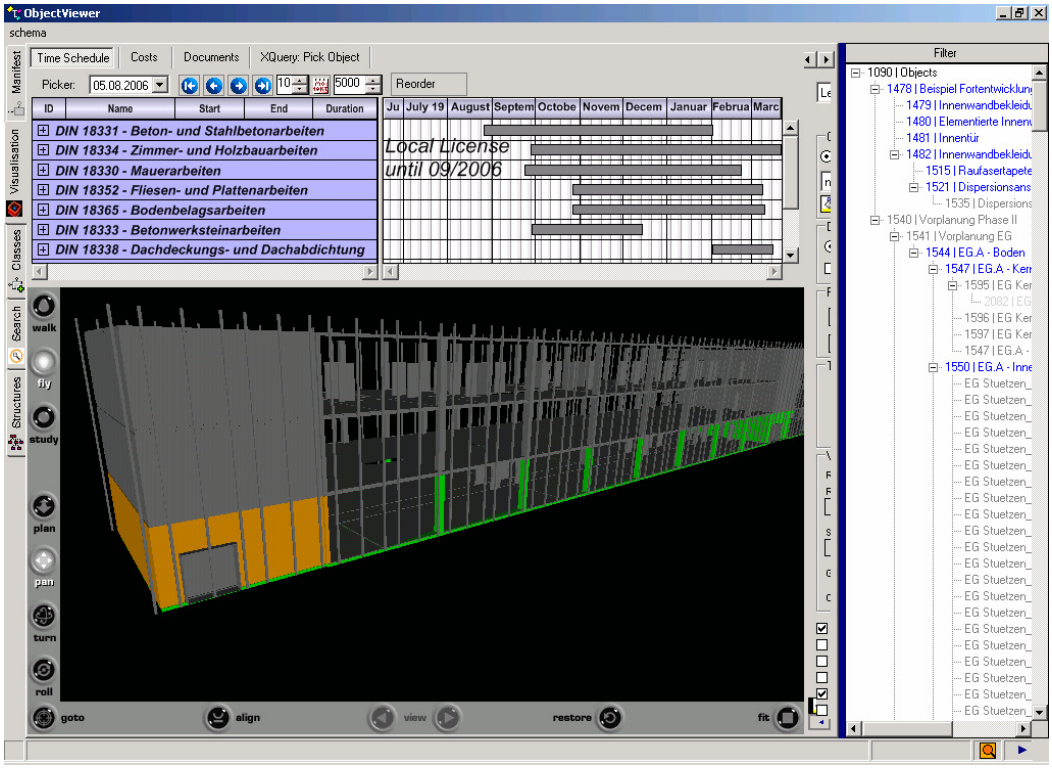
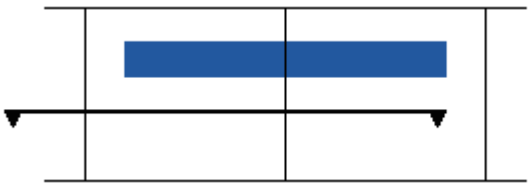
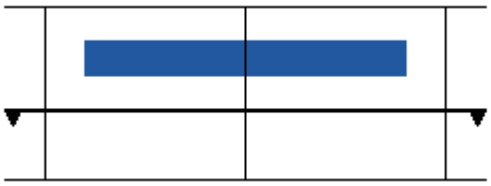
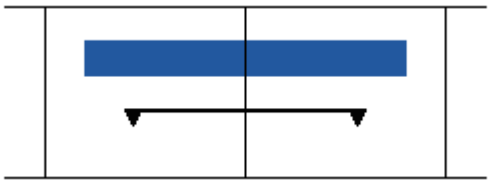
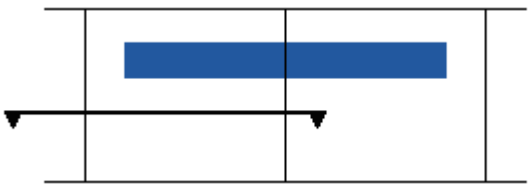
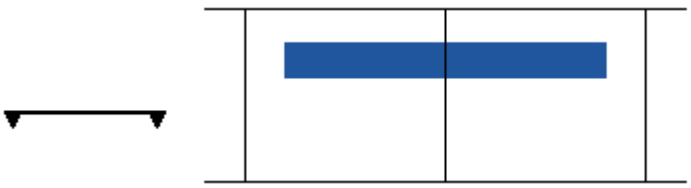
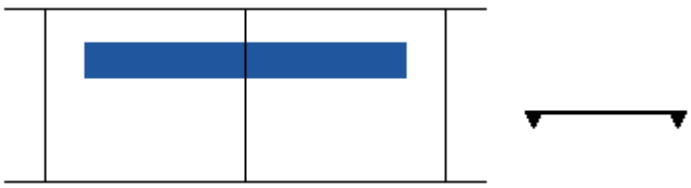


Abb. 140: Klassische Soll-Ist Darstellung der Termine erweitert um die Visualisierung der Geometrie (Soll: Grau; In Ausführung: Orange; Fertiggestellt: Grün).

Bei der Darstellung der Soll-Ist Werte ist es bei der Terminplanung üblich, den Ist-Balken über den Soll-Balken zu legen. Bei der Darstellung der Geometrie ist ein solches Vorgehen etwas schwieriger, da eine Gegenüberstellung wenig aufschlussreich ist und eine Überlagerung Informationen verdeckt. Eine gute Lösung ist die farbliche Darstellung von Objekten, die entweder in Verzug geraten oder früher als vorgesehen erstellt werden. Abbildung 141 führt zwecks Programmierung diese Möglichkeiten auf.

Nr.	Terminliche Situation	Kategorie*
1		P
2		V
3		P
4		P
5		V
6		P
7		V

* V-Verzug; P-Plan oder besser

Abb. 141: Kategorisierung terminlicher Soll-Ist Kombinationen zwecks Programmierung.

Zwecks Darstellung der flexiblen Auswertungsmöglichkeiten des Objektmodells, soll die Auswirkung einer Störung im Bauablauf auf das Gebäude visualisiert werden. Objekte, dessen Termine entsprechend der Planung verlaufen, ob im Soll- oder Ist-Zustand, werden farblich grün markiert. Objekte, deren Termine hinter den geplanten Werten zurück liegen, ob im Soll- oder Ist-Zustand, werden rot markiert. Diese Darstellung ist natürlich eine Vereinfachung der Situation, die üblicherweise durch Beschleunigungen, Verzögerungen und Änderungen gekennzeichnet ist.

Abbildung 143 führt eine solch vereinfachte Soll-Ist Darstellung auf. Diese Gegenüberstellung beruht auf der Annahme, dass Unterschiede zwischen den geplanten Terminen und den Ist-Terminen existieren. Es wird von einem Soll Terminplan ausgegangen und davon, dass alle Beschleunigungen in dem Ist-Terminplan nicht als terminliche Verbesserung gewertet werden, sondern anderweitig bewertet werden. Somit wird vermieden, dass ein sich stetig verändernder Soll-Terminplan entsteht, wodurch der Soll-Terminplan immer wieder der aktuellen Situation angepasst wird. Eine solche Situation würde eine komplexere Darstellung erfordern.

Unter der Annahme, dass späte Soll-Termine nicht zum Vergleich herangezogen werden, können die verschiedenen Zustände in Abbildung 141 auf zwei Zustände reduziert werden:

- P (Plan oder besser): Ist-Ende <= Soll-Ende (EarlyFinish-ActualFinish>=0)
- V (Verzug): Ist-Ende > Soll-Ende (ActualFinish-EarlyFinish>0)

In Anbetrachtnahme der späten Soll-Termine gilt Folgendes:

- P (Plan oder besser): Ist-Ende <= Soll-Ende (LateFinish-ActualFinish>=0)
- V (Verzug): Ist-Ende > Soll-Ende (ActualFinish-LateFinish>0)

Abbildung 142 stellt den Terminplan als Grundlage für die in Abbildung 143 dargestellte vereinfachte terminliche Abweichung des Gebäudes (entsprechend der Detaillierung der HOAI Leistungsphase 5) von der Soll-Planung dar.

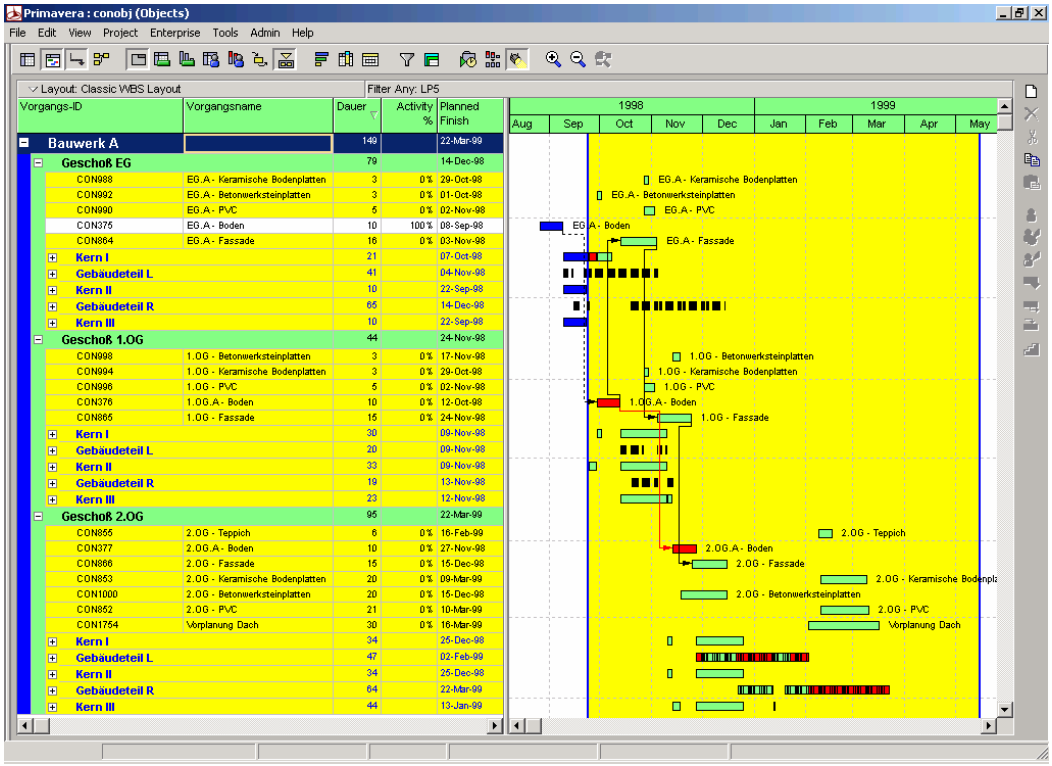


Abb. 142: Terminliche Soll-Ist Situation des Gebäudes mit einer Detaillierungstiefe entsprechend HOAI Leistungsphase 5.

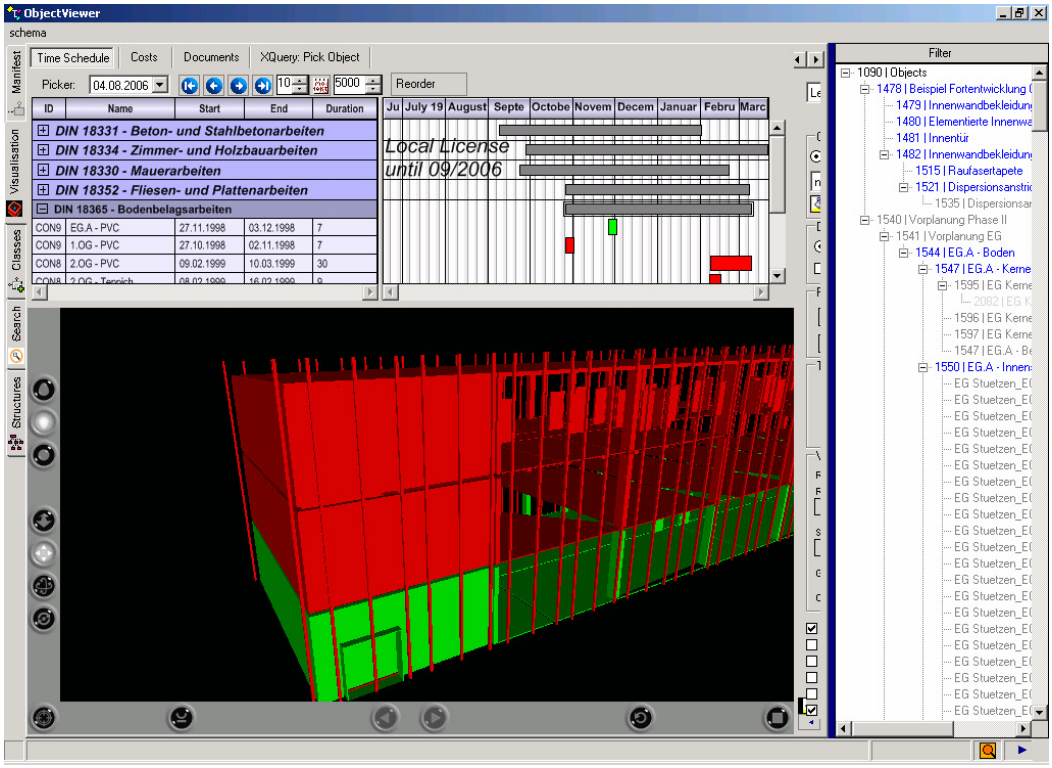


Abb. 143: Visualisierung des Objektmodells im vereinfachten Soll-Ist Zustand nach Abb. 142 entsprechend Terminierung Abb. 143. Gebäude in der HOAI Leistungsphase 5.[18]

Eine weitere wichtige Information ist die Feststellung, ob ein Vorgang auf dem kritischen Weg liegt [Langen und Schiffers, 2005. S. 28, RdNr. 61]. Diese Situation wird in Abbildung 144 dargestellt. Die Regel, nach der diese Objekte identifiziert werden, ist der sog. „Longest Path“, d.h. die Verkettung von Vorgängen, die im Projekt die längste Zeitdauer darstellen.

[18] Grün stellt Objekte dar, die nicht durch den Mangel im Objekt „EG Kerne EG_1“ in der Terminierung beeinflusst wurden. Bei diesen Objekten ist die Soll-Planung unverändert, bzw. erfolgte die Ist-Schreibung aufgrund der Soll-Planung. Werden alle Objekte dargestellt, die in der Ist-Schreibung einen Verzug gegenüber der Soll-Planung aufweisen, ebenso das Objekt „EG Kerne_EG_1“ mit einer Verschiebung der Soll-Planung. Mit dieser vereinfachten Darstellung kann die Einflusswirkung einer Abweichung von der Soll-Planung sowohl im Soll- als auch Ist-Zustand dargestellt werden.

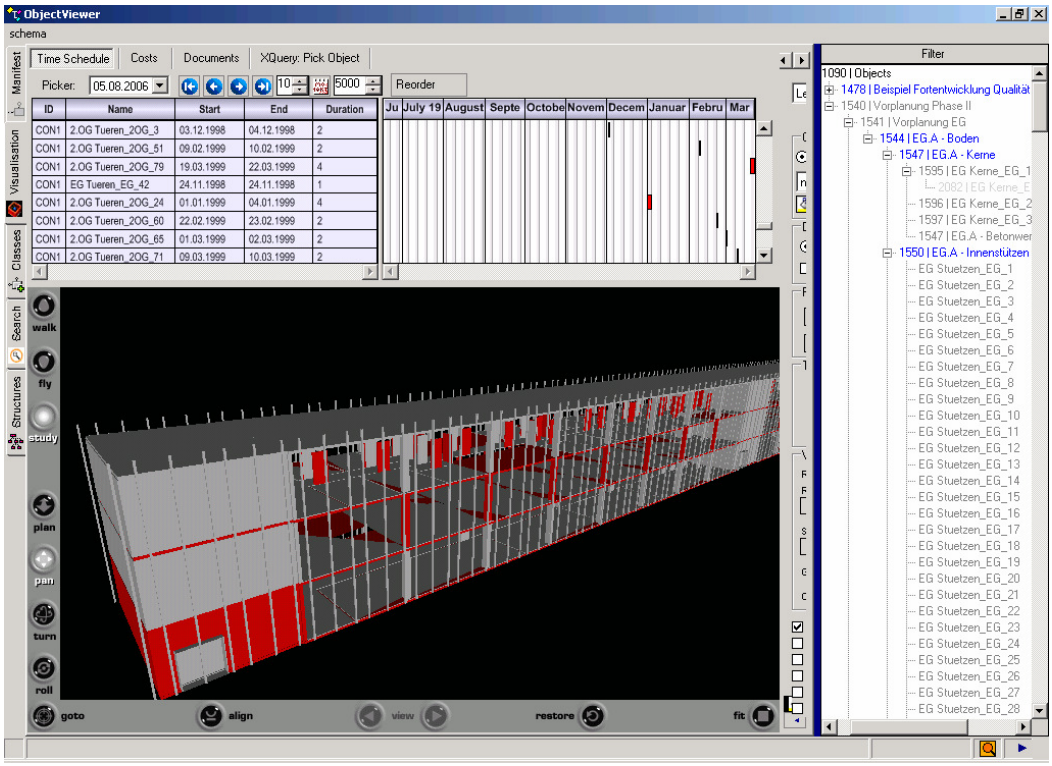


Abb. 144: Objekte, die auf dem kritischen Weg liegen (Longest Path) (Kritisch: Rot; Nicht Kritisch: Grau)

3.2.6 Simulation

Eine Simulation dient der Veranschaulichung des Bauprozesses, entweder in Bezug zu einer Soll-Planung, oder als Ablauf von Soll und Ist Vergleichen. In dem gekoppelten Objektmodell ist eine Simulation in Bezug auf eine Sicht der Daten unproblematisch, da alle Informationen zur Verfügung stehen. Die Umsetzung eines solchen Simulationsverfahrens ist allerdings etwas komplexer, da unterschiedliche Programmelemente wie das Steuerelement zur Darstellung der Terminplanungsinformationen, das Cortona Steuerelement (zur Darstellung der Geometrie), das DataGrid Steuerelement zur Darstellung der Kosten, sowie der Strukturbaum zur Gliederung des Gebäudes koordiniert werden müssen. Die Leistungsfähigkeit des Systems sinkt natürlich mit der Zunahme an Facetten, die berücksichtigt werden müssen. In Abbildung 145 und 146 werden zwei unterschiedliche Zustände des Gebäudes je nach Stand des geplanten Fortschritts dargestellt.

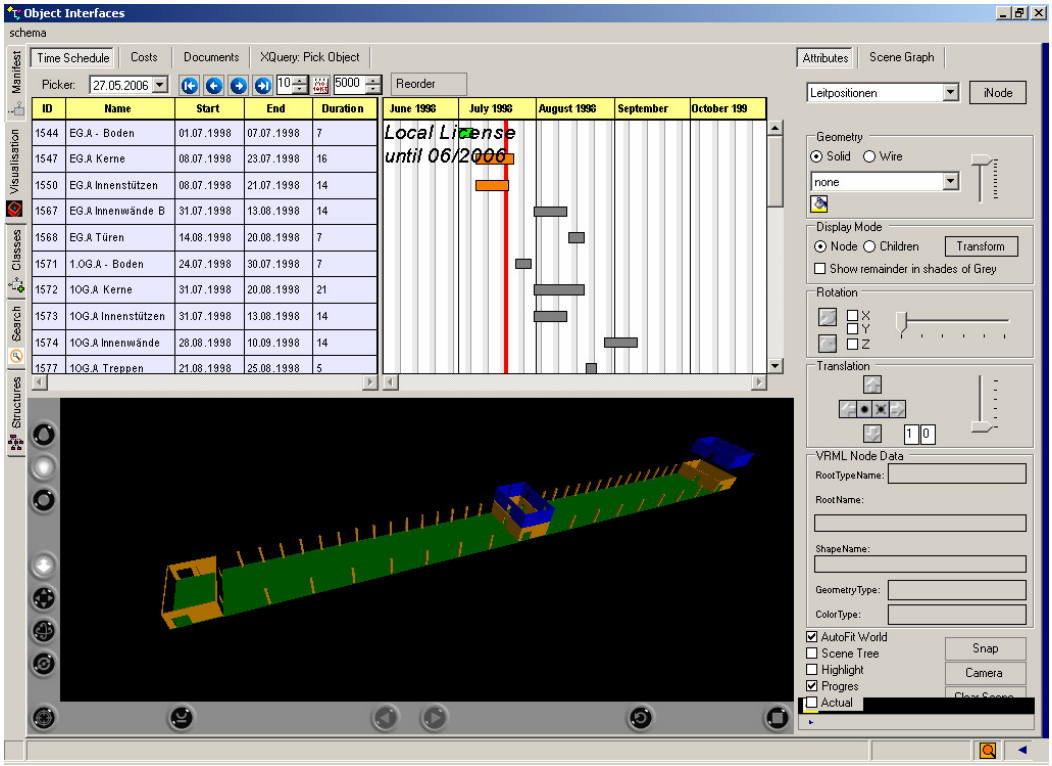


Abb. 145: Forschritt des Gebäudes (Fakultät + LP3) nach der Planung, drei Wochen nach Baubeginn.

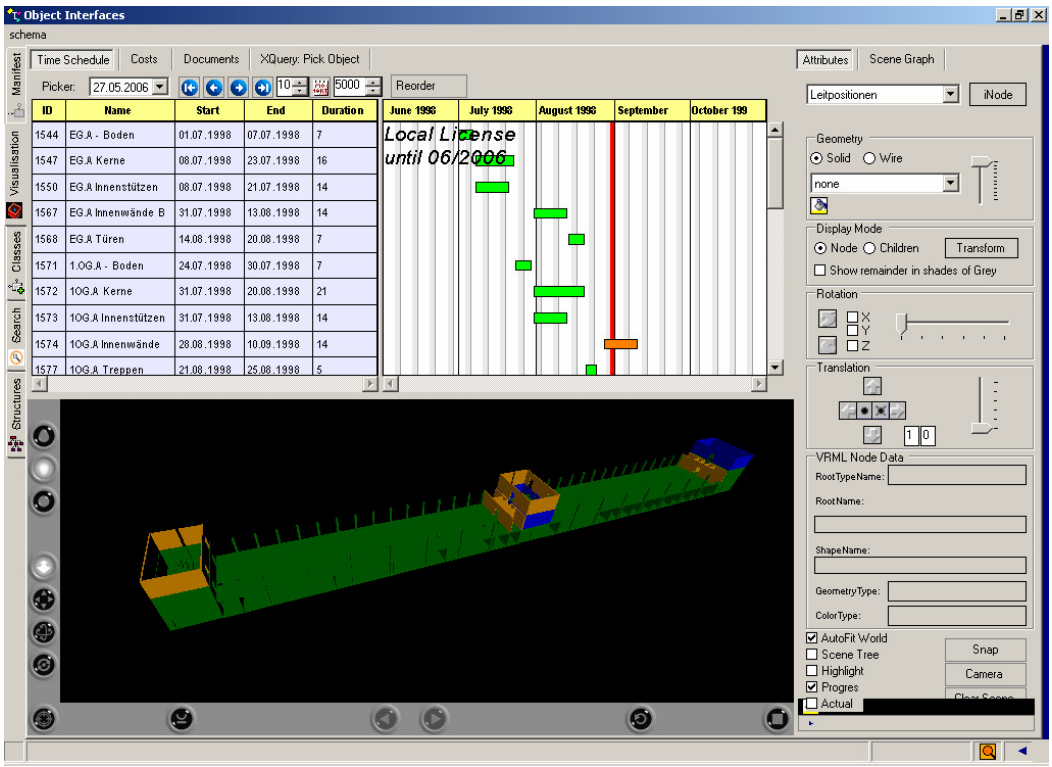


Abb. 146: Forschritt des Gebäudes (Fakultät + LP3) nach der Planung, neun Wochen nach Baubeginn.

Entsprechende Simulationen können auch mit einem Ist-Forschritt durchgeführt werden, um einen Vergleich mit dem geplanten Fortschritt zu ermöglichen und so tatsächlich auftretende Störungen visuell und in Kombination mit der Terminplanung oder Kosten zu identifizieren.

3.2.7 Aggregation

Aggregation wird in der Informatik normalerweise mit Sammlungen oder Collections wie Hashtables oder Arrays verbunden. Damit sind Listen von Werten oder Objekte gemeint. Die Aggregation spielt eine wichtige Rolle in der objektorientierten Programmierung. In der praktischen Anwendung, wie z.B. bei dem hier aufgeführten Objektmodell, kann die Aggregation als Verdichtung von Informationen oder sog. Sammeldarstellungen verstanden werden. Ein typisches Bespiel sind die Sammelbalken wie sie in der Terminplanung üblich sind.
Die Aggregation kann in drei unterschiedlichen Kontexten verstanden werden:

- Vereinfachung von Informationen durch z.B. Top-Down Entwicklung eines Projektes.
- Verdichtete Darstellung von detaillierten Informationen.
- Automatisierte Entwicklung von untergeordneten Informationen mittels Regelsystemen.

Das in dem hier aufgeführten Objektmodell enthaltene Klon-Vorgehen stellt eine Variante der Top-Down Entwicklung dar (siehe Kapitel C.2.3). Das Projekt wird mit den verfügbaren Informationen und unabhängig von Zwängen der Applikationen (Terminplanung, Konstruktion, Kostenermittlung) ausgehend von einer groben Detaillierung und Qualität aufgebaut. Dieser Ausbau geschieht immer im Kontext der vorherigen Informationen durch das Klonen von höheren und daher mit weniger genauen Informationen bestückten Objekten. Dadurch entsteht keine Diskrepanz zwischen der Informationsdichte und den datentechnischen Anforderungen des Objektmodells, was zwangsläufig bei starren Objektmodellen geschehen muss und eine der Schwächen von CAD-basierten Produktmodellen darstellt. Somit ist die Anforderung der Aggregation in dem hier vorgestellten dynamischen Objektmodell erfüllt.

Die verdichtete Darstellung von detaillierten Informationen ist eine Funktionalität der Benutzeroberfläche, wie sie typischerweise in Steuerelementen für Baumstrukturen, Terminplanprogrammen, oder Kalkulationsprogrammen zu finden ist. Diese Funktionalität dient zur Unterstützung des Berichtswesens, stellt jedoch keinen Gewinn an Prozessinformationen dar.

Die dritte Möglichkeit der Aggregation besteht aus einer regelbasierten, automatisierten Detaillierung bei der Top-Down Entwicklung. Diese Vorgehensweise ist jedoch immer dadurch begrenzt, dass die Rahmenbedingungen des Projekts entweder in das Programm integriert werden müssen oder mit einem relativ hohen aufwand eingepflegt werden müssen. Zusätzlich führt die automatisierte Produktion von Daten meistens zu einer unerwünschten und nicht realitätsgetreuen Modellierung des Projekts. Daher wird diese Betrachtungsweise hier nicht weiter verfolgt.

3.2.8 Positionierung einer virtuellen Kamera

Die Cortona API bietet Möglichkeiten zur Manipulation einer oder mehrerer virtueller Kamera(s). Diese dienen der Ansicht von Objekten im Cortona VRML Client. Die Positionierung einer Kamera erfolgt mit der Methode:

```
CameraPos
```

und der Schnittstelle

```
I3DViewServiceAdapterLibrary.I3DViewService3AdapterObject
```

Diese Methode gibt ein Objekt des Typs `VRMLMatrix` zurück. `VRMLMatrix` ist wie folgt aufgebaut:

```
VRML matrix:

| A11  A12  A13  A14  |
| A21  A22  A23  A24  |
| A31  A32  A33  A34  |
| A41  A42  A43  A44  |

Position:

[ A41, A42, A43, A11, A12, A13, A21, A22, A23, A31, A32, A33 ]
```

Ein `VRMLMatrix` Objekt kann mit einem Translations- und Rotations-Objekt initialisiert werden. Nach dem Start der Applikation steht die Kamera in der Ausgangsposition {x,y,z} = {10,0,0} mit der Orientierung entlang der z-Achse in Richtung –z, und waagerecht zur x-Achse. Als Erstes wird die Kamera auf das Gebäude gerichtet. Diese Position des Gebäudes kann aus folgender Eigenschaft des Cortona Steuerelements gelesen werden:

```
vec = engine.Bounds
```

wo `vec[0, 1, 2]` die Koordinaten der Hülle um das Gebäude beinhalten, die drei und the 3 Elemente `vec[3, 4, 5]` die Größe der Hülle bestimmen.

Die Ermittlung der Positionierung der Kamera erfolgt folgendermaßen:

```
Translation:
X0 y0 z0

Winkel in der xz-Fläche:
θ = Math.Atan2(x0, z0)

Winkel in der yz-Fläche:
θ = Math.Atan2(y0 / z0)
```

Nach einer Neupositionierung der Kamera nach {x1, y1, z1} muss eine Korrektur der Ausrichtung auf den Mittelpunkt des Gebäudes erfolgen. Dies erfolgt mittels folgender Ausdrücke:

```
Winkelkorrektur in der xz-Fläche:
Math.Atan2(x1, z1) - Math.Atan2(x0, z0)

Die Korrektur in der yz-Fläche lautet:
Math.Atan2(y1, z1) - Math.Atan2(y0, z0)
```

Die Navigation der Kamera mittels Zeitlupe (hier 4 Sek.) erfolgt mit der Methode

```
ForceCameraMovingGbl(NewMatrix.position, 4000)
```

Die Kamera Methode der Cortona API ermöglicht eine Steuerung der virtuellen Kamera um das Objekt herum. Prinzipiell kann das Objekt aus allen erdenklichen Winkeln und Positionen heraus untersucht werden, da eine Translation (Positionierung) und Rotation der Kamera auch innerhalb des Gebäudes möglich ist. Anders als bei der Steuerung der Orientierung und Position durch ein Eingabegerät wie einer Maus, ermöglichen die Methoden eine Fernsteuerung und Automatisierung in Zusammen-

hang mit anderen fachlichen Erwägungen. Typische Ansichten wie oben (Abb. 147), links (Abb. 148) und perspektivisch (Abb. 149) können nun automatisiert werden. Eine benutzerdefinierte dedizierte Steuerung der Kamera ist mit den oben beschriebenen Methoden möglich.

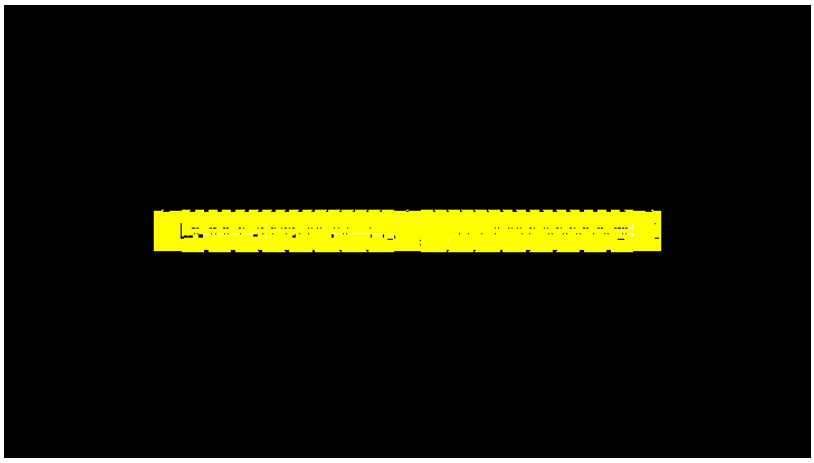


Abb. 147: Ansicht des Gebäudes von oben [0, 0, 15F; 1 0 0 0.0F].

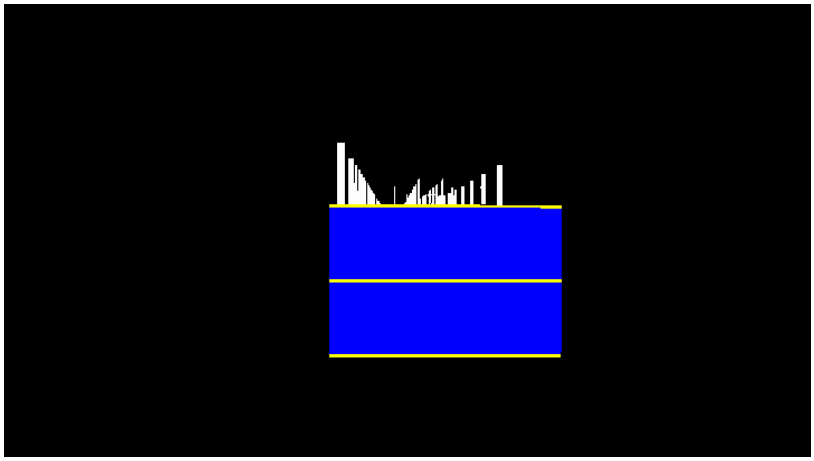


Abb. 148: Ansicht des Gebäudes von links [-10.F 0 0; 0 1 0 Math.Atan2(y,x)].

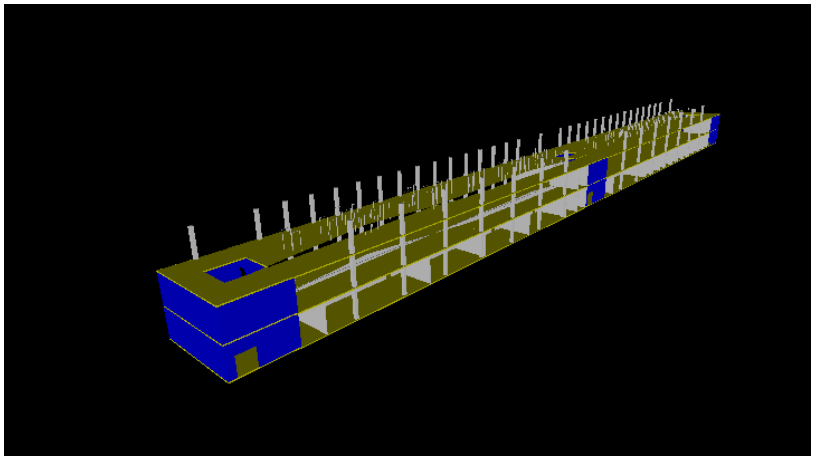


Abb. 149: Ansicht des Gebäudes aus der Perspektive.

3.2.9 Darstellungen der Ergebnisse

Durch den objektorientierten Ansatz und die konsequente Trennung von Objektmodell und Benutzeroberfläche, können Objekte grundsätzlich in jeder denkbaren Benutzeroberfläche dargestellt werden. Die vorherigen Darstellungen erfolgten überwiegend in der Applikation (Desktop GUI). Die Web Darstellung erfreut sich großer Beliebtheit, daher sollte das Objektmodel exemplarisch auch in einer WebForm dargestellt werden.

Einzelne Elemente können dadurch, dass sie in-sich geschlossene Datenobjekte sind, über einen Browser dargestellt werden (Abb. 150) oder im XML Format angezeigt werden (Abb. 151).

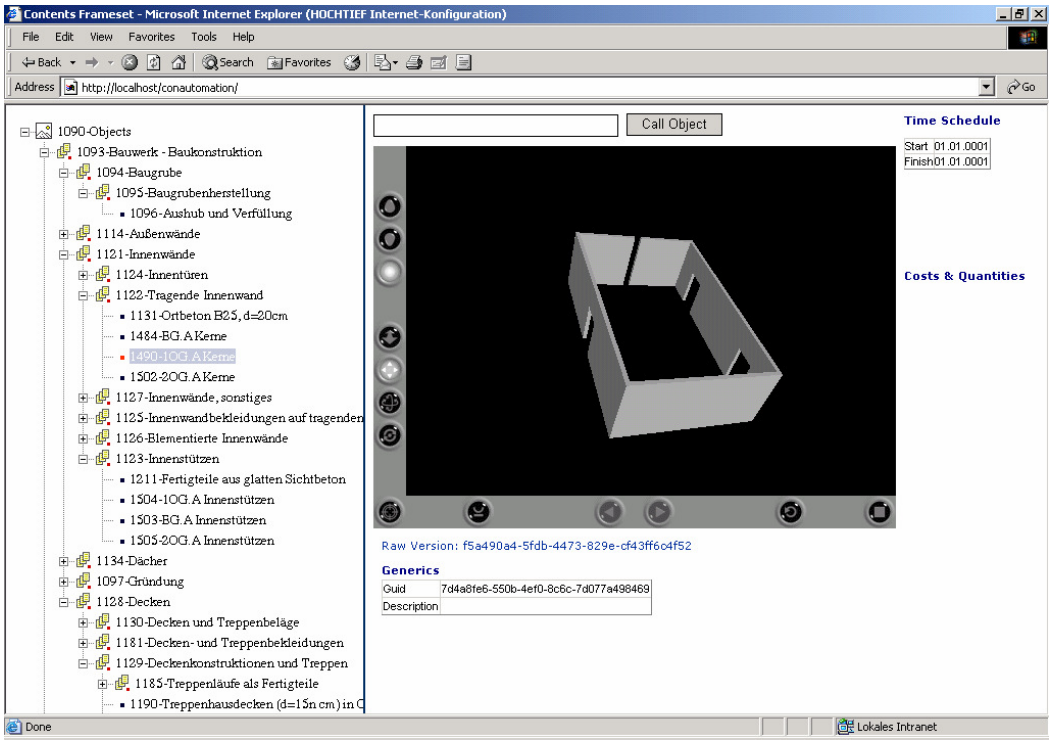


Abb. 150: Darstellung eines Objektes im Browser. Das Objekt kann entweder über den Objektbaum oder das Objektmodelle aufgerufen werden.

Wichtig bei einer webbasierte Darstellung ist die Möglichkeit, sich die Rohdaten anschauen zu können (Abb. 151).

