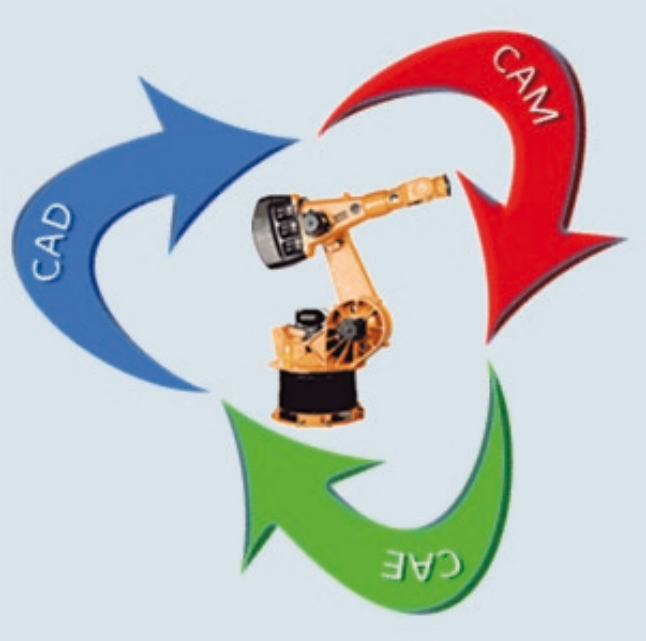
Schriftenreihe des Instituts für
Angewandte Informatik / Automatisierungstechnik
Universität Karlsruhe (TH)
Band 16

Shutao Li

Entwicklung eines Verfahrens zur Automatisierung der CAD/CAM-Kette in der Einzelfertigung am Beispiel von Mauerwerksteinen



universitätsverlag karlsruhe

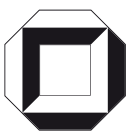
Shutao Li

Entwicklung eines Verfahrens zur Automatisierung der CAD/CAM-Kette in der Einzelfertigung am Beispiel von Mauerwerksteinen

Schriftenreihe des
Instituts für Angewandte Informatik / Automatisierungstechnik
an der Universität Karlsruhe (TH)
Band 16

Entwicklung eines Verfahrens zur Automatisierung der CAD/CAM-Kette in der Einzelfertigung am Beispiel von Mauerwerksteinen

von
Shutao Li



universitätsverlag karlsruhe

Dissertation, Universität Karlsruhe (TH)
Fakultät für Maschinenbau, 2007

Impressum

Universitätsverlag Karlsruhe
c/o Universitätsbibliothek
Straße am Forum 2
D-76131 Karlsruhe
www.uvka.de

Universitätsverlag Karlsruhe 2007
Print on Demand

ISSN: 1614-5267
ISBN: 978-3-86644-114-9

Entwicklung eines Verfahrens zur Automatisierung der CAD/CAM-Kette in der Einzelfertigung am Beispiel von Mauerwerksteinen

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Maschinenbau der

Universität Karlsruhe

eingereichte

Dissertation

von

Dipl.-Ing. Shutao Li

geboren am 20. April 1973 in Shandong, China

Hauptreferent:	Prof. Dr.-Ing. habil. G. Bretthauer
Korreferent:	Prof. Dr.-Ing. habil. J. Wernstedt
Tag der Einreichung:	05. 12. 2006
Tag der mündlichen Prüfung:	02. 02. 2007

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit im Institut für Angewandte Informatik des Forschungszentrums Karlsruhe.

Für die Möglichkeit an einem interessanten und umfangreichen Gebiet forschen zu können, möchte ich Herrn Prof. Dr.-Ing. habil. Georg Bretthauer herzlich danken. Er gab in wichtigen Gesprächen viele Anregungen und konstruktive Kritik zu dieser Arbeit.

Herrn Prof. Dr.-Ing. habil. Jürgen Wernstedt danke ich herzlich für sein der Arbeit entgegen gebrachtes Interesse und die Übernahme des Korreferats.

Ein besonderer Dank gilt Dr. Jörg Isele, der seit meinen ersten Tagen im Institut jederzeit bereit war, in intensiven fachlichen Diskussionen die Entwicklung der Arbeit zu begleiten, und das fleißige und rechtzeitige Korrekturlesen zu übernehmen. Ebenso möchte ich mich bei Herrn Karl-Heinz Häfele für die Anregungen und die wertvollen Hinweise bedanken.

Für die Unterstützung bei der softwaretechnischen Implementierung danke ich Herrn Franz-Josef Kaiser und Dr. Arnold Ludwig. Mein Dank gilt Herrn Walter Till für die Einführung in die KUKA-Roboterprogrammierung.

Für viele wertvolle und unterhaltsame Diskussionen gilt mein spezieller Dank meinem Zimmerkollegen Herrn Andreas Geiger sowie allen anderen Mitarbeitern des Instituts, die zum Gelingen dieser Arbeit beigetragen haben.

Weiterhin danke ich allen Studenten, die im Rahmen von Praktika, Studien- oder Diplomarbeiten einen Beitrag zu dieser Arbeit geleistet haben.

Mein herzlicher Dank gilt schließlich meinen Eltern und Schwiegereltern sowie meiner Frau für Rückhalt, Verständnis, Ermutigung und Unterstützung.