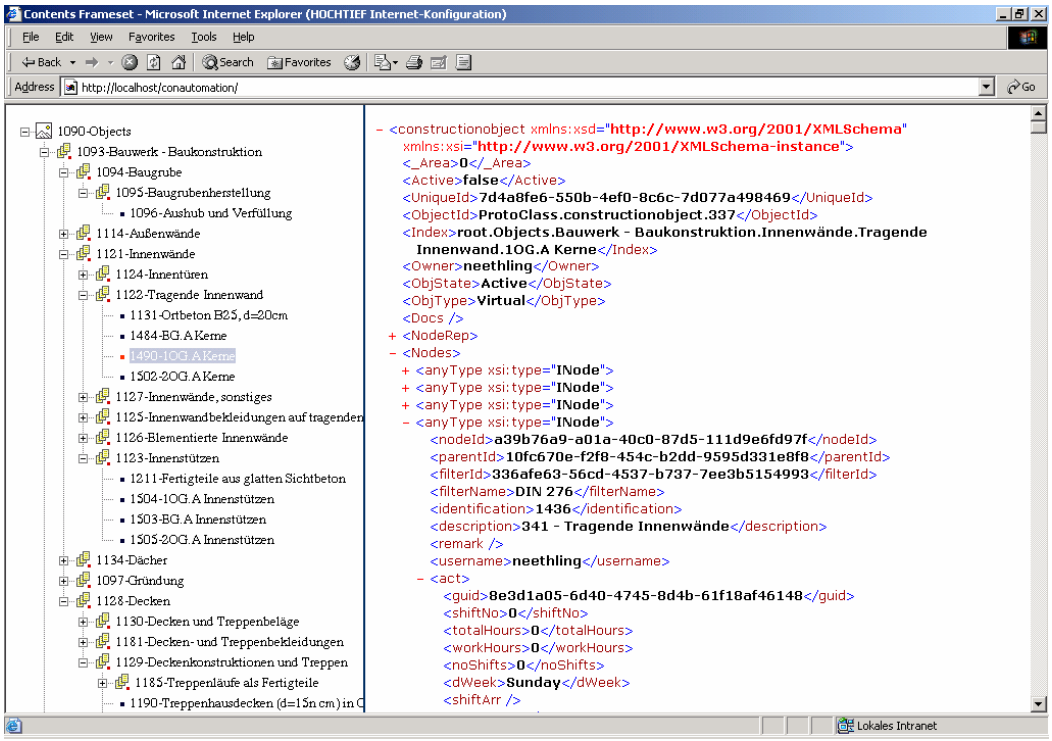


Abb. 151: Darstellung des Objekts im XML Format.

3.2.10 Anmerkungen zum Objektmodell

Die Umsetzung des Objektmodells führte zur Identifizierung der Schwachstellen des Modells. Erwartungsgemäß führte eine erhöhte Anzahl von Objekten zum Leistungsabfall. Diese Situation widerspricht der oft gestellten Anforderung der Skalierbarkeit, worunter nicht nur unbegrenzte Datenmengen zu verstehen sind, sondern auch ein Leistungsverhalten, was nicht von der Menge abhängig ist. In der Praxis tritt ein solches Verhalten kaum ein, jedoch sollte eine direkte oder sogar nicht-lineare Abhängigkeit vermieden werden.

Abbildung 152 führt die Reaktionszeiten bei der Ausführung von Abfragen zu den Leistungsphasen 1, 2, 3, und 5 auf. Das Verhalten weißt einen hohen Initialisierungszeitraum auf und ist danach im IT-technischen Sinne mehr oder weniger linear, was zumindest eine Verbesserung gegen über dem typischen nicht-linearen Verhalten von SQL-basierten Lösungen darstellt.

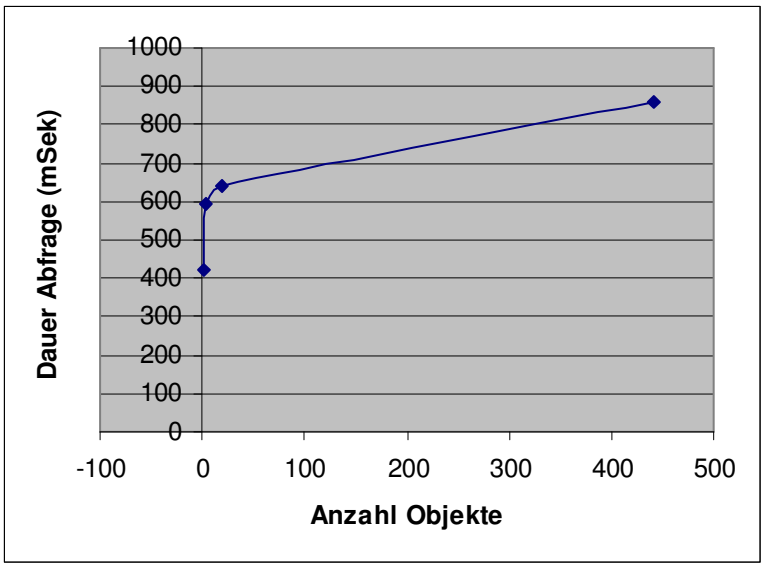


Abb. 152: Reaktionszeiten bei der Ausführung von Abfragen zu den Leistungsphasen 1, 2, 3, und 5.

Die Leistungsproblematik ist nur durch Einsatz von net.VE Technologien, wie in Anhang A.II gezeigt, zu verbessern. Dabei werden vier Aspekte im Wesentlichen verfolgt:

- Aufteilung des Objektmodells in mehrere Teilprojekte, die auf unterschiedlichen Maschinen ausgeführt werden können
- Reduzierung der Detaillierung auf das benötigte Minimum und erhöhte Detaillierung nur auf Anfrage
- Minimierung der Kommunikation zwischen den Applikationen. Generell sind einem System, wie dem hier entworfenen Objektmodell, das eine Zentralisierung der inter-Applikation Kommunikation voraussetzt, Grenzen gesetzt.
- Auswahl von verteilten Technologien (.NET Remoting, XML Services, JavaSpaces) möglichst niedrig auf dem ISO/OSI Stack. Es ist zu prüfen, ob jede Prozesskommunikation mittels TCP/IP und UDP nicht trotz der Aufwände bei der Integration in das Objektmodell den höheren Protokollen wie .NET Remoting oder XML Services überlegen ist.

Zusätzlich zu der Leistungsproblematik wurde auch klar, dass das Arbeiten mit proprietären Programmen Lizenzkomplikationen verursacht. Generell ist zu überlegen, ob der Umstieg auf Open Source Anwendungen die Entwicklung eines solchen Objektmodells vereinfacht, weil hierdurch unnötige technische Probleme mit der Lizenzhandhabung vermieden werden können.

II Diskussion und Ausblick

1 Grenzen eines dynamischen Objektmodells

Objektmodelle sind gewissen Begrenzungen unterworfen. Das liegt zum einen an der Notwendigkeit der Erweiterung entlang hierarchischer Strukturen (Vererbung) [Henckels, D., 2004. S. 10] und zum anderen daran, dass Typsicherheit zur Funktionstüchtigkeit dieser Modelle unerlässlich ist [Box, D., 2005. S. 2]. Box führt die Stärken von Web Services auf, um damit gleich auf Beschränkungen von Objektmodellen hinzuweisen:

- Grenzen sind explizit - Objektmodelle kennen nur Typgrenzen (Kapselung).
- Dienste sind autonom - Objektmodelle brauchen immer einen Kontext.
- Dienste interagieren mit Schemen und Verträgen - Objektmodelle interagieren mit Klassen und Typen.
- Leitlinien werden *a priori* eingesetzt, um Kompatibilität zu erreichen – die Leitlinien von Objektmodellen werden während der Entwicklung festgelegt.

Diese Schwächen von Objektmodellen werden auch weiterhin in einem dynamischen Objektmodell vorhanden sein. Deren Auswirkung auf eine praxisgerechte, nichthierarchische, nichtlineare und autonome Arbeitsweise wurde in dieser Arbeit nicht untersucht.

Ein Hauptproblem eines statischen Objektmodells besteht darin, dass alle Eigenschaften und deren Typinformationen vorher bekannt sein müssen. Während der Evolution eines Gebäudes sind im Normalfall viele Informationen nicht vorhanden oder fehlerhaft [Henckels, D., 2004. S. 21-37]. Darum können diese Informationen nicht anschließend mittels eines Messaging Systems (UDP/IP) ergänzt werden. Diese Situation stellt eine Schwäche des klassischen Objektmodells dar.

Im dynamischen Objektmodell kann dieses Problem umgangen werden, indem für zukünftig erwartete Eigenschaften das Objektmodell dynamisch erweitert wird (Vererbung) und die Typzuordnung, falls nicht bekannt, auf das generische Objekt gesetzt wird. Dieser Typ kann im Nachhinein verändert werden, was jedoch zu einer Vererbung führen wird. Diese Betrachtung macht schon klar, dass auch kleine Änderungen am Objektmodell (Eigenschaften, Methoden und Typen) zu Vererbungsvorgängen führen. Der Vorteil dieser Methode der Dokumentation von Veränderungen

besteht darin, dass das Objektmodell in seinen jeweiligen Zuständen zur Verfügung steht. Dieser Teil des Modells muss äußerst effizient und fehlerfrei arbeiten. Das aufgeführte Modell stellt rudimentäre Funktionalität dar. Es muss für den operativen Einsatz gründlich überarbeitet werden, um eine hohe Vererbungsfrequenz stabil zu verarbeiten.

Eine andere Möglichkeit der Dokumentation von Veränderungen besteht darin, Änderungen im Objektmodell mittels der eingebauten Ereignisbehandlung in einem internen oder referenzierten Speicher festzuhalten. Die Vorteile eines solchen Vorgehens liegen darin, dass das Objekt nur einmal vorhanden ist (Ersparnis von Speicher) und der Aufwand eines Klon-Vorgangs nicht nötig ist. Der Nachteil dieser Methode liegt jedoch darin, dass das Objekt bei Auswertungen immer wieder in seinem entsprechenden Zustand rekonstruiert werden muss, was einen erheblichen Leistungsnachteil darstellt und in Anbetracht der Aggregation problematisch werden kann.

Eine weitere Frage ist, ob die oben beschriebene strikte Handhabung des Modells bei Änderungen in dessen Schema (die Vererbung) gewollt ist. Falls dem nicht so sei, würde das eine grundlegende Untersuchung alternativer Methoden zur Darstellung eines Modells erfordern. Die von Box [Box, D., 2005. S. 2] genannten Vorteile einer dienst-basierten Plattform würde einen Ansatzpunkt für eine solche Untersuchung bereiten. Wie im jetzigen Objektmodell erfolgte die Steuerung eines solchen Systems auch textbasiert erfolgen.

2 Prototyp

In Teil C wurde das dynamische Objektmodell exemplarisch anhand eines in der Fakultät Bauwesen, Lehrstuhl für Bauorganisation, von Studenten erarbeiteten 3D Gebäudemodells erprobt. Zuerst wurde die qualitative Entwicklung eines Bauelements (Wand mit Tür) betrachtet. Dabei ging es darum, die systematische und logische Evolution dieses Bauteils anhand der DIN-Normen 276, 277, andere in den ATV (Allgemeinen Technischen Vertragsbedingungen für Bauleistungen der VOB Teil C) enthaltenen Normen und einer gebäudeorientierten Baubeschreibung mit dem dynamischen Modell abzubilden. Somit konnte auch die grundlegende Arbeitsweise des Modells geprüft werden.

Die gleiche Vorgehensweise wurde nun anhand des Fakultätsgebäudes durchgeführt. Somit sollte das Modell an einem wirklichkeitsnahen Objekt getestet werden.

Zusätzlich zu den funktionalen Tests konnten auch Performanceprobleme eruiert werden. Die Visualisierung aller Informationen im Kontext bedarf ferner einer komplexen und aufwendigen Benutzeroberfläche.

Die Entwicklung einer solchen Oberfläche ist typischer Bestand einer strategischen Produktentwicklung von Softwarefirmen. Sie sprengt den Rahmen einer Dissertation und ist vielmehr eine Betrachtung der Ergonomie und existierender GUI Technik. Zur Demonstration dieses Beispiels wurden rudimentäre Funktionalitäten der Oberflächen bereitgestellt.

Eine Betrachtung des Prototyps (Fakultätsgebäude) ist anhand einer Gliederung der Strukturen am besten zu machen. Eine Betrachtung des Modells kann wie folgt gegliedert werden:

- Datenhaltung
- Dynamisches Modell
- Schnittstellen zu Applikationen
- Schnittstellen zur Benutzeroberfläche (GUI)
- Benutzeroberfläche (GUI)
- Verhalten des Systems

2.1 Datenhaltung

Die Datenhaltung wurde in einer Relationalen Datenbank (SQL Server) realisiert. Die Objekte wurden im Binärformat serialisiert. Das Entity-Relation Modell der Datenbank ist primitiv, da das sog. Business Model in den Klassen objLib modelliert wurde. Die Datenbank dient ausschließlich der Datenhaltung der Objekte und all solcher Objekte, die für die Modellierung von Teilbereichen, wie etwa den Gliederungsstrukturen, zuständig sind.

Moderne Objektdatenbanken bieten Schnittstellen zur programmatischen Steuerung von Metadaten. Umfang und Art dieser Funktionalität ist von der jeweiligen Datenbank abhängig. Es gibt keinen verbindlichen öffentlichen Standard für diese Implementierungen. Die Datenbank Cache der Firma Intersystems ist ein gutes Beispiel einer solch objektorientierten Datenbank.

Diese Funktionalität könnte für ein dynamisches Objektmodell eingesetzt werden, würde jedoch die Implementierung an ein solches System binden. Somit käme die

Trennung von Datenhaltung und Modell abhanden, was dazu führte, dass aus Sicht der Wartung und Effizienz das Modell in der Datenbank implementiert würde.

Hinzu käme, dass solche Datenbanken Application Server und Web Server unterstützen, was bedeutet, dass die Benutzeroberfläche (GUI) dort auch etabliert würden. Tatsächlich werben diese Datenbankhersteller damit, dass Ihre Produkte komplette Entwicklungsumgebungen darstellen. Somit landet das gesamte Objektmodell in der Datenbank, was dem Mehrschichtenmodell widerspräche.

Um diese Situation zu vermeiden und die reine Datenhaltung in der Datenbank zu realisieren, genügt die Speicherung der serialisierten Objekte in der Datenbank. Die Implementierung einer beschränkten Datenhaltung in einer objektorientierten Datenbank ist in der Tat problematisch. Das liegt daran, dass sich alle gespeicherten Objekte einer Klassenbeschreibung gegenüber stehen müssen. Natürlich können auch nichtssagende binäre Felder verwendet werden, jedoch würden auf diese Weise die Vorteile einer solchen Datenbank und vor allem der Performancevorteil nicht genützt. Auch die hybriden Datenbanken, eine Mischung aus objektorientierter und relationaler Modelle, stellen keine befriedigende Lösung dar. DB2 (IBM) ermöglicht eine dynamische Änderung der Metadaten (zur Laufzeit). Diese Technologie ist jedoch noch nicht ausgereift. Daher besteht eine praktikable Lösung darin, die serialisierten Daten in Tabellen einer relationalen Datenbank zu speichern.

2.2 Dynamisches Modell

Die Funktionalität des dynamischen Modells beruht auf wenigen komplexen Klassen. Diese Klassen unterstützen folgende Funktionalitäten:

- Vorgänge zum Einlesen und Interpretieren von XML Schema Klassenbeschreibungen
- Methoden zur Generierung von Codes aus den Schema-Dateien heraus und Ablegen in einer Datei.
- Dynamische Kompilierung von Code, der in Dateien abgelegt wurde.
- Methoden zur Instanzierung von Objekten aus dynamisch generierten Klassen
- Methoden zur Nutzung der Eigenschaften, Methoden und Ereignisse der Objekte die im dynamischen Objektmodell generiert wurden.
- Schnittstellen für Ereignismodelle, Benutzeroberflächen und Dienste anderer Applikationen.

Die Erweiterung eines klassischen Objektmodells zur Laufzeit beruht auf den Klassen der Namespaces System.CodeDom und System.Reflection. Diese Klassen wurden entwickelt, um das dynamische Einbinden von XML Diensten im Visual Studio (.NET Entwicklungssystem von der Firma Microsoft) zu ermöglichen. Weiterhin werden diese Klassen in dem Tool wsdl.exe zur Generierung von Proxy Klassen für Web Dienste eingesetzt. Inzwischen erfreuen sich diese Klassen eines erweiterten Einsatzes in Situationen, in denen eine Dynamisierung des Objektmodells wünschenswert ist.

Gezwungenermaßen geschieht dies bei jeglicher Interaktion von XML Web Diensten mit Applikationen, die als allgemeine Portallösungen auftreten sollen und Objektmodellen, die aus fachlichen Gründen zur Laufzeit erweiterbar sein müssen. Diese Art der Programmierung ist auch als „Declarative Imperative Coding“ bekannt. Imperative Sprachen wie C++, Java oder die .Net Sprachen erfreuen sich der Typsicherheit und Effektivität. Deren Schwäche beruht darin, dass Metainformationen zur Zeit der Programmierung bekannt sein müssen. Deklarative Sprachen wiederum haben eine begrenzte Wirkung, sind aber textbasiert und können somit auch zur Laufzeit eingesetzt werden. Die Kombination von deklarativen und imperativen Sprachen ermöglicht die Dynamisierung des Objektmodells. Allerdings sind die Begrenzungen eines objektorientierten Ansatzes auch in diesem Modell enthalten.

2.3 Schnittstellen

Das Objektmodell lebt von Schnittstellen. Diese stellen im klassischen objektorientierten Modell einen Vertrag zwischen Modulen dar. Durch Schnittstellen (in der Programmierung als Interfaces bekannt) stehen den Modulen alle benötigten Informationen zu den Eigenschaften und Methoden zur Verfügung. Auf die Problematik der Starrheit dieser Schnittstellen wurde hingewiesen. Sie sind jedoch auch in der modernen dynamisierten Form unerlässlich zur Funktionstüchtigkeit einer Interaktion zwischen Softwaremodulen. Schnittstellen existieren zwischen den einzelnen Modulen eines Objektmodells, zwischen externen Diensten und dem Objektmodell und zwischen dem Objektmodell und einer grafischen Benutzeroberfläche oder einem Reportingsystem. Die dynamische Einbindung (late Binding) von Schnittstellen und Diensten könnte in Zukunft automatisiert werden. Dadurch kann ein großes Potential an Automatisierung und Effizienzsteigerung erschlossen werden. Das ist auch eine Voraussetzung für die Einbindung anderer Disziplinen außer Terminplanung, Kalku-

lation oder CAD.

3 Einsatz des Modells in existierenden Applikationen

Der Vorteil eines Objektmodells unabhängig von Applikationen liegt in der Möglichkeit, existierende Applikationen ohne Änderungen einzubinden, soweit diese Applikationen entsprechende Schnittstellen vorsehen. In dieser Arbeit wurden die Applikationen Allplan (CAD), P3e (Terminplanung) und Kubus (Kalkulation) im Zusammenspiel mit dem Objektmodell untersucht. Mit der Ausnahme von Kubus konnte ein praktikables Maß an Automatisierung erkannt werden. In Bezug auf Kubus werden Anstrengungen zur Entwicklung einer dynamischen Schnittstelle (Xml Web Dienst) Unternommen.

Somit könnte das Objektmodell mit diesen drei Applikationen, die auch in der Praxis häufig eingesetzt werden, integriert werden. Dadurch wären seitens der Anwender keine Änderungen erforderlich. Die Erweiterung der Schnittstellen der Applikationen liegt im normalen Umfang einer kontinuierlichen Softwareentwicklung. Der Hardwarebedarf eines solchen Systems für eine typische Baustelle liegt im Umfang einer ordentlichen IT Infrastruktur, die bei größeren Infrastrukturprojekten typischerweise vorhanden ist. Somit könnte das hier beschriebene System ohne nennenswerte Änderungen in der Praxis eingesetzt werden.

III Ausblick

Bei der Integration von Applikationen unterschiedlicher Disziplinen (Planung, Terminplanung, Kalkulation usw.) bieten sich im Wesentlichen zwei Möglichkeiten an. Zum einen kann auf ein gemeinsames Integrationsprotokoll gesetzt werden. Das IFC Protokoll ist ein Beispiel dieses Ansatzes. Wie schon erwähnt wurde, werden solche Protokolle über lange Zeiträume entwickelt. Sie stellen oft einen Kompromiss unterschiedlicher Anforderungen dar, sie lassen sich schlecht erweitern und es kann nicht von einem flächendeckenden Einsatz solcher Protokolle ausgegangen werden.
Eine andere Möglichkeit besteht darin, die unterschiedlichen Applikationen mittels Programmierschnittstellen (API's) in ein dynamisches Objektmodell zu integrieren. Abgesehen davon, dass die Verwaltung eines dynamischen Objektmodells komplex ist, bieten herkömmliche Applikationen nur unzureichende Schnittstellen an. Hinzu kommt, dass die unterschiedlichen Objektmodelle in ihrer Struktur sehr verschieden sind, was die Integration erheblich erschwert.

Es ist festzustellen, dass beide Ansätze in Bezug auf eine Integration der Applikationen mit viel Aufwand verbunden sind. Allerdings stellt die zweite Variante einen Lösungsansatz dar, der weniger abhängig von Standarden und deren langwierigen Entscheidungsprozessen ist. Sie verschafft daher der Implementierung mehr Spielraum. Dabei vergrößert sich jedoch die Abhängigkeit zu den Programmierschnittstellen. Dieser Ansatz eines dynamischen Objektmodells wurde in der vorliegenden Arbeit gewählt.

Dynamische Objektmodelle erfreuen sich eines wachsenden Einsatzes in vielen Aspekten der Informatik. Die Kombination einer deklarativen Beschreibung der Objekte, die sich an der menschlichen Sprache und somit an fachlichen Ausarbeitungen anlehnen, und die Ausführungsstärke und Typsicherheit von imperativen Sprachen bieten eine gute Lösung für die Problemstellung eines umfassenden Objektmodells eines Projektes. Die Stärken und Schwächen einer solchen Lösung liegen in der objektorientierten Modellierung [Henckels, D., 2004. S. 10; Box, D., 2005. S. 2]. Die von Box vorgeschlagenen Erweiterungen eines solchen Modells, welch die Nutzung neuer Plattformen wie Indigo benötigen, bedürfen einer systematischen Untersuchung bezüglich deren Effizienz.

Der Einsatz eines dynamischen Objektmodells, in Kopplung mit den für das Objektmodell relevanten Applikationen würde das Hauptproblem der Projektmodellierung lösen: Versionisierung. Versionisierung betrifft nicht nur die Aktualität der Projektdaten und die damit oft vorkommende Bearbeitung von veralterten Datenbeständen, sondern ist auch eine Dokumentation von Veränderungen der Planung und der Ausführung. Diese Veränderungen sind entweder gewollt und somit die Planung selbst, oder stellen Fehler, Fehlentscheidungen oder Nachlässigkeit dar. Daraus können zwei wesentliche Bestandteile eines umfassenden Models abgeleitet werden:

- Alle Teilnehmer müssen mit dem gleichen Datenbestand arbeiten.
- Änderungen müssen aussagekräftig dokumentiert werden.

Das dynamische Objektmodell kann in vollem Umfang beide Anforderungen erfüllen. Die Probleme, die bei der Erweiterung des Modells entstehen, wurden oben aufgelistet. Die Einführung eines solchen Modells veränderte die praktische Art der Projektarbeit nicht wesentlich. Existierende Applikationen für die Terminplanung, CAD, Kalkulation und Kostenkontrolle könnten in ihrer jetzigen Form erhalten bleiben. Das ist wünschenswert, weil nur eine Kompartmentalisierung der Aufwände den Betrieb eines komplexen Objektmodells in der Praxis ermöglicht. Die einzige Anforderung wäre eine Erweiterung der Schnittstellen zu vollwertigen Objektmodellen, die auch Ereignisbehandlung unterstützen. Die Primavera Integration API 5.0 erfüllt von allen Systemen diese Anforderung am ehesten. Nemetschek Allplan weist auch Elemente einer solchen API auf, ist jedoch erweiterungsbedürftig.

Das Objektmodell agiert unabhängig von den Applikationen. Die Verbindung erfolgt über die API's, die wiederum mit Remoting Technologien wie XML Services, .NET Remoting, RMI, CORBA oder JINI angesprochen werden können. Hier ist noch zu untersuchen, ob Technologien, die niedriger auf dem ISO/OSI Modell angesiedelt sind, wie TCP/IP oder UDP, nicht leistungsfähiger wären.

Bei der Projektarbeit werden oft Szenarien oder kleinere temporäre Projekte aus verschiedenen Gründen eingesetzt. Bei einem dynamischen Objektmodell sollte darauf geachtet werden, dass offizielle und operative Projektdaten in einem öffentlichen Bereich gespeichert werden. Das soll verhindern, dass Szenarien, private Daten oder temporäre Projekte in das dynamische Objektmodell gelangen.

Weiterhin sollten alle Anwender damit vertraut sein, dass die komplette Integration der Daten dazu führt, dass die jede einzelne Veränderung in dem Modell größere Folgen nach sich ziehen kann. Zum Beispiel führen Änderungen im Terminplan zu einer Verlängerung der Bauzeit, was wiederum einer Erhöhung der indirekten Kosten in der Kalkulation führt.

Die Strukturierung des Objektmodells spielt eine herausragende Rolle. Obgleich viele Strukturen parallel eingesetzt werden können, gibt es immer eine Leitstruktur für die Sicherstellung der Eindeutigkeit des Gebäudemodells.

Das dynamische Objektmodell bietet eine wesentliche Verbesserung gegenüber nicht-gekoppelten Systemen oder Systemen, die um eine Disziplin entwickelt wurden (4DCAD, Projekt Management Systeme, ERP Lösungen). Die Stärke des Systems liegt darin, dass existierende Arbeitsweisen nicht beeinträchtigt werden, keine neuen Applikationssuites entwickelt werden müssen, und eine umfassende Dokumentation der Veränderungen (Fortschritt, Fehler) im Projekt erreicht wird. Weiterhin wird durch die Aufteilung des Objektmodells in fachliche Disziplinen eine Aufgabenverteilung erreicht, was den Pflegeaufwand reduziert.

Die Schwächen des dynamischen Objektmodells liegen technisch in den Begrenzungen des objektorientierten Ansatzes und fachlich in der Notwendigkeit der fortlaufenden Strukturierung des Gebäudemodells. Die erhöhte Komplexität und Interaktion zwischen den Applikationen setzen rudimentäre Kenntnisse der objektorientierten Programmierung voraus.

IV Glossar

Activity-Code – Schlüsselsysteme	Activity – Codes sind in “ConObj.” anzulegende Schlüsselsysteme zur Einstufung der Objekte. Die Activity-Codes sind Attribute mit festen Inhalten, die je nach Detaillierungsgrad dem Objekt zugewiesen werden. Sie strukturieren entscheidend das System und sind in dieser Arbeit bauspezifischer Natur (HOAI, DIN, etc.).
API	API (**Application Programming Interface**) stellen die Schnittstellen des Systems mit externen Applikationen dar. Ein API stellt Routinen und Dienste bereit und erleichtert die

	Anpassung externer Programme an das System. Die API's gewährleisten einen verlustfreien Datenaustausch.
Assoziationen/ Beziehungen	Es gibt zwei Grundarten von Assoziationen: Die Art von Beziehungen zwischen Objekten, die aus der Realität entstammen und entsprechend im System abgebildet werden und die entwicklungs- und systembedingten Beziehungen von Objekten. Die Assoziationen drücken das Verhältnis zwischen ansonsten selbständigen Objekten gleicher Klasse aus. Eine Assoziation geht durch eine Vererbung nicht verloren.
Attribut	Attribute sind die Merkmale der Objekte. Ein Attribut kann verschiedene Attributwerte annehmen. Diese Werte können mit Hilfe von Methoden, also kurzen vom Anwender erstellten Programmen, oder durch externe Applikationen bestimmt werden. Neben den objektbezogenen Attributen gibt es auch projektbezogene Attribute. Die projektbezogenen Attribute beschreiben die abstrakten Objekte, wie z.B. „Risiko", „Vertrag" oder „Gesamtdauer". Attribute sind auch Daten, die einem Objekt zugeordnet werden, z.B. geometrische Daten, Kostengruppen, Leistungsbereiche etc.

Bausoll

Bausoll ist die aus der Ausführungsplanung entwickelte Produktionsvorgabe. Es wird durch den Bauvertrag nach Bauinhalt und nach Bauumständen zu differenzierender, auszuführender Bauleistung. Der Bauinhalt stellt das visualisierte Bauobjekt, also „Was“ gebaut wird dar, die Bauumstände, beschreiben das „Wie“ gebaut wird.

Daten

Gemäß ISO 2382-1 sind Daten[19] formalisierte Fakten, Konzepte oder Instruktionen zur Kommunikation, Interpretation oder Verarbeitung durch Menschen oder Maschinen. Werden diese Daten zielgerichtet eingesetzt und durch den Menschen im Zusammenhang interpretiert, wandeln sich die Daten in Informationen um.

Einfachvererbung

Im Gegensatz zur Mehrfachvererbung werden bei der Einfachvererbung nur die Attribute, Schnittstellen und Methoden einer Mutterklasse übernommen. Dies schränkt die Möglichkeiten der Vererbung ein, kann jedoch durch den Einsatz von Schnittstellen umgangen werden.

Die Einfachvererbung (C#, Java VB.NET) wurde eingeführt, um das Problem der Mehrfachbelegung bei C++ auszuschalten. Die in Schnittstellen definierten Methoden müssen beim Einsatz einer Schnittstelle neu definiert werden, was eine Mehrfachbelegung verhindert, jedoch die Möglichkeit zusätzlicher Erbquellen erschließt.

---

[19] Definition des Begriffes *data* nach ISO 2382-1, Stand 1993

„A representation of facts, concepts or instructions in a formalized manner, suitable for communication, interpretation or processing by humans or automatic means.“ aus Hölkermann, Oliver, Modell eines prozessorientierten Informationssystems zur Steuerung von Bauunternehmen, Weißensee Verlag, Berliner Beiträge zum Bauwesen Band 1, 2002, S.15f.

Ereignis (Event)	Ein Ereignis löst einen im System hinterlegten Befehl oder hinterlegte Methode aus. Auf der Systemebene sind i.d.R. bauspezifische Ereignisse definiert. Das Ereignis wird durch ein Eingabe–Medium ausgelöst, diese „Botschaft“ wird durch das System aufgenommen und unter Mithilfe des API verarbeitet.
http	Das Hyper Text Transfer Protocol, über das die Abwicklung des überwiegenden Teils des Datenverkehrs im Internet erfolgt, basiert wiederum auf TCP/IP. Es definiert HTML und XML Dokumente, die über das Internet übertragen werden. HTTP-fähige Server und Clients wie Webserver und Browser kommunizieren mit Hilfe von HTTP. HTTP stellt gewisse Bedingungen zur Auflösung von IP Adressen und TCP Ports. In der aktuellen W3C Spezifikation HTTP 1.1 werden eine ganze Reihe Features behandelt, einschließlich Pipelinefunktionen, Segmentierung (Chunking), Authentifizierung, Vorauthentifizierung, Verschlüsselung, Unterstützung von Proxies, Überprüfung von Serverzertifikaten und Verbindungsverwaltungen.
I-Collection	Eine gefilterte Gruppierung von Objekten mit gleichen Attributwerten wird für die Weiterverarbeitung zu einer I-Collection zusammengefasst.
Informationen	Informationen sind zweckbezogene oder zielgerichtet eingesetzte und entscheidungsrelevante Daten. Sie sind somit eine Voraussetzung für eine Detaillierung der Planung und für deren Entscheidung.

Instanzierung	Ein Objekt, das zu einer bestimmten Klasse gehört, bezeichnet man als Instanz. Eine Instanzierung ist der systeminterne Vorgang, bei dem das (modifizierte) Objekt in eine Klasse instanziert wird, wo die Klasse in den systeminternen Speicher geladen und initialisiert wird. Ein Instanz kann serialisiert werden.
JVM	Java Virtual Machine – eine Programmierumgebung, die sich durch Plattformunabhängigkeit und starke Objektorientierung auszeichnet. Bietet auch automatische Speicherverwaltung. Wird mit Hilfe der Sprache Java programmiert.
Klasse	Signatur eines Objektes, wonach die Instanzierung des Objektes folgt. Beschreibt die Vererbung, Interfaces, Attribute und Methoden der Klasse.
Merkmale	Die zu einem Objekt gehörigen Attribute und Methoden.
Methode	Beispielle: GetObjectSurface(): die Berechnung von abgeleiteten Größen eines Objektes. Validate(): Validier,t ob ein Objekt gemäß der Klassenbeschreibung noch koherent ist.
Methoden	Methoden werden in den Objekten gekapselt, mitvererbt oder überschrieben. Sie können auch als Abstract definiert werden, müssen in einem solchen Fall überschrieben werden. Sie stellen objektinterne Routinen dar, die sowohl objektinterne als auch globale Attribute bedienen und meistens als Ergebnis wiederum Attribute modifizieren. Zum Beispiel: GetObjectSurface(): die Berechnung von abgeleiteten Größen eines Objektes. Validate(): Validiert ob ein Objekt gemäß der Klassenbeschreibung noch kohärent ist.
Objekt	Objekte sind die grundlegenden Bestandteile des Systems,

	sie bilden bauspezifische Identitäten ab („Stütze“, „Fenster“, „Risiko“) und werden durch Eigenschaften, Attribute und Methoden definiert. Ein Objekt ist das kleinste Element innerhalb eines Projektes und stellt den gegenwärtigen Planungs- oder Ausführungsstand dar. Objekte mit wenigstens einem gleichen Attributwert können in Objektgruppen = I-Collections zusammengefasst werden.
Objektorientierte Datenbanken	Objekte werden nicht wie in relationalen Datenbanken zerlegt, sondern als binäre Kontinui gespeichert. Der Zugriff erfolgt über einen fortlaufend aktualisierten Zeiger-Cache, womit erhebliche Performance-Steigerungen gegenüber relationalen Datenbanken erzielt werden. Nach der Referenzierung eines Objektes, werden dessen Attribute, Eigenschaften und Methoden mittels üblicher OO Techniken abgerufen.
OO	Objektorientiert . Eine Form der Programmierung bei der Erstellung von Software, die sich am objektorientierten Paradigma anlehnt. Es wird mit Hilfe von Vererbung eine Klassenhierarchie entwickelt, Instanzen dieser Klassen bilden die Objekte, die vom Programm eingesetzt werden, um den gewünschten Zielen zu entsprechen.
OSI	Open Standards Interconnection. Ein vom ISO (International Standards Organisation) definiertes Referenzmodell für die Kommunikation offener Systeme.
Polymorphismus	Mehrfachvererbung. Die Möglichkeit einer Klasse von mehreren anderen Klassen, Schnittstellen, Methoden und Attribute zu erben. Bei der Vererbung von Schnittstellen müssen deren Methoden neu implementiert werden.
Projekt	Der Begriff Projekt wird hier ausschließlich synonym mit dem Bauprojekt als solchem verwendet.
Property	Properties (Eigenschaften) sind durch Regeln eingeschränkte Attribute.

Beispiel:

```
int höhe; // stellt die Höhe einer Stütze dar, sollte immer einen Wert >= 0 aufzeigen

 public int Höhe
  {
  get(return höhe;)
  set(höhe = value<0?höhe=0:höhe=Math.Abs(value);)
  /* Wert wird gesetzt, der Wert muss >= 0 sein,
  daher  nehmen wir 0 falls der Wert <0 ist,
 sonst den absoluten Wert.*/
  }
```

RMI — Remote Method Invocation, ähnlich wie RPC.

TCP/IP — Transmission Control Protocol/Internet Protocol. TCP arbeitet auf Ebene 4 des OSI Modells, setzt direkt auf IP (Schicht 3) auf. TCP ist ein verbindungsorientiertes End-to-End Protokoll, das eine folgerichtige und zuverlässige Datenübertragung gewährleistet. Zur Übertragungssicherung ist ein großer Protokoll Overhead nötig, der die Leistung etwas reduziert. Jedes Datagramm wird mit einem Header, der eine Mindestlänge von 20 Oktets hat, versehen. In diesem Header befindet sich auch eine Folgenummer, mit der jedes Datagramm so durchnummeriert wird, dass die richtige Reihenfolge der Pakete erkannt werden kann.

Bei einem Netzwerkverbund können so die einzelnen Pakete auf unterschiedlichem Weg zu ihrem Ziel gelangen.

IP deckt OSI Schicht 3 ab. IP hat die Aufgabe Datagramme über mehrere Rechner hinweg an den Zielrechner zu liefern, überlässt jedoch Reihenfolge und Sicherung der Ablieferung dem TCP.

Tools — Als Tools werden externe Applikationen bezeichnet, die Daten aus einem System aufnehmen und weiterverarbeiten. Die

	Tools dienen zur Visualierung und der Bearbeitung der Objekte. Mit einigen Tools wird daher eine bidirektionale Beziehung aufgebaut, so dass die Veränderungen und Ergebnisse aus der externen Bearbeitung wieder in das System eingelesen werden können.
Vererbung:	Auch Mehrfachvererbung (Polymorphismus). Die Möglichkeit einer Klasse von einer oder mehreren anderen Klassen, Schnittstellen, Methoden und Attributen zu erben. Bei der Vererbung von Schnittstellen müssen deren Methoden neu implementiert werden.
Viewers	Ein Software Tool oder Plugin zur anschaulichen Darstellung eines geometrischen Gebildes, oft auf den X3D Standard bezogen, steht jedoch auch im Zusammenhang mit CAD zur Verfügung.
Visualisieren	Visualisieren ist die Darstellung von geometrischen Objekteigenschaften als optische Wahrnehmung.
Virtuell	Virtuell wird in der IT Welt die Bezeichnung für Gegenstände, Geräte und Vorgänge eingesetzt, die nicht tatsächlich (also nur als Illusion) vorhanden sind.
Virtuelle Realität	Unter virtueller Realität (VR) fasst man alle Methoden und Techniken zusammen, die benötigt werden, um einen Menschen in eine vom Computer erzeugte künstliche, dreidimensionale Umgebung zu versetzen. Um diese dreidimensionalen Angebote bereitstellen zu können, musste eine Sprache entwickelt werden, mit deren Hilfe dreidimensionale Objekte beschrieben werden können. Es entstand die Virtual Reality Modelling Language (VRML, neuerdings X3D). Diese Beschreibungssprache ist plattformübergreifend (d.h. an keinen Rechnertyp gebunden), erweiterbar und auch bei niedrigen Übertragungsbandbreiten einsetzbar. Die VR ist ein Teilgebiet der künstlichen Intelligenz. Vgl. Bibliographisches Institut & F. A. Brockhaus AG, Wies-

baden, 2002.

V Literaturverzeichnis

Adachi, Y. ; Forester, J. ; Hyvarinen, J. (2002)
Industry Foundation Classes: IFC 2X – International Alliance for Interoerability, IAI Specification, Model Support Group IAI, 2002

Ahmad, I. ; Azhar, S. (2002)
Data Warehousing in Construction: From Conception to Application, First International Conference on Construction in the 21st Century, (CITC2002), Miami, 25-26 April, 2002

Anderson, R. (1999)
Professional XML, Birmingham, Wrox, 1999
ISBN: 1-861003-11-0

Anderson, R. (2002)
Professional ASP.NET, Birmingham, Wrox, 2002
ISBN: 1-861004-88-5

Anfortas (2004)
The trouble with abstract classes, Livejournal: Hackward
URL: http://www.livejournal.com/community/hackward/3671.html, Zugriff am 11.03.2005

Ardestani, K., Ferracchiati, F., Gopikrishna, S. et al. (2002)
Visual Basic.NET Threading, Birmingham, Wrox, 2002
ISBN: 1-861007-13-2

Arnold, K. ; O'Sullivan, B. ; Scheifler, B. (1999)
The JINI Specification, Reading, Massachusetts, Addison-Wesley, 1999
ISBN: 0-201-61634-3

Auld, C. (2003)
Building an XSL Transformation Filter in ASP.Net, ASPToday Magazine, Apress,

2003
URL: http//www.asptoday.com/Forums.aspx?Subject=O&Content=2247

Aust, E. ; Niemann, H-R. ; Seeliger, D. (1998)
Meeresroboter, Heidelberg, Spektrum Der Wissenschaft, Dossier 4, SS 86-89, 1998

Bakkmoen, K. 1996.
The Relationship between services offered by ICIS members and International and Regional Standards. Paper to the ICIS Delegates Assembly in Sydney, 15.04.1996. Norwegian Council for Building Standardization.