Karlsruhe, im Februar 2007

Shutao Li

INHALTSVERZEICHNIS

BILDVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGEN

AP	**A**pplication **P**rotocol
API	**A**pplication **P**rogramming **I**nterface
AS	**A**blauf**s**prache
ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
AWL	**An**weisungs**l**iste
B-Rep	**B**oundary **Rep**resentation
CAAD	**C**omputer **A**ided **A**rchitectural **D**esign
CAD	**C**omputer **A**ided **D**esign
CAE	**C**omputer **A**ided **E**ngineering
CAM	**C**omputer **A**ided **M**anufacturing
CAN	**C**ontroller **A**rea **N**etwork
CIM	**C**omputer **I**ntegrated **M**anufacturing
CNC	**C**omputerized **N**umerical **C**ontrol
COM	**C**omponent **O**bject **M**odel
CSG	**C**onstructive **S**olid **G**eometry
DCOM	**D**istributed **C**omponent **O**bject **M**odel
DLL	**D**ynamic **L**ink **L**ibrary
DXF	**D**rawing E**x**change **F**ormat
FUP	**Fu**nktions**p**lan
GML	**G**eography **M**arkup **L**anguage
GUID	**G**lobal **U**nique **Id**entification
HMI	**H**uman **M**achine **I**nterface
IAI	**I**nternational **A**lliance for **I**nteroperability
IEC	**I**nternational **E**lectrotechnical **C**ommission
IFC	**I**ndustry **F**oundation **C**lasses
IGES	**I**nitial **G**raphics **E**xchange **S**pecification
ISO	**I**nternational **O**rganization for **S**tandardization
KCP	**K**UKA **C**ontrol **P**anel
KOP	**Ko**ntakt**p**lan
KRC	**K**UKA **R**obot **C**ontrol
KRL	**K**UKA **R**obot **L**anguage
MAC	**M**edia **A**ccess **C**ontrol
MFC	**M**icrosoft **F**oundation **C**lasses
NC	**N**umerical **C**ontrol
OID	**O**bject **Id**entification
OLE	**O**bject **L**inking and **E**mbedding
OPC	**O**LE for **P**rocess **C**ontrol
OpenGL	**Open G**raphics **L**ibrary
PDM	**P**roduct **D**ata **M**anagement
POE	**P**rogramm**o**rganisations**e**inheit
PTP	**P**oint **t**o **P**oint
RAM	**R**andom-**a**ccess **m**emory
SPS	**S**peicher**p**rogrammierbare **S**teuerung
ST	**S**trukturierter **T**ext
STEP	**St**andard for the **E**xchange of **P**roduct Model Data
TCP	**T**ool **C**enter **P**oint (in Robotik)
TCP/IP	**T**ransmission **C**ontrol **P**rotocol/ **I**nternet **P**rotocol
UML	**U**nified **M**odeling **L**anguage
VDI	**V**erein **D**eutscher **I**ngenieure
XML	**E**xtensible **M**arkup **L**anguage

1 Einleitung

1.1 Motivation und Bedeutung

„Ein Dach über dem Kopf“ ist eines der menschlichen Grundrechte und Grundbedürfnisse. Die meisten Normalbürger können aber die Kosten für den Bau eines Hauses normaler Größe für eine Durchschnittsfamilie kaum bezahlen. Im Vergleich mit der industriellen Serienfertigung der meisten sonstigen vom Menschen benötigten Güter, ist die bisherige Bauweise im Hausbau zu teuer. Die oft noch mittelalterlich traditionell anmutende handwerkliche Arbeit auf der Baustelle und die lange Bauzeit verursachen die hohen Baukosten.

Kostengünstiges Bauen und insbesondere der kostengünstige Wohnungsbau ist ein zentrales Ziel aller am Bauprozess Beteiligten [EHLT]. Der weltweit steigende Bedarf an preisgünstigen und individuellen Ansprüchen befriedigenden Wohnungen stellt Bauplaner, Bauherren und Investoren vor die Aufgabe, die Bauzeit und die Baukosten im Rahmen der wirtschaftlichen und technischen Möglichkeiten einzusparen, ohne dabei Qualitätseinbußen hinzunehmen. Aus organisatorischer, wirtschaftlicher und technischer Sicht gibt es dazu verschiedene Lösungsvorschläge [ROMH]. In dieser Arbeit soll eine neue Lösung vorgestellt werden, nämlich die computergestützte Vorfertigung aller Wandelemente mit integrierter Haustechnik.

Statt verschiedene Arbeitsvorgänge seriell auf der Baustelle zu erledigen, werden die Beiträge aller Gewerke in einer Vorfertigungsfabrik produziert. Von der industriellen Vorfertigung profitiert die Bauwirtschaft mit einer ganzen Reihe von Vorteilen:

- Schnellere und einfachere Montage auf der Baustelle
- Passgenaue sichere Qualität und geringer Materialverlust
- Witterungsunabhängigkeit
- Rationalisierung ermöglicht Produktivitätssteigerung
- Verkürzte Bauzeit und geringerer Personalbedarf sowie geringere Lohnkosten
- Befriedigung individueller Kundenwünsche.

Entscheidend für die Wirtschaftlichkeit der Vorfertigung ist dabei die enge Zusammenarbeit von Konstruktion, Produktionsplanung und Fertigung. Im Gegensatz zur Automobilindustrie, wo große Stückzahlen produziert werden, ist im Bauwesen nach wie vor die Einzelfertigung vorherrschend [ROMH]. Um eine weitgehend automatisierte Fertigung zu realisieren, müssen moderne elektronische Datenverarbeitungsprinzipien in die Planung und Produktion einbezo-

gen werden. Eine rationelle Herstellung erfordert den Einsatz automatisierter Produktionsanlagen, die durchgängig mit der Bauplanung verknüpft sind.

1.2 Darstellung des Entwicklungsstandes

1.2.1 Geschichtlicher Rückblick der Vorfertigung

Die Definition des Begriffs Vorfertigung ist durch das Wort selbst gegeben. Diese Art der Rationalisierung besteht aus der Herstellung von Teilen, die zusammengefügt ein fertiges Produkt ergeben. Dieses Produkt wird zunächst als Ganzes geplant. Dann wird es im Hinblick auf Produktion und Montage in sinnvolle Teile aufgeteilt, produziert und schließlich in logischer Reihenfolge zusammengesetzt. In Verbindung mit der Vorfertigung wird häufig die Forderung erhoben, dass das vorgefertigte Element einen nahezu endgültigen Zustand aufweisen sollte, damit die Folgearbeiten vor Ort so gering wie möglich sind [EHLT].