Banerjee, A. ; Corera, A. ; Greenvoss, Z. (2001)
C# Web Services, Birmingham, Wrox, 2001
ISBN: 1-861004-39-7

Bar-Cohen, Y. ; Mavroidis, C. ; Bouzit, M. (2001)
Virtual reality robotic telesurgery simulations using MEMICA haptic system, Proceedings of SPIE's 8th Annual International Symposium on Smart Structures and Materials, P 1-7, Newport, California, 2001

Barwell, F. ; Blair, R. ; Case, R. (2001)
Professional VB.NET, Birmingham, Wrox, 2001
ISBN: 1-861004-97-4

Beshear, F. ; Raju, S. ; Evdemon, J. (2005)
IMS General Web Services UML to WSDL Binding Auto-generation Guidelines, Version 1.0 Public Draft Specification, IMS Global Learning Consortium, Inc., 2005

Booch, G. ; Rumbaugh, J. ; Jacobson, I. (2000)
The Unified Modeling Language User Guide, Reading, Massachusetts, Addison-Wesley, 2000
ISBN: 0-201-57168-4

Bouchard, O. (1996)

JAVA – Objektorientiert Programmieren für das WWW, München, dtv, 1996
ISBN: 3-423-50185-5

Box, D. (1998)
Essential COM, Reading Massachusetts, Addison- Wesley, 1998
ISBN: 0-201-63446-5

Box, D. (2001)
House of Com: Migrating Native Code to the .NET CLR, Microsoft Developer Network Magazine, P 1-7, Redmond, Microsoft Corporation, 2001

Box, D. (2001)
House of Web Services: Moving to .NET and Web Services, Microsoft Developer Network Magazine, P 1-13, Redmond, Microsoft Corporation, 2001

Box, D. (2005)
Code Name Indigo: A Guide to Developing and Running Connected Systems with Indigo, Microsoft Developer Network Magazine, P 1-9, Redmond, Microsoft Corporation, 2005

Brutzman, D. (2002)
X3D Examples and VRML Sourcebook Examples, USA, Web 3D Consortium, Extensible 3 D, 2002

Building Design and Construction (2000)
Construction trends and the Virtual World, Building Design and Construction, vol. 1, P 12, 2000

Burdea, G. (1993)
Virtual Reality Systems and Applications, In Electro 93, International Conference, P 164-, New York, Edison, 1993

Cazzulino, D. (2004)
Code Generation in the .NET Framework Using XML Schema, Microsoft Developer

Network Library, Redmond, Microsoft Corporation, 2004

Cazzulino, D. (2005)
Customizing XSD->Classes code generation, the „easy“ way, eXtensible Mind, 2005
URL: http://weblogs.asp.net/cazzu/archive/2003/10/24/33302.asp

Cepa, V.; Mezini, M. (2004)
Language Support for Model-Driven Software Development, Paper – Department of Computer Science, Darmstadt University of Technology, Germany, 2004

Clayton, M. ; Warden, R. ; Parker, W. (2002)
Virtual Construction of Architecture using 3D CAD and simulation, Automation in Construction, Vol. 11, PP 227-235, Amsterdam, Elsevier-Science Publisher, 2002

Clifton, M. (2005)
Comparing Declarative and Imperative Programming, The Code Project, 2005.
URL: http://www.codeproject.com/csharp/CompareDecImp.asp.

Citrin, W. (2004)
Calling Java Classes Directly from .NET, devx-Jupitermedia Corporation, 2004
http://www.devx.com/interop/Article/19945

Corno, F. (2004)
VRML and X3D: Modeling, Torino, Politecnico di Torino, Dipartimento di Automatica e Informatica, 2004

Dai, J. ; Su, X. (2002)
Simulated Robotic Hot-line Maintenance Based on Virtual Reality, Proceedings of the 4th World Congress on intelligent Control and Automation, Shandong University of Science and Technology, 2002
ISBN: 0-7803-7268-9 701

Davis, L. (1998-2007)
CAN Bus. Controller Area Network. ISO 11898/11519. Internet, URL:

http://www.interfacebus.com/Design_Connector_CAN.html. Retrieved: 30.08.2007

Dawood, N. ; Sripsrasert, E. ; Mallasi, Z. (2002)
Development of an integrated information resource base for 4D/VR construction processes simulation, Automation in Construction, Vol.12, P P 123-131, Amsterdam, Elsevier-Science Publisher, 2002

DeCou, J. ; Timberlake, T. ; Dooley, R. (2002)
Virtual reality modeling and computer-aided design in pediatric surgery: applications in laparoscopic pyloromyotomy, Pediatric Surgery International,
Vol. 18, no. 1, PP 72-74, Amsterdam, Elsevier-Science Publisher, 2002
ISSN: 0179-0358

Dhawan, P. (2002)
Performance Comparison: .Net Remoting vs. ASP.NET Web Services, S 10, Redmond, Microsoft Developer Network, 2002

Ferretti, G. ; Filip, S. ; Maffezzoni, C. (1999)
Modular Dynamic Virtual-Reality Modeling of Robotic Systems, IEEE Robotics & Automation Magazine, 1070-9932/99, London, The Building Publisher, 1999

Fielding, R. (2000)
Architectural Styles and the Design of Network-based Software Architectures, Dissertation, Irvine, University of California, 2000

Fischer, M. (1993)
Linking CAD and Expert Systems for Constructability Reasoning, Proceedings of the 5th Conference on Computing in Civil and Building Engineering, ICCCBE, Anaheim, 1993

Fourie, W. (2004)
An XSD to .NET language code class generator that adds real punch to Microsoft's XSD Tool, South Africa, Halwayhouse, 2004
URL: http//www.codeproject.com/soap/CodeXS.asp

Freeman, H. (1999)
JavaSpaces, Principles, Patterns and Practice, New York, Allison Wesley, 1999
ISBN: 0-201-30955-6

Gehringer, H. ; Rust, H. (1998)
Fernbediente Brandbekämpfung, Heidelberg, Spektrum Der Wissenschaft, Dossier 4, SS 72-73, 1998

Geiger, C. (2004)
Stereoskopische Techniken und autostereoskopische Displays: Helft mir, Obi-Wan Kenobi, iX - Magazin für Professionelle Informationstechnik, Bd. 5, Hannover, Heyse Verlag, 2004

Getz, K. (2005)
Advanced Basics: Doing Async the Easy Way, The Microsoft Journal for Developers, Microsoft Corporation, Redmond, March 2005
URL:http://msdn.microsoft.com/msdnmag/issues/05/03/AdvancedBasics/default.aspx

Gibbs, W. (1994)
Software's Chronic Crisis: Trends in computer, Scientific America, P P1-13, 1994
URL: http://pdownland.see.plymouth.ac.uk/modules/COMP212/gibb.htm

Gralla, M. (2001)
Garantierter Maximalpreis, Stuttgart, Teubner Verlag, 2001
ISBN: 3-519-05056-0

Grimes, R. (1997)
Professional DCOM Programming, Birmingham, Wrox, 1997,
ISBN: 1-861000-60-X

Grimes, R. ; Stockton, A. ; Reilly, G. (1998)
Beginning ATL COM Programming, Birmingham, Wrox, 1998
ISBN: 1-861000-11-1

Gunderloy, M. ; Chipman M. (1999)
MS SQL Server 7, Düsseldorf, Sybex Verlag, 1999
ISBN: 3-8155-5517-5

Gunnerson, E. (2002)
A Programmer's Introduction to C#, New York, Apress, 2002
ISBN: 1-893115-86-0

Hartman, J. ; Wernecke, J. (1996)
The VRML 2.0 Handbook, Reading, Massachusetts , Addison Wesley Developers Press, 1996,
ISBN: 0-201-47944-3

Helmerich, R. ; Schmidt, P. (1985)
CAD-Grundlagen, Würzburg, Vogel-Buchverlag, 1985
ISBN: 3-8023-0805-0

Henckels, D. (2004)
Fehldatenmodellierung: Eine intuitives Informationsaustauschmodell für den rechnergestützten Bauwerkslebenszyklus, Karlsruhe, Institut für Industrielle Bauproduktion Universität Karlsruhe (TH), SS 1-104, 2004

Hepperle, M. (2004)
An XML Airfoil Geometry Format, MH-AeroTools, 2004
URL: http://ww.mh-aerotools.de/airfoils/xml_airfoil_format.htm

Herbst, C. (1998)
Roboter in der Sicherheits- und Kerntechnik, Heidelberg, Spektrum Der Wissenschaft, Dossier 4, SS 36-38, 1998

Herold, H. ; Unger, W. (1988)
C Gesamtwerk, München, Te-wi Verlag, 1988
ISBN: 3-89362-015-X

Hirzinger, G. (1998)
Intelligente Roboter: Raumfahrtroboter – Prototypen für die Industrie?, Heidelberg, Spektrum Der Wissenschaft, Dossier 4, SS 54-60, 1998

Hölkermann, O. (2002)
Modell eines Prozessorientierten Informationssystems zur Steuerung von Bauunternehmen, Weißensee Verlag, Berliner Beiträge zum Bauwesen Band 1, 2002

Hunter, D. (2000)
Beginning XML, Birmingham, Wrox, 2000
ISBN: 1-861003-4-12

JNBridge (2004)
User Guide, 2004

Kapellmann, D. u. Schiffers, K.-H. (2000)
Vergütung Nachträge und Behinderungsfolgen beim Bauvertrag, 3. Aufl., Düsseldorf, Werner-Verlag, 2000
ISBN: 3-8041-4965-0

Kass, M. (2001)
X 3D Conformance Test Suite, USA, National Institute of Standards and Technology, 2001

Kay, M. (2000)
XSLT Programming Reference, Birmingham, Wrox, 2000
ISBN: 1-861003-12-9

Kohler, N. ; Lockemann, P. (2003)
Arbeits- und Entwicklungsbericht: Planungsplattform für Dynamische Gebäude, KO 1488/3-1, 3-2, SS 2-18, Karlsruhe, Institut für industrielle Bauproduktion (IFIB), 2003

Körner, H. ; Volkhard, F. ; Böttcher, P. (1991)

Fertigungstechnik im Ingenieurbau: Planung und Steuerung von Prozessen durch Simulation, Forschungsbericht T 2353, SS 1-108, Kassel, Gesamthochschule Universität Kassel, IRB-Verlag, 1991

Kuntze, H-B. u. Haffner, H. (1998)
Inspektion von Abwasserkanälen, Heidelberg, Spektrum Der Wissenschaft, Dossier 4, SS 82-85, 1998

Kupernas, J., Um, J. and Booth, C. (1995)
Use of CAD Three-Dimensional Modeling as a Constructability Review Tool, Proceedings of the 2nd Congress on Computing in Civil and Building Engineering, Atlanta, 1995

Langen, W. und Schiffers, K.-H. (2005)
Bauplanung und Bauausführung, München, Werner Verlag, 2005

Leithold, L. (1968)
The Calculus with analytic Geometry, New York, Harper & Row, 1968

Li, S. and Econopoulos, P. (1998)
Professional COM Applications with ATL, Birmingham: Wrox, 1998
ISBN: 1-861001-7-03

Lipman, R. and Reed, K. (2000)
Using VRML In Construction Industry Applications, Web3D – VRML 2000 Symposium, Monterey, California,
February 21-24, 2000

Loughran, S. and Smith, E. (2005)
Rethinking the Java SOAP Stack, IEEE International Conference on Web Services (ICWS) 12-15 July 2005, Orlando, Florida, USA, Bristol: Internet Systems and Storage Laboratory, HP Laboratories Bristol, 2005

Maine, S. (2003)

Don Box on the future of distributed applications (a.k.a. the Indigo Tech Preview), Brain.Save (), Steve maine's blog, 2003

Manavazhi, M. (2001)
A software architecture for the virtual construction of structures, Advances in Engineering Software, Vol. 32, SS 545-554, Amsterdam: Elsevier Science Publishing, 2001

Maruyama, Y and Iwasw, Y. (2000)
Development of virtual and real-field construction management systems in innovative, intelligent field factory, Automation in Construction Vol. 9, PP 503-514, Amsterdam: Elsevier Science Publishing, 2000

Mechnig, M. (1998)
Die Anpassungsfähigkeit der baubetrieblichen Produktionsplanung und –Steuerung an interne und externe Einflüsse, Dissertation, Dortmund: Universität Dortmund, 1998

Model Support Group of the IAI (2000)
INDUSTRY FOUNDATION CLASSES - 2x , International Alliance for Interoperability, IAI, 2002
URL: http://www.iai-international.org/ifcXML2/RC2/IFC2X2_FINAL/ifcXML%20Implementation%20Guide%20v1-0.pdf

Moore, D. and Tunnicliffe, A. (1995)
An Automated Design Aid (ADA) for Constructability, Proceedings of the 2nd Congress on Computing in Civil Engineering, New York, 1995

Müller, S., u. Weber, K-H. (1998)
Roboter – Basis der industriellen Produktionstechnik, Heidelberg: Spektrum Der Wissenschaft, Dossier 4, SS 12-15, 1998

Neethling, D. ; Horn, T. (2004)
Information und Kommunikation für Baugroßgeräte, Fachtagung zu logistischen Di-

Dienstleistung im Bauwesen insbesondere bei der Ver- und Entsorgung, Dortmund, Fraunhofer Institut für Baulogistik, 2004

Nishigaki, S. ; Maruyama, Y. ; Iwase, Y. (1999)
Development of virtual and real-field construction management systems in innovative, intelligent field factory, Proceedings of the 16th IAARC/IFAC/IEEE International Symposium on Automation and Robotics in Construction, Automation and Robotics in Construction XVI, Madrid, UC3M, 1999

Obasanjo, D. (2003)
XML Schema Design Patterns: Is Complex Type Derivation Unnecessary?, O'Reilly Media Inc., 2003
URL: http://www.xml.com/pub/a/2003/10/29/derivation.html

Och, C. (2000)
A Component-based Data Mediator Definition Language to Support Heterogeneous Database Integration and Evolution, Thesis, Department of Computer Science University of Colorado, Colorado, 2000

O'Conner, J. ; Rush, S. ; Schulz, M. (1987)
Constructability Concepts for Engineering and Procurement, Construction Engineering and Management, Vol. 113, no. 2, P P235-248, 1987

Parallelgraphics, Cortona Software Development Kit 4.1 (2002-2003). ParallelGraphics Ltd.

Partridge, C. (1996)
Being There, Newspaper for the Design & Construction, London, The Building Publisher, Vol. 261, no.19, PP 46-49, 1996

Patty, B. ; McCullough, B. ; Vanegas, J. (1995),
Automating Constructability during Design, Proceedings of the 1995 Congress on Computing in Civil Engineering, San Diego, 1995

Platt, S. (1999)
COM+ verstehen: Die Möglichkeiten der Unternehmens-Programmierung mit COM+, Unterschleißheim, Microsoft Press, 1999
ISBN: 3-86063-610-3

Prescod, P. (2002)
Rest and the Real World, USA, O'Reilly Media Inc, Published on XML.Com URL: http://www.xml.com/pub/a/ws/2002/02/02/rest.html, 2002

Primavera Systems Inc. (1996)
Primavera Project Planner, Reference, Bala, Primavera Systems Inc., 1996
ISBN: 1-57408-022-9

Radeck, K. (2003)
C# and JAVA: Comparing Programming Languages, USA, Windows Embedded MVP, 2003

Rains, D. (2003)
4D Modeling on the Burnett River Dam Tender, Australia: A Case Study in Virtual Design and Construction, Australia, South Brisbane, Thiess Pty Ltd, 2003

Rammer, I. (2004]
.Net Remoting Use-Cases and best Practices, Internet, Thinktecture, 2004
URL: http://www.thinktecture.com/Resources/RemotingFAQ/RemotingUseCases.html

Richter, J. (2002)
Applied Microsoft .NET Framework Programming, Redmond, Microsoft Press, 2002
ISBN: 0-7356-1422-9

Russell, J. (2005)
Generate .NET Code in Any Language Using CodeDOM, Article/15678, Internet, Jupitermedia Corporation: DevX, 2005
URL: http//www.devx.com/dotnet/Article/15678

Saxon, S. (2003)
Xpath Querying Over Objects with ObjectXPathNavigator, Redmond, Microsoft Developer Network, Microsoft Corporation, 2003
URL: http://msdn.microsoft.com/library/en-us/dnexxml/html/xml0317200

Schittko, C. (2004)
Troubleshooting Common Problems with the XmlSerializer, Microsoft Developer, Redmond, USA, Network, Microsoft Corporation, 2004
URL: http://msdn.microsoft.com/library/en-us/dnxmlnet/html/trblshtxsd.asp

Schmid, D. (1998)
Virtuelle und telematische Roboter, Heidelberg, Spektrum Der Wissenschaft, Dossier 4, S S32-35,1998

Schmitt, A. ; Deussen, O. ; Kreeb, M. (1994)
Graphisch geometrische Algorithmen, 2. Aufl., Vorlesungsskript, Karlsruhe, Institut für Betriebs- und Dialogsysteme Abteilung Dialogsysteme und graphische Datenverarbeitung Universität Karlsruhe, 1994

Schraft, D. ; Müller, E. (1998)
Roboter gestern, heute, morgen, Heidelberg, Spektrum Der Wissenschaft, Dossier 4, Essay, SS 6-11, 1998

Schweimeier, R.
XML Schema, Lecture 4 Notes, P 1-12, Berlin, Fachhochschule für Technik u. Wirtschaft,
URL: http://www.cs.rpi.edu/~puninj/XMLJ/classes/class4/all.html

Schweitzer, G. ; Zlatnik, D. ; Honegger, M. (1998)
Roboter als Partner – das neue Paradigma, Heidelberg, Spektrum Der Wissenschaft, Dossier 4, SS 61- 63, 1998

Seitz, S. (2001)

Maxon Cinema 4D Art und XL7: Visualisierung und Animation von CAD-Konstruktion, München, Addison Wesley Publisher, 2001
ISBN: 3-8273-1907-2

Singhal, S. ; Zyda, M. (2000)
Networked Virtual Environments: Design and Implementation, Reading, Massachusetts, Addison-Wesley, 2000
ISBN: 0-201-32557-8

Sokolnikoff, I. (1966)
Mathematics of Physics and Modern Engineering, New York, McGraw-Hill, 1966
ISBN: 65-28518

Sorcus (2001)
MAX5pci, MAX6isa, MAX3pc104, und MAX-Module incl. MAX-PC, Infoblatt, Heidelberg, Sorcus Computer GmbH, 2001

Stewart, S. (2005)
re: Has .Net Remoting become an exercise in academia?, Internet, dotnetjunkies, 2005
URL:http://dotnetjunkies.com/WebLog/ssteward/archive/2005/01727/48819.aspx

Stewart, S. (2003)
Working with Events Over Remoting, Internet, dotnetjunkies, 2003
URL: http://www.dotnetjunkies.com/Tutorial/BFB598D4-0CC8-4392-893D-30252E2B3283.dcik

Strauss, L. (1982)
Wiskunde vir Natuurwetenskaplikes, Johannesburg, McGraw-Hill, 1982
ISBN: 0-07-450623-4

Sussman, D. ; Homer, A. (1998)
ADO 2.0, Programmer's Reference, Birmingham, Wrox, 1998

ISBN: 1-861001-83-5

Thabet, W. (2002)
Design/Construction Integration through Virtual Construction for Improved Constructability, Paper, PP 1-4, Virginia, Virginia Tech, 2002

Thielens, J. (2003)
Exposing custom-made classes through a Webservice and binding them to a DataGrid., CodeProject, 2003
URL: http://www.codeproject.com/vb/net/LeaditWebServiceWrapper.asp

Trowbridge, D. ; Hohpe, G. ; Newkirk, J. (2003)
Enterprise Solution Patterns using Microsoft .NET: patterns & practices,
Version 1.0, Microsoft Corporation, 2003

Vince, J. (1998)
Essential Virtual Reality fast: How to Understand the Techniques and potential of Virtual Reality, London, Springer Verlag, 1998
ISBN: 1-85233-012-0

Von der Heyde, M. (2001)
A Distributed Virtual Reality System for Spatial Updating. MPI Series in Biological Cybernetics, No. 2, January 2001.
ISBN 3-89722-781-9

Walsh, A. ; Bourges-Sevenier, M. (2001)
Core Web3D, New Jersey, Prentice Hall, 2001
ISBN: 0-13-085728-9

Walsh, A. (2002)
Understanding Scene Graphs, do7wals.x.g2db, 2002.
http://www.ddj.com/documents/s=7217/ddj0207a/0207a.htm

Weyer, C. (2004)

A.NET Library to dynamically call Web services at runtime, Thinktecture's, Version 1.5, PP 1-4, 2004

Willenbacher, H. (2002)
Interaktive Verknüpfungsbasierte Bauwerksmodellierung: als Integrationsplattform für den Bauwerkslebenszyklus, Diss., Weimar, Bauhaus-Universität Weimar, 2002

Wörn, H. (1998)
Tendenzen in der Robotik: Industrieroboter- billiger, leichter, intelligenter, Heidelberg, Spektrum der Wissenschaft, Dossier 4, SS 16-20, 1998

Workshop Virtual Construction (2003)
Essen, Hochtief AG, 19 März, 2003

Wylie, C. (1966)
Advanced Engineering Mathematics, 3rd ed., New York, McGraw-Hill, 1966

Yang, J. (1999)
Introduction to OQL, Computer Science 145 Lecture Notes, #15, USA, 1999

Zenk, A. (1999)
Lokale Netze: Die Technik fürs 21. Jahrhundert, 6. Aufl., München,
Addison-Wesley, 1999
ISBN: 3-8273-1592-1

D Anhang

Die CD anbei enthält alle Anhänge

Anhang Teil A – Details zu den Kapiteln

Anhang Teil B – Klassen für die Modellierung

Anhang Teil C – Listing des Objektmodells

Anhang Teil A – Details zu den Kapiteln

A.I – Aspekte der objektorientierten Programmierung

1.1 Imperative und Deklarative Programmiersprachen

Folgende Tabelle (Abb. 153) führt einige aktuelle imperative Sprachen auf:

Sprache	Beschreibung	VM	OS
C++	Objektorientierte Version von C	-	Unix, Solaris, Linux, Win32/Win64
Java	SUN Microsys proprietäre Sprache, erste multi-plattform VM-fähige Sprache	Java Virtual Machine	Unix, Solaris, Linux, Win32, BEOS, Mac
MC++	(Managed C++) VM-fähige Version von C++	.NET Common Language Runtime	Win32, Solaris, Linux (Mono)
C#	Weiterentwicklung von C++ zur reinen objektorientierten Sprache mit Entfernung aller Elemente die in C++ zur Instabilität und Unsicherheit der Lösungen beitrugen.	.NET Common Language Runtime	Win32, Solaris, Linux (Mono)
VB.NET	Weiterentwicklung von Visual Basic zur objektorientierten Sprache mit Erzwingung der Typisierung.	.NET Common Language Runtime	Win32, Solaris, Linux (Mono)

Abb. 153: Gegenüberstellung von Programmiersprachen der Generation IV

1.1.1 Imperative Programmiersprachen in .NET

.NET unterstützt eine wachsende Anzahl imperativer Sprachen, u.a. MC++, C#, VB.NET, OBERON, Component Fortran, JScript.NET. Diese Sprachen wurden in ihrer ursprünglichen Syntax modifiziert, um komponentenbasiert eingesetzt werden zu können. Das war wesentlich, um das Ziel eines gemeinsamen Frameworks für alle Sprachen zu ermöglichen. Es ermöglicht beim Programmieren das Nutzen von Klassen, die in anderen Sprachen geschrieben wurden. Damit geriet die Wahl einer Sprache eher zu einer Geschmacksfrage und nicht, wie sonst, einer Systemfrage. Alle Sprachen sollten ähnliche Möglichkeiten bieten, was sich jedoch nicht realisieren lässt auf Grund der Historie der verschiedenen Sprachen.

C# gilt als Sprache der Wahl durch ihre Nähe zu C++ und Java und die stark objektorientierte Syntax.

1.1.2 Deklarative Programmiersprachen

Die meisten W3C Sprachen sind deklarativer Art, beschreibender Natur. Sie eignen sich zur Umschreibung statischer Elemente, wie z.B. Dokumenten oder Geometrie. Dadurch, dass sie nicht imperativer Natur sind, d.h. nicht an einen Compiler gebunden, besitzen sie auch nicht die Möglichkeit der Anweisung von Rechenschritten, d.h. es gibt keine Syntax für Methoden. Das stellt natürlich eine Einschränkung beim Einsatz in das hier beschriebene System dar, da alle Objekte auch mit Methoden versehen werden sollten.

Steuerung imperativer Programmiersprachen mittels deklarativer Elemente moderner Programmierumgebungen wie die .NET Framework, ermöglichen die dynamische Generierung von Code und dessen Kompilierung und Instanzierung zur Laufzeit. Somit können Objekte im Arbeitsspeicher mit deklarativen Mitteln gesteuert werden. Das erlaubt die direkte Kopplung von fachspezifischen Klassenbeschreibungen und einsatzfähigen Objekten zur Laufzeit. In der .NET Framework befinden sich die dazu notwendigen Klassen in den Namespaces:

- System.CodeDom
- System.Runtime.Serialization.

Diese technologische Lösung des Dilemmas der einerseits zwar verständlichen aber abstrakten und nicht zur Objektsteuerung einsetzbaren deklarativen Sprachen, und der andererseits zur Steuerung benötigten aber dynamisch schwer veränderbaren imperativen Sprachen, bietet die Möglichkeit, komplexe Objektwelten dynamisch beeinflussen zu können und bildet somit die Grundlage, der in dieser Arbeit beschriebenen Lösung der objektorientierten Projektmodellierung.

1.2 Objektorientierte Technologien

1.2.1 Verteilte objektorientierte Programmierung

Der flächendeckende Einsatz von Datenbanken hat ein brauchbares Werkzeug zur Modellierung und Speicherung von Daten von Bauprojekten ermöglicht. Rechen- und datenintensive Vorgänge wie Statik, Kalkulation oder Terminplanung wurden aus offensichtlichen Gründen frühzeitig modelliert.

Die übliche Modellierung bestand aus einer Abbildung des Arbeitsprozesses mit der entsprechenden Datenhaltung mit Hilfe einer Datenbank. Dieser Ansatz, trotz des heutigen Einsatzes objektorientierter Technologien noch immer weit verbreitet, verkörpert ein in sich logisches Vorgehen, führt aber zu eher prozessorientierten, inkompatiblen und stark markenbezogenen Lösungen, z.B. AutoCAD, Primavera, Allplan, Technodata, PowerProject und Kubus.

Dadurch, dass Projekte multifunktional und multipersonell bestückt und bearbeitet werden, findet ein wachsender Transfer von Daten zwischen Rechnern und Applikationen statt [Henckels, D., 2004. S. 6]. Am Anfang der 90'er waren verteilte Protokolle wie CORBA, DCOM und auch schon JINI entsprechend weit entwickelt, um den Aufbau von intelligenten, interaktiven und verteilten Applikationen zu ermöglichen. Damit wurde auch die Integration von anderen Ressourcen wie z.B. (Bau-) Maschinen, Anlagen, Peripherie-Geräten und verteilten Informationssystemen (auch Internet/ Intranet) mit dem organisationsspezifischen PIS (Projekt Information System) und MIS (Management Informations- System) durchgeführt. Jedoch beschränkten sich die Lösungen durch die hohe Komplexität und den großen Aufwand und die mithin hohen Kosten auf relativ wenige Systeme.

Ein zusätzliches Problem stellten die sehr unterschiedlichen Arten der Speichermedien dar, da meistens Markenlösungen bei der Wahl der Datenbanken bevorzugt wurden. Verteilte Systeme mit hohen Datenvorkommen erfordern komplexe Datentransformationen, um verschiedene Aspekte eines Modells zu beleuchten. Sie können bei hohen Datenvolumen unhandlich werden.

Die Spezifizierung von XML als Public Domain Datenformat durch das W3C und die große Unterstützung durch die Industrie und öffentliche Organisationen hat Lösungen zum Austausch sehr unterschiedlicher Datenbankformate erheblich beschleunigt. Dieses einheitliche Datenformat, und dessen Manipulationssprache XSLT und XPath lösen zwar Differenzen in den Modellierungsarchitekturen nicht auf und vereinfachen nicht immer den Einsatz verteilter Protokolle (z.B. direct Coupling), ermöglichen aber die Überbrückung unterschiedlicher Datenbank Schemata. Die offene und frei zugängliche Art der XML Spezifikation sowie die allgemeine Erkenntnis der Software-Industrie, dass eine XML-Datenbasis oder zumindest -Kompatibilität die Langlebigkeit der Lösungen sichert, hat auch zur rasanten Annahme der Technologie geführt und damit die Attraktion der Implementierung erhöht.

Der Erfolg des XML Vorgängers, ASCII-Format, stützt diese Feststellung. Ein großer Vorteil der XML-basierten Modellierung ist, dass die Wahl der Datenbank dadurch sehr vereinfacht wird, da damit auch Objektorientierte Modellierung in einer relationalen Datenbank ermöglicht wird, und die strikte Objektdefinition der Objektdatenbanken auch relativiert wird. Entscheidender werden Eigenschaften wie die Unterstützung von verteilten Protokollen, die Leistungsfähigkeit und Skalierbarkeit des DBMS, Zugriffsmöglichkeiten, Handhabung und Kosten. Die Kopplung der RPC Technik mit XML hat zur Einführung der sehr erfolgreichen XML Services (XML Dienste), mit SOAP als führendem Protokoll, geführt. Da SOAP textbasiert ist und nicht auf den Arbeitsspeicher begrenzt ist, kann dadurch ein viel weiteres Spektrum von Problemstellungen im verteilten Bereich gelöst werden. Allerdings gibt es Schwierigkeiten bei diesem Protokoll, unter anderem die Inkompatibilität zwischen XML Schema Typen und programmatischen Typen (programmatic types). Diese hat Ihren Ursprung in der Ungleichheit von in-memory und serialisierten Daten [Box, D., 2001. S. 4].

Verteilte (cross-app/cross-machine) Protokolle ergeben andere Probleme. Die Technologie ist komplex und die Rivalität der wichtigsten Lösungsanbieter immens. Es werden sowohl Public Domain wie Markenlösungen für unterschiedliche Szenarien angeboten (messaging, non-critical RPC, real-time RPC). Technologisch sind die meisten Lösungen ausgereift, so dass eine erfolgreiche Implementierung, jedoch nicht-trivial, leistungsfähig möglich ist. Diese Art der Technologie ist im Modellierungs- und Erfassungsbereich entscheidend und wird der Motor vieler weiterer Entwicklungen sein. Nur das Zusammenspiel verteilter Applikationen führt zur Überwindung der Schranken des Client-Server Paradigmas. Allerdings sollte auch darauf geachtet werden, dass die Unterschiede in der Performance der Technologien abhängig sind von dem anvisierten Einsatz, den Datenmengen, Wiederholungsraten und anderen Faktoren, so dass bei dem Entwurf eines Systems der Zieleinsatz bedacht werden sollte [Dhawan, 2005. S. 9].

Nach der Wahl eines einheitlichen Datenformats und einer geeigneten RPC Technologie (HTTP, CORBA, DCOM, JINI, etc.) steht als Nächstes die Modellierung an. Diese Aufgabe ist nicht trivial, da weitere Lösungen automatisch zur Datenprofilierung führen und für weitere Entwicklungen gesorgt werden muss. Kriterien, die gründlich untersucht werden müssen, beinhalten:

- Eindeutigkeit der Klassen,
- Objekt Verfolgung,
- Erzwingung von kosten- und terminlicher Überarbeitung/Aktualisierung bei Entwurfsänderungen,
- Verträglichkeit zwischen den Phasen,
- Allgemeingültigkeit,
- Einhaltung des Schemas, Verteilung

1.2.2 Schnittstellen (Interfaces)

Bei den neuen Framework-orientierten Sprachen (JAVA, C#) ist nur eine Einfachvererbung vorgesehen. Die Begründung liegt in der Unübersichtlichkeit, die bei der Polymorphie (C++, MC++) entsteht. Hier kommt es oft vor, dass Attribute und Methoden aus unterschiedlichen Klassen aufgrund gleicher Benennung überschrieben werden, so dass nicht immer eindeutig klar ist, welche Attribute oder Methoden bei einem Aufruf zur Geltung kommen.

Durch die Einfachvererbung entsteht ein Verlust an Flexibilität. Zusätzlich schränkt die Einfachvererbung die Anzahl der Quellen ein. In manchen Situationen ist eine Polymorphie wünschenswert, diese kann mit Hilfe von Interfaces (Schnittstellen) erzielt werden. Im Gegensatz zur Polymorphie, bei der eine implizite Deklaration aller Methoden stattfindet (die Signaturen werden automatisch vererbt, nur nach Wunsch überladen), werden hier alle mittels der Interface übernommenen Attribute, Properties und Methoden neu deklariert. Dies verhindert das versehentliche und ungewollte Überladen einer Signatur.

1.2.3 XML Schema als Klassenbeschreibung

Das in 1995 gegründete W3C hat sich zur Standardisierungs–Institution des Internets entwickelt. Namhafte Anbieter von System- und Entwicklungssoftware wie Microsoft, Sun, Oracle, Telekom und auch öffentliche Institutionen wie führende Institute, Universitäten, Behörden und die Open Source Gemeinde sind in den Gremien des W3C repräsentiert, unterstützen deren Standards und setzten diese konsequent in die Praxis um. Das Konsortium wird hauptsächlich vom Massachusetts Institute of Technology (MIT), the Institut National de Recherche en Informatique et Automatique (INRIA) und dem Center for European Particle Research (CERN) betrieben.

Mittlerweile kann W3C Konformität als Standard in der Softwareentwicklung eingesetzt werden, vor allem im Bereich Internet-Technologien. Das W3C entwickelt eine Fülle von Spezifikationen zur Beschreibung von Datenformaten (XMLSchema, CSS, XSLT), Grafik (SVG, X3D) und Datenträgern (XML, HTTP, TCP). Für die Entwicklung eines dynamischen Objektmodells hat sich das XMLSchema als führendes Schema ausgeprägt.
XMLSchema befindet sich z.Zt. in Version 1.1. Der Standard umfasst folgende Bereiche:

- XML Schemas for Structures - http://www.w3.org/TR/xmlschema-1/ - Unterstützung von Schemazuordnung und -überprüfung.
- XML Schemas for Data Types - http://www.w3.org/TR/xmlschema-2/ - Unterstützung von Datentypen für XSD.

Das XMLSchema ist eigentlich eine Definitionssprache für XML. Ein XMLSchema Dokument wird mit der Endung XSD gespeichert, was aus dem XML Schema hervorgeht. Sämtliche Datenhaltungssysteme, ob Datenbanken, Schnittstellen, RTF, Dokumente aller Art, unterliegen Schemen. Diese sind sehr unterschiedlich und werden von vielen Anbietern meist proprietär eingesetzt. Offene Standards wie GAEB, ASCII, IFC werden auch nach gewissen Schemata definiert und verifiziert. Datenbanken bauen auf proprietären Schemata auf. Oft besteht die Hauptarbeit bei der Interaktion zweier Datenbanken durch die Übersetzung der beiden Schemata. ODBC ist eine Programmierschnittstelle, die den Zugriff auf beliebige Datenbanken, mit SQL als Datenzugriffssprache, ermöglicht. ODBC wurde von vielen Datenbank Betreibern unterstützt und kann als ein Vorgänger von XMLSchema gesehen werden. Im Übrigen gibt es eine W3C Working Draft für XQL, wobei die Universalität von SQL im XML Umfeld eingesetzt werden sollte.

XMLSchema beinhaltet alle Aspekte einer Schema Sprache. Dort werden Strukturen, Elemente und Datentypen unterstützt. Der Vorteil liegt in der relativ einfachen und kostenneutralen Handhabung, sowie der Möglichkeit, mit Hilfe von öffentlich zugänglichen Parser XML Dokumenten, anhand von XMLSchema zu verifizieren. Da mittlerweile alle namhaften Datenbanken das XMLSchema unterstützen, und es eine Fülle leistungsfähiger Parser gibt, ist das Verifizieren von XML aus einer Datenbank, sowie der Datenaustausch zwischen den Datenbanken unproblematisch geworden. Das Augenmerk hat sich auf den Datenaustausch im fachlichen Kontext verschoben. Wichtig ist, dass komplette Objekte serialisiert bzw. deserialisiert werden und nicht einzelne Datenströme die erst im Kontext einen fachlichen Sinn geben.

Zur Veranschaulichung der Vielfältigkeit des XMLSchemas ein Beispiel:
Die Definition einer einfachen Stütze. Die Stütze besitzt folgende Attribute:

- Höhe in m
- Breite in m
- Länge in m
- Materialangabe

Es können auch beliebig weitere Attribute definiert werden.

Die XMLSchema Repräsentation sieht folgendermaßen aus:

```
<xs:schema id="stuetze"
      targetNamespace="http://tempuri.org/stuetze.xsd"
      elementFormDefault="qualified"
      xmlns="http://tempuri.org/stuetze.xsd"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:complexType name="stuetze">
            <xs:sequence minOccurs="1">
                  <xs:element name="height" type="xs:double"/>
                  <xs:element name="width" type="xs:double"/>
                  <xs:element name="length" type="xs:double"/>
                  <xs:element name="material" type="xs:string"/>
            </xs:sequence>
      </xs:complexType>
</xs:schema>
```

Eine Instanz dieser Klasse „stuetze“ würde im XML Format folgendermaßen aussehen:

```
<stuetze>
      <height>3.0</height>
      <width>0.2</width>
      <length>0.3</length>
      <material>B35</material>
</stuetze>
```

Also eine Stütze mit den Dimensionen: Höhe 3 m, Breite 0,2 m und Tiefe 0,3 m und der Materialbezeichnung (DIN 276) B35. Das Interessante an dieser XML Repräsentation der Stütze ist, dass sie auch ein Pendant in C# (also einer imperativen Sprache) besitzt, nl.