Die Vorzüge der Vorfertigung werden seit vielen Jahrhunderten überall auf der Welt genutzt. Blickt man in die Geschichte der Vorfertigung [JUNG], so werden immer wieder entscheidende gesellschaftliche Ereignisse wie die Errichtung von Kultbauten, Kriege, Kolonialisierung oder Naturkatastrophen, die eine Vorfertigung im Wohnungsbau forcierten, gefunden. Aber auch die Entwicklung neuer Baustoffe inspirierte die Pioniere immer wieder zur Entwicklung neuer Verfahren und Systeme.

Als frühe Beispiele der handwerklichen Vorfertigung sind die ägyptischen Kultbauten anzusehen. In Steinbrüchen wurden die für Tempel und Pyramiden benötigten Stützen und Quadersteine vorgefertigt und meist über große Entfernungen mittels Holzschlitten zu den Montageorten transportiert. Klassisches Beispiel der antiken Vorfertigung ist der griechische Tempelbau. Große tragende Bauteile wie Säulen und Balken wurden im Steinbruch weitgehend vorgefertigt und als Fertigteile an die Einbaustellen transportiert. Auf der Baustelle wurden die vorgefertigten Bauteile maßgenau nachgearbeitet und mit Dübeln und Klammern aus Bronze oder Eisen als Verbindungsmittel zusammengefügt [EHLT]. Am Ende des 15. Jahrhunderts hat Leonardo da Vinci für die Herzogin Isabella Sforza einen Gartenpavillon in Tafelbauweise entworfen. Um 1600 wurden in Moskau größere Mengen von Wohnhäusern in der dort üblichen Blockbauweise produziert.

Der wichtigste Auslöser für Vorfertigung und Montage war anfänglich überhaupt das Militärwesen. So wurden 1788 während des Türkenkrieges in Wien 24 Lazarett-Baracken zusammengestellt und auf der Donau nach Slawonien in das Kampfgebiet verschifft. 1790 brachte die britische Flotte ein vorgefertigtes Krankenhaus nach Australien. Während des

Krimkrieges 1854-56 errichteten die Engländer bei Renkioi ein vorbildliches Etappenlazarett. Es bestand aus 64 in England vorgefertigten Holzfachwerkbaracken, die in drei getrennten Gruppen an überdachten foyerartigen Gängen aufgereiht worden waren. Im amerikanischen Sezessionskrieg 1865 war der Einsatz von vorgefertigten, leicht transportierbaren Baracken wegen der geringen Besiedlung des Landes eine dringende Notwendigkeit. Es wurden damals 214 Barackenlazarette mit über 100.000 Betten errichtet [LANG]. Durch die Anforderungen des Militärs nahm die Barackenproduktion einen heftigen Aufschwung, der die Entwicklung von neuen für den Transport und die Montage besonders geeigneten Bausystemen zur Folge hatte.

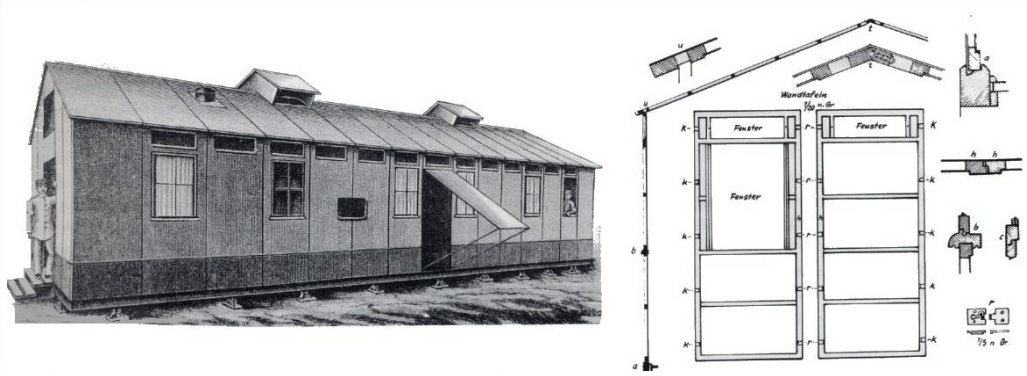


Bild 1.1: Sanitätsbaracke, System Doecke, 1885 [JUNG]

Des Weiteren war auch der große Wohnraummangel in den Kolonien ein starker Anreiz zur Entwicklung der Vorfertigung, denn der stark wachsende Bedarf an Wohnraum konnte vor Ort durch den üblichen Massivbau nicht gedeckt werden. Dort fehlte es vor allem an den Bauhandwerkern und Fachkräften für die Herstellung der herkömmlichen Baukonstruktion. Dadurch erhöhten sich die Facharbeiterlöhne und die Wohnungsmieten extrem stark. Die Kosten für die Vor-Ort-Fertigung waren so hoch, dass die Vorfertigung von Wohnhäusern, Läden und Geschäftshäusern in Europa und der Transport nach Übersee zu einem gewinnbringenden Geschäft werden konnten.

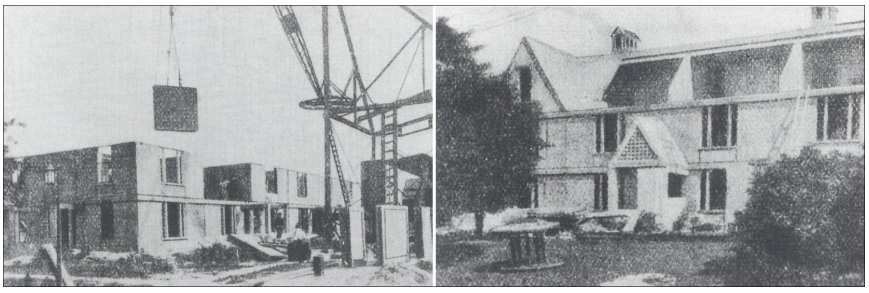


Bild 1.2: Großplattenbauweise von Grosvenor Atterbury, Montage und Rohbau [JUNG]

1918 wurde die von dem Ingenieur Grosvenor Atterbury entworfenen Bauweise eingeführt. Er fasste die vielen Teilarbeiten auf der Baustelle in wenige Arbeitsgänge zusammen, indem

er statt der bisher üblichen, kleinen Betonblöcke geschoßhohe und raumgroße Wandplatten stampfen und mit einem Kran versetzen ließ. Bild 1.2 zeigt die Großplattenbauweise und ein so entstandenes Fertighaus.

Ein Pionier des Einsatzes der Großenplattenbauweise in Deutschland war Martin Wagner. Durch das Studium der Bautechnik und der Bauwirtschaft in den Vereinigten Staaten und den Niederlanden war er von den Vorteilen der Rationalisierung des Bauens überzeugt. Das von ihm gegründete Unternehmen hat in Berlin-Friedrichsfelde einige Gebäude gebaut. Die großformatigen Platten wurden in einer Feldfabrik direkt vor dem Gebäude in Holzschalungen gefertigt (Bild 1.3).

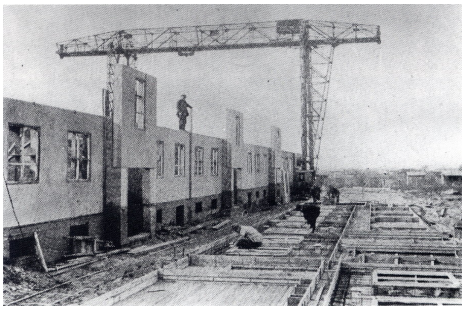


Bild 1.3: Großplattenbauweise von Martin Wagner [JUNG]

Auf der Basis der Erfahrung Wagners in Berlin-Friedrichsfelde entwickelte Ernst May, der 1926 Stadtbaurat von Frankfurt am Main wurde, eine Fertigteilbauweise mit Großblockbauweise (Bild 1.4). Die einzelnen Wandelemente waren nicht mehr geschoßhoch, sondern es gab Brüstungs-, Fenster- und Sturzblöcke. So konnte der Bewehrungsanteil erheblich reduziert und statt eines Portalkrans ein einfacher Drehkran eingesetzt werden. Diese Bauweise vereinfachte den Wandaufbau und reduzierte das Wandgewicht.

Bild 1.4: Großblockbauweise von Ernst May [JUNG]

1941 entwickelten Gropius und Wachsmann zusammen in den USA ein universelles Bausystem. Das „General Panel System“ bestand aus vorgefertigten Kleinholztafeln zum Bau von ein- und zweigeschossigen Gebäuden (Bild 1.5). Zeitgleich entstand in den USA mit großen öffentlichen Investitionen die Lustron Corporation, welche die Techniken der Automobilin-

dustrie auf den Hausbau übertrug und ca. 2000 Häuser aus emaillierten Blechen verkaufte, bevor sie schon 1950 wieder unterging [LUST].

Bild 1.5: "General Panel System", Aufbau und Plattenwagen [JUNG]

Die enormen Kriegzerstörungen machten in Deutschland einen raschen Wiederaufbau des benötigten Wohnraumes erforderlich. Bereits 1960 wurden die ersten Fertighäuser vorgestellt. Da eine Produktivitätssteigerung mit der konventionellen Bautechnik nicht zu bewältigen war, entschied sich die DDR auf außerordentlich konsequente Weise zur vollständigen Industrialisierung des gesamten Baugewerbes mit dem Ziel besser, billiger und vor allem schneller zu bauen [GRA1]. Über die Verwendung von Block- und Tafelbauweise wurde die Plattenbauweise in der DDR sehr weit verbreitet (siehe Bild 1.6). 1985 wurden 83 Prozent der Wohnungsneubauten in der DDR in Plattenbauweise errichtet [RIOL].

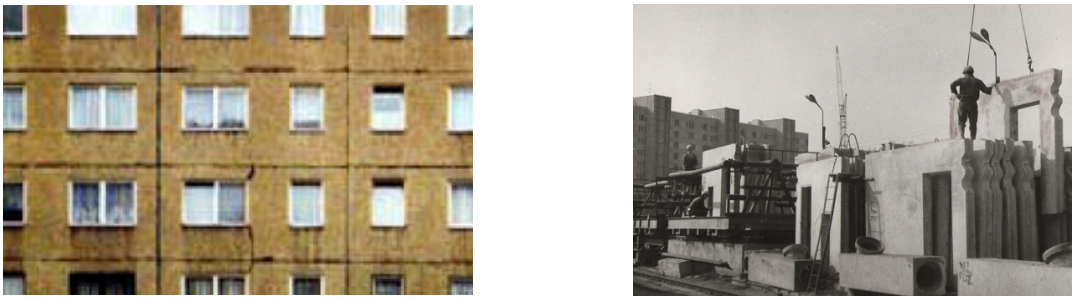


Bild 1.6: Plattenbauweise (Quelle: Links: DB 3/97, (Zs.), S.87; Rechts: Lexitv)

Wegen mangelnder Varianz der großformatigen Betonelemente hat die Plattenbauweise wenig Gestaltungsvielfalt. Die häufige Verwendung der Schalelemente führte zu mangelhafter Maßhaltigkeit der Fassadenelemente. Solche Faktoren haben der Plattenbauweise einen schlechten Ruf eingebracht.