Die SOAP Serialization:

```
[C#]
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Soap;
using System.Runtime.Serialization.Formatters.Binary;

namespace bau
{
      [Serializable()]
      public class stuetze
      {
            public int id;
            public double height;
            public double width;
            public double length;
            public string material

            public stuetze(int id)
            {
                  this.id = id;
            }

            public void serialize()
            {
            //Opens a file and serializes the object into it in
            //binary format.
                  Stream stream = File.Open(this.id.ToString()+".xml",
                        FileMode.Create);
                  SoapFormatter formatter = new SoapFormatter();

                  formatter.Serialize(stream, this);
                  stream.Close();
            }

            public stuetze deSerialize(int id)
            {
            //Opens „stuetze.xml" and deserializes the
                  Stream stream = File.Open(id.ToString()+".xml",
                        FileMode.Open);
                  SoapFormatter formatter = new SoapFormatter();

                  stuetze obj = (stuetze)formatter.Deserialize(stream);
                  stream.Close();
                  return (stuetze) obj;
            }

            public void Print()
            {
                  Console.WriteLine("height = '{0}'", height);
                  Console.WriteLine("width = '{0}'", width);
                  Console.WriteLine("length = '{0}'", length);
                  Console.WriteLine("material = '{0}'", material);
            }
      }
}
```

1.2.4 Serialisierung/DeSerialisierung der Klassen

Die Methoden *serialise()* und *deSerialise()* entkodieren die Objekte in XML-basierten Dateien, bzw. stellen die Objekte aus solchen Dateien wieder her mit Hilfe von einem SOAP Wrapper. Das serialisierte Objekt wird im XML Format wie folgt dargestellt:

```
<SOAP-ENV:Envelope xmlns:xsi="...">
      <SOAP-ENV:Body>
            <a1:stuetze id="ref-1" xmlns:a1="...">
                  <height>3</height>
                  <width>0.2</width>
                  <length>0.3</length>
                  <material>B35</material>
            </a1:stuetze>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Durch die Serialisierung/DeSerialisierung entsteht eine Schnittstelle zwischen dem leicht umgänglichen und vielfältigen XML Format und kompilierten Klassen. Das ermöglicht nicht nur die lokale Serialisierung von Objekten, sondern deren Weitergabe und Verteilung über ein Netzwerk mit Protokollen wie SOAP und .NET Remoting (in Anhang A.I.1.10 und A.I.1.11 beschrieben).

Das XMLSchema dient auch zur Verifizierung von solchen Objekten, da wegen der beliebigen Handhabungsmöglichkeiten eines solchen Objektes die Integrität des Objektes anhand einer Klassenbeschreibung garantiert werden muss.

Eine solche Verifizierung erfolgt mühelos:

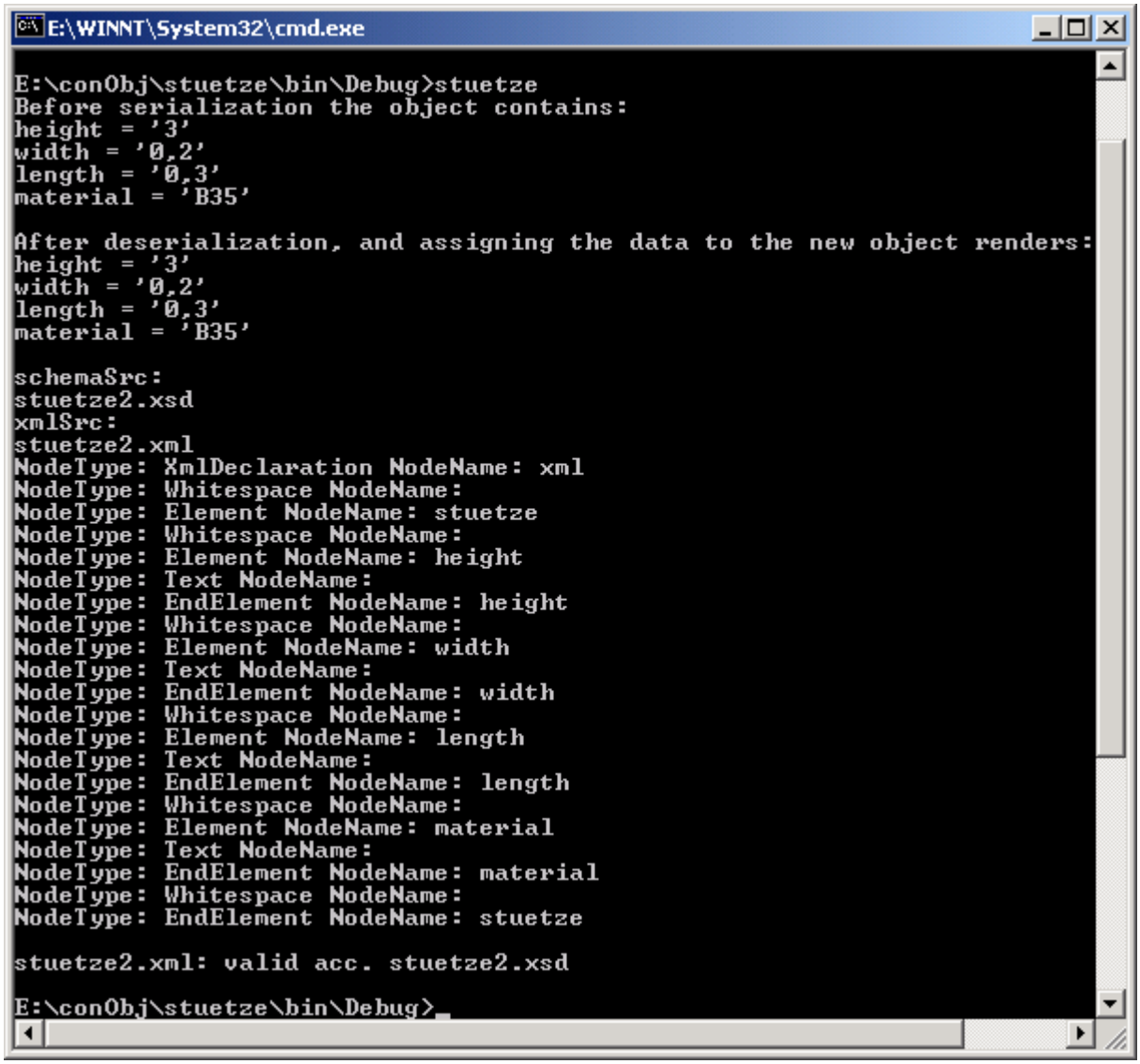


Abb. 154: Verifizierung einer XML Beschreibung eines Bauobjekts

Diese technologische Innovation hat im Baubetrieb große Anwendungsmöglichkeiten, wenn es gelingt, fachliche Prozesse objektorientiert auszurichten und diese fachlichen Vorgaben zu implementieren.

Zwei Anwendungsmöglichkeiten sind von besonderem Interesse:

- Integration von Daten aus verschiedenen Quellen zu einem fachlich sinnvollen Konstrukt
- Arbeiten mit verteilten Objekten (distributed computing)

Die Integration von Daten aus verschiedenen Quellen führt zu einer fachlichen Bereicherung, da erst im Kontext die einzelnen Informationen wirklich sinnvoll sind.

Das Arbeiten mit verteilten Objekten trägt der Tatsache Rechnung, dass viele Teilnehmer räumlich getrennt und auf unterschiedlichen Maschinen arbeitend, mit Hilfe von verteilten Technologien, interaktiv arbeiten können. Die starre Aufgabenteilung Client/Server wird dadurch auch aufgehoben, da bei verteilten Objekten alle Clients als Server, und alle Server als Clients agieren können. Die jeweilige Situation und Programmfolge bestimmt, ob ein Objekt als Client oder Server agiert.

1.2.5 .NET XML Serializer-Klasse

[Microsoft.NET Framework-Klassenbibliothek, 2003. WS]

> *Die XML-Serialisierung ist die Konvertierung der öffentlichen Eigenschaften und Felder eines Objekts in ein serielles Format (in diesem Fall XML) zum Zweck der Speicherung und Übermittlung. Die Deserialisierung erstellt das Objekt aus der XML-Ausgabe in seinem ursprünglichen Zustand erneut. Die Serialisierung kann daher als eine Möglichkeit zum Speichern des Zustands eines Objekts in einen Stream oder Puffer aufgefasst werden. Beispielsweise verwendet ASP.NET die* `XmlSerializer`*-Klasse zum Codieren von XML-Webdienstmeldungen.*
>
> *Die Daten in den Objekten werden durch Konstrukte der Programmiersprache beschrieben, z. B. Klassen, Felder, Eigenschaften, primitive Typen, Arrays oder auch eingebettetes XML in Form von XmlElement-Objekten oder XmlAttribute-Objekten. Es können eigene, mit Attributen versehene Klassen erstellt, oder Klassen anhand eines vorhandenen XML-Schema-definitionsdokuments (XSD-Dokument) mit XML Schema Definition-Tool (Xsd.exe) erzeugt werden. Mit dem Tool Xsd.exe können anhand des XML-Schemas eine Reihe von Klassen erstellt werden, die für dieses Schema streng typisiert und mit Attributen versehen sind, so dass sie nach der Serialisierung dem Schema entsprechen.*

Eine Datenübertragung zwischen Objekten und XML setzt eine Zuordnung der Konstrukte der Programmiersprache zum XML-Schema und umgekehrt voraus. `XmlSerializer` *und verwandte Tools wie Xsd.exe stellen sowohl zur Entwurfszeit als auch zur Laufzeit einen Übergang zwischen diesen beiden Technologien bereit. Zur Entwurfszeit kann mit der Xsd.exe aus den benutzerdefinierten Klassen ein XML-Schemadokument (.xsd) erstellt, oder Klassen aus einem angegebenen Schema generiert werden. In beiden Fällen sind die Klassen mit benutzerdefinierten Attributen versehen, um* `XmlSerializer` *anzuweisen, wie die Zuordnung zwischen dem XML-Schemasystem und der Common Language Runtime vorzunehmen ist. Zur Laufzeit können Instanzen der Klassen in XML-Dokumenten serialisiert werden, die dem angegebenen Schema entsprechen. Ebenso können diese XML-Dokumente wieder in Laufzeitobjekte deserialisiert werden. (das XML-Schema ist optional und weder zur Entwurfszeit noch zur Laufzeit erforderlich).*

Zum Steuern des generierten XML können spezielle Attribute auf Klassen und Member angewandt werden. Um z. B. einen anderen XML-Elementnamen anzugeben, werden XmlElementAttribute einem öffentlichen Feld oder einer Eigenschaft zugewiesen und somit die ElementName-Eigenschaft festgelegt. Eine Liste aller Attribute findet sich unter Attribute für die Steuerung der XML-Serialisierung.

Wenn der generierte XML Abschnitt 5 des Dokuments "Simple Object Access Protocol (SOAP) 1.1" des World Wide Web Consortium (www.w3.org) entspricht, muss der `XmlSerializer` *mit XmlTypeMapping erstellt werden. Um das codierte SOAP-XML weiter zu steuern, sollen die in Attribute für die Steuerung der Serialisierung von codiertem SOAP aufgeführten Attribute verwendet werden.*
Mit `XmlSerializer` *können die Vorteile der Arbeit mit streng typisierten Klassen genutzt werden, ohne auf die Flexibilität von XML verzichten zu müssen. Wenn in den streng typisierten Klassen Felder oder Eigenschaften vom Typ* `XmlElement, XmlAttribute` *oder XmlNode verwendet werden, können Teile des XML-Dokuments direkt in XML-Objekte eingelesen werden.*

Wenn mit umfangreichen XML-Schemas gearbeitet wird, können auch mit den Attributen XmlAnyElementAttribute und XmlAnyAttributeAttribute Elemente oder Attribute serialisiert bzw. deserialisiert werden, die nicht im ursprünglichen Schema gefunden wurden. Zum Verwenden des Objektes sollten `XmlAnyElementAttribute` *auf ein Feld angewandt werden, das ein Array von* `XmlElement`*-Objekten zurückgibt, oder* `XmlAnyAttributeAttribute` *auf ein Feld angewandt werden, das ein Array von* `XmlAttribute`*-Objekten zurückgibt.*

Wenn eine Eigenschaft oder ein Feld ein komplexes Objekt zurückgibt (z. B. ein Array oder eine Klasseninstanz), wird dieses vom `XmlSerializer` in einem Element konvertiert, das innerhalb des XML-Hauptdokuments geschachtelt ist.

Die erste Klasse im folgenden C#-Code gibt beispielsweise eine Instanz der zweiten Klasse zurück:

```
public class MyClass
{
      public MyObject MyObjectProperty;
}

public class MyObject
{
      public string ObjectName;
}
```

Die serialisierte XML-Ausgabe lautet wie folgt:

```
<MyClass>
      <MyObjectProperty>
      <Objectname>My String</ObjectName>
      </MyObjectProperty>
</MyClass>
```

Die Serialisierung einer beliebigen Gruppe von Objekten und der zugehörigen Felder und Eigenschaften kann überschrieben werden, indem eines der entsprechenden Attribute erstellt, und dieses den XmlAttributes hinzugefügt wird. Die Serialisierung auf diese Weise zu überschreiben, bietet zweierlei Nutzen: Erstens kann die Serialisierung von Objekten in einer DLL gesteuert und erweitert werden, auch wenn nicht auf den Quellcode zugegriffen werden kann. Zweitens kann eine einzige Gruppe von serialisierbaren Klassen erstellt werden, deren Objekte jedoch auf verschiedene Art serialisiert werden können. Für diese Technik gibt es die XmlAttributeOverrides-Klasse.

Die Serialize-Methode wird aufgerufen, um ein Objekt zu serialisieren. Die Deserialize-Methode deserialisiert ein Objekt.

Die Klasse XmlSerializerNamespaces ermöglicht das Hinzufügen von XML-Namespaces zu einem XML-Dokument.

*Hinweis: Klassen, die IEnumerable oder ICollection implementieren, werden von XmlSerializer besonders behandelt. Eine Klasse, die **IEnumerable** implementiert, muss eine öffentliche **Add**-Methode mit einem einzigen Parameter implementieren. Der Parameter der **Add**-Methode muss von demselben Typ sein, den die **Current**-Eigenschaft für den von **GetEnumerator** zurückgegebenen Wert zurückgibt, oder muss ein Basistyp dieses Typs sein. Eine Klasse, die **ICollection** zusätzlich zu **IEnumerable** implementiert, z. B. CollectionBase, muss eine öffentliche und mit **Item** indizierte (Indexer in C#) Eigenschaft mit einer ganzen Zahl sowie eine öffentliche **Count**-Eigenschaft vom Typ Integer aufweisen. Der Parameter der **Add**-Methode muss von demselben Typ sein, den die **Item**-Eigenschaft zurückgibt, oder muss ein Basistyp dieses Typs sein. Bei Klassen, die **ICollection** implementieren, werden die zu serialisierenden Werte nicht durch einen Aufruf von **GetEnumerator**, sondern über die indizierte **Item**-Eigenschaft abgerufen.*

1.2.6 Synchronisierung

Die Synchronisierung garantiert die Eindeutigkeit eines Objektes. In einer Multiuser-Umgebung besteht die Möglichkeit einer Undeutlichkeit von Variablen, Attributen und dadurch Objekten. Diese ist nicht mit der Benutzersynchronisierung zu verwechseln, d.h. der Aktualisierung von Objektattributen durch unterschiedliche Projektteilnehmer ohne gegenseitige Absprache. Letztere ist durch ein Berechtigungssystem zu lösen.

Die Aktualisierungsproblematik entsteht durch das Multitasking eines modernen Betriebssystems. Durch die Timeslots, die einem Prozess zugeordnet werden, wird der Prozess immer wieder unterbrochen.

Wenn mehr als eine Applikation auf dasselbe Objekt zugreift, wird für jede Applikation, zum Zugriff auf das Objekt, ein Prozess gestartet:

```
[C#]
public long updateOd(string projId, string objId, long duration)
{
      stor s = new storUtils();
      return s.updateOd (projId, objId, duration);
}
```

Benutzer A führt eine Aktualisierung der Objektdauer mit Hilfe von Primavera aus, indem ein Proxy Objekt für Primavera (P3e) auf dem Objekt zur Aktualisierung der Zeitschiene zugreift:

```
[C#]
conObj c = new conObj();
long confirmOd =
            c. updateOd („18E96AF7-ECC1-4FF5-B532-A9187F18C1B“, „223F0AD1-
            C46E-46BD-BDDA-9093FB343C0A„, 10);
```

Die Dauer der Ausführung des Objektes mit der Identifikationsnummer „223F0AD1-C46E-46BD-BDDA-9093FB343C0A“, eine runde Stütze, wird von 5 auf 10 Tage erhöht.
Die Methode c.UpdateOd(...) greift auf das Attribut OD des Objektes zu. Es erhält den Wert 5. Bevor es eine Inkrementierung auf 10 vornehmen kann, wird der Thread vom Betriebssystem pre-empted, d.h. angehalten.

Zwischenzeitlich aktiviert Benutzer B einen Prozess, ähnlich wie oben beschrieben, der wiederum als Proxy für Primavera fungiert (bei anderen Applikationen, z.B. PowerProject oder MS Project, wird es ähnlich vorgehen). Hier wird der Wert auf 20 gesetzt. Das Betriebssystem führt die Anweisung bis kurz vor der Bestätigung des Wertes zurück. Hier findet ein Interrupt seitens des Betriebssystems statt. Der aktuelle Wert der Dauer des Objektes ist 20. Jetzt führt das Betriebssystem den Thread des Prozesses A weiter aus. Der Wert der Dauer wird entsprechend der Anweisung der Methode auf 10 gesetzt, die Methode gibt zur Bestätigung den Wert 10 zurück, der Benutzer A ist zufrieden und terminiert den Prozeß.

Nach Terminierung des Prozesses A, aktiviert das Betriebssystem den Thread B und führt ihn bis zum Ende aus. Da allein die Rückgabe des aktuellen Wertes der Dauer fehlte, liefert Thread B jetzt den Wert 10 zurück. Der Benutzer B ist natürlich verwirrt, da der Wert 20 eingegeben wurde, jedoch 10 als Bestätigung erhalten wird.

Um dieses Problem zu lösen, stehen einige Techniken zur Verfügung wie Channelling, Critical Sections und andere Synchronisierungsmechanismen. Diese werden weiter unten beschrieben.

1.2.7 Synchronisierungsebene

Bei der Objektaktualisierung entsteht die Frage der Synchronisierungsebene, da Objektattribute auch parallel aktualisiert werden können. Entscheidend ist, dass jegliche Aktualisierung durch nur einen Thread erfolgt. Die Ebene der Synchronisierung ergibt sich aus der Überschneidung gegenseitiger Tendenzen:

- Aus fachlicher Sicht sollte die Aktualisierungsebene so hoch wie möglich sein, da ab einem bestimmten Grad der Partikularität (Einzelheiten der Attribute) kein fachlicher Zusammenhang mehr besteht.
- Da die Bearbeitung von einfachen Datentypen immer der Aktualisierung von komplexen Attributen vorzuziehen ist, sollte aus technischer Sicht eine möglichst niedrige Ebene gefunden werden.
- Eine niedrige Ebene ermöglicht eine größere Zahl paralleler Aktualisierungen (Benutzer).

Da in einem dynamischen System nicht abzusehen ist, wie viele Kategorien für Attribute schließlich vorhanden sein können und da die Attribute untereinander auch Abhängigkeiten besitzen (Dauer, Anfang, Ende), ist es sinnvoll, fachlich abhängige Attribute möglichst zu kapseln (in einer ArrayList), um eine möglichst feingliedrige Aktualisierung ohne fachliche Inkohärenz zu ermöglichen.

Da jedoch Grauzonen der fachlichen Inkohärenz existieren können, sollte die Aktualisierungsebene durch den Administrator steuerbar sein.

1.2.8 Aktualisierung und Wiederherstellung

Zur Sicherung einer Aktualisierung wird in statischem Code (nicht-dynamischer Vererbung) eine Critical Section eingesetzt. Dieser Code Absatz erlaubt den Zugriff nur eines Threads auf einer kritischen Variable (die nur von einem Benutzer auf ein Mal aktualisiert werden darf) [Ardestani, K. et al., 2002. S. 86]:

```
[VB.NET]
Imports System
Imports System.Threading

Namespace MonitorEnterExit
      Public Class EnterExit
            Private result As Integer = 0

            Public Sub CriticalSection()
                  Monitor.Enter(Me)
                  Dim i as Integer
                  For i = 1 to 5
                        result += 1
                  Next i
                  Monitor.Exit(Me)
            End Sub

            Public Overloads Shared Sub Main(ByVal args() as [String])
                  Dim e as New EnterExit()
                  Dim ct1 as New Thread(New ThreadStart(AddressOf
                  e.CriticalSection))
                  ct1.Start()
            End Sub
      End Class
End Namespace
```

Ein solches Verfahren erlaubt den Zugang nur eines Threads auf ein Mal zum aktuellen Objekt mit der Variabel `result`. Da das Objektmodell von *conObj* außerhalb der Framework Klassen erfolgt, irgendwo zwischen dem .NET Framework und der Datenbank, also innerhalb des *conObj* Objektmodells, kann die oben beschriebene Methode nicht eingesetzt werden. Stattdessen wird eine Klasse, die zusätzlich zu dem Objektstream (Klassenbeschreibung – XML Schema - und Instanzierung - XML) mit den relevanten Informationen zur Aktualisierung der Objektattribute versehen wird, eingesetzt. Diese Klasse registriert den Zugriff auf das Objekt und verwaltet die Anzahl Threads, die auf verschiedene Attribute der Objekte wirken, um eine fachlich kohärente Synchronisierung zu gewährleisten. Zusätzlich zu der Steuerung der Anzahl Zugriffe muß die Klasse auch gewährleisten, dass Prozesse, worin die aktiven Threads laufen, noch leben. Sonst werden Attribute unzugänglich.

Dazu gibt es einige Methoden:

- Das kontinuierliche Pingen (Abfragen) des ausführenden Threads
- Ein Thread Lease mit Timeout
- FIFO mit Timeout

Die Art des Timeouts ist zu beachten. Falls sich die Timeout auf die absolute Systemzeit bezieht, und der Thread, worin eine Instanz der Synchronisierungsklasse läuft, vom Betriebssystem entsprechend lange aufgehalten wird (pre-emptive interrupt), so dass der auszuführende Code nicht in der verfügbaren Zeit zu schaffen ist, tritt ein Fehler ein. Das Objekt kann nicht aktualisiert werden.

Daher sollten sich die Timeouts auf die Laufzeit des Thread beziehen, d.h.

TimeOut >= |Thread Lifetime| - (|ΣInterrupts | + |Σ Sleeps|)

Thread Lifetime =die absolute Lebensdauer des Threads
|Σ Interrupts| = vom Betriebssystem erzwungene Interrupts
|Σ Sleeps| = vom Betriebssystem erzwungene sleeps

Interrupts sind vom System erzwungene Unterbrechungen der Threadberechnungen. Sie werden ausgelöst durch externe Steuergeräte (Eingabegeräte, Analog-Digitalwandler) oder durch das threadmanagement des (multithreading) Betriebssystems.

Ein Sleep wird erzwungen, nachdem ein Interrupt den thread angehalten hat. Der Thread wartet auf die Erlaubnis zur Wiederaufnahme der Berechnungen.

Falls ein ausführender Prozess oder ein ausführender Thread abrupt beendet wird oder sich selbst beendet, muss eine Methodik den ursprünglichen Zustand des Objektes oder des Segmentes des Objektes, das in diesem Thread bearbeitet worden ist, wiederherstellen. Das ist unumgänglich, da sonst keine Haftung des Systems für die Kohärenz der Objekte besteht (das nach Ausführung aller verfügbaren Methoden der erwartete Zustand des Objektes erhalten, wird).

1.3 Objektverhalten in einer Multiuser Umgebung

Moderne VR Systeme werden so ausgelegt, dass verteiltes Rechnen oberste Priorität ist. Dadurch können Rechenoperationen vor Ort ausgeführt werden und somit die Auslastung der Haupt-Server reduziert werden. Clients berechnen Objektinformationen in einem LOD lokal und senden diese Informationen an ein Subset von Usern oder an alle User, entweder direkt oder über einen Verteilungsserver, je nach Multiuser Strategie. Einige Techniken, die bei der Multiuserarchitektur helfen sollten, sind:

- Multiplayer Client Server logische Architektur (Abb. 155)
- Multiplayer Client Server mit Multiple-Server Architekturen
- Peer-to-peer Architekturen
- AOIM – Area of Interest Management (Abb. 156).

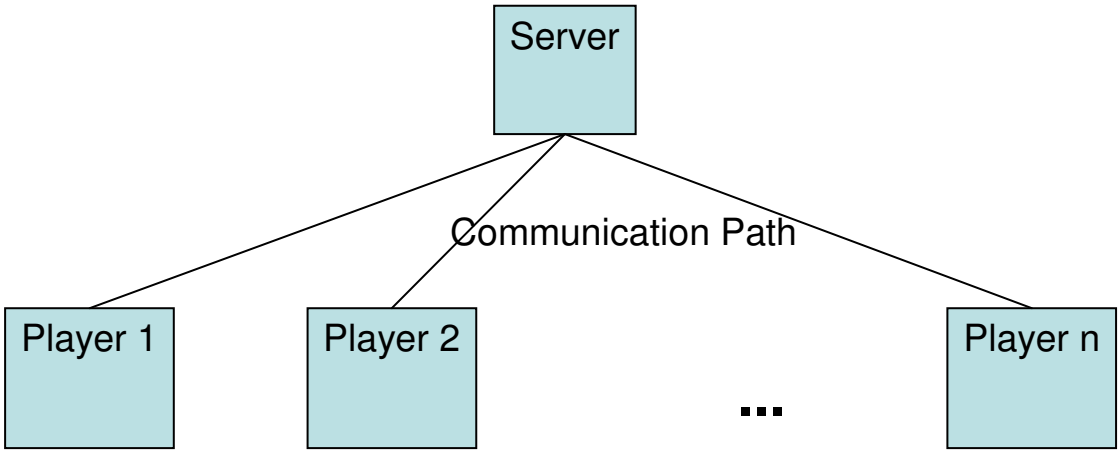


Abb. 155: Multiplayer Client Server logische Architektur [Singhal, S. and Zyda, M., 1999. S. 92]

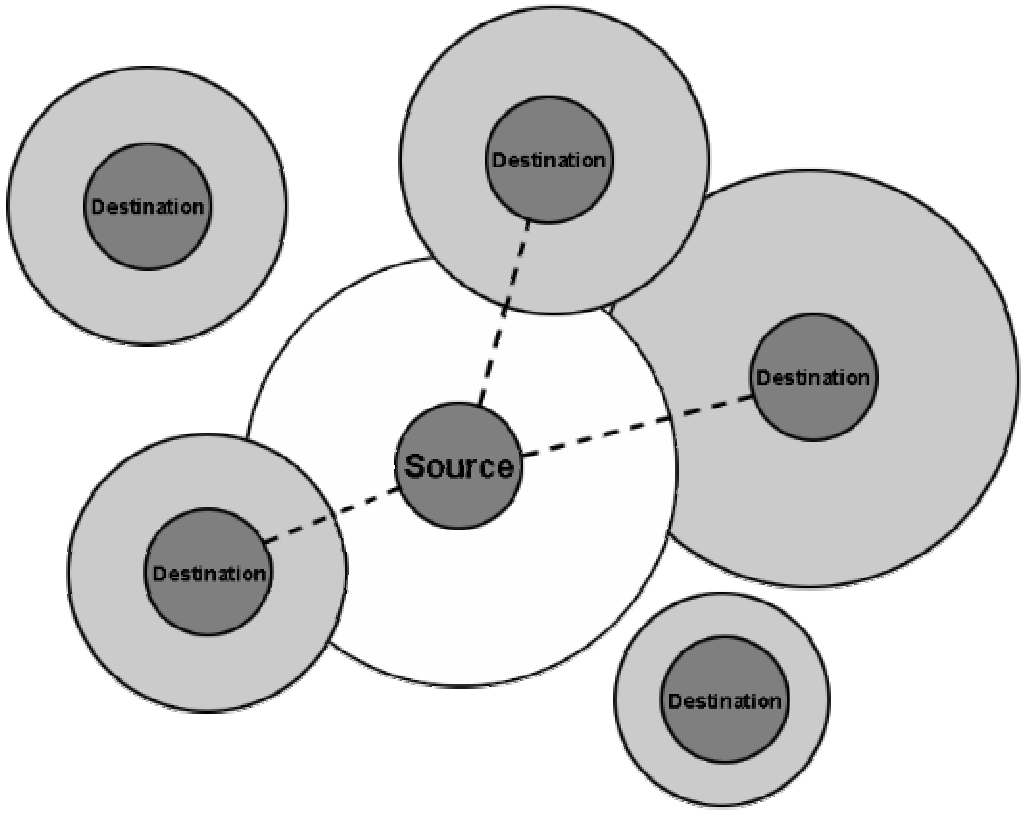


Abb. 156: Räumliche Partitionierung eines VE mittels AOIM (auch als Aura-Nimbus bekannt [Singhal, S und Zyda, M., 1999. S. 197]

Zusätzlich zu den erwähnten Überlegungen sollten auch Probleme, die sich bei der Aktualisierung und dem asynchronen Zugriff auf Objekte einstellen könnten, bedacht werden.

1.4 Racing, Deadlocks, lost Objects, Timeouts

Synchronisierung ist der Begriff für die Verwaltung von Zugriffen auf Objekte. Diese Verwaltung ist essentiell in einer Objektorientierten Architektur, um zu verhindern, dass

- Objektzustände unbekannt sind,
- um Threadsicherheit zu gewährleisten
- und um sicherzustellen, dass Race Conditions verhindert werden.

Nicht jede Klasse sollte threadsicher sein, da es dadurch zu einem Verlust der Leistung kommt, aber bei einem Multithreaded Zugriff auf kritische Ressourcen (Objekte), sollte die Threadsicherheit gewährleistet sein. Abbildung 157 zeigt eine typische *Race Condition* auf. Mr X und Mrs. X wollen zeitgleich von zwei verschiedenen ATM's 1000 USD vom gleichen Konto abheben. Eine solche Situation, wo mehrere Threads

zeitgleich auf die gleiche Ressource zugreifen, kann unvorhergesehene Situationen herbeiführen, ist als Race Condition bekannt und sollte vermieden werden. Folgende Strategien können eingesetzt werden, um Racing zu vermeiden:

- Synchronisieren Critical Sections im Code
- Das Objekt (oder die Ressource) unveränderlich machen (immutable)
- Einen Thread-Safe Wrapper einsetzen

Bei einer Critical Section wird nur der Zugriff eines Threads auf ein Mal zugelassen. Da es immer möglich ist, die Zugriffsreihenfolge zeitlich zu messen (auch wenn der Unterschied den Bruchteil einer Sekunde ausmacht), können interessierte Threads synchronisiert werden. Somit sind alle Transaktionen atomisch. Andere Threads müssen das Ende der Transaktion abwarten.

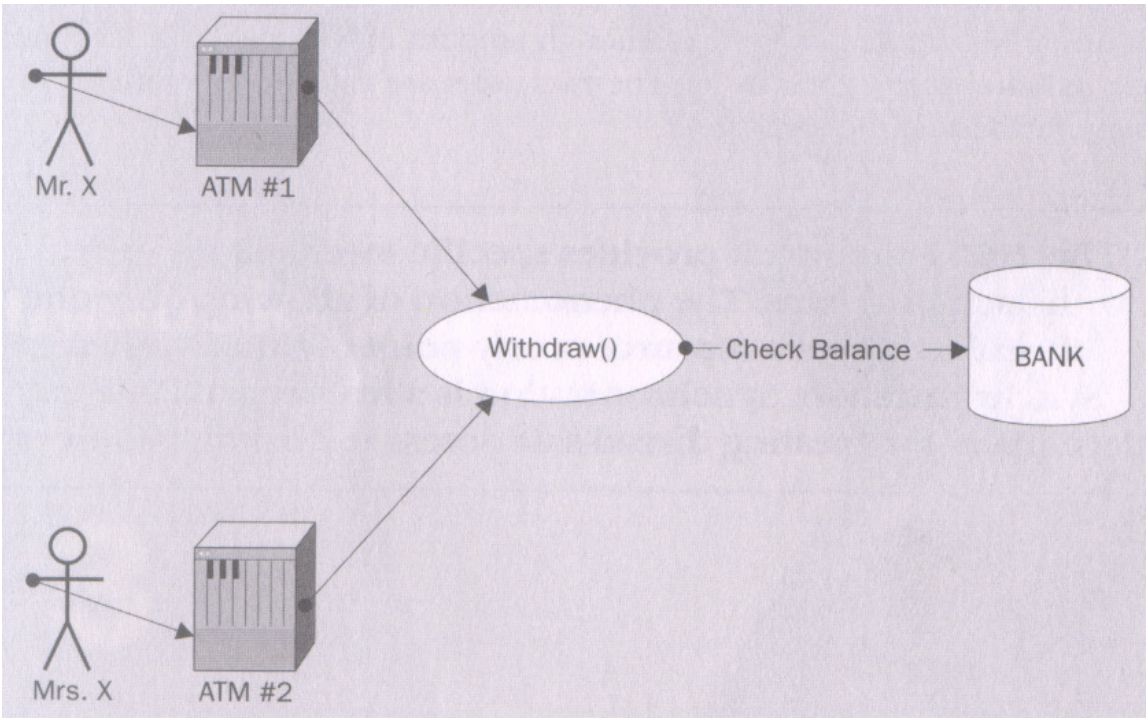


Abb. 157: Typische Situation einer Race Condition [nach Ardestani et al., 2002. S. 80]

Abbildung 158 zeigt ein typisches *Deadlock* Szenario. Eine solche Situation tritt typischerweise bei der Synchronisierung von Objekten auf. In der Abbildung ergreift Thread 1 einen Lock L1 auf das Objekt durch den Eintritt in die Critical Section. In dieser Critical Section sollte Thread 1 den Lock 2 ergreifen. Thread 2 ergreift Lock 2 und sollte Lock L1 ergreifen. Jetzt kann Thread 2 nicht L1 ergreifen, weil es im Besitz von Thread 1 ist, und Thread 1 kann nicht L2 ergreifen, weil es im Besitz von Thread 2 ist. Dadurch begeben sich beide Threads in einen Zustand des unbegrenzten Wartens oder Deadlocks.

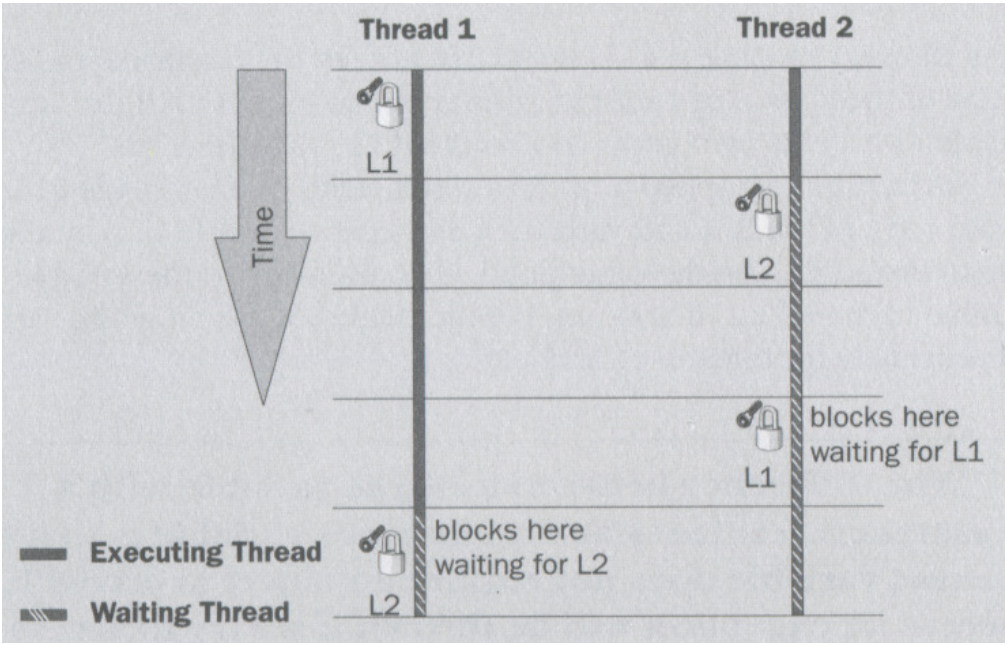


Abb. 158: Typisches Deadlock Szenario [nach Ardestani et al. , 2002. S. 108]

Die beste Strategie, Deadlocks zu vermeiden, ist zu verhindern, dass mehr als ein Deadlock auf ein Mal ergriffen wird. Falls nicht möglich, sollten Deadlocks synchron ergriffen werden.

Lost Objects beschreiben Objekte, die aufgrund einer Objektmodell-Anomalie eine Referenz verlieren und somit nicht mehr programmatisch auffindbar sind. Das geschieht typischerweise wenn GUID's oder Datenbank ID's als Referenznummern verwendet werden und bei Aktionen wie Löschen oder Verschieben diese Referenzen ungültig werden, einige Objekte jedoch durch diese Referenzen verknüpft sind. Wenn bei einer solchen Situation keine sinnvollen Aktualisierungsalgorithmen existieren, werden Objekte in dem Datenspeicher verloren gehen. Dies führt dazu, dass der Datenspeicher unnötigen Ballast bekommt. Daher sind lost Objects ein Problem der Architektur, es sollten alle Methoden dahingehend geprüft werden, ob sie lost Objects erzeugen.

Timeouts werden typischerweise bei Transaktionen wie Datenbankzugriffen oder Diensten eingesetzt, um eine Endlosschleife zu unterbrechen. Dadurch kann entweder eine andere Methode durchgeführt werden, oder der Aufruf abgebrochen werden. Timeouts werden meist durch die Framework oder Datenbank auf einen Standardwert gesetzt und sollten bei komplexen Aufrufen dynamisch gestaltet werden, um sich an der zu erwartenden Last zu orientieren.

Die System.Threading Namespace der .NET Framework enthält eine Timeout- Klasse, die allerdings dazu verwendet wird, Threads für eine unbestimmte Zeit zu unterbrechen.

1.5 Object Caching

Das Objektmodell ist nicht in einem herkömmlichen, statischen Klassenmodell eingebettet, sondern in der conObj.Framework, einer Laufzeit-Umgebung auf einer höheren Abstraktion. Diese höhere Abstraktion führt zwangsläufig zu Einbußen bei der Performance. Hinzu kommt der Aufwand der dynamischen Vererbung. Jedes neue Objekt, d.h., in der Klassenbeschreibung noch nicht vorhandene Objekt, muss erst mit einer Klassenbeschreibung (Klassen Definition) versehen werden, bevor es instanziert werden kann. Diese Prozedur erfolgt zur Laufzeit, d.h. wird von dem aufrufenden Prozeß mit Hilfe von einem Worker Thread während der Interaktion des Benutzers mit *conObj* lanciert und ausgeführt. Nach Erstellung der Klassendefinition wird das Objekt instanziert. Diese Aktionen erfolgen teils in Objekten, teils auf der Datenbank. Der Aufwand ist relativ groß und die Performance leidet. Um diese Situation etwas zu mildern, kann von einem Objekt Caching Gebrauch gemacht werden. Der Einsatz von Caching–Verfahren hat sich im Internet bewährt. Dort werden vor allem Komponenten einer Webseite regelmäßig sowohl auf dem Server wie auch auf dem Client in einem Cache zwischengespeichert, um den schnellen Wiederaufbau einer Seite zu ermöglichen. Der .NET Framework, sowie diverse Java Implementierungen bieten diese Technik auch für XML-basierte Seiten an. Da die Objektinstanzierung mit Hilfe von XML erfolgt, bietet sich diese Technik an. Sie kann natürlich nur für Objekte mit schon existierenden Klassen eingesetzt werden.

Eine interessante Variante des Cachings wird bei den ASP.NET Seiten angeboten. Hier gibt es die Möglichkeit eines parameter-gesteuerten Cachings. Da Objekte auch als Strukturen mit Variablen (Attribute, Eigenschaften) gesehen werden können, bietet sich eine solche Verfeinerung an.

Diese Technik kann in zwei Szenarien eingesetzt werden:

- Mit Hilfe von einem Parameter (Attribut, Eigenschaft) kann gesteuert werden, wann ein existierendes Objekt verwendet wird, oder eine neue Klassendefinition erfolgt.
- Da Objekte der gleichen Klasse unwesentliche Unterschiede haben dürften (Objekt 1: Radius 0,5 m; Objekt 2: Radius 0,6 m), könnte das Objekt im Cache eingesetzt werden, das ungleiche Attribut kann mittels eines Parameters aktualisiert werden.

Caching ist oft die einzige Möglichkeit einer Steigerung der Performance bei komplexen Systemen. Ein intelligentes Caching ermöglicht überdies den Einsatz einer größeren Anzahl existierender Objekte.

1.6 Verteilte Objekte

Der Begriff verteilte Objekte (Distributed Objects) wird im Zusammenhang mit Distributed Computing verwendet. Darunter ist ein Netzwerk von über mehreren Maschinen verteilten Objekten und Methoden zu verstehen. Der Gewinn liegt in zwei Punkten:

- Ein Verbund paralleler Rechner weist ein besseres Leistungs-/Kostenniveau auf, als eine Großrechneranlage.
- Daten können dezentral fachspezifisch verarbeitet werden und durch die Vernetzung in einen gesamtheitlichen Kontext gebracht werden.

Verteilte Systeme sind Kernbestand moderner Fertigungsanlagen oder im sog. Network-centric Warfare. Sie werden häufig bei komplexen Rechenprozessen eingesetzt. Bei einem Zusammenspiel verteilter Applikationen kann folgendes zusammenwirken:

- Die Kostenschätzung berechnet die Kosten für gewisse, im Objektmodell vorhandene Objekte, auch mit Hilfe sog. job objects. Es können sich mehrere Kostenschätzer die Objektwelt eines Projektes aufteilen und in einem verteilten Verbund arbeiten.
- Parallel dazu werden Gliederungsknoten, die wiederum den Objekten zugeordnet wurden, mit einem Terminplanungsprogramm von einem Terminplaner mit einer Zeitschiene versehen. Dadurch entstehen automatisch Informationen wie ein Cashflow.

- Der Arbeitsvorbereiter kann mit Hilfe von Visualisierungstechnologien wie X3D ermitteln, ob Konflikte bei der Zusammensetzung der Objekte im Projekt entstehen.
- Planung kann durch die Zuordnung von Planungsinformationen zu den jeweiligen Gliederungsknoten und Objekten aufgedeckt werden.
- Das Vertragsmanagement kann durch Planspiele im Objektmodell potentielle Störungen erkennen.
- Die Logistik der Bauausführung kann vom Umfang her und in der Wirkungsweise im Objektmodell modelliert werden
- Das Objektmodell dient als Dokumentation jeglicher Objektänderungen und somit als Basis des Vertragsmanagements.

Verteiltes Arbeiten löst den Widerspruch auf, der sich aus dem Zwang einer möglichst einfachen Bearbeitung durch die jeweiligen Sachbearbeiter und dem Gesamtnutzen der Informationen ergibt, der nur dadurch zu erreichen ist, dass Daten im Kontext, d.h. integriert bearbeitet werden. Im verteilten Verbund kann mit schlanken Applikationen einfach gearbeitet werden, jedoch im Verbund entsteht dadurch, dass alle das gleiche Objektmodell verwenden, ein gesamtheitlicher Kontext.

In einem verteilten System müssen Objekte nach Bedarf über die Leitung innerhalb der Host Applikation zur Verfügung stehen. Diese verteilte Instanzierung geschieht mit Hilfe von einem der Protokolle für verteilte Objekte. Die Art der Protokolle bestimmt den Modus der Instanzierung, ob das Objekt lokal anhand eines Objektgraphs oder auf der Ursprungsmaschine erstellt und dann über die Leitung zur Verfügung gestellt wird. Die Unterschiede haben Auswirkungen hinsichtlich Kenntnis und Leistung. Eine Kombination der Protokolle je nach Bedarf der einzelnen Applikationen ist durchaus möglich.

1.7 Protokolle für verteilte Anwendungen

Ein integriertes und komplexes Objektmodell kann grundsätzlich nur in einer verteilten Architektur implementiert werden. Das liegt daran, dass die Teilnehmer an einem solchen Objektmodell, seien es die Projektbeteiligten oder die Objekte selbst, auf verschiedenen Systemen und Maschinen gelagert sind. Daher ist eine gründliche Betrachtung von verteilten Technologien eine Vorraussetzung für die gelungene Implementierung eines verteilten Objektmodells.

Die Vernetzung von Rechnern bedingte den Einsatz von Protokollen zur gegenseitigen Verständigung. Daher ist die Entwicklung und Hoheit über Protokolle auch eine Frage des Einflusses in diesem Segment der Informatik. Über die Jahre gab es verschiedene Initiativen. Einige fungierten als öffentlich zugängliche Standards wie z.B. CORBA, andere wiederum waren als proprietäre Lösungen zu erwerben, z.B. DCOM oder RMI. Öffentliche Standards waren zwar allgemein zugänglich und von Gremien bestückt aus mehreren Firmen und Institutionen erarbeitet, jedoch oft in deren Implementierung wiederum proprietär, z.B. CORBA. Wichtige Aspekte bei der Entwicklung der Standards waren unter anderem:

- der Standort der Objektinstanzierung und somit, ob das Objekt übertragen wird, oder nur ein Schema des Aufrufs
- und somit auch die Notwendigkeit einer Registrierung des Objektmodels,
- das Abtasten der Schnittstelle des verteilten Objektes (Interface),
- Sicherheit, sowohl in Bezug auf Berechtigung und Verschlüsselung,
- das Ereignismodell der verteilten Objekte und das damit verbundene Übertragungsprotokoll,
- die Leistungsfähigkeit des Protokolls und damit auch dessen Schichtmodell und
- die Fähigkeit, in einer Internet-Umgebung mit Firewalls und Proxies funktionsfähig zu sein.

In der Dekade 1990-2000 hatten sich vorwiegend die Protokolle CORBA, DCOM und RMI (JINI) durchgesetzt [Banerjee, A. et al, 2001. S.14 f.] Diese waren meistens kompliziert in der Implementierung und leistungsanfällig. JINI ist eine Besonderheit, da das Objekt Anagramm nicht auf dem Zielrechner vorhanden sein muss, sondern allein die Anfrage nach einer Methode in einem JINI Verbund ausreicht, um das entsprechende Objekt übertragen zu bekommen. Natürlich ist der verbundene Aufwand von Dienst- und Objektfindung, sowie Objekttransfer mit einer Leistungsminderung verbunden. Technologisch jedoch markiert JINI ein Maß an Minimalismus und Effizienz, was auch nicht von XML Diensten erreicht wird.

DCOM ist durch .NET Remoting und XML Dienste ersetzt worden, da das COM Objektmodell durch die JVM-ähnliche .NET CLR ausgetauscht wurde .NET Remoting basiert auf TCP und HTTP als Übertragungsschichten, wobei die Leistungsverbesserung zwischen diesen Protokollen einen Faktor 10 beträgt. XML Dienste sind auch umfangreich in der JVM implementiert, zusätzlich zu RMI und JINI (JavaSpaces).

CORBA wird weiterhin durch die JVM und die .NET CLR unterstützt. Somit existieren netzwerkzentrische Protokolle wie CORBA, JINI und .NET Remoting, die ein homogenes Netzwerk voraussetzen. Demgegenüber können XML Dienste sowohl über Firewalls als auch in heterogenen Netzen funktionieren.

Bei der Wahl eines geeigneten Protokolls müssen folgende Aspekte berücksichtigt werden [Grimes, R., 1997. S.14 ff] (folgende Auflistung auf English):

- Object Locator: Distributed objects must be uniquely identified in the network. This obtains through IP Adresses, local Names (COM) or central object administrators such as CORBA 2's OMG Transaction Service, or DCOM's MS Transaction Server, local services information (.NET Remoting, XML Services) or associative Lookup (JavaSpaces, JINI).
- Communication
- Data Typing – strong and weak Typing
- Data Representation – Definition of Network data types for marshalling and unmarshalling of data. DCE RPC defines NDR (Network Data Representation)
- Synchronous and Asynchronous Communication
- State Persistance: Depending on connection-based or connectionless transport mechanism. State persistance is supported by the IIS with Web Services, and with .NET Remoting. JINI also supports state persistance.
- Security: Object creation and destruction rights. Stream Encryption.
- Reliability and Availability: Object Reliability is machine independent scope expectation. Availability deals with object redundance by utilising transaction monitors such as CORBA OMG Transaction Service, or the MS Transaction Server. Fault Tolerance is also supported by Isis RDO (CORBA). JINI and .NET Remoting also support object redundancy. XML Services can do so by leveraging alternative service URL's.
- Load Balancing: not offered in DCOM, however supported by CORBA 2, .NET Remoting and JINI. Load Balancing redirects object requests in order to stabilise network traffic. Only works in an object server cluster.

.NET Remoting bietet seit der Version 2.0 wieder die volle Palette technologischer Möglichkeiten, inkl. Ereignisbehandlung die in .NET Version 1.1 noch nicht unterstützt war. Die Leistungsfähigkeit von .NET Remoting ist allerdings aufgrund eines Implementierungsfehlers nicht wesentlich besser als XML Dienste, liegt jedoch dar-

über und bietet auch weitere Vorteile. .NET Remoting ist wesentlich einfacher zu Implementieren als CORBA oder DCOM, und deswegen auch als Technologie für das Objektmodell geeignet. Da diese Technologie mit der .NET Framework standardmäßig ausgeliefert wird, bietet sie sich bei der Implementierung eines verteilten Objektmodells an.

1.8 JINI and JavaSpaces

SUN Microsystems ist führend in der Entwicklung verteilter Systeme. Java wurde entwickelt, um eine Plattform-unabhängige, leichte Software für den Einsatz auf allen erdenklichen Endgeräten und Servern zu ermöglichen. Bei der Entwicklung von verteilten Protokollen hat SUN folgende Aspekte stark berücksichtigt:

- Latenz (Latency): Die Reaktionszeiten im Netzwerk sind um einiges höher als auf einer lokalen Maschine. Dieser Faktor sollte berücksichtigt werden, wenn Prozesse auf unterschiedlichen Maschinen miteinander kommunizieren.
- Synchronisierung (Synchronisation): Prozesse in einem verteilten Einsatz müssen untereinander synchronisiert werden, um sinnvoll zu agieren. Algorithmen verlangen eine gewisse Sequenz der verteilten Prozesse, Daten sollten in einer gewissen Sequenz verändert werden. Da die Prozesse jedoch meistens asynchron operieren, wird deren Synchronisierung mathematisch sowie rechnerisch ein komplexes Unterfangen.
- Partielles Versagen (Partial Failure): Sowohl durch den Einsatz zunehmender, verteilter Komponenten, als auch bei steigender Zeit des Einsatzes, nimmt die Wahrscheinlichkeit, dass einige von den Komponenten versagen könnten zu, entweder weil der Prozess versagt oder die Kommunikation getrennt wurde. Ein verteiltes System sollte auf eine solche Situation sanft reagieren können, entweder durch eine ausweichende Aktion, oder indem es den fehlenden Prozess ersetzt.

1.8.1 JINI

JINI basiert auf Java und dessen Remoting Protokoll RMI und wurde von SUN in 1996 entwickelt.

1.8.2 JavaSpaces

Javaspaces setzt auf der JINI oder RMI Technologie auf. Es unterscheidet sich von herkömmlichen Protokollen, die auf dem Aufruf entfernter Methoden basieren. Javaspaces betrachtet eine Applikation als ein Zusammenwirken verteilter Prozesse als Objekte, die in einem oder mehreren spaces rein- oder rausfließen. Diese space-basierte Technologie hat ihren Ursprung in der Linda Coordination Language, die Gelernter [Arnold, K. et al, 1999. S. 258] entwickelte.

Ein space ist eine gemeinsame, netzwerkzugängliche Repository für verteilte Objekte. Prozesse nutzen die *Repository* als einen persistenten Objekt– Speicher und einen Austausch–Mechanismus. Anstatt direkt miteinander zu kommunizieren, wird über einen Austausch von Objekten kommuniziert. Wenn ein gewünschtes Objekt in der *Repository* nicht unmittelbar vorhanden ist, kann ein Prozess auf dessen Ankunft warten. Anders als bei herkömmlichen Objektspeichern, modifizieren Prozesse die Objekte in dem *space* nicht direkt. Um Objekte verändern zu können, müssen diese aus dem *space* geladen werden, explizit verändert werden und dann wieder in dem Space eingeschoben werden.

Space-basierte Applikationen erfordern verteilte Datenstrukturen (distributed data structures) und verteilte Protokolle (distributed protocols), die über diese Strukturen agieren. Verteilte Datenstrukturen bestehen aus einem oder mehreren Objekten, die in einer oder mehreren *spaces* wohnen. Die Darstellung von Daten, als eine Sammlung von Objekten in einem gemeinsamen *space,* ermöglicht mehreren Prozessen den parallelen Zugriff und die Modifizierung der Daten. Verteilte Protokolle definieren, wie Prozesse koordiniert und parallel auf die Daten zugreifen können.

Verteilte Protokolle, die mit spaces geschrieben wurden, haben den Vorteil, loosely coupled (loosely coupled) zu sein. Weil Prozesse indirekt mit Hilfe von Objekten in einem *space* kommunizieren, brauchen diese deren gegenseitige Identitäten nicht zu kennen, oder zur gleichen Zeit im *space* aktiv zu sein. Konventionelle Netzwerk Prozesse verlangen Auskunft

- über Zielprozess (wer?)
- auf einer bestimmten Maschine (wo?)
- zu einem bestimmten Zeitpunkt (wann?).

Mit der Javaspaces Technologie können Objekte in einen *space* reingestellt werden, mit der Erwartung, dass ein Prozess, residierend auf einer beliebigen Maschine, zu einem gegebenen Zeitpunkt das Objekt nutzen wird. Die Entkopplung von Sender und Empfänger führt zu einem einfachen, flexiblen und zuverlässigen Protokoll. JavaSpaces besitzen folgende Merkmale:

- *Spaces* werden gemeinsam genutzt, verfügen über gemeinsame Speicher
- *Spaces* sind persistent, Objekte leben dort
- *Spaces* sind assoziativ, Objekte werden anhand von Templates gesucht
- *Spaces* sind transaktionssicher, Operationen werden atomisch ausgeführt
- *Spaces* ermöglichen den Austausch von *executable content*, lokale Objekte, die aus einem space geladen wurden, besitzen Methoden, durch die lokale Prozesse erweitert werden können.

1.8.3 Beispiel einer JavaSpaces Anwendung

In Anhang A.I.1.2.3 wird ein einfaches Beispiel beschrieben und mit Diensten implementiert. Es handelt sich um eine einfache Stütze. Das gleiche Beispiel wird hier mit JavaSpaces behandelt.

Ein *space* speichert Einträge (Entries). Ein Eintrag (Entry) ist eine Sammlung (Collection) typisierter Objekte, die das „Entry“ Interface implementieren:

```
public class Stuetze implements Entry{
        private String status = “unchanged”;
        public Integer id;
        public Double height;
        public Double width;
        public Double length;
        public String material = “B35”; //default Wert
        public Stuetze(Integer id){
                this.id = id;
        }
        public String setDimensions(Double height, Double width, Double
                length){
                if(this.height==height || this.width==width ||
                        this. length == length)
                        this.status = „stuetze altered“;
                this.height = height;
                this.width = width;
                this. length = length;

                return this.status;
      }
        public String setMaterial(String material){
                if(this.material==material)
                        this.status = „stuetze altered“;
                this. material = material;
                return this.status;
      }
        public String dump(){
                String str = “stuetze obtains as: ”
                str += “height: “ + this.height.toString();
                str += “; width: “ + this.width.toString();
                str += “; length: “ + this.length.toString();
                str += “; material: “ + this.material;
                return this.str;
      }
}
```

Eine Eintragung zu einer Stuetze kann wie folgt instanziert werden:

```
Stuetze st = new Stuetze(22556);
st. SetDimensions(3, 0.3, 0.2);
```

Die Interaktion mit einem *space* erfolgt mit verschiedenen Methoden:

- Write – platziert eine Kopie des Eintrags in dem space,
- Read – liest eine Kopie des Eintrags mit einem Template aus dem space,
- Take – agiert wie read, entfernt jedoch einen Eintrag vom space.

Der Zugriff auf ein Object space folgt:

```
JavaSpace space  = SpaceAccessor.getSpace();
space.write(st, null, Lease.FOREVER);
```

SpaceAccessor.getSpace() instanziert ein JavaSpaces Objekt. Der SpaceAccessor wird in A.I.1.8.4 im Detail erläutert. Nachdem der Eintrag mit write in den space geschrieben wurde, kann jeder Prozess mit einem Template das Objekt aus dem space lesen. Ein Eintrag gleicht einem Template, wenn der Typ oder Subtyp gleich ist, und für jeden Nicht-null Eintrag im Template decken sich die Parameterfelder exakt:

```
Stuetze template = new Stuetze (22556);
```

Mit dem Template wird ein read auf dem space ausgeführt:

```
Stuetze result = (Stuetze) space.read(template, null, Long.MAX_VALUE);
```

Ein Konsolenausdruck des Resultats

```
System.out.println(result.status);
```

ergibt:

```
„stuetze altered“
```

Mit take wird der Eintrag vom space entfernt:

```
Stuetze result = (Stuetze) space.take(template, null,
      Long.MAX_VALUE);
```

Result.content hat den gleichen Wert wie zuvor, der Eintrag jedoch wurde vom space gelöscht.

Ein anderer Prozess, z.B. *Projektleiter*, kann sich in dem space einklinken und das Objekt lesen:

```
Stuetze template = new Stuetze (22556);
Stuetze result = (Stuetze) space.read(template, null, Long.MAX_VALUE);
System.out.println(result.dump());
```

Das ergibt:

```
„stuetze obtains as: height: 3; width: 0.3; length: 0.2;
          material: B35“
```

Der Statiker klinkt sich im space ein, ändert die Materialeigenschaft des Objektes, und liest den Objektstatus raus:

```
Stuetze template = new Stuetze (22556);
Stuetze result = (Stuetze) space.read(template, null, Long.MAX_VALUE);
result.setMaterial(“B40”);
space.write(result, null, Lease.FOREVER);

result = (Stuetze) space.read(template, null, Long.MAX_VALUE);
System.out.println(result.dump());
```

Das ergibt:

```
„stuetze obtains as: height: 3; width: 0.3; length: 0.2;
  material: B40“
```

1.8.4 Der SpaceAccessor

JavaSpaces enthalten eine Utility Klasse, die den Zugriff auf einen space ermöglicht. Der code ist in eine utility verlagert worden, da ein Zugriff auf ein *space* auch mit anderen Methoden wie einem JINI lookup service oder einer RMI Registrierung möglich ist. Die SpaceAccessor utility Klasse erlaubt daher Zugriff auf ein *JavaSpace* mit beiden Methoden. Hier ist auch zu sehen, wie JavaSpaces auf JINI aufsetzt. Die Version der Klasse im SUN JavaSpaces Toolkit hat folgenden code:

```
public class SpaceAccessor {
      public static JavaSpace getSpace(String name) {
            try {
                  if(System.getSecurityManager() == null) {
                        System.setSecurityManager(
                              new RMISecurityManager());
                  }
                  if(System.getProperty("com.sun.jini.use.registry")
                              == null) {
                        Locator locator = new
                              com.sun.jini.outrigger.LookupFinder();
                        return (JavaSpace) finder.find(locator, name);
                  }
                  else {
                        RefHolder rh = (RefHolder)Naming.lookup(name);
                        return (JavaSpace) rh.proxy();
                  }
            }
            catch(Exception e) {
                  System.err.println(e.getMessage());
            }
            return null;
      }
      public static JavaSpace getSpace() {
            return getSpace("JavaSpaces");
      }
}
```

Die Klasse SpaceAccessor bietet die statische Methode getSpace, die ein Objekt, das wiederum das `JavaSpaces` Interface implementiert, zurückgibt. Zwei Formen dieser Methode sind vorhanden, die erste nimmt ein String Parameter des Namens des gewünschten *space* auf, die zweite gibt den Accessor für die Default *space* „JavaSpaces“ zurück.

Der Security Manager in der ersten if Abfrage der `getSpace(String Name)` Version sichert das Herunterladen von nicht vorhandenen Code Paketen.
Als nächstes wird der Wert der Eigenschaft

```
com.sun.jini.use.registry
```

abgefragt. Ist dieser Wert gleich null, wird JINI als lookup service eingesetzt, sonst eine RMI Registrierung. Beim Einsatz von JINI können zwei Klassen, die sich im Paket

```
com.sun.jini.outrigger
```

befinden, eingesetzt werden:

- `DiscoverLocator` – lokalisiert einen JINI lookup service
- `LookupFinder` – liefert ein Interface auf einen service mit der Methode find, die als Parameter den locator und service Namen nimmt.

Wenn der Zugriff auf ein space mit einer RMI Registrierung erfolgt, kann die statische `Naming.lookup` Methode des Pakets `java.rmi` eingesetzt werden. Wird eine Registrierung gefunden, wird eine Referenz auf ein entferntes Objekt geliefert. In der Referenzimplementierung von SUN muß ein Aufruf mit der proxy Methode des Objektes

```
com.sun.jini.mahout.binder.RefHolder
```

erfolgen. Diese Methode liefert ein lokales Proxy Objekt, was die `JavaSpaces` Interface implementiert.
In beiden Fällen wird das Proxy Objekt durch den Aufruf getSpace zurückgegeben. Dieses Proxy Objekt kann nun dazu genutzt werden, um mit spaces interaktiv zu agieren.

1.8.5 Zusammenfassung

JINI und JavaSpaces eignen sich hervorragend für ein verteiltes Objektmodell dank:

- Loose-coupling (lose Kopplung)
- Indirekte Kommunikation, Prozessidentität ist nicht nötig

Die Problematik bei Javaspaces ist, dass diese Technologie auf die Sprache Java und die JVM (Java Virtual Machine) zugeschnitten ist. Alle Teilnehmer eines solchen Systems müssen in dieser Sprache implementiert worden sein. Java unterstützt zwar Plattformunabhängigkeit, ist jedoch unflexibel im Umgang mit anderen Systemen. Ein weiteres Problem ist die Kommunikation über das Internet. JavaSpaces sprechen andere Ports als 80 an, benötigen daher das Öffnen weiterer Ports auf der Firewall, eine Anforderung, die wegen der Sicherheitsrisiken meistens strikt unterbunden wird. Daher eignen sich diese Technologien eher als sekundäre Implementierung beim Einsatz des Objektmodells in einem geschlossenen Netzwerk unter gleichen technologischen Bedingungen.

1.9 Messaging

Messaging (Abb. 159) hat sich im Internet vor allem als Instrument zum „Chatten“ beliebt gemacht. In Wirklichkeit handelt es sich um eine ernstzunehmende Technologie in einer verteilten Architektur. Da die Technik jedoch auf dem Senden und Empfangen von Objekten beruht, sind einige Merkmale zu beachten:

- Asynchrones Objektverhalten – das Eintreffen von Objekten ist nicht deterministisch
- Entkoppeltes Objektmodell – Sender und Empfänger können Datenpakete und Objekte ausschließlich aufgrund des einheitlichen Protokolls verarbeiten. Es fehlt eine dedizierte Verbindung.
- Abwesenheit einer Verbindung – Objekte werden als serialisierte Datenpakete verschickt, ähnlich wie bei einem POST Datagramm
- Objekte können Methoden enthalten, die wiederum beim Empfänger ausgeführt werden - Sicherheitsaspekte.

- Einheitliche Infrastrukturen – Sender und Empfänger müssen einen Message-queue Server bereitstellen

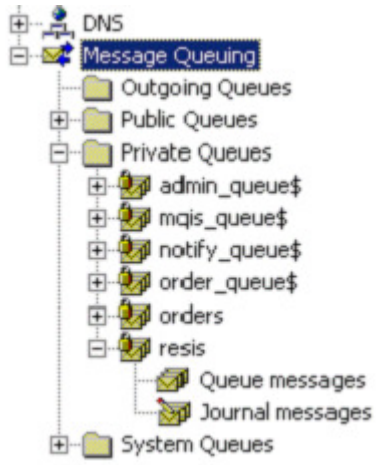


Abb. 159: Message Queue Server von Microsoft

Der Code eines Message Queue Servers ist einfach, und sieht wie folgt aus:

Server Code:

```
using System;

string queuePath = ".\\Private$\\resis";
EnsureQueueExists(queuePath);
MessageQueue queue = new MessageQueue(queuePath);
((XmlMessageFormatter)queue.Formatter).TargetTypes
= new Type[]{typeof(ResI)};

while(true)
{
      ResI newRes = (ResI)queue.Receive().Body;
      newRes.setResI();
   }
```

Das ResI Objekt

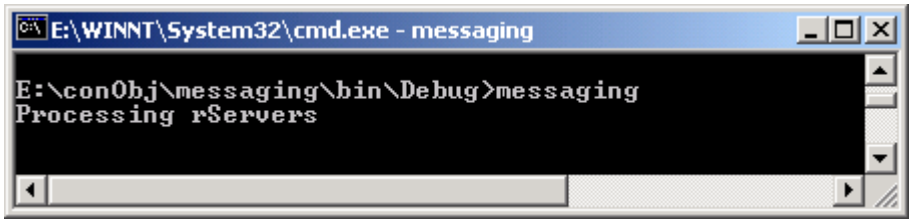
```
public class ResI
{
      public int itemId;
      public int quantity;
      public string address;
      public string IP;

      protected genSpHandler sp;
      protected ArrayList parmArr;

      public ResI(){
            IPAddress ip = IPAddress.Parse("127.0.0.1");
            IP = ip.ToString();
            sp = new genSpHandler("sa","","resS","127.0.0.1");
            parmArr = new ArrayList();
      }

      public void setResI()
      {
      // Add resI Object to the database.
            try
            {
                  parmArr.Clear();
                  parmArr.Add(itemId.ToString());
                  parmArr.Add(quantity.ToString());
                  parmArr.Add(address);
                  parmArr.Add(IP);
                  sp.execNonQuery("setResI", parmArr, false);
            }
            catch(OleDbException err)
            {
                  throw err;
            }
      }
}
```

Der Messagequeue Server wird aktiviert und wartet auf ein eintreffendes Objekt:



Der Client code ist analog dem Server code einfach strukturiert:

Client code

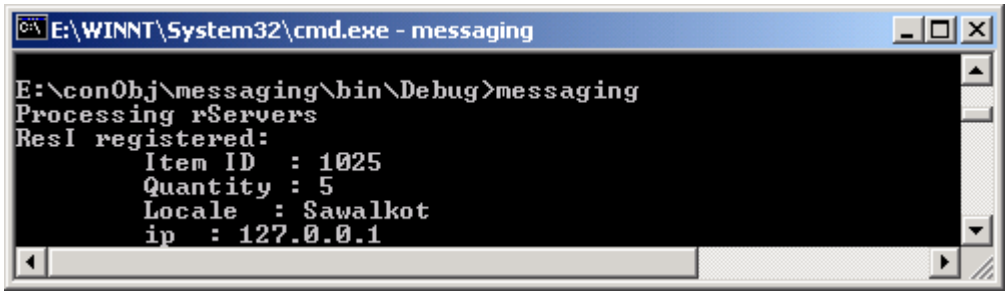
```
public static void Main()
{
      string queuePath = ".\\Private$\\resis";
      EnsureQueueExists(queuePath);
      MessageQueue queue = new MessageQueue(queuePath);

      ResI resRequest = new ResI();
      resRequest.itemId = 1025;
      resRequest.quantity = 5;
      resRequest.address = "Sawalkot";

      queue.Send(resRequest);
}

// Creates the queue if it does not already exist.
public static void EnsureQueueExists(string path)
{
      if(!MessageQueue.Exists(path))
            MessageQueue.Create(path);
}
```

Der Client initialisiert das zu sendende Objekt und verschickt es an den wartenden Message Queue Server, der beim Eintreffen des Objektes die Methode `setResI()` ausführt, in diesem Fall speichert sich das Objekt selbst in einer Datenbank:



Messaging bietet den Vorteil der losen Kopplung, ähnlich wie es bei Webservices oder Javaspaces der Fall ist. Es eignet sich daher für Anwendungsbereiche, die nicht Echtzeitanforderungen haben, wie es oft im Bauwesen in der Projektarbeit der Fall ist. Die entsprechende Infrastruktur und Kodierung ist gering, allerdings ist die Vor-

aussetzung, dass kompatible Message Queue Server zum Einsatz kommen. In diesem Fall ist die Aufteilung Client Server irreführend, da beide in wechselseitiger Beziehung agieren und jeweils als Client oder Server fungieren können.

Ein solches System ist denkbar in einem dezentralen Objektmodell wenn aus Sicherheits- oder Kostengründen eine Dauerverbindung nicht sinnvoll ist. Aufgrund des entkoppelten Objektmodells ist es jedoch nicht wünschenswert und nur als Alternative bei gestörten oder instabilen Leitungen in Betracht zu nehmen.

1.10 .NET Remoting

.NET Remoting ersetzt DCOM und beruht wie auch DCOM auf dem TCP Stack (Abb. 160). Unterschiede zu DCOM ergeben sich wie folgt:

- Der Ping Mechanismus von DCOM ist nicht Internet-skalierbar. .NET Remoting folgt einem lease-based lifespan Prinzip, das verteilte Objekt besitzt eine vorab festgesetzte und wieder erneuerbare Lebensdauer.
- Anstatt mittels eines Proxy Objektes verteilt aufrufbar zu sein, besitzt ein verteiltes Objekt unter .NET Remoting eine verteilte Identifikation, dessen Referenz ist vom ganzen Netzwerk aus aufrufbar.
- Der Aufruf Kontext basiert auf SOAP, in dessen Header zusätzliche Informationen vorhanden sind.

Layer	Description	Protocols
Application	Defines TCP/IP application protocols and how host programs interface with transport layer services to use the network.	HTTP, Telnet, FTP, TFTP, SNMP, DNS, SMTP, X-Windows, other application protocols
Transport	Provides communication session management between host computers. Defines the level of service and status of the connection used when transporting data.	TCP, UDP, RTP
Internet	Packages data into IP datagrams, which contain source and destination address information that is used to forward the datagrams between hosts and across networks. Performs routing of IP datagrams.	IP, ICMP, ARP, RARP
Network interface	Specifies details of how data is physically sent through the network, including how bits are electrically signaled by hardware devices that interface directly with a network medium, such as coaxial cable, optical fiber, or twisted-pair copper wire.	Ethernet, Token Ring, FDDI, X.25, Frame Relay, RS-232, v.35

Abb. 160: .NET Remoting basiert auf dem TCP Stack [Microsoft.NET Framwork Developer Guid, 2003. WS]

Das TCP Protokoll zeichnet sich durch Leistungsfähigkeit und Übertragungsintegrität aus.

.NET Remoting ist nicht überall sinnvoll einzusetzen. In einigen Szenarien kann es technisch nicht funktionieren, z.B. im Internet ohne Freischaltung zusätzlicher Ports. Rammer [Rammer, I., 2004. S.1] führt folgende Matrix zur Entscheidungsfindung beim Einsatz von Remoting auf:

	Singlecall	Singleton	CAO	Sponsors@Server	Sponsors@Client	Events	TCP	HTTP	CustomHost	IIS
AppDomain	X	X	X	X	X	X			X	
Process/Local	X	X	X	X	X	X	X	X	X	
Process/LAN	X	+	*	*				X		X
Process/WAN	X	+	*	*				X		X

X: Perfectly ok

+: Could affect scalability due to the possibility to hold cross-method state. If designed and developed carefully, no issues should turn up.

*: Negatively affects scalability. Don't use if you want to scale out to multiple servers.

Others/empty: Not recommended.

1.10.1 Technische Beschreibung

Ein verteilter Aufruf instanziert ein Proxy Objekt mit exakt den gleichen Methoden des entfernten Objektes. Beim Aufruf der Methoden des Proxies werden Botschaften initiiert, die mittels einer Formatter Klasse z.B.

- `System.Runtime.Serialization.Formatters.Binary`
- `System.Runtime.Serialization.Formatters.Soap`

serialisiert und über einen Client Channel an der entfernten Maschine (hier: Server) übermittelt werden. Dort empfängt ein Server-Channel die Botschaft und deSerialisiert diese mit Hilfe des gleichen Formatters, um die Botschaft an das entfernte Objekt weiterzuleiten. Die Architektur ist flexibel gestaltet, und kann aus verschiedenen Formattern und Channels bestehen, unter anderem:

- HTTP
- TCP
- SOAP

Alle .NET Remoting Objekte leiten von System.MarshalByRefObject ab. Damit sind der Aufbau, der Wrapper und der Channel gesichert. MarshalByRefObject beinhaltet auch die leasing Methoden.

Rammer vermerkt, dass obgleich der TCP Channel leistungsfähiger sein sollte dieser jedoch fehlerhaft ist, daher sollte die Kombination HTTP Channel mit BinaryFormatter eingesetzt werden.

Weitere Aspekte der Remoting Architektur:

- Verteilte Objekte: well-known und client-activated
- Aktivierung: SingleCall und Singleton
- Marshalling: Marshall-By-Value und Marshall-By-Reference
- Interception: sink objects

1.10.1.1 Beispiel einer .NET Remoting Anwendung

```
 [C#]
using System;
namespace conObj
{
        public class remoteObj : MarshalByRefObject
        {
                public remoteObj()
                {
                        Console.WriteLine(„constructor called“);
                }

                public string init()
                {
                        return „machine log initialised“;
                }
        }
}
```

Dieses remote Objekt steht auf einem Server zur Verfügung und kann mit Hilfe von einem Server Configuration File, dessen Information auch über einem directory service kenntlich gemacht werden kann, aufgerufen werden. Das Server Configuration File sieht wie folgt aus:

```
<configuration>
      <system.runtime.remoting>
            <application>
                  <service>
                        <wellknown mode="SingleCall" type="
                              Wrox.Samples.remoteObj ,
                              remoteObj " objectUri="remoteObj" />
                  </service>
                  <channels>
                        <channel id="tcp server" Port="9000"/>
                  </channels>
            </ application >
      </system.runtime.remoting>
</configuration>
```

Dieses Objekt kann mit folgender URL aufgerufen werden:

tcp://localhost:9000/remoteServer/remoteObject

Der Server aktiviert das verteilte Objekt mit der Anweisung

RemotingConfiguration.Configure()

Der Server Code sieht wie folgt aus:

```
[STAThread]
static void Main(string[] args)
{
        RemotingConfiguration.Configure("remServer.exe.config");
        Console.WriteLine("press return to exit");
        Console.ReadLine();
}
```

Die Client Applikation beinhaltet die Information zum Standort des Servers, der wiederum um die Position des verteilten Objektes weiss:

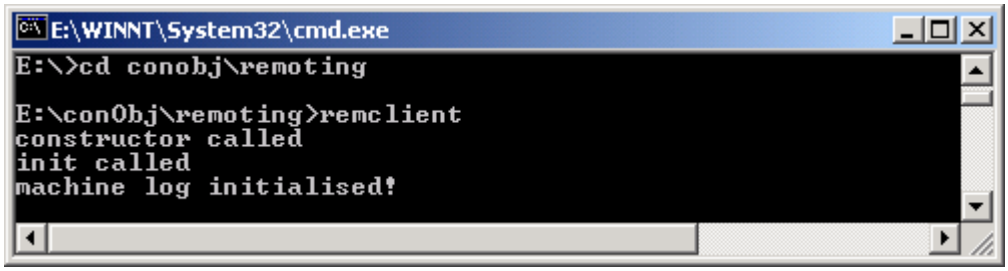
```
[STAThread]
static void Main(string[] args)
{
      RemotingConfigura-
      tion.Configure(@"e:\conObj\remClient\remClient.exe.config");
      remObject obj = new remObject();
      Console.WriteLine(obj.message());
      Console.WriteLine("serialized to: "
            + obj.serializeXml());
}
```

Die Server Lokationsinformationen liegen im Client configurations file:

```
<configuration>
      <system.runtime.remoting>
            <application name="remClient">
                  <client url="tcp://127.0.0.1:9000/remServer">
                        <wellknown type="Wrox.Samples.remObject, remObject"
                        url="tcp://127.0.0.1:9000/remServer/remObject" />
                  </client>
                  <channels>
                        <channel ref="tcp client" />
                  </channels>
            </application>
      </system.runtime.remoting>
</configuration>
```

Server wird aufgerufen:

Client wird aufgerufen:



Channels

Der Channel ermöglicht die Kommunikation zwischen Client und Server. Der Channel schickt eine Botschaft mit Hilfe eines Formates zwischen Client und Server. Als grundlegendes Protokoll wird HTTP oder TCP eingesetzt. HTTP kann auch über eine Firewall eingesetzt werden, TCP nur im Intranet, ist jedoch wesentlich effizienter. Weitere custom Protokolle können auch konfiguriert werden, wie z.B. SMTP oder Message Queueing.

1.10.2 Zusammenfassung

.NET Remoting erstellt eine attraktive Möglichkeit, ein verteiltes Objektmodell aus Sicht der Leistung und Skalierbarkeit zu modellieren. Im Allgemeinen kann jedoch von einem heterogenen und nicht-öffentlichen Netwerk ausgegangen werden, sei es durch Einsatz von Verschlüsselungstechnologien wie VPN.

Ab Version 2.0 unterstützt .NET Remoting wieder Ereignisse. Das ermöglicht die Weitergabe von Ereignissen aus den API's der teilnehmenden Applikationen, soweit diese eine solche Ereignisbehandlung unterstützen. Authentifizierung oder Verschlüsselung werden von .NET Remoting nicht per default unterstützt. Da Teilnehmer jedoch als Vertrauenswürdig eingestuft werden müssen, ist eine Verschlüsselung in einem ungeschützten Netzwerk oder dem Internet unabdingbar.

1.11 XML Services

„A web service is a piece of functionality exposed through a web interface“ [Banerjee, A., 2001. S. 15]

Web Services, auch XML Services oder XML Dienste genannt, ermöglichen den Aufruf von Methoden über das Internet. Diese Funktionalität ist erreichbar durch Standard Protokolle wie HTTP. Dadurch können beliebige Clients RPC-ähnliche Aufrufe auf Servern im Internet tätigen, das Ergebnis wird im XML Format zurückgegeben. Die Botschaften, die zwischen Client und Server geschickt werden, sind in einem XML Dialekt namens SOAP kodiert. Dieses Protokoll definiert eine standardisierte Vorgehensweise beim Zugriff von Funktionalität auf fernen Maschinen.

Das grundlegende Konzept der Dienste ist die freie Wahl eingesetzter Technologien, Sprachen und Geräte. Diese Angelegenheiten unterliegen der Entscheidung des Entwicklers und dem Medium des Gerätes. Nur die Schnittstelle eines jeden Dienstes wird explizit festgelegt. Daher ist es unerheblich, ob der Client ein Java Servlet, ein VB.NET client, eine Perl Applikation oder ein WAP-Telefon ist. Es ist auch unerheblich, ob die Applikation ein Kalkulationsprogramm, ein Terminplanungsprogramm oder ein Zeichnungsprogramm ist. Die Ortung der Applikation ist auch unwichtig, sie sollte jedoch am Internet angebunden sein.

Der Unterschied zu proprietären Protokollen wie DCOM, CORBA, RMI oder JINI liegt in der Unabhängigkeit der Technologien der Teilnehmer. Das kommt durch den zugrundeliegenden Update Mechanismus. Bei DCOM, CORBA und RMI (JINI) wird ein Ping Mechanismus zum Herstellen und Aufrechterhalten der Verbindung eingesetzt. Dieser Ping Mechanismus braucht entweder spezifische Ports z.B. 9000, vergibt diese nach einem im Protokoll vorgesehenen Mechanismus, oder wählt den Port beliebig aus. Das hat auch den Hintergrund erhöhter Sicherheit, da sonst das Abhören eines Ports zur Kompromittierung des Datenverkehrs führt. Das HTTP Protokoll, das oft mit SOAP eingesetzt wird, basiert auf dem Austausch von Datagrammen, wie es im normalen Postverkehr geschieht. Eine lebendige Verbindung gibt es hier nicht, lediglich eindeutige Anschriften (URL und Port 80). Dadurch können auch Teilnehmer miteinander agieren, die auf völlig verschiedenen Technologien basieren.

Web Dienste sind primär für den Einsatz in Applikationen gedacht und nicht für Anwender. Sie können auch als ferne, wieder verwendbare Bausteine gedacht werden. Dadurch kann Redundanz vermieden werden und eine Bündelung von Know-how um gewisse Dienste erfolgen. Anbieter (auch als fachliche Anbieter zu verstehen, z.B. Brancheninformationen, typische Statikfälle, wirtschaftliche Eckdaten) können sich auf das Angebot gewisser fachspezifischer Dienste spezialisieren, diese können wiederum von Integrationssystemen, wie das cs, genutzt werden, um Informationen in einen gesamtheitlichen Kontext zu setzen.
Dienste haben folgende Vorteile:

- Plattform Unabhängigkeit (MS Windows, Linux, Unix, Solaris, BeOS..)
- Allgegenwärtiger Kommunikationskanal (HTTP, SMTP, Message Queueing)
- Wiederverwendbarkeit der Funktionalität
- Erweiterung des affektierten Aktionsraums
- Server Neutralität
- Sichere Verbindungen

Beim Einsatz von Web Diensten in der Objektmodellierung wird eine Abbildung (Mapping) zwischen dem SOAP Protokoll und dem Objektmodell gemacht. Diese Abbildung erfolgt händisch. Beim Einsatz von System.Reflection, kann sie auch automatisch erfolgen, anhand von den Namen von Eigenschaften oder sonstigen Attributen. Hierbei ist auf das Problem der Dopplungen und der Typinkompatibilität zu achten.

Diese Vorgehensweise ist starr und stößt bei der Weiterentwicklung des SOAP Protokolls und Web Dienste auf Skepsis. Maine [Maine, 2003. S.1] sieht das Problem wie folgt:

> *"…object-oriented programming was a good metaphor for building systems that reside in memory, but that metaphor breaks down when you think about trying to build distributed systems that are stable over time."*
>
> *"…OO systems assume too much infrastructure and don't have a way of answering the question "what happens when that infrastructure changes?""*
>
> *"…distributed systems don't respect logical boundaries. In the real world there are boundaries that govern human interaction…If there are logical boundaries that govern human interaction, why aren't there similar logical boundaries that govern the interaction of software components."*

Indigo (ein ß-Produkt von Microsoft) ist ein Sammelname für einen Satz API's die auf folgenden Prinzipien programmiert wurden [Box, D., 2005. S. 2]:

1. Grenzen sind explizit.
2. Dienste sind autonom.
3. Dienste interagieren mit Schemas und Verträgen, nicht Klassen und Typen.
4. Leitlinien werden eingesetzt um Kompatibilität zu erreichen.

Interessant ist, dass parallele Überlegungen in der Java Entwicklungs-Community vorkommen, obgleich Java bei der Implementierung des Objektmodells nicht zum Einsatz kommt. Laut Loughran und Smith [Loughran and Smith, 2005. S. 2] liegt dies der folgenden Ursache zu Grunde:

> "Undermining it all (Object/XML Mapping) is a fundamental difference between the type systems of XML (especially that of XML Schema and that of Java, making any mapping both complex and brittle."

Typische Inkompatibilitäten sind [vgl. Loughran and Smith, 2005. S. 2-5]:

Bindung von XML Elementen zu Java Klassen: Im Gegensatz zu modernen Objekt-orientierten Sprachen wie Java oder C#, die eine Vererbung durch Erweiterung einer Klasse zulassen, besteht die Möglichkeit in XML eine Vererbung durch die Einengung einer Klasse zu erreichen (derivation by restriction).

Mappen von XML Namen zu Java Identifizierern: XML Namen können beliebige Formen annahmen, z.B. können XML Namen mit einem Unicode Buchstaben, einem Ideographen oder einem Unterstrich „_“ anfangen. Solche Konstrukte sind nur bedingt in Java möglich. Somit würde ein automatisches Mapping der XML Namen `schrödinger`, `_unknown.type-set` oder `String` auf Java zu einem Fehler führen.

Enumerations: In Java wird die Handhabung von Enumerations ab der Version 1.5 intern durch Indices festgehalten. Somit führen Änderungen an der Enumeration zu einer Neuordnung der Einträge. Schon vorhandene Zuordnungen werden somit um einen oder mehrere Indices verschoben.

Nicht-übertragbare Typen: Einige Java Typen sind analog C# nicht transportierbar. Typen wie eine Datenbankverbindung oder `java.util.Calender` können nicht vermittelt, geschweige denn automatisch zugeordnet werden.

Serialisierung eines Objektgraphs: Java Objekte verweisen oft auf andere Objekte und können somit zyklisch referenziert sein. Da XML Daten nur hierarchisch darstellen kann, tritt hiermit eine Inkompatibilität auf. Diese kann mittels Querverweise in der XML Botschaft, zur Weiterverarbeitung am empfangenden Ende, gelöst werden. In der Praxis jedoch hat sich eine Variante des SOAP Protokolls durchgesetzt, zumindest in JAX-RPC, was eine solche Vorgehensweise nur mit einer proprietären Entwicklung zulässt.

XML Metadaten und Namespaces: XML erlaubt die Angabe einer namespace unmittelbar auf Ebene eines tags. In Java ist ein solches Konstrukt nicht möglich. Somit kann Java die Granulation von XML nicht erreichen.

Message Validierung: Typische SOAP Konstrukte validieren nicht Botschaften (Messages) bei einer Übertragung. Die Message wird nicht gegen das Schema validiert und die Anzahl der benötigten Botschaften auch nicht geprüft. Das führt entweder zum ignorieren des Problems oder dem Einsatz von proprietären Codes.

Inkorrekte Verquickung von XML und serialisierte Daten: Bei der Deserialisierung durch JAX-RPC und JAXM

Fehlerbehandlung: Die Implementations- und Plattformabhängigkeit in der JAX-RPC Fehlerbearbeitung steht im Kontrast zur generischen XML-basierten SOAP Fehlerbehandlung, die in der JAX-RPC durch den Typ `SOAPFaultException` zugänglich ist.

SOAP wurde entwickelt als eine Art RPC in XML über HTTP. Somit sollten die zwei großen Einschränkungen von CORBA/DCOM, nämlich die Plattformabhängigkeit und die Unfähigkeit auf Port 80 übertragen werden zu können, umgangen werden. XML-RPC war ein Vorläufer dieser Entwicklung.

SOAP wird jedoch oft in einem Szenario eingesetzt, wo typische RPC Überlegungen nicht greifen. Bei der Übertragung von großen Datenmengen etwa, oder dem Einsatz in einem schwachen Netzwerk oder WAN, spielen Überlegungen wie asynchrones Verhalten eine überragende Rolle. Die jetzige Asynchrone Implementierung besteht darin, dass der SOAP Aufruf in einem separaten Thread abgearbeitet wird, womit das asynchrone Verhalten vom Framework geleistet wird und nicht in der SOAP Implementierung vorhanden ist.

Eine mögliche Lösung, laut Loughran and Smith [Loughran and Smith, 2005. S. 8], besteht darin, eine einfache SOAP Implementierung auf XML aufzubauen. Somit können die in XML vorhandenen Technologien (vermutlich XPath, XSLT, XQuery) verwendet werden, um ein dynamisches Mapping zu ermöglichen.

Diese Lösung verfolgt einen ähnlich dynamischen Ansatz wie die neue Dienste-Plattform von Microsoft namens *Indigo*. Diese Implementierung führt weg von einem beschrittenen starren SOAP/Objekt Mapping. Die Auswirkung dieser Entwicklung ist, dass sich das starre objektorientierte Denken auflösen muss, um eine vernünftige Zuordnung zwischen den Objektmodellen teilnehmender Applikationen und dem verteilten System zu ermöglichen. Somit ist das Arbeiten mit dynamischen Objektmodellen möglich, was der Realität in der Baupraxis eher entsprechen würde.

Allerdings sind diese Überlegungen erst im Anfangsstadium und können z.Z. nicht für technische Problemlösungen eingesetzt werden, da auch nicht abzusehen ist, ob sie sich durchsetzen werden.

1.11.1 Technische Beschreibung

Web Dienste basieren ausschließlich auf W3C Standards. Alle Botschaften werden ausschließlich mit dem HTTP Protokoll verschickt. Die Botschaften zwischen Clients und Servern werden mit XML enkodiert. Das SOAP Protokoll bestimmt, wie ein WebDienst kodiert wird. SOAP wurde von einem Konsortium (Microsoft, Develop-Mentor, IBM, UserLand) entwickelt und vom W3C standardisiert. Mit SOAP kann jeglicher WebDienst erreicht werden, somit können bekannte Aufrufe beantwortet werden. Zusätzlich zu SOAP wirken eine Hand voll Protokolle, um WebDienste funktionstüchtig zu machen:

- XML – Extensible Markup Language, eine standardisierte Darstellung von Daten
- WSDL – Web Service Description Language, spezifiziert das Interface eines Web Services. Darin werden die Signaturen der Methoden aufgelistet, und mit diesem Dokument können die validen SOAP Botschaften ermittelt werden.
- DISCO – Discovery Protocol agiert als Pointer auf alle WebDienste, die sich auf einem bestimmten Webserver befinden. Damit können WebDienste einer Firma oder eines Instituts dynamisch publiziert werden.
- UDDI – Universal Description, Discovery and Integration, agiert im Internet als zentral zugängliches Register für WebDienste (Abb. 161).

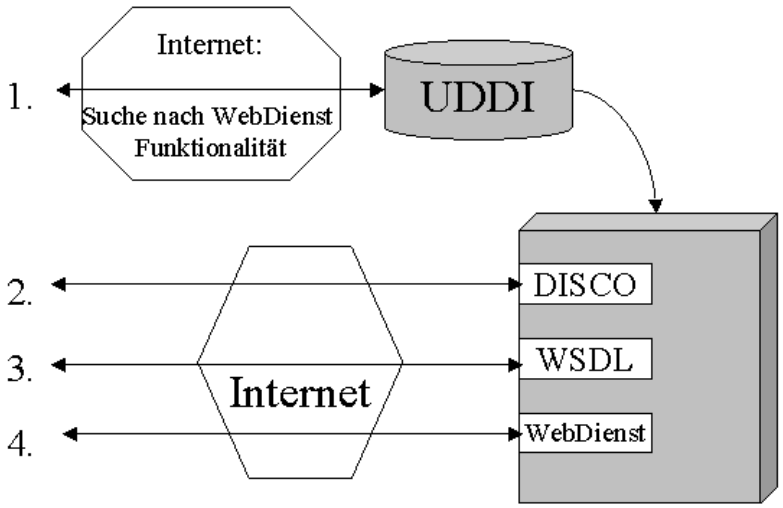


Abb. 161: Vorgehensweise einer Web-Dienst-Identifikation und das Nutzen der einzelnen Protokolle [Banerjee, A. et al, 2001. S. 19]

1. Mit UDDI kann der benötigte WebDienst ermittelt werden. Das UDDI Register zeigt auf eine Domäne mit dem gewünschten Dienst.
2. Auf der Domäne steht die entsprechende DISCO Datei zur Verfügung, mit einer Auflistung der auf der Domäne befindlichen Dienste.
3. Die WSDL eines gewünschten Dienstes enthält die Signaturen der WebDienste.
4. Die SOAP Botschaften werden in XML encodiert und beinhalten die Methodenaufrufe und resultierenden Daten, die wiederum an den Client zurückgegeben werden.

1.11.2 SOAP und WSDL

SOAP und WSDL sind von zentraler Wichtigkeit beim Einsatz von WebDiensten, die wiederum das Arbeiten mit verteilten Objekten ermöglichen. Darum ist eine gesonderte Betrachtung dieser Protokolle hilfreich.

1.11.2.1 SOAP

SOAP spezifiziert das Format der Methodenaufrufe und der übertragenden Parameter. SOAP beinhaltet die Regeln zur Serialisierung, zur Sicherung der Sendung, zur korrekten Information. SOAP basiert auf XML und setzt XMLSchema und Namespaces in großem Umfang ein. Aktuelle Informationen zu SOAP befinden sich unter http://w3.org/2000/xp.

Eine SOAP Botschaft kann einer der folgenden drei entsprechen:

- Method Call
- Response message
- Fault Message,

und hat in einfacher Form das Format:

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Header>
            <!-- Transaction-specific header details -->
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
            <!-- Transaction-specific elements -->
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Alle SOAP Botschaften haben das gleiche Format, einen Header und einen Body, und alle Elemente der SOAP Spezifikation scopen zu

```
http://schemas.xmlsoap.org/soap/envelope/
```

Ein Beispiel ist die ferne Instanzierung eines Objektes der Klasse „stuetze“. Nach der Änderung der Attribute wird das Objekt serialisiert. Der SOAP Aufruf gleicht:

```
POST /remStuetze/colMethods.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: http://tempuri.org/setDimensions

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="...">
  <soap:Body>
    <setColumn xmlns="...">
      <id>int</id>
      <height>double</height>
      <width>double</width>
      <length>double</length>
    </setColumn>
  </soap:Body>
</soap:Envelope>
```

Die Methode setColumn instanziert das Objekt mit der Identifikation id. Es wird den Werten `height, width` und `length` entsprechend Höhe, Breite und Tiefe zugewiesen.

Die Elemente haben folgende Funktion:

`<Envelope>`: Der oberste Knoten in einer SOAP Botschaft. Als erstes kommt ein `<Header>` Tag, gefolgt von einem `<Body>` Tag. Danach können Benutzer-definierte Tags folgen.

<Header>:Meta-Informationen. Können ein „mustUnderstand“ Attribut enthalten, das, wenn auf 1 gesetzt, den Empfänger zwingt, die Header Informationen ohne FehlerException zu interpretieren. Nützlich für Authentifizierungen.

<Body>: Enthält die serialisierten Methodenaufrufe sowie deren Parameter.
Ein Beispiel veranschaulicht einen einfachen Aufruf (gibt das *INode* Objekt eines Eintrags in dem Filter Objects wieder):

http://127.0.0.1/objServices/iNodes.asmx?op=getNodeByDescription

Der Methodenaufruf wird im Body Tag serialisiert. Der Namespace stammt vom WSDL was benutzt wird, um das Interface des Webdienstes und die SOAP Botschaften zu konstruieren.

Das Ergebnis dieses SOAP Aufrufs (mit dem Parameter=“Vorplanung Phase II“) ist:

```
<?xml version="1.0" encoding="utf-8"?>
  <base64Binary
    xmlns="http://tempuri.org/">77u/PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0
    idXRmLTgiPz48SU5vZGUgeG1sbnM6eHNk PSJodHRwOi8vd3d3LnczLm9yZy8yMDAxL1h
    NTFNjaGVtYSIgeG1sbnM6eHNpPSJodHRwOi8vd3d3 LnczLm9yZy8yMDAxL1hNTFN jaGVt
    YS1pbnN0YW5jZSI+PG5vZGVJZD5iNjVhYWRiOC1hZWMyLTQy NWEtYjJhZS1lMmE1NjhmNm
    Y3NGQ8L25vZGVJZD48cGFyZW50SWQ+YTNkZWM4ZTktZDcxYy00ZTY1 …
```

Der WebDienst hat das Ergebnis des Methodenaufrufs wiedergegeben.

Bei einem fehlerhaften Methodenaufruf würde ein <Fault> Tag im Body Tag erscheinen. Der Fault Tag kann folgende Elemente beinhalten:

- Faultcode
- Faultstring
- Faultactor
- Detail

Diese Informationen können wiederum von einem intelligenten Client ausgewertet werden, um entweder einen verbesserten Aufruf zu generieren, einen anderen WebDienst anzufragen, oder einen sanften Ausstieg zu ermöglichen.

XMLSchema listet die in modernen Programmiersprachen häufig verwendeten Datentypen. Einige wichtige davon sind:

- `string`
- `boolean`
- `float`
- `int`
- `dateTime`
- `binary`

Eine komplette Auflistung ist unter http://w3.org/TR/xmlschema-2/ zu finden.

Die meisten Aufrufe auf einem WebDienst werden mit dem HTTP POST command angefragt. Folgende Bedingungen müssen erfüllt werden:

- Der Content Type muß text/xml sein
- Ein zusätzlicher SOAPAction HTTP header muss vorkommen. Die URL entspricht dem WebDienst, dass von diesem SOAP Paket angefragt wird.

Der SOAPAction Header kann von Firewalls verwendet werden, um einkommende SOAP Pakete zu identifizieren und ohne ein scanning des gesamten XML Payloads passieren zu lassen, um dadurch den Durchsatz zu erhöhen. Der Header kann auch zum Blockieren von SOAP Paketen eingesetzt werden.

1.11.2.2 WSDL

Das WSDL beschreibt die Elemente eines WebDienstes, die Methodensignatur, die Parameter, die URL und Informationen zum WebDienst.
WSDL ist standardisiert und XML basiert. Es kann daher auch von Maschinen gelesen und interpretiert werden. WSDL Dateien werden oft von WebDiensten wie .Net WebServices implementiert, um ein lokales Proxy Objekt zu instanzieren, womit die Methodenaufrufe getätigt werden.

Ein WSDL Dokument ist komplex, es besteht aus 5 Sektionen, die alle im <definitions> Tag vorkommen:

- types – Definiert die unterschiedlichen benutzerdefinierten Datentypen des Dokuments.
- message – Eine abstrakte Repräsentation der logischen Botschaften, die vom Dienst akzeptiert und zurückgegeben werden.
- portType – Eine Liste abstrakter Operationen, die ungefähr der Signatur des Methodenaufrufs entsprechen. Es definiert die logischen Botschaften, die jede Operation akzeptiert und zurückgibt.
- binding – Definiert das Format und Protokoll eines portTypes. Diese Sektion definiert, welche Protokolle, wie z.B. SOAP, dieser Dienst interpretiert und versteht, um kommunizieren zu können.
- service – Die physische Adresse des WebDienstes, der mit einem named port die spezifische Adresse referenziert.

Alle WSDL Definitionen müssen innerhalb des <definitions> Tag vorkommen.
Die <binding> Sektion gibt das Übertragungsprotokoll an: SOAP
Der <portType> Tag beinhaltet den Methodenaufruf z.B.: setHeight
Der <portType> Tag enthält die Segmente des Aufrufs durch eine Referenz auf das <messages> Tag. Diese wiederum kann eine Referenz auf <types> haben, die die Typinformationen für die Parameter liefert. Die <binding> Sektion enthält Informationen über den Namespace, der für den Methodenaufruf erforderlich ist.
Der <service> Teil führt die URL auf, die zur Kommunikation mit dem WebDienst aufgerufen werden muss. Der SOAPAction HTTP header muss dem Wert des soapAction Attribut im <binding> Sektion des WSDL entsprechen. Das ermöglicht der Firewall, einkommende SOAP Pakete überprüfen zu können.

Folgend wird der Aufruf des einfachen Dienstes für die Instanzierung und Änderung einer Stütze erläutert:

```
POST /remStuetze/colMethods.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/changeHeight"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="...">
  <soap:Body>
    <changeHeight xmlns="...">
      <id>int</id>
      <newHeight>double</newHeight>
    </changeHeight>
  </soap:Body>
</soap:Envelope>
```

Die Rückgabe der Methode:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="...">
  <soap:Body>
    <changeHeightResponse xmlns="...">
      <changeHeightResult>double</changeHeightResult>
    </changeHeightResponse>
  </soap:Body>
</soap:Envelope>
```

A.II Einzelheiten zur Robotik, Aspekte einer VE.NET Umgebung und Simulationsverfahren

VR-Technologien integrieren viele IT-Facetten und Methoden, um die virtuellen Darstellungen möglichst realitätsnah zu gestalten. Eine Vielzahl von Eingabegeräten und sogar sensorbestückte Räume dienen als Interaktionsinstrumente zwischen den Benutzern und der Anwendung. Typische Eingabemedien sind [Vince, J., 1998. S.76]

- Tracking
  - Mechanische Tracker, z.B. BOOM3C, FS², MEDVIEW und PUSH
  - Optische Bewegungstracker wie HiRES von ExpertVision
  - Ultrasonische und Elektromagnetische Sensoraufnehmer wie FASTRACK, ISOTRACK, LONG RANDER

- Eingabegeräte
  - 3D Maus
  - Handschuhe, CyberGlove von Virtual Technolgies, Haptic Glove [Bar Cohen, Y. et al., 2001. S. 4]

- Ausgabegeräte
  - Force feedback sensors z.B. Super Expanded Workspace PHANToM
  - 3D Brillen, wie die CrystalEyes Shutter Glasses
  - Displays: 3D Screens wie ZScreen oder die moderneren Polarisationsfilter [Geiger, C., 2004. S. 97]
  - HMD – Head Mounted Device wie V6, V8 und Cyclops
  - BOOMs, Geräte zur 3D Positionierung eines Benutzers
  - Retinale Displays, ein Laser wird direkt in die Retina eines Benutzers geleuchtet, um so die Orientierung des Benutzers zu orten. Ein System wurde im Human Interfaces Laboratory der University of Washington entwickelt.
  - Panoramische Bildschirme, die seit Jahren in den Flugsimulatoren verwendet wurden und virtuelle Tische (z.B. Fakespace's Immersive Workbench), wo VR Gebilde durch einen durchsichtigen Tisch auf 2 Flächen, die wiederum verstellbar sind, beleuchtet werden können (SIGGRAPH).
  - CAVE – Cave Automatic Virtual Environment der Universtiy of Illinois. Der Benutzer wird in einer 3D Landschaft versetzt.
  - Augmented Reality – computergenerierte Bilder werden auf einer realen Sicht

der Welt aufgelegt. So können Landschaften mit zusätzlichen Informationen aus Satellitenaufnahmen oder Gesteinsbohrungen angereichert werden, um Geologen die Suche nach Ressourcenvorkommnissen zu erleichtern.

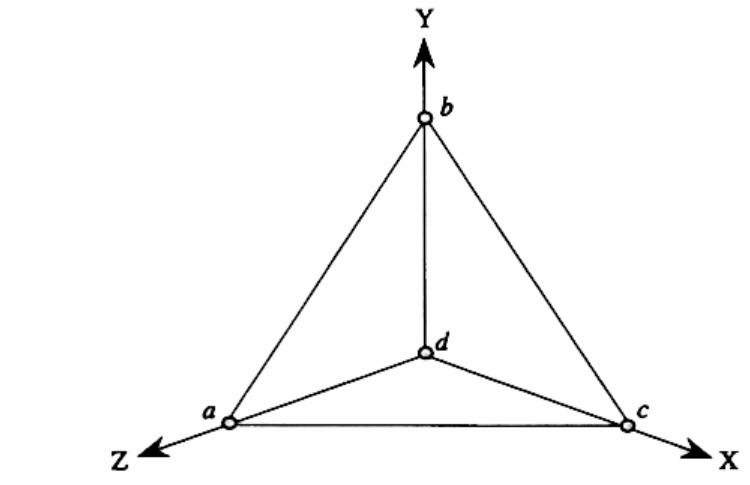
- Audiogeräte
  - Virtuelle Landschaften können mit Audioinformationen angereichert werden. Dazu gehören die Geräusche von Motoren, Radios, Uhren, Menschen. Von der Heyde [Von der Heyde, 2001. S.2] hat Untersuchungen unternommen, um die räumliche Orientierung bei Menschen anhand von Audioinformationen zu verstehen. DSP (Digital Signal Processing) Systeme wie Huron und CP4 benutzen Chips und Software, um Audioorientierung zu unterstützen.

Weitere Eigenschaften, die die Qualität einer VR Software maßgeblich beeinflussen:

- LOD – Level of Detail zur Minimierung des benötigten Rechnerbedarfs, oder zum schrittweise Laden von großen Datenbanken
- Objekt Skalierung, Rotation und Translation
- Randbedingungen
- Artikulierte Eigenschaften
- Animation
- Kollisionsidentifizierung
- Parallele Welten
- Lichtquellen
- Ereignisbehandlung
- Audio
- Kontrollsprachen
- Simulation
- Sensoren
- Stereosichten
- Tracking
- Networking

Im CAD werden mathematische Abbildungen eingesetzt, um maßgetreue Darstellungen von Objekten zu erzielen. Der Nachteil besteht darin, dass solche Gebilde mit sehr unterschiedlichen Formeln abgebildet werden, je nach der Form eines solchen Gebildes. Darum ist es unmöglich, allgemeingültige Formeln für die Abbildung von 3D-Geometrien zu erstellen.

In einer VR Welt werden geometrische Abbildungen als Mosaik von Polygonen oder Dreiecken beschrieben. Dreiecke sind wünschenswert, da sie einfach berechnet und manipuliert werden können und zusätzlich nicht durch Torsion verformt werden. Die Trennung eines Dreiecks in einem sog. Triangle Table (Liste von Vektoren, die die Dreieckspunkte darstellen) und Vertex Table (kartesische Koordinate der Punkte) in Abbildung 162, ermöglicht die Änderung einer Geometrie mit minimalem Aufwand. In Abbildung 162 wird ein 3D Körper als Zusammensetzung von 4 Dreiecken dargestellt (ein Dreieck am Boden, zwei vertikale an den Seiten, ein Dreieck zum Beobachter). Wenn Punkt (Vertex) b in Abbildung 162 verschoben wird, braucht nur die Vertextabelle geändert zu werden, da sich die geometrischen Formen der Dreiecke nicht verändert haben, lediglich die Position des einen Punktes b. Die Triangletabelle bleibt unverändert.
Polygone und Dreiecke können mit geringem Aufwand bearbeitet werden, und farbige Darstellungen schnell berechnet werden.



Triangle Table			
name	Vertices		
abc	*a*	*b*	*c*
cbd	*c*	*b*	*d*
adb	*a*	*d*	*b*
acd	*a*	*c*	*d*

Vertex Table			
name	*x*	*y*	*z*
a	0	0	1
b	0	1	0
c	1	0	0
d	0	0	0

Abb. 162: Interner Speicher von Dreiecken als Mosaiksteine eines 3D-Gebildes [Vince, J., 1998. S. 30].

2 (Tele-) Robotik

Die moderne Robotik beschränkt sich nicht nur auf Fertigungsanlagen, Schwerindustrieroboter, Tiefseegreifer, Raumfahrtrobotik oder Roboter für gefährliche Einsätze (Strahlung, Hitze, Gift). Verstärkt wird nach interaktiven, mit den menschlichen Sinnesorganen steuerbaren Maschinen und zunehmend die Integration von Mensch und Maschine verlangt.

Von der Heyde [Von der Heyde, 2001. S.19 ff] beschreibt den Einsatz eines VR-gekoppelten Motion Labs zur Feststellung der menschlichen Orientierungsfindung beim Einsatz folgender Sinne:

- Vision
- Akustik
- Tasten
- Vestibularsinn des inneren Ohres

Die VR-Technologie des Motion Labs wurde genutzt, um den Einfluss dieser Sinne auf die räumliche Orientierung separat und im Zusammenspiel zu messen und zu identifizieren. Bei den biometrischen Untersuchungen handelte es sich um Regelkreise und Kreislaufsimulationen. Das Subjekt betätigt ein Eingabegerät (Steuerung). Diese Daten werden über einen Analog-Digital Wandler durch das Simulationsprogramm geschleust, dieses wiederum steuert die Plattform und den VR-Helm des Subjekts. Da die Eingabe und Ausgabemedien unterschiedliche Frequenzen haben, entsteht die Notwendigkeit:

- verteilte Systeme anzuwenden, für jedes signifikante Gerät wird ein Server bereitgestellt;
- asynchroner Aktualisierungsverfahren, mit der Regel, dass kein Instrument auf eine sofortige Datenlieferung vertrauen kann.

Von der Heyde [Von der Heyde, 2001. S. 42] beschreibt weiterhin Technologien, die bei einer Differenz der Eingabe- und Ausgabe-Frequenzen einzusetzen sind, z.B. Extrapolationen. Diese Problemstellungen und Lösungen kommen grundsätzlich in VR-Welten vor und werden in Anhang A.II.3.1.1.2 vertieft.

Roboter gelten als *de rigeur* in der Fertigungsindustrie. Die virtuelle Simulation von Robotern gilt der Planung eines Fertigungseinsatzes, sowie der Konstruktion von Hilfsgeräten und -flächen. Hinzu kommt, dass Steuerer ausgebildet werden sollten. Weiterhin sollte VR eine intuitive Methode darstellen, um die Arbeitsabläufe und Bewegungen von Robotern zu simulieren und somit zu programmieren. Solche Verfahren werden auch eingesetzt, um Roboter und deren Steuerer für gefährliche Aufgaben, wie die Wartung eines Stromkabels vorzubereiten [Dai u. Su, 2002. S. 1].

Die Simulation von Robotern wird mittels objektorientierter Module mit 3D Visualisierungsschnittstellen vereinfacht. Ferretti [Ferretti et al., 1999. S. 16] beschreibt den Einsatz einer standardisierten Objekt-Bibliothek, bei der Komponenten je nach Bedarf kombiniert werden können, um eine Simulation mittels VRML als Visualisierungsprotokoll zu ermöglichen. Nishigaki, S. und Maruyama, Y. [Nishigaki, S. und Maruyama, Y., 1999. S.88] bzw. Maruyama, Y. und Iwase, Y. [Maruyama, Y. und Iwase, Y., 2000. S. 503 ff] beschreiben den virtuellen und physikalischen Aufbau einer intelligenten Feldfabrik. Ziel ist die Automatisierung des Bauablaufs einer typischen Hochbaustelle. Zuerst werden die logistischen Abläufe analysiert und systematisiert. Danach wird ein VR-Modell des Ablaufes entwickelt, um ihn zu optimieren. Es werden sog. Software Agents eingesetzt, die mittels künstlicher Intelligenz gewisse Aufgaben zur Stabilisierung des Systems erfüllen. Sie dienen somit als Immunsystem zur Kompensation von Störungen während des geplanten Ablaufes. Beispiele sind:

- Control-Recovery
- Fault-Warning
- Procedure-knowledge
- Data Mining
- Network agents

Diese VR-Simulation des Bauprozesses wurde benutzt, um die Machbarkeit des sog. Takada Modells (Kaffeetisch) zu erproben (Abb. 163). Diese Technik sieht die Vorfertigung von Konstruktionselementen in einer Gruppe vor (Stützen, Balken, Platten), die wiederum mit einem Kran automatisch positioniert werden. Die optimierte VR-Simulation dient als Vorlage für eine Umsetzung einer vollautomatisierten Hochbaustelle der Fa. Shimizu, Japan.

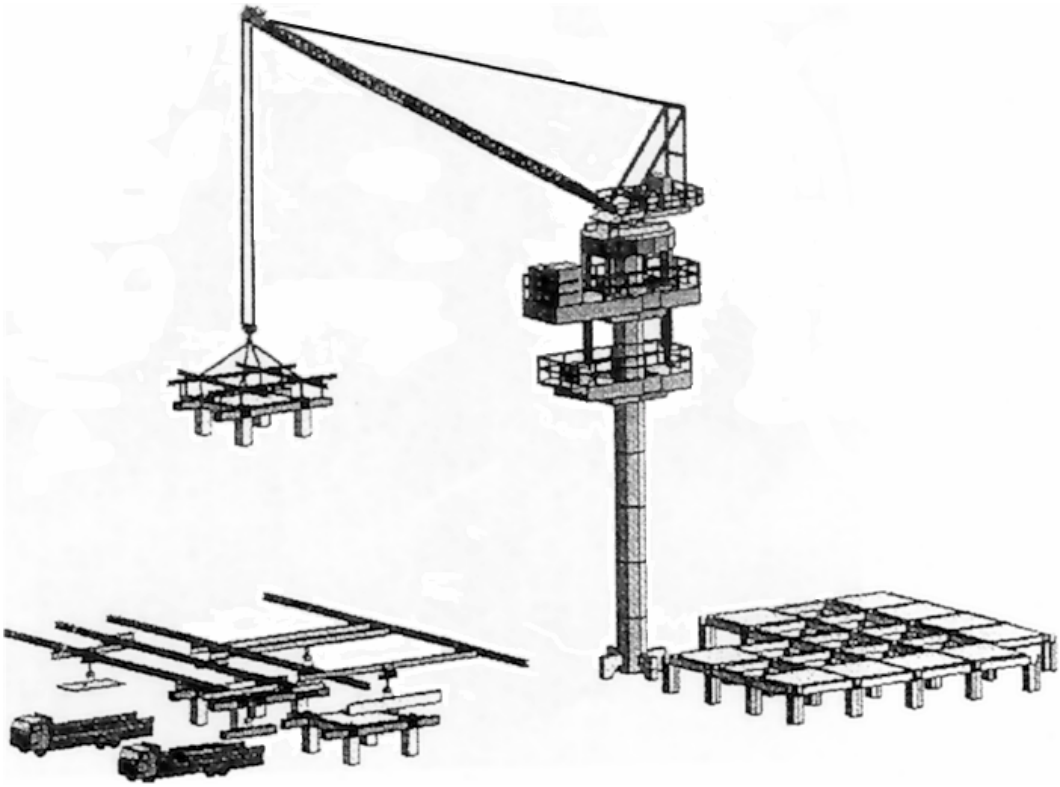


Abb. 163: Virtueller dynamischer Bauablauf mit der Takada Methode der Fa. Shimizu [Maruyama, Y. Iwase, Y., 2000. S. 510]

VR-Modellierung wird zunehmend in der medizinischen Robotik eingesetzt. Sie dient zur Optimierung des Entwurfs klinischer Geräte, fernoperativer Eingriffe, sowie der Planung und dem Einüben komplexer Operationen, z.B. in der Neurochirurgie. Meistens dient die VR-Modellierung im klinischen Bereich zur Einübung in die chirurgische Praxis, vor allem im Bereich der minimalinvasiven Chirurgie [DeCou et al., 2002. S. 73 ff; Bar-Cohen et al., 2001. S.1 f.].

Die Technologien einer net-VE Umgebung, die hauptsächlich im Bereich des militärischen Einsatzes und der Spieleindustrie entwickelt wurden, nehmen schrittweise Einzug in baubetriebliche Applikationen.