1.2.2 Systeme des Fertighausbaus

Es gibt am Markt drei Arten von Montagebau, nämlich Skelettbauweise, Wandelementbauweise und Raumzellenbauweise [KOTU][BRET].

Die **Skelettbauweise** beruht auf einer tragenden Konstruktion aus Stützen und Riegeln z.B. aus Stahlbeton, Stahl oder Holz, auf die die raumabschließenden, nicht tragenden Wände und

Decken montiert bzw. aufgelegt werden. Neben den tragenden Elementen kann auch die Ausfachung vorgefertigt werden. Im Vergleich zu den anderen Montagebauweisen ist der Arbeitsaufwand auf der Baustelle relativ groß. Bild 1.7 (Quelle: Richter System GmbH) zeigt ein Beispiel für die Stahlskelettbauweise. Die Skelettbauweise ist im Industriebau sehr weit verbreitet.

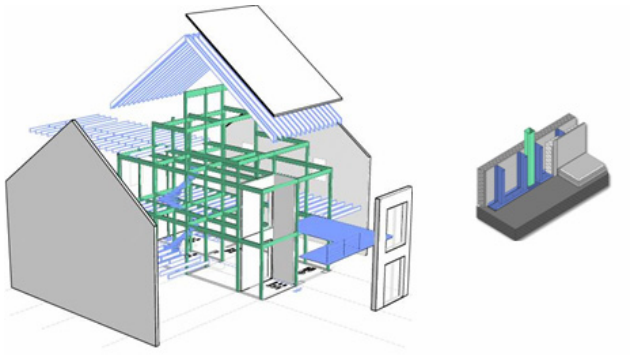


Bild 1.7: Stahlskelettbau: Gesamtsicht mit Wandaufbau

Die Wandtafeln bei der **Wandelementbauweise** sind zumeist raumgroß, so dass die Elemente nur an den Raumecken aneinander stoßen. Als Materialien werden Normal- oder Leichtbeton, Mauerwerksteine oder Holz eingesetzt. Ein Beispiel zeigt Bild 1.8 (Quelle: Dennert KG). Die Installations- und Ausbauarbeiten wie z.B. Kanäle, Leerrohre oder Einrichtungen wie Elektro- und Wasserleitungen, Wandheizungen, Fenster und Türen können vorbereitet werden. Eine Dämmung und Putz kann ebenfalls in der Vorfertigung aufgebracht werden. Damit verringert sich der Aufwand an notwendigen Ausbauarbeiten auf der Baustelle erheblich. Bei dieser Bauweise gibt es aber zwei Probleme. Zum einen stellt die Realisierung individueller Grundrisse vor allem höhere Anforderungen an die Vorfertigungsplanung und führt zu relativ hohen Herstellungskosten. Zum anderen sind die Anforderungen an den Transport wegen der genormten Abmessung eines Lastwagens zu berücksichtigen.

Bild 1.8: Wandelementbauweise

Bei der **Raumzellenbauweise** ist das Prinzip der werkseitigen Vorfertigung am weitesten verwirklicht. Die Raumzellenbauweise erlaubt den annähernd vollständigen Innenausbau im

Werk, so dass sich auf der Baustelle die Arbeit im Wesentlichen auf die Montage beschränkt. Anwendung findet sie vor allem bei kleinen (z.B. Garage) oder hochinstallierten Räumen (z.B. Sanitärzellen) sowie temporären Gebäuden (z.B. Bauunterkünfte). Darüber hinaus werden Raumzellen bei typisierten Gebäuden wie Krankenhäusern, Schulen und Verwaltungsgebäuden eingesetzt (siehe Bild 1.9, Quelle: Meisterstück Baumeister GmbH (links), Verband Österreichischer Beton- und Fertigteilwerke (rechts)). Die Realisierung individuell geplanter Wohngebäude mit Raumzellen ist zumindest derzeit nicht konkurrenzfähig.

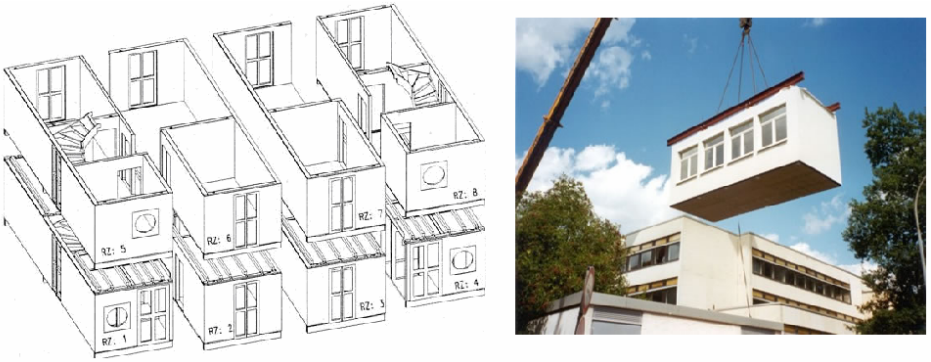


Bild 1.9: Raumzellenbauweise

Die Fertigteile sind in ihrer Größe beschränkt. Die Straßenverkehrsordnung (StVO) gibt genaue Richtlinien vor, welche Abmessungen ein Lastwagen samt seiner Ladung haben darf.

„§ 22 Ladung

(2) Fahrzeug und Ladung dürfen zusammen nicht breiter als 2,55m und nicht höher als 4m sein. (…)

(4) … Fahrzeug oder Zug samt Ladung darf nicht länger als 20,75m sein. (…)“

Bei größeren Elementen muss eine Sondergenehmigung beantragt werden, eventuell muss der Transport dann unter Polizeibegleitung durchgeführt werden [HAHN].

1.2.3 Computer Aided Design (CAD)

Anfang der 80-er Jahre wurde das rechnergestützte Entwerfen und Konstruieren (CAD) in Architekturbüros eingeführt und ist mittlerweile als Standardtechnik nicht mehr wegzudenken. Als Entwurfs- und Zeichenwerkzeug bietet es eine Vielzahl an Einsatzmöglichkeiten und erleichtert die Arbeit von Architekten. Zunächst wurde unter dem Begriff CAD die rechnergestützte Zeichnungserstellung verstanden. Heute zielen die CAD-Systeme über die reine Zeichnungserstellung hinaus auf die geometrische Modellierung von Objekten des Ingenieurwesens. Gearbeitet wird zunehmend häufiger mit einem dreidimensionalen geometrischen Modell eines Projektes, also nicht mit einem zweidimensionalen Plan.

Im Vergleich zu allgemeinen CAD-Programmen, die nicht allein für bauspezifische Applikationen geschaffen sind, bieten die in den letzten Jahren entwickelten CAAD-Programme (Computer Aided Architectural Design) mehr bauspezifische Anpassungen, z.B. Architectural Desktop (Autodesk), Allplan (Nemetschek) und ArchiCAD (Graphisoft). Mit solchen objektorientierten, computergestützten Entwurfssystemen lässt sich ein Haus mit Wänden, Decken, Dach, Treppen und den Bauelementen wie z.B. Fenstern, Türen usw. „aufbauen“. Eine Wand besteht z.B. aus Wand, Wandöffnungen, Tür- und Fensterelementen (siehe Bild 1.10). Die Elemente enthalten neben Produkteigenschaften wie Material, Farbe usw. auch die Beziehungen untereinander.

„Das rechnerunterstützte Konstruieren hat zum Ziel, alle Informationen, die im Produktlebenszyklus anfallen, zu speichern und immer zur Verfügung zu stellen“ [GRA2].

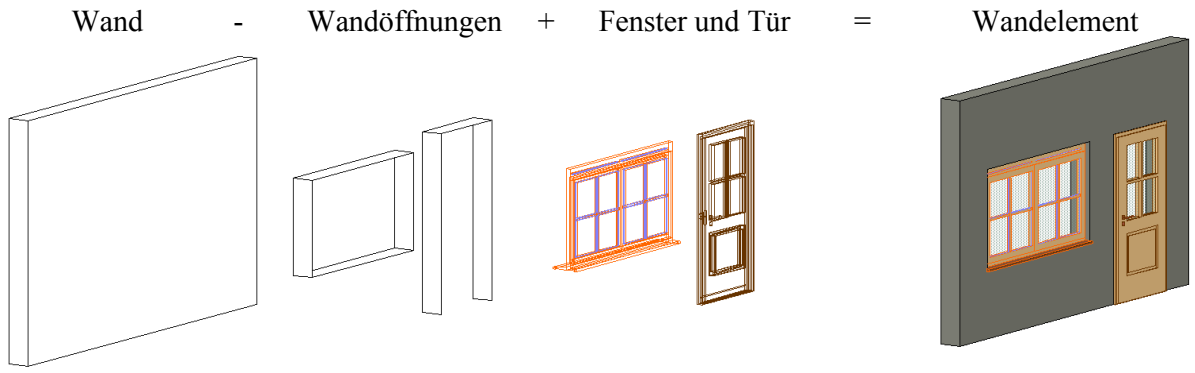


Bild 1.10: Konstruktion eines Wandelements [BRET]

1.2.4 Datenschnittstelle im CAD

Datenaustausch ist ein typisches Problem in CAD-Anwendungen. Es ergibt sich bereits innerhalb eines Unternehmens z.B. bei einem Systemwechsel. Häufig gehen viele Informationen beim Datenaustausch verloren, was dazu führt, dass die vorhandenen Datenbestände komplett neu erfasst werden, bevor sie wieder verwendet werden können. Die durch die mangelhafte Qualität der ausgetauschten Daten erforderliche Nachbearbeitung stellt einen erheblichen Kostenfaktor dar. Im Bauwesen ist der Datenaustausch von großer Bedeutung, weil die Bauplanung hochgradig interdisziplinär ist. Während des gesamten Lebenszyklus werden viele Akteure beteiligt wie z.B. Architekt, Bauherr, Bauausführende, Statiker, Haustechniker usw. Die einzelnen Beteiligten setzen dabei unterschiedliche CAD-Systeme ein. Der Datenaustausch zwischen diesen Systemen funktioniert, wenn überhaupt, mit erheblichem Informationsverlust und vielen Fehlern.

Konventioneller Datenaustausch reiner 2D-Daten erfolgt über DXF (Drawing Exchange Format), ein weit verbreitetes Datenformat der Firma Autodesk. Fast jedes CAD- oder CAE-System (Computer Aided Engineering) unterstützt den Datenaustausch über das DXF–Format. Mit Hilfe von DXF ist es möglich, Zeichnungen von einem CAD-System auf ein nahezu beliebiges CAD-System zu übertragen. Heute wird es häufig zum Austausch von Zeichnungsdaten in digitaler Form verwendet. Aber ein dreidimensionales Volumenmodell kann damit kaum repräsentiert werden. Mit IGES (Initial Graphics Exchange Specification) kann ein 3D-Modell transportiert und zumindest grundlegend manipuliert werden. Aber solche Datenformate beschränken sich auf dem Austausch von reinen Geometrieinformationen eines Gebäudes. Sie sind keine durchgängigen Lösungen hinsichtlich des gesamten Lebenszyklus eines Gebäudes.