3 Aspekte einer net-VE Umgebung

3.1 Ressourcenverwaltung

Singhal und Zyda geben das Informationsprinzip einer virtuellen Umgebung (*Networked Virtual Environment Principle* – auch als net-VE bekannt) wie folgt an [Singhal, S. and Zyda, M., 2000. S. 184]:

> *"The resource utilization of a Net-VE is directly related to the amount of information that must be sent and received by each host and how quickly that information must be delivered by the network."*

Diese Formulierung kann wie folgt konkretisiert werden:

Ressourcen ≈ M x H x B x T x P

wobei:

M = Anzahl Botschaften, die im net-VE gesendet werden

H = Durchschnittliche Anzahl Zielhosts für jede Botschaft

B = Durchschnittliche Bandbreite pro Botschaft

T = Priorität der Lieferung einer Botschaft. Große T-Werte implizieren eine unverzögerte Lieferung, kleine T-Werte erlauben Verzögerungen

P = Anzahl der Prozessorzyklen zur Sendung und zum Empfang jeder Botschaft

Anhand dieser Formel können verschiedene Ansätze der Ressourcenverwaltung zusammengefasst werden (Abb. 164):

Methodik	Auswirkung auf das Informationsprinzip	Beschreibung
Dead-Reckoning	M-, P+	Host-basierte Projektion potentieller Positionierungsdaten mittels Vektor-Arithmethik. Aktualisierungspakete können dadurch reduziert werden; Host-Prozessoraktivität wird erhöht.
Packet Compression	B-, P+	Größenreduzierung der Pakete durch Einsatz von *lossy* oder *lossless* Techniken
Packet Aggregation	M-, B-, P+, T+	Mehrere Pakete werden aggregiert mittels *Timeout* oder *Quorem* Techniken
Multicasting	H-, M+	Durch Gruppierung von Ursprungsentitäten und net-VE Regionen können die Anzahl der Zielhosts bei der Aktualisierung reduziert werden.
Area-of-Interest Filtering	H-, M+, P+, T+	Explizite Informationsfilter werden definiert und an einem Abonnementmanager abgearbeitet.
Projection Aggregations	H-, B-, M+, P+, T+	Granuliert dosierte Multicast Gruppen werden für bestimmte Abbo-Filter predefiniert.
Reduced level-of-detail (LOD)	H-, B-, M+, P+	Detaillierungsgrad wird proportional zur Sichtweite justiert
Temporal Contour	T-, P+	Weit verzweigte Entitäten werden proportional zur wahrgenommenen Netzwerklatenz aktualisiert
Server Clusters	P-, T+	Serverhierarchien; Peer-to-peer Server Kommunikation
Peer-server communication	T-, B-, M-, T+, B+ M+	Die Netzwerkkommunikation schaltet zwischen peer-to-peer und client-server Modus entsprechend der Netzwerkeigenschaften

Abb. 164: Möglichkeiten der Ressourcenverwaltung [Sighal, S. and Zyda, M., 2000. S. 186]

Im Folgenden werden nur solche Techniken behandelt, die das hier entwickelte System für den Multiuser Einsatz wesentlich optimieren können.

3.1.1 Projektionsalgorithmen

Fieldname	Contents	Field Size (bytes)
PDU header	Protocol version, exercise ID, PDU type, protocol family, time stamp, length, padding	12
Entity ID	-	6
Force ID	-	1
Number of articulation parameters	-	1
Entity type	Entity kind, domain, country, category, subcategory, specific, extra	8
Alternative Entity Type	Entity kind, domain, country, category, subcategory, specific, extra	8
Entity linear velocity	x,y,z	12
Entity location	x,y,z	24
Entity orientation	Psi, theta, phi	12
Entity appearance	-	4
Dead reckoning parameters	Dead reckoning algorithm, other parameters, entity linear acceleration, entity angular velocity	40
Entity marking	Character set, marking	12
Capabilities	-	4
Articulation paramaters	Parameter type, change, ID-attached to, paramater type, parameter value	N * 16

Abb. 165: Distributed Interactive Simulation (DIS) Entity State PDU [IEEE93] [Sighal, S. und Zyda, M., 1999. S. 28]

Bei dynamischen Objekten wie Baugeräten (z.B. Kräne, Bagger) können Projektionsalgorithmen eingesetzt werden, um den Aktualisierungsaufwand der Position und Ausrichtung der Geräte zu minimieren. Diese Art Vereinfachung des Aktualisierungspakets führt zu stark reduzierter Paketgröße und damit zu Gewinn in der Bandbreite und wird im DIS-Protokoll eingesetzt (siehe Abb. 165).

So könnte ein Turmdrehkran wie in Abbildungen 166 und 167 als aktiver Teilnehmer des Objektmodells modelliert werden (Neethling, D. und Horn, J. 2004. S.6):

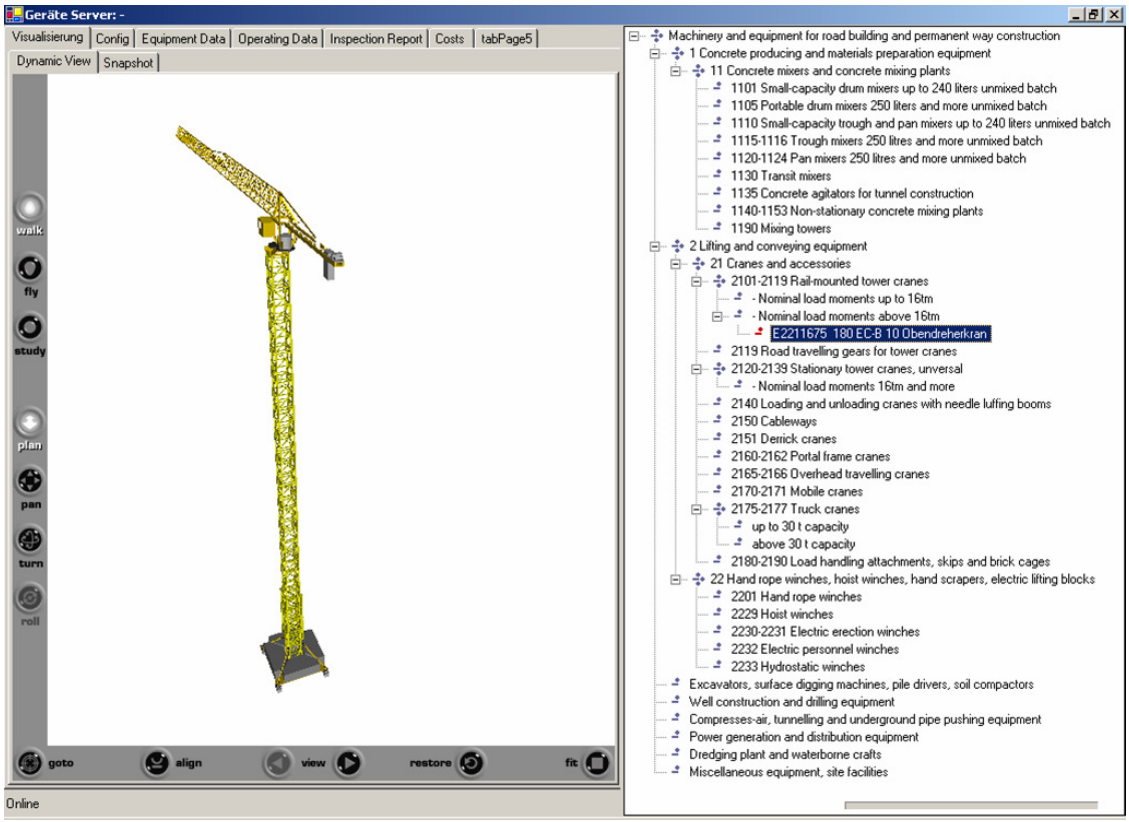


Abb. 166: VRML Darstellung eines Turmdrehkranes. Modell zur Aktualisierung durch Echtzeit-Daten aus CAN-Bussen [Neethling und Horn, 2004. S. 6].

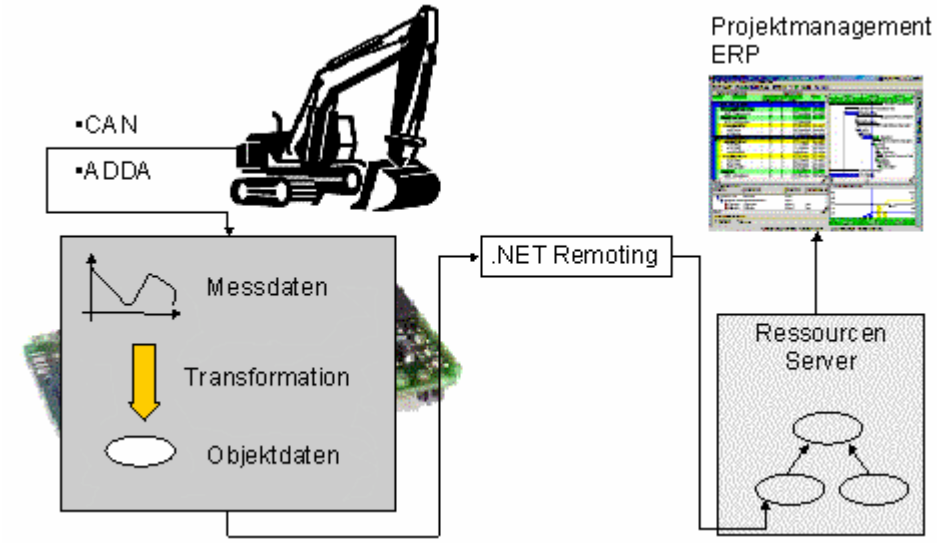


Abb. 167: Zyklus einer automatisierten Produktionsgrößenüberwachung für eine Baustelle [Neethling und Horn, 2004. S. 4].

Da es sich um ein konstantes Objekt im Sinne einer Veränderungshierarchie handelt, können die Informationspakete mit Verfahren wie DIS modelliert werden. Nachteile wären der zu erwartende Verlust an Qualität, die Komplexität der Algorithmen und die benötigten objekt-spezifischen Einstellungen. Diese Methode der Optimierung lohnt sich bei begrenzter Bandbreite, verfügbarer lokaler Rechnerkapazität und bei zahlreichen net-VE-Teilnehmern. Bei erhöhter Präzision und Konsistenz sollte darauf eher verzichtet werden.

3.1.2 Multicasting

IP Multicasting ist ein Netzwerk-Protokoll zum Senden von Paketen an eine Multicastgruppe, die mittels einer Multicast-Adresse identifiziert wird. Dadurch wird die Anzahl der Empfänger eingegrenzt. Ein Multicast-Paket wird entlang der zu sendenden Baumstruktur von einem abonnierten Knoten kopiert. Dadurch wird der Fluss des gesendeten Pakets nicht behindert. Da Multicastgruppen eine Vielzahl an nicht miteinander vernetzten Informationen erhalten können, sollte die Möglichkeit der Informationsflut und damit reduzierter Bandbreite bei der Architektur eines Multicast-Systems berücksichtigt werden.

Eine Optimierungsmöglichkeit besteht darin, gewisse Hosts in einer Multicast Subscriptiongruppe zu beteiligen. So werden nur die dieser Gruppe zugehörigen Hosts über den Zustand eines Sender-Hosts informiert (Group-per-Entity Technique). Das ist vor allem einsetzbar, wenn es in der net-VE sehr unterschiedliche Interessengruppen gibt, die mit getrennten Objektgruppen agieren. Hosts erfahren Adressinformationen über relevante interessierte Hosts über einen Multicast-Directory Service, wie es z.B. im Diamond-Pack des MERL (Mitsubishi Electric Research Laboratory) implementiert wurde. Solche Systeme haben folgende Limitationen:

- Multicast Adressen müssen mit anderen Applikationen, die auch auf diesen Adressen basieren, geteilt werden
- Große Multicast-Gruppen erzeugen viele *Join* und *Leave* Anfragen
- Aktuelle Netzwerkkarten können nur eine begrenzte Anzahl Multicast Host Adressen verwalten.

3.1.3 Area-of-Interest Filtering

Eine Anzahl von Subscription Servern verwaltet die von den Hosts angegebenen Area-of-interest Filter. Jedes Datenpaket wird evaluiert und Hosts, die für dieses Datenpaket entsprechend dessen subscriptions in Frage kommen, werden mit diesem Paket versehen. *Area-of-interest Filtering* ist eine vereinfachte Form des *aura-nimbus* models. Dieses Modell beschreibt Datenfluss-Optimierungen. Entitätsinformationen sollten nur solchen Entitäten, die diese Informationen auch wahrnehmen können, zur Verfügung gestellt werden. Diese Einflusssphäre einer Entität stellt deren *aura* dar. Zielentitäten werden aufgrund von

- Virtual world location
- Entity type
- Sensor capabilities
- Other characteristics

festgelegt.

Ebenso interessiert sich eine Entität nur für einen gewissen Satz Entitäten in der virtuellen Welt. Die oben genannten Eigenschaften definieren diesen Satz Entitäten. Dieser Satz wird der *nimbus* genannt. Jedes Stück Information wird individuell verarbeitet und nur den Entitäten bereitgestellt, deren *nimbi* sich mit der *aura* der Ursprungsentität schneiden.
Dieses Model wurde im MASSIVE-1 System [Singhal, S. und Zyda, M., 1999. S. 284] eingesetzt, konnte jedoch aufgrund folgender Faktoren nicht skaliert werden:

- Hoher Rechenbedarf
- Dadurch, dass jedes Paket mit einer festen Anzahl Zielentitäten assoziiert ist, können effiziente Verteilungsmechanismen wie Multicasting nicht eingesetzt werden.

Datenflussmanagement wird in drei Kategorien eingeteilt: *area-of-interest* Filter, *Multicasting*, und *Subscription-based* Aggregation.

3.1.4 Aggregations

3.1.4.1 Packet Aggregation

Anders als bei der Packet Kompression, bei der Pakete komprimiert werden, um die zu übertragende Größe zu minimieren, werden bei der Packet Aggregation Pakete zusammengefahren, um die Anzahl der zu übertragenden Pakete zu minimieren. Dadurch, dass die gesamte Anzahl Packet Headers reduziert wird, wird Netzwerk-Bandbreite eingespart. Packet Headers für UDP/IP und TCP/IP betragen 28 bzw. 40 byte, daher kann je nach net-VE Anwendung bis zu 50% Paketgröße eingespart werden. Wenn ein zyklisches Packetupdate pro Host unternommen wird, werden alle Pakete aggregiert und mit einem Header versendet. Wenn jedoch verschiedene Algorithmen für einzelne Entitäten auf einem Host verwendet werden, wie es z.B. beim Dead-Reckoning Verfahren der Fall ist, müssen Paketsendungen künstlich aufgehalten werden, bis eine genügende Anzahl solcher Pakete für eine Paketaggregation zur Verfügung steht.

Paketaggregation stellt einen Kompromiss zwischen der Aktualität einer Information und der Anzahl gesendeter Pakete (Bandbreite) dar. Je länger auf Pakete zur Aggregation gewartet wird, desto weniger Bandbreite wird aufgenommen, jedoch desto wertloser sind die Informationen.

Verschiedene Strategien bieten sich für die Paketaggregation an:

- Timeout-based Transmission policy – der Paket Aggregator wartet auf Pakete und sendet diese nach dem Ablauf eines Timeouts
- Quorum-based Transmission policy – der Packet Aggregator aggregiert Pakete bis zu einem Quorum (gewisse Anzahl von Paketen), danach wird das aggregierte Paket gesendet.
- Kombinierte Transmission Policy – Der Paket Aggregator sendet Pakete nachdem das Quorum erreicht wurde, oder bis der Timeout abgelaufen ist, je nachdem, welche Bedingung vorher eintritt.

3.1.4.2 Aggregation Server

Dadurch, dass viele Hosts nur eine Entität auf einmal verwalten können, wird die Einsparung durch Packet Aggregation bedingt durch die Qualitätsanforderungen. Besser ist es, hier Aggregation Server für ein net-VE einzusetzen. Diese können auch nach Entitätstyp spezialisiert werden, z.B. Entitäten mit schnellen Bewegungen werden an einem Server gesichtet, stationäre Entitäten an einem anderen Server. Auf diesen Servern können die oben beschriebenen Aggregationsstrategien eingesetzt werden, bevor die Pakete an die anderen Hosts weitergeleitet werden. Packet-Aggregations können auch nach den Kriterien Lokation, Typ oder LAN gestaltet werden, um somit eine der Architektur entsprechend optimale Verdichtung zu erlangen.

3.1.5 Reduced LOD

Darstellungen von Objekten können je nach Sichtweite vom Beobachter mit verschiedenen Detaillierungsstufen versehen werden. So können Objekte, die weit entfernt liegen, mit einer geringeren Detaillierung als nahe-liegende Objekte versehen werden, um Speicher und Bandbreite zu sparen. Die Berechnung des Level Of Detail erfolgt proportional zur Distanz zum Beobachter und wird, je nach Qualitätsanforderungen, an die geometrische Darstellung angepasst. Bei manchen net-VE LOD Verfahren werden Polygone aus der Geometrie gestrichen je nach Distanz zum Beobachter [Singhal und Zyda, 2000. S. 157], bei anderen Darstellungssprachen wie VRML (X3D) werden Objekte je nach Distanz vom Benutzer durch mehr detaillierte Darstellungen ersetzt.

Folgendes X3D Fragment beschreibt 4 Knotenpunkte, je nach Distanz zum Beobachter [Brutzman, 2002. URL: http://www.web3D.org/ TaskGroups/x3d/ translation/ examples/Vrml2.0Sourcebook/Chapter25-Level Of Detail/ Figure 25.04 Three Torches Side BySide.x3d. WS]:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE X3D PUBLIC "http://www.web3D.org/TaskGroups/x3d/translation/x3d-
compact.dtd"

"file:///C:/www.web3D.org/TaskGroups/x3d/translation/x3d-compact.dtd">
<X3D version='3.0'>
      <head>
            <meta name='translator' content='Don Brutzman' />
            <meta name='description' content='Three torches within an LOD
            node' />
      </head>
      <Scene>
            <NavigationInfo speed='2' type='"FLY" "EXAMINE" "ANY"' />
                  <LOD range='4, 8, 16' center='0 0 0'>
                  <Inline url='"Figure25.01TorchHighDetail.wrl" "..."' />
                  <Inline url='"Figure25.02TorchMediumDetail.wrl" "..."' />
                  <Inline url='"Figure25.03TorchLowDetail.wrl" "..."' />
                  <WorldInfo info='null node' />
            </LOD>
      </Scene>
</X3D>
```

Die unterschiedlichen Abbildungen, wenn nebeneinander aufgeführt, stellen unterschiedliche Darstellungsqualitäten dar (Abb. 168):

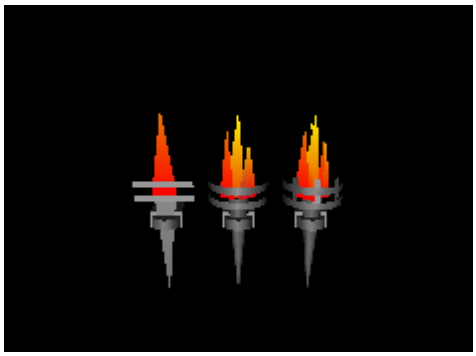


Abb. 168: Aus Vrml 2.0 Sourcebook, Chapter 25 - Level Of Detail: Figure 25.04 Three Torches Side By Side

Wenn diese qualitativ unterschiedlichen Formen in Korrelation zur Beobachtungsdistanz eingesetzt werden, entsteht ein realistisches, jedoch vom Speicher und der Bandbreite her optimiertes Bild (Abb. 169):

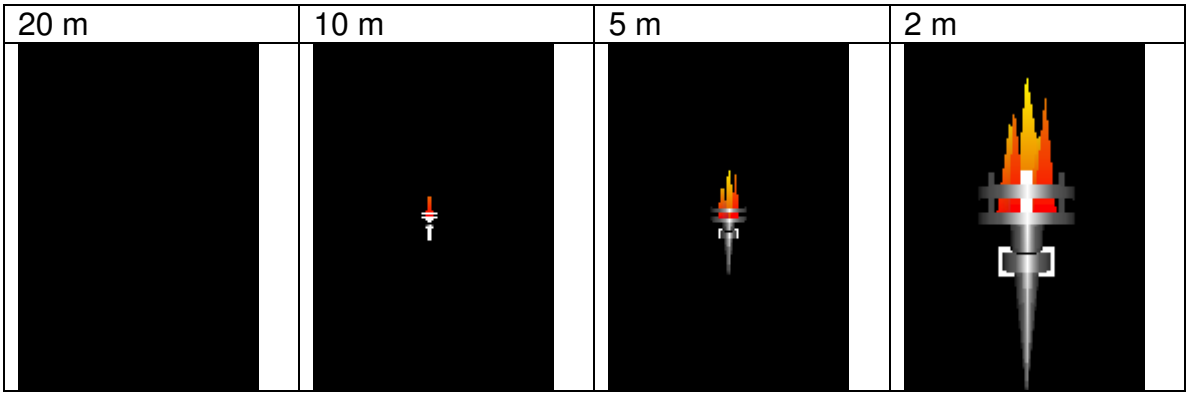


Abb. 169: Snapshots aus 4 X Positionen. Aus Vrml 2.0 Sourcebook, Chapter 25 - Level Of Detail: Figure 25.05 Three Torches Single LOD
[Brutzman, 2002. URL: http://www.web3D.org/TaskGroups/x3d/translation/examples/Vrml2.0Sourcebook/Chapter25-LevelOfDetail/Figure25.05ThreeTorchesSingleLOD.x3d. WS]

So werden geometrische Formen mit unterschiedlicher Darstellungsqualität je nach Distanz zum Beobachter eingesetzt, um Speicher und Bandbreite zu optimieren, ohne effektiven Verlust an Qualität.
LOD Verfahren können erheblich zu der Performance einer net-VE Applikation beitragen.

Eine effektive Methode zur Manipulation der LOD Wahrnehmung ist die Bereitstellung einer Multiple Channel Architecture, darunter

- Rigid-Body channel – Position, Orientierung und Struktur werden nur aktualisiert, wenn signifikante Änderungen vorgekommen sind. Diese Art der Darstellung ist am effizientesten und sollte nur bei distanzierten oder passiven Entitäten eingesetzt werden, da nicht-signifikante Strukturänderungen nicht aktualisiert werden.
- Approximate-body channel – Ein vereinfachtes Modell der Entität wird mit der Aktualisierung von radialer Länge, Artikulationsvektor, lokaler Koordinate erzeugt. Die Entitätsstruktur wird zwischen den Paketpunkten interpoliert.
- Full-body channel – Die Entitätsstruktur wird durch die Lieferung aller Entitätsinformationen beschrieben. Dadurch wird eine hohe Darstellungsqualität erreicht, jedoch zum Preis erhöhter Bandbreite.

Die Benutzerwahrnehmung kann in passive und aktive Elemente eingeteilt werden, und mittels Local Temporal Contours ausgenutzt werden, um Bandbreite zu sparen, bei hoher Darstellungsqualität. Drei Eigenschaften müssen erhalten bleiben:

- Der lokale Benutzer muss die Möglichkeit besitzen, mit passiven Elementen zu interagieren.
- Der lokale Benutzer muss die Interaktion von passiven und aktiven Objekten beobachten können.
- Der lokale Benutzer muss die Trajektion von entfernten Objekten als geglättet beobachten.

Ressourcen Management mit Methoden wie Local Temporal Contours sind seit dem Einsatz des Internets für Träger von net-VE Systemen immer wichtiger geworden. Dadurch sollten die Begrenzungen in Bandbreite und die hohe Latenz des Mediums umgangen werden.

4 Simulation

Die Simulation dient eher einer Betrachtung von Prozessabläufen, ob linear und sequenziell, asynchron und parallel oder stochastischer Art. Eine Simulation der Planungs- und Bauprozesse ist im Hinblick auf die stärkere Integration dieser Phasen interessant. Thabet [Thabet, 2002. S. 3 f.] stellt ein Modell vor, wo ein sog. Project Product Module eine Vielzahl von 3D Modellen, Teilfertigungen und Komponenten enthält, der Simulator diese nach dem Baukastenprinzip auswählt und bei dem Einbau im Hintergrund die Terminierung aktiviert. In diesem Sinne wird der Terminplan mit den Interaktionen der Objekte generiert.

Dawood [Dawood et al., 2002. S.126] beschreibt ein mit größtenteils herkömmlichen Technologien gefertigtes Steuerinstrument zur Simulation eines Bauprozesses (School of Helath Construction Project). Im Gegensatz zu herkömmlichen 4D CAD Produkten, wurde AUTOCAD2000 als Visualisierungsinstrument eingesetzt, die Logik wurde in einer sog. VIRCON-kompatiblen Datenbank erfasst. Ein Simulator liest Informationen aus der VIRCON Datenbank und visualisiert diese Information im VIRCON-Kontext VIRCON in AUTOCAD. Es können auch weitere Informationen, wie z.B. Fertigstellungsgrad, gefertigter und genutzter Raum, dargestellt werden. Interessant bei dem VIRCON Projekt ist auch die Gliederung der Qualitätsfindung mittels eines „layer coding“ (Abb. 170).

A-	Element	Presenta-	Sector	Status	Scale	User defined
A	G252	M	01-	_	F	DEMOUNTABLE
Archi-	Internal Partition	Model Gra-	Level 1, all building	Not	1:100	Demountable Partition Walls

Abb. 170: Beispiel eines „Layer Coding“ mit erforderlichen und optionalen Informationsfeldern [Dawood et al., 2002. S. 128].

Um das aufwändige Erstellen von Simulationsvarianten zu systematisieren und vereinfachen, wird ein Vorgehen bei der Simulation mittels eines Objektmodells und dem Einsatz von allgemeinen Simulationssprachen wie MODSIM II und Tools wie GPSS/H von Manavazhi propagiert [Manavazhi, 2000. S. 547]. Die Innovation dieser Methode liegt nicht nur im Einsatz eines objektorientiert zentrischen Ansatzes, sondern auch in der Modellierung der für den Bau benötigten Ressourcen. Teilnehmer der Simulation werden als Klassen definiert (Abb. 171). Instanzen dieser Klassen (Objekte – Abb. 172) werden mit Methoden gesteuert zur Simulation eines Bauprozesses.

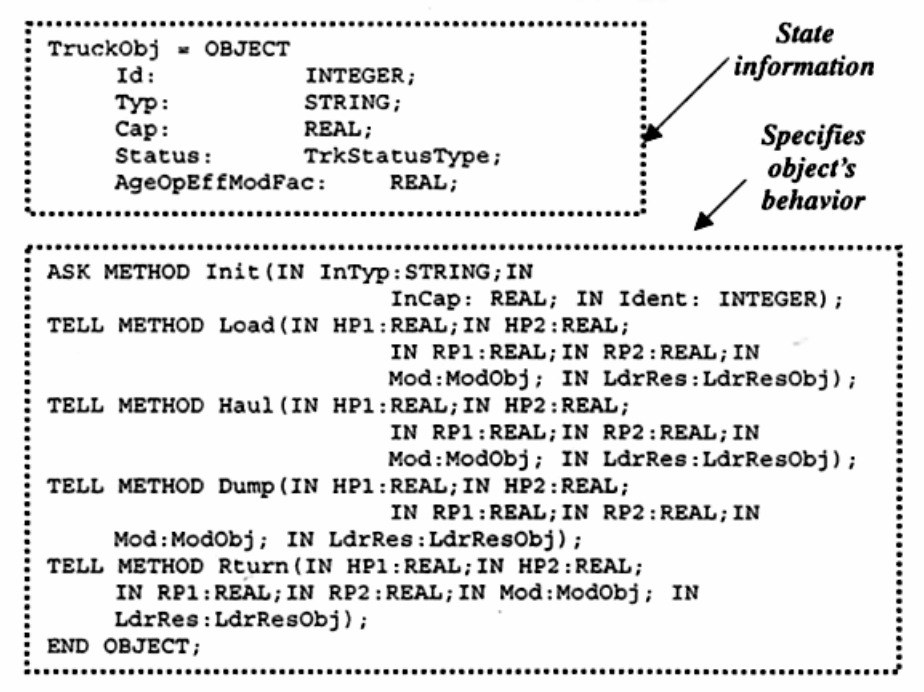


Abb. 171: Klassendefinition eines Lastwagens (Truck) [Manavazhi, M., 2000. S. 548].

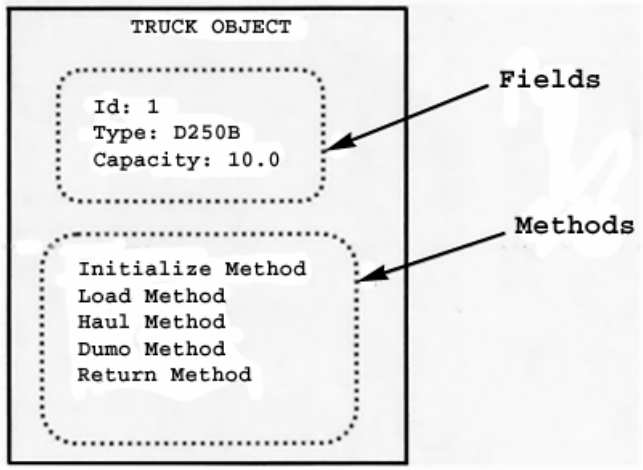


Abb. 172: Ein instanziertes Lastwagen (Truck) Objekt [Manavazhi, M., 2000 S. 548].

MODSIM II bietet über die Objektorientierte Definition von Baukörpern und Ressourcen auch die Möglichkeit, Methoden und Ereignisse zu definieren. ASK, TELL und WAITFOR Ereignisse werden eingesetzt zur Steuerung der Simulation. Die Simulation des Baus einer einfachen Dammschüttung zeigt auch die Problematik der Massenbilanz auf, wenn Laster bei der letzten Schüttung nur einen Teil ihrer Last verbrauchen (Abb. 173).

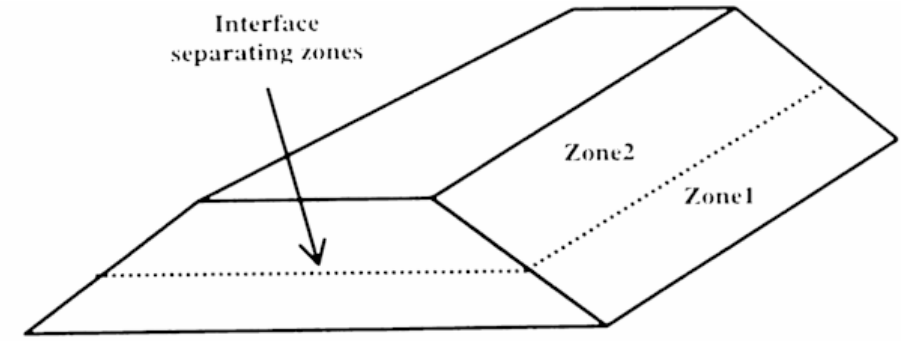


Abb. 173: Dammschüttung einer Simulation mit MODSIM II. Probleme mit der Massenbilanz bei der Abgrenzung der Materialien für Zone I und II, weil Lastwagen ihre Ladungen nicht teilweise entleeren können [Manavazhi, M., 2000. S.552].

Simulationen sind in den Planungsphasen wertvoll, um mögliche Lösungsvorschläge zu eruieren und zu testen. Sie können auch die Kopplung mit anderen Disziplinen wie die Statik vorsehen, um integrierte Betrachtungen zu ermöglichen. Auch die Erarbeitung von Varianten einer Kalkulation stellt Simulation dar. Es geht darum, die Auswirkung von Rahmenbedingungen und Größen, die Einfluss auf die Planung haben, zu eruieren. Die Kopplung mit der geometrischen Visualisierung soll die Wahr-

nehmung und somit die Verständigung stärken. Simulationen leiden jedoch daran, dass alle Modelle begrenzt sind und nur Teilmengen der Wirklichkeit darstellen können. Eine Gefahr der Simulation ist die Extrapolation von zukünftigen Werten aus einem begrenzten Modell heraus. Daher sollten Simulationen immer mit Vorsicht interpretiert werden.

A.III.1.3 Objekt-relationales Beispiel

```
using System.Data.ObjectSpaces;
using System.Xml.Query;

namespace cObj.estimating
{
/// <summary>
/// <c>provides the base element currency as builsing block for more
            complex estimating structures</c>
/// </summary>
      public class currency
      {
            public char[] RegionCode = new char[3];
            public char[] CurrencyCode = new char[3];
            public string Description;
            public char[] RefRegionCode = new char[3];
            public char[] RefCurrencyCode = new char[3];
            public string RefDescription;
            public DateTime Date = DateTime.Now;
            public double Val;
            public double RefVal;
            public Uri Source;

            static string path =
                  @"C:\projects\cObj\objectMaps\currency\currency.xml";
            static string connStr ="Data Source=localhost;Integrated
                  Security=SSPI; Database=dice";
            SqlConnection conn;
            ObjectSpace os;

      /// <remark>default currency constructor</remark>
            public currency() { }

      /// <remark>obtain currency value from service</remark>
            public currency getNCombo(Uri uri, char[] currencyCode)
            {
                  this.Date = DateTime.Now;
                  this.Source = uri;
                  XmlReader reader = new XmlTextReader(uri.AbsoluteUri);
                  this.Val =Convert.ToDecimal(reader.ReadElementString
                        ("currency"));
                  return this;
            }

            /// <remarks>provide ObjectSpaces object persistence</remarks>
            public string setObj()
            {}

            /// <remarks>select object from persisting store</remarks>
            public currency getObj()
            {}
      }
}
```

Die Methode zur Persistierung des Objektes, hier *setObj()* ist kompakt, liefert jedoch alle nötigen Funktionalitäten:

```
public string setObj()
{
      string str = "ok";
      SqlConnection conn = new SqlConnection(connStr);
      ObjectSpace os = new ObjectSpace(path, conn);

      try
      {
            os.StartTracking(this, InitialState.Inserted);
            ObjectContext ctx = ObjectContext.GetInternalContext(os);
            ObjectState objs = ctx.GetObjectState(this);
            if (objs != ObjectState.Inserted)
                  os.PersistChanges(this);
            else
                  os.Resync(this, Depth.ObjectGraph);
      }
      catch (PersistenceException e)
      {
            foreach (Object o in e.Errors)
                  str += "_" + o.ToString();
      }
      return str;
}
```

Gleichermaßen die Methode zur Deserialiserung des Objektes:

```
public currency getObj()
{
      currency lingo = null;

      conn = new SqlConnection(connStr);
      os = new ObjectSpace(path, conn);

      ObjectReader reader;

      try
      {
            reader = os.GetObjectReader(new ObjectQuery(typeof(currency),
                  "CurrencyCode='" + this.CurrencyCode
                  + "' AND RefCurrencyCode='"
                  + this.RefCurrencyCode + "'"));
            foreach (currency l in reader)
            {
                  lingo = l;
            }

            reader.Close();
      }
      catch (PersistenceException e)
      {
            throw e;
      }
      return lingo;
}
```

A.IV - Abstrakte Klasse *constructionobject* als XML Schema

Schema der Klasse *constructionobject*

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="conobj" targetNamespace="http://www.conobj.net/conObj.xsd"
elementFormDefault="qualified" xmlns="http://www.conobj.net/conObj.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="class">
            <xs:complexType>
                  <xs:sequence minOccurs="0" maxOccurs="unbounded">
                        <xs:element name="attributes">
                              <xs:complexType>

Siehe CD-ROM

                              </xs:complexType>
                        </xs:element>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:simpleType name="status">

Siehe CD-ROM

      </xs:simpleType>
      <xs:simpleType name="constraintType">

Siehe CD-ROM

      </xs:simpleType>
      <xs:simpleType name="levelingConstraintType">

Siehe CD-ROM

      </xs:simpleType>

Siehe CD-ROM


</xs:schema>
```

Instanzierte xml Form der Klasse conobj

```
<?xml version="1.0" encoding="utf-8" ?>
<objModel xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <guid>23257c67-4421-426a-b697-88fd180b394f</guid>
      <id>root.conObj.10</id>
      <internalId>0</internalId>
      <className>root.conObj</className>
      <creator />
      <dateCreated>2003-12-28T21:47:06.4097440+01:00</dateCreated>
      <dateChanged>2003-12-28T21:47:06.4097440+01:00</dateChanged>
      <node>
            <class>

Siehe CD-ROM

            </class>
      </node>
</objModel>
```

B.I.3.3 – Schema-basierte Erzeugung einer Klasse

```
[C#]
using System;
using System.Data;
using System.Data.OleDb;
using System.Xml;
using System.Xml.Schema;
using System.Xml.Serialization;
using System.Xml.XPath;
using System.Xml.Xsl;
using System.IO;
namespace inc.svr.factory
{
      /// <summary>
      /// schemaFactory generates signatures for conObj classes including
            inheritance
      /// functionality, and returns a XmlSchema object.
      /// This object can then be used to generate usable elements in the
            element
      /// db, from which objects are constructed to be used in the project
            model.
      /// </summary>
      public class schemaFactory : MarshalByRefObject
      {
            public System.Xml.XmlDocument xmlObj;
            public System.Xml.XmlNode xmlNode;
            public System.Xml.XmlTextReader txtReader;
            public System.Xml.XmlTextWriter txtWriter;
            public System.Xml.XmlWriter xmlWriter;
            public System.Xml.Schema.XmlSchema xmlSchema;

            public schemaFactory()
            {}

            public XmlSchema genSchema(string parentClass, string conObjC
                  lass, string currentNamespace)
                        {
                              XmlSchema schema;
                              XmlSchemaElement element;
                        }
            ...
            }
```

Diese Klasse kann wie folgt instanziert werden:

```
using System;
using  inc.svr.factory;

public schemaFactory  sf;
public class conObjImpl(string guid) : conObj
{
      sf = new schemaFactory ();
      sf.load(guid);
}
```

Implementierung in Java

```
import System;
import inc.svr.factory;

public class conObjImpl(string guid) extends conObj
{
      public schemaFactory sf = new schemaFactory ();
      sf.load(guid);
}
```

B.I.3.4 – Schema-basierte Erzeugung einer Klasse

statische Methode der Klasse *objmodel*

```
public static CompilerResults CompileCode(string filepath)
{
// Obtains an ICodeCompiler from a CodeDomProvider class.
      CSharpCodeProvider provider = new CSharpCodeProvider();
      ICodeCompiler compiler = provider.CreateCompiler();
// Configures a compiler parameters object which links System.dll and
// generates a file name based on the specified source file name.
      string exepath =  filepath.Substring
      (0,filepath.LastIndexOf(".")+1)+"dll";
      CompilerParameters cp = new CompilerParameters(new string[]
            {"System.dll","System.Xml.dll"}, exepath, false);
// Indicates that a.dll rather than an executable should be generated.
      cp.GenerateExecutable = false;
// Invokes compilation.
      CompilerResults cr = compiler.CompileAssemblyFromFile(cp, filepath);
// Returns the results of compilation.
      return cr;
}
```