Die Produktdatenmodellierung ist die Basis für eine durchgängige Datenverarbeitung in allen Lebensphasen eines Gebäudes. Mit den IFC (Industry Foundation Classes) hat die International Alliance for Interoperability (IAI) ein integrales Produktdatenmodell für Gebäude entworfen [BRET]. Es enthält neben den Geometriedaten auch topologische und technologische Informationen. Dadurch kann der gesamte Prozess vom ersten Entwurf über die Bauausführung bis hin zur Gebäudeverwaltung vereinfacht und wesentlich beschleunigt werden. Das IFC Produktdatenmodell ermöglicht, die Daten in jeder Bauphase zu sammeln und in alle nachfolgenden Schritte der Bauwerkerstellung mit einzubinden [DIAI]. Es ermöglicht auch, Informationen über die Eigenschaften eines Gebäudes zu speichern und programmübergreifend zu übertragen. Die dabei übermittelten Eigenschaften gehen über rein geometrische Informationen weit hinaus: Materialeigenschaften, Oberflächeneigenschaften etc. können im Produktdatenmodell enthalten sein. Alle Elemente wie Wände, Decken, Fenster, Türen, Fliesen, Heizungen, Sanitärobjekte etc. bis hin zu den Steckdosen werden im Hausmodell festgelegt.

1.2.5 CAD/CAM-Einsatz in der Bauindustrie

Um ein CAD-Systems zu einem leistungsfähigen CAD/CAM-System zu erweitern, ist im Allgemeinen eine offene Programmierschnittstelle erforderlich. Sie stellt eine Verbindung zwischen dem CAD-System und dem jeweiligen Fertigungssteuerungssystem her. Dadurch ermöglicht sie eine Schnittstelle zur CNC (Computerized Numerical Control)-Steuerung, die den Austausch aller fertigungsrelevanten Daten zur Arbeitsvorbereitung und Arbeitsdurchführung gewährleistet. Ein CAD/CAM-System kann entweder auf einem lokalen Rechner oder einem Netzwerk aus mehreren Rechnern realisiert werden. Eine CAD/CAM-Lösung bietet dem Unternehmen den Vorteil, alle Arbeitsgänge, von der Angebotsbearbeitung bis hin zur

Fertigung, zentral zu steuern. Falls während eines Fertigungsprozesses, was häufig vorkommt, Änderungen vorgenommen werden müssen, ist eine CAD/CAM-Lösung von besonderem Nutzen [GANZ]. Heutzutage verfügen viele große CAD-Systeme im Maschinenbaubereich über einen so genannten „Postprozessor“. Dabei werden die Daten, welche für die Steuerung der CNC-Werkzeugmaschine notwendig sind, anhand des vorhandenen CAD-Modells erzeugt. Die in einer ASCII-Datei abgelegten maschinenneutralen Steuerungsinformationen können wiederum mit Hilfe spezieller Postprozessoren in die NC-Bearbeitungsdaten konvertiert werden. Mit dem abschließenden Schritt erfolgt die maschinenspezifische Anpassung an die zu verwendende Werkzeugmaschine. Die entsprechende Fertigungsbahn (üblich für das Fräsen) kann auch gleich generiert und visualisiert werden. Auf dem Markt gibt es zahlreiche solche Systeme wie z.B. Pro/Engineer, NX Unigraphics, Solidworks, Vector CAD/CAM, Mastercam usw. Sie sind meistens spezifisch für den Maschinenbau und insbesondere die Automobilindustrie entwickelt und werden auch meist dort betrieben. Der NC-Sprachumfang wird allerdings auf die NC-Sätze beschränkt, die in der zur Fertigungsanlage gehörenden Steuerung vordefiniert sind. Um den Postprozessor auf eine spezifische Fertigungsanlage anzupassen, ist für eine entsprechende Schnittstelle zu sorgen, damit die Anlage die Daten „verstehen“ kann.

Im Bauwesen werden wegen der unterschiedlichen Materialien und des unterschiedlichen Zwecks verschiedene CAD/CAM-Systeme in der Vorfertigung eingesetzt. Zahlreiche CAD/CAM-Lösungen sind inzwischen für den Holzbau auf dem Markt zu finden. Alle für den Holzbau benötigten Stücklisten und Zeichnungen, z.B. Holzlisten, Lamellenpläne usw., lassen sich automatisch erzeugen. Fertigungsunterlagen für die Herstellung von Bauteilen des Holzbaues können dann abgeleitet werden. Ferner werden die Bearbeitungsdaten der Einzelteile wie Sparren, Pfosten, Riegel usw. für die Abbundanlage aufbereitet. Hierbei werden die eingesetzten CNC-Bearbeitungsmaschinen von der CAM-Software unterstützt [ROMH].

Bei einer Schwellen-/Pfetten-Station (Quelle: Fa. Homag) wird das Holz während der Zuführung zu den Bearbeitungsstationen mit einem NC-gesteuerten Greifer automatisch vermessen. Bearbeitungen wie Anreißen, Bohren, Fräsen, Einpressen von Nagelplatten sowie winkelgerechtes Absägen werden programmgesteuert vollautomatisch ausgeführt. Danach wird das bearbeitete Holz weitertransportiert und der Weiterbearbeitung zur Verfügung gestellt (Bild 1.11 links). Das Riegelwerk wird in einer Riegelstation automatisch erstellt. Dazu gehören das Zuführen und Auflegen aller Hölzer, Absägen der Ober- und Untergurte, Ausrichten und Abnageln der Hölzer, Fräsen und Bohren im Gurt (Bild 1.11 rechts).

Bild 1.11: Links: Schwellen-/Pfetten-Station, Rechts: Riegelwerkstation

In einer Multifunktionsbrücke (Bild 1.12, Quelle: Fa. Homag) ist ein vollautomatischer Werkzeugwechsler integriert. Hierbei werden die Holzplatten ver-/bearbeitet sowie befestigt. Zu den Aufgaben der Multifunktionsbrücke gehören u.a. [ROMH]:

- Automatisches Auflegen von Holzplatten wie Spanplatten, Gipskartonplatten
- Automatisches Abnageln, Schrauben und Klammern der aufgelegten Platten
- Automatisches Sägen bzw. Fräsen von Fenster- und Türausschnitten
- Automatisches Bohren von Elektrodosen sowie Wandanschlussbohrungen.