B.I.3.4.2 – Systematik zur dynamischen Klassengenerierung und Instanzierung

```
/// <summary>
/// Converts the default public fields into properties backed by a private
field.
/// </summary>
public class FieldsToPropertiesExtension : ICodeExtension
{
      #region ICodeExtension Members

      public void Process( System.CodeDom.CodeNamespace code, Sys-
      tem.Xml.Schema.XmlSchema schema )
      {
            foreach ( CodeTypeDeclaration type in code.Types )
            {
                  if ( type.IsClass || type.IsStruct )
                  {
                  // Copy the colletion to an array for safety. We will be
                  // changing this collection.
                        CodeTypeMember[] members = new CodeTypeMem-
                        ber[type.Members.Count];
                        type.Members.CopyTo( members, 0 );

                        foreach ( CodeTypeMember member in members )
                        {
                        // Process fields only.
                              if ( member is CodeMemberField )
                              {
                                    CodeMemberProperty prop = new
                                          CodeMemberProperty();
                                    prop.Name = member.Name;

                                    prop.Attributes = member.Attributes;
                                    prop.Type = ( ( CodeMemberField )
                                          member ).Type;

                              // Copy attributes from field to the property.
                                    prop.CustomAttributes.AddRange
                                          ( member.CustomAttributes );
                                    member.CustomAttributes.Clear();

                              // Copy comments from field to the property.
                                    prop.Comments.AddRange( member.Comments
);
                                    member.Comments.Clear();

                              // Modify the field.
                                    member.Attributes =
                                          MemberAttributes.Private;
                                    Char[] letters =
                                          member.Name.ToCharArray();
                                    letters[0] = Char.ToLower(letters[0]);
                                    member.Name = String.Concat
                                          ( "_", new string(letters ) );

                                    prop.HasGet = true;
                                    prop.HasSet = true;
```

```
                        // Add get/set statements pointing to field.
                              Generates:
                        // return this._fieldname;
                              prop.GetStatements.Add(
                                    new CodeMethodReturnStatement(
                                    new CodeFieldReferenceExpression(
                                          new CodeThisReferenceExpres-
                                          sion(),
                                    member.Name ) ) );
                              // Generates:
                              // this._fieldname = value;
                              prop.SetStatements.Add(
                                    new CodeAssignStatement(
                                    new CodeFieldReferenceExpression(
                                          new CodeThisReferenceExpres-
                                          sion(),
                                    member.Name ),
                                    new CodeArgumentReference
                                          Expression( "value" ) ) );

                              // Finally add the property to the type
                              type.Members.Add( prop );
                        }
                  }
            }
      }
            #endregion Turn Public Fields into Properties
}
```

Der Konstruktor für den Processor, der die Steuerung der Generierung übernimmt, ist wie folgt aufgebaut:

```
static Processor()
{
      XPathNavigator nav = new XmlDocument().CreateNavigator();
      // Select all extension types.
      Extensions = nav.Compile(
            "/xs:schema/xs:annotation/xs:appinfo/ProtoClass:Code/
                  ProtoClass:Extension/@Type" );
      References = nav.Compile(
            "/xs:schema/xs:annotation/xs:appinfo/ProtoClass:Code/
                  ProtoClass:Reference/@Type" );

      // Create and set namespace resolution context.
      XmlNamespaceManager nsmgr = new XmlNamespaceManager( nav.NameTable );
      nsmgr.AddNamespace( "xs", XmlSchema.Namespace );
      nsmgr.AddNamespace( "ProtoClass", ExtensionNamespace );
      Extensions.SetContext( nsmgr );
      References.SetContext( nsmgr );
}
```

Die Einbindung der Namespaces erfolgt über eine Iteration durch den ComplexType Tag, der für References vorgesehen ist:

```
Type baseType = null;
// Iterate schema top-level elements and export code for each.
foreach ( XmlSchemaObject element in xsd.Items ){
      if(element is System.Xml.Schema.XmlSchemaComplexType) {
            System.Xml.Schema.XmlSchemaComplexType obj = element
                  as System.Xml.Schema.XmlSchemaComplexType;

            if(obj.QualifiedName.Name == "References") {
                  System.Xml.Schema.XmlSchemaSequence seq = obj.Particle
                        as System.Xml.Schema.XmlSchemaSequence;
                  foreach(System.Xml.Schema.XmlSchemaObject obj_ in
                        seq.Items) {
                        System.Xml.Schema.XmlSchemaElement el  = obj_
                              as System.Xml.Schema.XmlSchemaElement;
                        ns.Imports.Add( new CodeNamespaceIm-
                        port(el.QualifiedName.Name) );
                  }
            }
            else if(obj.QualifiedName.Name == "Derivation") {
                  System.Xml.Schema.XmlSchemaSequence seq = obj.Particle
                        as System.Xml.Schema.XmlSchemaSequence;
                  System.Xml.Schema.XmlSchemaObject obj_ = seq.Items[0];
                  System.Xml.Schema.XmlSchemaElement el  = obj_
                        as System.Xml.Schema.XmlSchemaElement;
                  try {
                        Object o = null;
                        string path =
                        @"E:\conObj2\ProtoClasses\bin\Debug\ProtoClasses.dl
                        l";
                        Assembly a = Assembly.LoadFrom(path);
                        Type[] mytypes = a.GetTypes();
                        BindingFlags flags = (BindingFlags.NonPublic | Bin-
                        dingFlags.Public |
                              BindingFlags.Static | BindingFlags.Instance |
                              BindingFlags.DeclaredOnly);
                        foreach(Type t in mytypes) {
                              if(t.FullName == el.QualifiedName.Name) {
                                    baseType = t;
                                    MethodInfo[] mi = t.GetMethods(flags);
                                    o = Activator.CreateInstance(t);
                              }
                        }
                  }
                  catch(Exception exc) {
                        throw exc;
                  }
            }
      }
}
```

Der Konstruktor wird über eine kurze Anweisung generiert:

```
#region Constructor
// Iterate schema top-level elements and export code for each.
foreach ( XmlSchemaElement element in xsd.Elements.Values )
{
// Import the mapping first.
      XmlTypeMapping mapping = importer.ImportDerivedTypeMapping(
            element.QualifiedName, baseType, true );

// Export the code finally.
      exporter.ExportTypeMapping( mapping );
}
#endregion
```

Eigenschaften können mit der `FieldsToPropertiesExtension` Implementierung erweitert werden:

```
#region Execute Extensions
using ( FileStream fs = new FileStream( xsdFile, FileMode.Open ) )
{
      nav = new XPathDocument( fs ).CreateNavigator();
}

it = nav.Select( Extensions );
while ( it.MoveNext() )
{
      Type t = Type.GetType( it.Current.Value, true );
// Is the type an ICodeExtension?
      Type iface = t.GetInterface( typeof( ICodeExtension ).Name );
      if (iface == null)
            throw new ArgumentException( "Invalid extension type '" +
            it.Current.Value + "'." );

      ICodeExtension ext = ( ICodeExtension ) Activator.CreateInstance( t );

// Run it!
      ext.Process( ns, xsd );
}
#endregion
```

Mit diesen Methoden kann aus einer XmlSchema Datei eine abgeleitete Klasse dynamisch generiert werden. Entsprechend der früher beschriebenen Vorgehensweise mit Xml und XmlSchema, kann dieses Verfahren auch während der Laufzeit angewandt werden, obgleich der praktische Nutzen im Baubereich begrenzt ist. Die Konvertierung der Datei ProtoClass.xsd

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:NameSpace1="http://weblogs.asp.net/cazzu">
      <xs:annotation>
            <xs:appinfo>
                  <Code xmlns="http://weblogs.asp.net/cazzu">
                        <Reference Type="XsdGenerator.
                              References.InheritanceExtension,
                              XsdGenerator.Library" />
                        <Extension Type="XsdGenerator.
                              Extensions.FieldsToPropertiesExtension,
                              XsdGenerator.Library" />
                  </Code>
            </xs:appinfo>
      </xs:annotation>
      <!-- References -->
      <xs:complexType name="References">


Siehe CD-ROM

            </xs:complexType>
      </xs:element>
</xs:schema>
```

Führt zur Generierung der Klasse ProtoClass.derviedconstructionobject, die die zusätzliche Eigenschaft Mängel aufweist:

```
namespace ProtoClass {
    using System;
    using System.Drawing;
    using System.CodeDom;
    using System.Collections;
    using System.Threading;
    using System.Text.RegularExpressions;
    using System.ComponentModel;
    using System.Timers;
    using System.Xml;
    using System.Xml.Schema;
    using System.Xml.XPath;
    using System.Xml.Xsl;
    using System.IO;
    using System.Xml.Serialization;
    using System.Runtime.Serialization;
    using System.Text;
    using System.Reflection;
    using System.Security.Principal;
    using System.Data;
    using System.Data.OleDb;
    using System.Data.SqlClient;
    using ILOG.Views.Gantt.Data;
    using structs;
    using objLib.chrono;
    using objLib.db;
    using objLib.configuration;
    using objLib.enumerations;
    using objLib.items;
    using objLib.collection;
    using objLib.combo;
    using objLib.dStack;
    using objLib.costs;
    using objLib.resources;
    using RemoteLogger;
    /// <remarks/>
    [System.Xml.Serialization.XmlRootAttribute(Namespace="",
            IsNullable=false)]
    public class derivedconstructionobject : ProtoClass.constructionobject
{

        private string _maengel;

        /// <remarks/>
        public string Maengel {
            get {
                return this._maengel;
            }
            set {
                this._maengel = value;
            }
        }
    }
}
```

B.I.3.5 – Implementierung der Musterklasse *constructionobject*

Eigenschaften:

```
#region properties
[XmlIgnore]
private Guid _id;
public Guid Id
{
      get{if(this._id == Guid.Empty)
                  this._id = Guid.NewGuid();
            return this._id;}
      set{this._id = value;}
}
[XmlIgnore]
public string IdString
{
      get{return this._id.ToString();}
}
[XmlIgnore]
public string IdXmlString
{
      get{return this._id.ToString() + ".xml";}
}
[XmlIgnore]
private string _description;
public string Description
{
      get{return this._description;}
      set{this._description = value;}
}
[XmlIgnore]
private int _delay;
[XmlIgnore]
public int Delay
{
      get{//min 1 minute delay
            return (int) Math.Max(1000.0M*6.0M, (decimal) this._delay);}
      set{this._delay = value;}
}
[XmlIgnore]
private int _interval;
[XmlIgnore]
public int Interval
{
      get{//min 1 minute interval
return (int) Math.Max(1000.0M*6.0M, (decimal) this._interval);}
      set
      {
            this._interval = value;
      }
}
```

```
[XmlIgnore]
private Interface[] _Interfaces;
[XmlIgnore]
public Interface[] Interfaces
{
      get{return this._Interfaces;}
      set{this._Interfaces = value;}
}

//properties pertaining to obj geometry
[XmlIgnore]
private XmlNode _geometry;
public string Geometry
{
      get{
            XmlNode node = this._geometry;
            return (node!=null?node.InnerXml:string.Empty);}
      set{  //here we call the delegate if we detect a change in geometry
            //using string comparison
            XmlDocument doc = new XmlDocument();
            XmlNode cdata = doc.CreateElement("CDATA");
            cdata.InnerXml = value;
            this._geometry = cdata;
      }
}
[XmlIgnore]
private double _volume;
public double Volume
{
      get{return this._volume;}
      set{  //here we call the delegate if we detect a change in volume
            if(this._volume != value)
            {
                  double diff = this._volume - value;

                  // Create an instance of the class that will
                  // be firing an event.
                  constructionobject co = new constructionobject();

                  //use our class to raise a few events and
                  //watch them get handled
                  co.ActivateObjectAlarm(this.Id, "Volume");
                  updateRefControl("textBox_Volume", diff.ToString());
            }
            else{
                  this._volume = value;
            }
      }
}
```

```
[XmlIgnore]
private object _sync = null;
[XmlIgnore]
public object SynchronizingObject
{
      get { return _sync; }
      set { _sync = value; }
}
[XmlIgnore]
private long tickCounter;
[XmlIgnore]
public string Xml
{
      get{  byte[] b = this.serialize();

            //writer as file to check data
            string validate = @"c:\temp\" + this.IdXmlString;
            FileStream fs = new FileStream(validate, FileMode.OpenOrCreate);
            fs.Write(b, 0, b.Length);
            fs.Close();

            //Create the XmlNamespaceManager.
            NameTable nt = new NameTable();
            XmlNamespaceManager nsmgr = new XmlNamespaceManager(nt);

            //Create the XmlParserContext.
            XmlParserContext context = new XmlParserContext(null, nsmgr,
                        null, XmlSpace.Preserve);

            XmlDocument doc = new XmlDocument();
            doc.Load(validate);
            string ret = doc.DocumentElement.OuterXml;

            return ret; }
}
[XmlIgnore]
FileInfo[] GeometryStack
{
      get{DirectoryInfo di = new Directory-
Info(@"E:\conObj2\objInterface\wrl");
            return di.GetFiles("*.wrl");}
}
[XmlIgnore]
FileInfo[] _geometryStack;
#endregion
```

B.II.2.1.2 – Beispiel einer mit Koordinaten und Linien beschriebenen XML Datei

```
<?xml version="1.0" encoding="utf-8" ?>
<geom>
	<nodes>
		<node ref="0" x="0" y="0" z="0"></node>
		<node ref="1" x="-1" y="0" z="0"></node>
		<node ref="2" x="-1" y="0" z="-1,5"></node>
		<node ref="3" x="-2,5" y="0" z="-1,5"></node>
		<node ref="4" x="-2,5" y="0" z="1,5"></node>
		<node ref="5" x="1,5" y="0" z="1,5"></node>
		<node ref="6" x="1,5" y="0" z="-1,5"></node>
		<node ref="7" x="0" y="0" z="-1,5"></node>
		<node ref="8" x="-1" y="1" z="-1,5"></node>
		<node ref="9" x="0" y="1" z="0"></node>
		<node ref="10" x="-1" y="1" z="0"></node>
		<node ref="11" x="-1" y="1" z="-1,5"></node>
		<node ref="12" x="-2,5" y="1" z="-1,5"></node>
		<node ref="13" x="-2,5" y="1" z="1,5"></node>
		<node ref="14" x="1,5" y="1" z="1,5"></node>
		<node ref="15" x="1,5" y="1" z="-1,5"></node>
	</nodes>
	<arcs>
		<arc ref="0" lineType="0" p1="0" p2="1"></arc>
		<arc ref="1" lineType="0" p1="0" p2="7"></arc>
		<arc ref="2" lineType="0" p1="7" p2="6"></arc>
		<arc ref="3" lineType="0" p1="6" p2="5"></arc>
		<arc ref="4" lineType="0" p1="5" p2="4"></arc>
		<arc ref="5" lineType="0" p1="4" p2="3"></arc>
		<arc ref="6" lineType="0" p1="3" p2="2"></arc>
		<arc ref="7" lineType="0" p1="2" p2="1"></arc>
		<arc ref="8" lineType="0" p1="2" p2="7"></arc>
		<arc ref="9" lineType="0" p1="1" p2="10"></arc>
		<arc ref="10" lineType="0" p1="10" p2="9"></arc>
		<arc ref="11" lineType="0" p1="0" p2="9"></arc>
		<arc ref="12" lineType="0" p1="9" p2="8"></arc>
		<arc ref="13" lineType="0" p1="8" p2="11"></arc>
		<arc ref="14" lineType="0" p1="10" p2="11"></arc>
		<arc ref="15" lineType="0" p1="7" p2="8"></arc>
		<arc ref="16" lineType="0" p1="2" p2="11"></arc>
		<arc ref="17" lineType="0" p1="8" p2="15"></arc>
		<arc ref="18" lineType="0" p1="6" p2="15"></arc>
		<arc ref="19" lineType="0" p1="15" p2="14"></arc>
		<arc ref="20" lineType="0" p1="5" p2="14"></arc>
		<arc ref="21" lineType="0" p1="12" p2="14"></arc>
		<arc ref="22" lineType="0" p1="13" p2="4"></arc>
		<arc ref="23" lineType="0" p1="13" p2="12"></arc>
		<arc ref="24" lineType="0" p1="3" p2="12"></arc>
		<arc ref="25" lineType="0" p1="12" p2="11"></arc>
	</arcs>
</geom>
```

B.II.2.2.1 – Zugriff auf native CAD Objekte mittels Allplan NOI

```
CNOI_DBObjectPtr pObj = vec[i];
if (! pObj) continue;

// execute lazy-evaluation
pDB->read(pObj);
CNOI_ColorARGB argb = pObj->GetCommonStyle()->Color();

if (pObj->IsKindOf (NOI_RUNTIME_CLASS (CNOI_ArchWall)))
      GeoForWallLayers (VRMLwriter, (CNOI_ArchWallPtr &) pObj, dim);
else if (pObj->IsKindOf (NOI_RUNTIME_CLASS (CNOI_ArchUpstand)))
      GeoForUpstandLayers (VRMLwriter, (CNOI_ArchUpstandPtr &) pObj, dim);
else if (pObj->IsKindOf (NOI_RUNTIME_CLASS (CNOI_BasisMacroPlacement)))
      GeoForBasisMacroPlacement (VRMLwriter, (CNOI_BasisMacroPlacementPtr
      &) pObj, dim);
else if (pObj->IsKindOf (NOI_RUNTIME_CLASS (CNOI_BIE_MacroPlacement)))
      GeoForBIEMacroPlacement (VRMLwriter, (CNOI_BIE_MacroPlacementPtr &)
      pObj, dim);
else
{
      CNOI_GeoMatrix_4x4Ptr pMatrix;
      if (dim == e2D)
            VRMLwriter->writeGeo2D (pObj, pMatrix);
      else
            VRMLwriter->writeGeo3D (pObj, pMatrix);
}
```

B.II.2.2.2 – Remote Utility Call Stack

```
// Initialise the data structure for remote call
RemoteAllplanProcedureCallCopyDataStruct_t pRemoteInfo;
// Event ID for the Astron remote DLL
pRemoteInfo.Event = RemoteDLLEventID;
// DLL name, which will called by Allplan
_tcsncpy (pRemoteInfo.RemoteDLLName, RemoteDLLName,
      sizeof (pRemoteInfo.RemoteDLLName));
// DLL name, which will called by Allplan
_tcsncpy (pRemoteInfo.RemoteDLLArgument, RemoteDLLArgumentString,
      sizeof (pRemoteInfo.RemoteDLLArgument));
COPYDATASTRUCT cds =
{ 0, sizeof (RemoteAllplanProcedureCallCopyDataStruct_t),
      static_cast<void*> (&pRemoteInfo)
};
MSG msg;
msg.hwnd    = hWnd;
// HWND of allplan window
msg.message = WM_COPYDATA;
// message typ
msg.wParam  = UIATOOL_REMOTE_ALLPLAN_SERVICE_CALL;
// ID for internal EVENT
msg.lParam  = (LPARAM) (PCOPYDATASTRUCT)&cds;
// Remote call data
// SendMessage must be used for WM_COPYDATA
// PostMessage not possible (see Microsoft documentation)
retVal = ::SendMessage(msg.hwnd, msg.message, msg.wParam, msg.lParam);
```

B.II.2.1.2 Klassen zur Geometrie

4.1 Definition der Punkte

Die Punkte werden als kartesische Raumkoordinaten definiert, diese bestehen aus x, y, und z Werten. Diese können relativ zum projektspezifischen rechtwinkeligen Achsensystem angegeben werden, oder auch als globale Koordinaten gelten:

4.2 Definition der Linien

4.2.1 Gerade Linien

Linien werden als geradlinige Verbindungen zwischen kartesischen Punkten definiert. Sie stellen Vektoren dar, die wiederum Segmente zwischen zwei kartesischen Punkten sind,

$P1(x1, y1, z1)$ und $P2(x2, y2, z2)$.

Der entsprechende Vektor

$A = \langle a1, a2, a3 \rangle$

hat die Größe

$$|A| = \sqrt{[a1^2 + a2^2 + a3^2]} = \sqrt{[(x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2]}$$

Die Richtungskosinusse werden wie folgt ermittelt:

$$\cos \alpha = a1/|A|$$
$$\cos \beta = a2/|A|$$
$$\cos \chi = a3/|A|$$

Nach der Definition der Eckpunkte eines geometrischen Gebildes, reicht die Angabe von zwei Punkten zur Erstellung einer Linie

$P1(x1, y1, z1)$ und $P2(x2, y2, z2)$.

4.2.2 Kurvensektoren

Drei Knoten beschreiben z.B. einen Kreis im Raum. Der Kreis liegt in einer Fläche, die gegeben wird durch die Gleichung

$$\alpha x + \beta y + \chi z = \delta$$

oder spezifisch

$$\begin{vmatrix} x & y & z & 1 \\ x1 & y1 & z1 & 1 \\ x2 & y2 & z2 & 1 \\ x3 & y3 & z3 & 1 \end{vmatrix} = 0$$

wobei (x1, y1, z1), (x2, y2, z2), (x3, y3, z3) die 3 Knoten in einem basischen Achsensystem bezeichnen.

Wählt man ein Achsensystem x‘, y‘, z‘ mit z-Achse entlang der Gerade, die vom Ursprung des Basissystems senkrecht auf die Fläche trifft, so wird der Kreis von der Gleichung

$$Z' = 0,\ x'^2 + y'^2 + ax' + by' = c'$$

gegeben.
Richtungskosinusse von der z‘ – Achse

$$\cos \theta xz' = \alpha/\kappa$$
$$\cos \theta yz' = \beta/\kappa$$
$$\cos \theta zz' = \gamma/\kappa$$

mit $\kappa = \sqrt{\alpha^2 + \beta^2 + \gamma^2}$
Voraussetzung für einen Kreis

$$\begin{vmatrix} 1 & 1 & 1 \\ x1 & x2 & x3 \\ y1 & y2 & y3 \end{vmatrix} \neq 0$$

Distanz zwischen Ursprüngen der beiden Systeme ist

$$d = \delta/\kappa$$

4.2.3 Transformation eines Achsensystems

Zur Veranschaulichung eines Baukörpers soll ein Modell dargestellt werden, welches aus verschiedenen Blickpunkten betrachtet werden kann. Hierzu sind zwei Koordinatensysteme notwendig: Ein Baustellensystem, Ξ, und ein System, Ξ', welches Drehen und Verschiebung des Gebildes ermöglicht. Die Transformation von einem zum anderen muss dann angegeben werden. Die Baukörper werden zusammengesetzt aus Knoten, die durch Kurven, geradlinig oder gebogen, verbunden werden.

Gefragt ist also eine Transformation von dem kartesischen Achsensystem

Ξ: xyz

mit Ursprung O zu dem kartesischen System

Ξ′: x'y'z'

mit Ursprung O' wobei Ξ' aus Ξ entsteht durch Verschiebung und Drehung (Abb. 174).

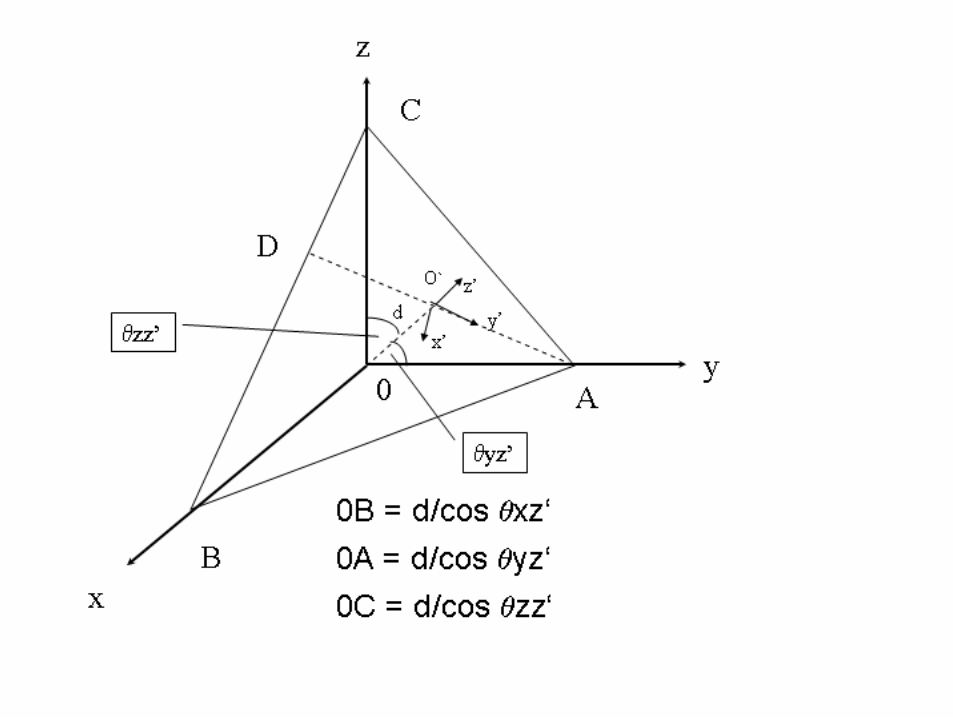


Abb. 174: Achsensysteme in xyz und x'y'z'

Die Distanz OO' sei mit d bezeichnet und z' liege in der Richtung von OO'. Die Drehwinkel seien $\theta_{xz'}, \theta_{yz'}, \theta_{zz'}$ wovon natürlich nur 2 frei gewählt werden können, da

(1) $$\cos^2 \theta_{xz'} + \cos^2 \theta_{yz'} + \cos^2 \theta_{zz'} = 1$$

gelten muss. Die x'- und y'-Achsen liegen offenbar auf der Fläche

(2) $$x \cos \theta_{xz'} + y \cos \theta_{yz'} + z \cos \theta_{zz'} - d = 0.$$

B bzw. C sei der Schnittpunkt der Fläche (1) mit der x- bzw. z-Achse. Nimmt man nun y' längs der Verbindungslinie von O' und dem Schnittpunkt A dieser Linie mit der y-Achse, so liegt die x'-Achse parallel zu CB.

Um das zu bestätigen, bestimmt man die Richtungskosinusse der Linien AD (siehe Abb. 174) und BC.

Unter Berücksichtigung von (1) errechnet man

$$\cos \alpha_{AD} = \cos^{-2} \theta_{yz'} - 1, \cos\beta_{AD} = -\sin \theta_{yz'}, \cos \gamma_{AD} = \cos\theta_{zz'}$$
$$\cos \alpha_{BC} = \cos \theta_{zz'} \sin^{-1} \theta_{yz'}, \cos\beta_{BC} = 0, \cos \gamma_{BC} = -\cos\theta_{xz'}$$
$$\sin^{-1} \theta_{yz'}.$$

Aus $\cos \alpha_{AD} \cos \alpha_{BC} + \cos\beta_{AD} \cos\beta_{BC} + \cos \gamma_{AD} \cos \gamma_{BC} = 0$ zeigt sich dann, dass $\angle$ADC ein rechter Winkel ist. Mit den Zwischenergebnissen:

$$\operatorname{ctg} \angle BCA = \cos \theta_{xz'} \cos \theta_{yz'} \cos^{-1} \theta_{zz'}$$

$$\cos (\angle CBO) = \cos \theta_{zz'} \sin^{-1} \theta\theta_{yz'} \qquad \sin (\angle CBO) = \cos \theta_{xz'} \sin^{-1} \theta_{yz'}$$

findet man sodann

(3) $$x = (d + z') \cos \theta_{xz'} + x' \cos \theta_{zz'} \sin^{-1} \theta_{yz'} - y' \cos \theta_{xz'} \operatorname{ctg} \theta_{yz'}$$

(4) $$y = (d + z') \cos \theta_{yz'} + y' \sin \theta_{xz'},$$

(5) $$z = (d + z') \cos \theta_{zz'} - x' \cos \theta_{xz'} \sin^{-1} \theta_{yz'} - y' \operatorname{ctg} \theta_{yz'} \cos \theta_{zz'}.$$

Die umgekehrte Transformation ist meist nicht so aktuell, kann aber dadurch erreicht werden, daß das Gleichungssystem (3)–(5) für x', y', z' aufgelöst wird.

Will man nun z.B. einen Kreis in Ξ' haben, so wähle man drei Knoten x'_l, y'_l ($l = 1,2,3$) mit der Bedingung

$$\begin{vmatrix} 1 & 1 & 1 \\ x'_1 & x'_2 & x'_3 \\ y'_1 & y'_2 & y'_3 \end{vmatrix} \neq 0.$$

4.3 Umsetzung

Die Umsetzung gelingt mit Hilfe folgender .NET Klassen die im Detail weiter unten aufgeführt werden:

- `pnt` – beschreibt einen Endpunkt einer Kurve
- `straightLine` – beschreibt eine Gerade

die beide in *arcs.cs* wohnen. Hinzu kommen die Klassen

- `geom` – führt Punkte, Sektoren oder Linien und deren Relationen in Matrizen zusammen
- `node` - beschreibt eine dreidimensionale Koordinate
- `transform` – beschreibt die Transformation eines Achsensystems in einem anderen Achsensystem und somit die nötige Arithmetik für die Transformation eines Körpers
- `arc` – beschreibt eine Kurve

die wiederum in *geom.cs* liegen.

```
namespace objLib.math
{
/// <summary>
/// Zusammenfassende Beschreibung für math.
/// </summary>
      [Serializable]
      public class Pnt
      {
            public double x;
            public double y;
            public double z;
            public Pnt(double x1, double y1, double z1)
            {
                  x = x1;
                  y = y1;
                  z = z1;
            }
      }

      [Serializable]
      public class straightLine
      {
            public Pnt p1;
            public Pnt p2;
            public double x1;
            public double y1;
            public double z1;
            public double x2;
            public double y2;
            public double z2;
            public double a;
            public double b;
            public double c;
            public double t=1;
            public straightLine()
            {}

            public straightLine(double x_1, double y_1, double z_1, double
                  x_2, double y_2, double z_2)
            {
                  p1 = new Pnt(x_1, y_1, z_1);
                  p2 = new Pnt(x_2, y_2, z_2);
                  a = x_2 - x_1;
                  b = y_2 - y_1;
                  c = z_2 - z_1;
            }

            public Pnt getX3(double l)
            {
                  t = l/Math.Sqrt(Math.Pow(a,2) + Math.Pow(b,2) +
                        Math.Pow(c,2));
                  return (new Pnt(x1+a*t, y1+a*t, z1+a*t));
            }
      }
}
```

```
namespace objLib.geometry:
[Serializable]
public class geom
{
      public node n;
      public transform t;
      public arc a;
      public straightLine l;
      public ArrayList relations;
      public Array nodes;
      public Array arcs;
      public Array nodeAssociations;
      string persistFileName;

      //class instantiation
      public geom()
      {}
      public geom(int nosNodes, int arcNos){
            relations = new ArrayList();

            nodes = Array.CreateInstance(typeof(object),nosNodes, 2);
            arcs = Array.CreateInstance(typeof(object), arcNos, 2);
            nodeAssociations = Array.CreateInstance(typeof(object), nosNo-
                              des, 2);
      }
      //node
      public node setNode(int nos, double x, double y, double z){
            n = new node(nos, x, y, z);
            return n;
      }

      //transform
      public transform setTransform(double x, double y, double z, double
                  rx, double ry, double rz, double rot){
            t = new transform(x, y, z, rx, ry, rz, rot);
            return t;
      }

      //relation
      public ArrayList setRelation(double rel){
            relations.Add(rel);
            return relations;
      }

      public int getRelation(double rel){
            return (int) relations[relations.IndexOf(rel)];
      }

      public void delRelation(double rel){
            relations.RemoveAt(relations.IndexOf(rel));
      }

      //arc
      public arc setArc(arc.arcType type, straightLine ln, int stop){
            a = new arc(type, ln, stop);
            return a;
      }
```

```
public void setArcs(int index, objLib.geometry.arc.arcType aType,
            straightLine ln, ArrayList indexedSet){
      Array sArc = Array.CreateInstance(typeof(object),3);
      sArc.SetValue(aType, 0);
      sArc.SetValue(ln, 1);
      sArc.SetValue(indexedSet, 2);
      arcs.SetValue(index, index, 0);
      arcs.SetValue(sArc, index, 1);}

//setArcs overloaded
public void setArcs(int index, objLib.geometry.arc.arcType aType,
            straightLine ln, ArrayList indexedSet, int stop){
      Array sArc = Array.CreateInstance(typeof(object),4);
      sArc.SetValue(aType, 0);
      sArc.SetValue(ln, 1);
      sArc.SetValue(indexedSet, 2);
      sArc.SetValue(stop, 3);
      arcs.SetValue(index, index, 0);
      arcs.SetValue(sArc, index, 1);}

public Array getArcs(int index){
      Array sArc = Array.CreateInstance(typeof(object),4);
      sArc.SetValue(arcs.GetValue(index,0),0);
      sArc.SetValue(arcs.GetValue(index,1),1);
      sArc.SetValue(arcs.GetValue(index,2),2);
      sArc.SetValue(arcs.GetValue(index,3),3);
      return sArc;}

      //nodes
public void setNodes(int index, int nos, double x, double y, double
            z){
      nodes.SetValue(nos, index, 0);
      nodes.SetValue(setNode(nos,x,y,z), index, 1);}

public Array setNodes(int index, node N, ArrayList rel){
      nodes.SetValue(N, index, 0);
      nodes.SetValue(rel, index, 1);
      return nodes;}

public Array getNodes(int index){
      Array sNode = Array.CreateInstance(typeof(object),2);
      sNode.SetValue(nodes.GetValue(index,0),0);
      sNode.SetValue(nodes.GetValue(index,1),1);
      return sNode;}

public void serialize(string path){
      //write SOAP (XML)
      persistFileName = path + ".xml";
      Stream streamWrite = File.Create(persistFileName.ToString());
      SoapFormatter soapWrite = new SoapFormatter();
      soapWrite.Serialize(streamWrite, this);
      streamWrite.Close();}
```

```
	public geom deSerialize(string path){
		//read SOAP (XML)
		geom tmpGeom = new geom();
		persistFileName = path; // + ".xml";
		if(File.Exists(persistFileName.ToString()))
		{
			Stream streamRead = Fi-
			le.OpenRead(persistFileName.ToString());
			SoapFormatter soapRead = new SoapFormatter();
			tmpGeom = (geom) soapRead.Deserialize(streamRead);
			streamRead.Close();
			return tmpGeom;
		}
		else
			return null;
	}

	public Array retAssnodes(){
		for(int i = 0;i<=nodes.GetUpperBound(0); i++)
		{
			ArrayList assAr = new ArrayList();
			string nodeIndex = nodes.GetValue(i,1).ToString();
			for(int j = 0;j<=arcs.GetUpperBound(0); j++)
			{
				Array nodeCache = Array.CreateInstance
					(typeof(object), 3);
				nodeCache = (Array) arcs.GetValue(j,1);
				ArrayList indexedSet = (ArrayList)
					nodeCache.GetValue(2);
				foreach(Object o in indexedSet)
				{
					if(o.ToString()==nodeIndex)
						assAr.Add(o.ToString());
				}
			}
			nodeAssociations.SetValue(nodeIndex,0);
			nodeAssociations.SetValue(assAr,i,1);
		}
		return nodeAssociations;
	}
}
```

```
//class node
[Serializable]
public class node
{
      public int pos;
      public double X;
      public double Y;
      public double Z;

      public node(int position, double x, double y, double z){
            Pos = position;
            X = x;
            Y = y;
            Z = z;
      }

      public int Pos{
            get{
                  return pos;
            }
            set{
                  pos = Math.Abs(value);
            }
      }
}
```

```
//class transform
[Serializable]
public class transform
{
      public double tranX;
      public double tranY;
      public double tranZ;
      public double rotX; //unit vector X
      public double rotY; //unit vector Y
      public double rotZ; //unit vector Z
      public double rot; //degrees

      public transform(double x, double y, double z, double rX, double rY,
            double rZ, double rot){
            this.tranX = x;
            this.tranY = y;
            this.tranZ = z;
            this.RotX = rX;
            this.RotY = rY;
            this.RotZ = rZ;
            this.rot = rot;
      }

      public double RotX{
            get {return rotX;}
            set {if(rotX/360 > 1)
                        rotX = Math.Abs(value -
                              Math.Round(360*(value/360),0));
                  else rotX = value;
            }
      }

      public double RotY {
            get {return rotY;}
            set {if(rotY/360 > 1)
                        rotY = Math.Abs(value -
                              Math.Round(360*(value/360),0));
                  else rotY = value;
            }
      }

      public double RotZ {
            get{return rotZ;}
            set {if(rotZ/360 > 1)
                        rotZ = Math.Abs(value -
                              Math.Round(360*(value/360),0));
                  else rotZ = value;
            }
      }
}
```

```
[Serializable]
public class arc
{
	public enum arcType
	{
		straightLine=1,
		sector=2,
		exponential=3,
		ln=4,
		inverseLn=5,
		polynomial=6
	}

	public arcType aType;
	public straightLine l;
	public int stop;

	public arc(arcType typ, straightLine ln, int s)
	{
		AType = typ;
		l = ln;
		stop = s;
	}

	public arcType AType
	{
		get{
			return aType;
		}
		set
		{
			aType = value;
		}
	}
}
```

Anhang Teil C - Listing des Objektmodells

Auf Anfrage können Source Codes und Applikationen für das Objektmodell geliefert werden, soweit sie nicht Urheberrechtsbeschränkungen unterliegen. Folgendes sollte beachtet werden:

- Cortona und die Cortona SDK sind proprietäre Programme der Firma Parallel-graphics
- VARCHART.NET ist ein proprietäres Programm der Firma NETRONIC
- P3e und die Primavera SDK sind proprietäre Programme der Firma Primavera.
- Kubus ist ein proprietäres Programm der Firma Cap Gemini
- Allplan ist ein proprietäres Programm der Firma Nemetschek
- JNBridge ist ein proprietäres Programm der Firma Jnbridge
- Die überwiegenden Anzahl Applikationen wurden mit Visual Studio 2003 und .NET 1.1 entwickelt.

Lebenslauf

Name	Dirk Neethling
E-mail	dirk.neethling@gmx.net
Anschrift	Barkhorstrücken 1, 45239 Essen, Germany
Geburtsdatum	1 Dezember 1965
Geburtsort	Pinelands, South Africa
Staatsangehörigkeit	Deutsch
Konfession	Evangelisch

Bildung

1984 – 1986	BSc, Civil Engineering, University of Pretoria, South Africa
02/1987 – 12/1987	BSc, Civil Engineering, University of Cape Town, South Africa
01/1988 – 06/1988	BSc, Civil Engineering, University of Pretoria, South Africa Post-graduate studies in Numerical Methods
07/1988 – 06/1989	Geotechnical Engineering, University of Witwatersrand, Johannesburg, South Africa
1989	Graduate Diploma in Engineering
07/1989 – 01/1991	Master of Science in Engineering (MSc. Eng), University of Witwatersrand, Johannesburg, South Africa
2004-2006	Master of Computer Information Systems, University of Phoenix, USA
2007+	Fernstudium Maschinenbau an der TU-Dresden, Deutschland

Wehrdienst

07/1993 – 07/1994	National Service in der SADF. Leutnant im Corps of Engineers, Kroonstad, South Africa.

Beruf

01/1989 – 04/1991	Engineering Scientist in Division for Building Technology at the Council for Industrial and Scientific Research (CSIR), Pretoria, South Africa
05/1991 – 06/1993	Bauingenieur in der HNA Ausland der Firma Hochtief AG, Essen.
08/1994 – 10/1994	Trans-Afrika Motoradreise von Südafrika nach Deutschland
11/1994 – 03/1995	Tätigkeit im Ingenieurbüro Maidl & Maidl, RUB, Bochum
04/1995 – 09/1999	HNA Ausland der Firma Hochtief AG, Essen: Abteilungen AVA, Kalkulation der TES
Ab 01/11/1999	Bauingenieur / Programmierer in der Abteilung Risikomanagement der Firma HOCHTIEF Construction AG, Essen