Bild 1.12: Multifunktionsbrücke

Auch in der Betonfertigteilindustrie wird die CAD/CAM-Idee eingesetzt. Beim Porenbetonhersteller XELLA werden z.B. Fertigwände produziert. Auf der Fertigungsstraße werden die Steine auf bestimmte Maße abgesägt. Danach werden anhand der Wand-Zeichnung die liegenden Steine mit Berücksichtigung der Tür- und Fensteröffnungen zu einer Wand zusammengeklebt. Bei Dachschrägen wird die Schräge abschließend mit einer Seilsäge abgesägt.

1.3 Ziele und Aufgaben

Das Ziel dieser Arbeit besteht darin, ein Verfahren zur Erweiterung der technologischen Vorfertigung von Wandelementen zu entwickeln und den gesamten Prozess vom individuellen CAD-Modell bis zur computergestützten Fertigung in Losgröße Eins zu automatisieren.

Als Lösungsansatz werden alle Wände aus dem CAD-Hausmodell herausgefiltert. Unter der Berücksichtigung aller Wandöffnungen, vorhandener Steingröße sowie anderer technischer Sichten wird jede Wand in einzelne geschosshohe Mauerwerksteine aufgeteilt. Neben dem Zusägen auf die benötigen Maße werden die Steinblöcke in der Vorfertigung auch durch das Fräsen von Installationsschlitzen usw. bearbeitet. Dazu dient die Geometrie dieser Blöcke als Grundlage für die Bearbeitungsmakros eines Säge- bzw. Fräsroboters.

Die Entwicklung des Verfahrens erfordert folgende Entwicklungsaufgaben:

1. Durch den Vergleich marktüblicher CAD-Schnittstellen im Bauwesen ist ein 3D-Datenformat auszuwählen, das ein Gebäudemodell objektorientiert beschreibt und zu den CAAD-Marktführern kompatible ist.
2. Algorithmen zur Bearbeitung des Gebäudemodells z.B. Wandmanipulation, Aufteilung in einzelne geschosshohe Steine usw. sind auf dem Basis vom vorhandenen Softwareframework zu entwickeln und implementieren.
3. Ein eigenes Datenaustauschformat zwischen dem Gebäudemodell und der Fertigung ist zu erschaffen. Um die wichtige Datenquelle der Fertigung zu gewährleisten, soll das neue Datenformat eine Validierungsfunktion zur Verfügung stellen. Eine grafische 3D-Vsualisierung ist sinnvoll zu entwickeln, um die Funktionalität und die Regenerierbarkeit der Gebäudeentwurf mit dem neuen Datenformat aussagekräftig zu nachweisen.
4. Eine automatisierte Fabrik zur Vorfertigung von Mauerwerksteinen soll als Prototyp konzipiert und realisiert werden. Die Bearbeitungsmakros für Sägen und Fräsen sind in der Robotersteuerung bereitzustellen. Ein Steuerungsprogramm ist neu zu entwickeln, um die gesamte Anlagensteuerung zu vereinen, die Informationen im neu entwickelten Datenformat zu „empfangen“.

Kapitel 2 stellt Produktdatenmodelle zu dem individuellen Hausentwerfen vor. Das IFC-Produktdatenmodell ist ein gut eingeführter Standard, auf dem diese Arbeit aufbaut. Die zu der Arbeit relevanten IFC-Objekte werden hier untersucht und analysiert.

Kapitel 3 ist von zentraler Bedeutung für die Entwicklung eines Verfahrens zur Automatisierung der CAD/CAM-Kette in der Einzelfertigung von Mauerwerksteinen. Hier wird das neutrale Datenaustauschformat - BauXML als Brücke von IFC-Produktdatenmodell zur Fertigung neu konzipiert und spezifiziert. BauXML soll folgende Aufgaben lösen:

- Vermittlung zwischen der Wand (visuelles IFC-Objekt) und dem Stein (reales Bauobjekt)
- Nutzung des verbreiteten Beschreibungsstandards XML
- Aus BauXML-Daten soll eine grafische Gesamtansicht erstellt werden können
- Datendefinition mit Hinsicht auf die Fertigung.

Ein vom Architekt entworfenes Hausmodell wird nach dem IFC-Standard exportiert. Zur Bearbeitung müssen Wandobjekte und ihre Öffnungen aus dem Modell herausgefiltert und anschließend fertigungs- und montagegerecht bearbeitet werden. Nach bestimmten technischen Randbedingungen sollen die Wände dann in einzelne Steine segmentiert werden. Jeder Stein ist der Geometrie der Wand anzupassen. Die neu erzeugten Steininformationen sind in dem neuen BauXML-Format zu speichern. Die dafür entwickelten Algorithmen und die softwaretechnische Implementierung werden komplett in Kapitel 4 dargestellt. Das Ergebnis wird anhand mehrerer Beispiele demonstriert und die Lösung evaluiert.

Das Kapitel 5 beschreibt die Zellensteuerung einer prototypischen CIM-Wandvorfertigungsfabrik. Sie umfasst die Robotersteuerung, die Kommunikation und die Integration des Roboters mit Förderbahnen und allen notwendigen Peripherien. Darüber hinaus ist das neu entwickelte Steuerungsprogramm „WallProducer“ hier vorgestellt.

Das Kapitel 6 fasst die wichtigsten Ergebnisse der Arbeit zusammen, gibt einen Ausblick auf die Verwendung des in der Arbeit entwickelten Verfahrens im Bauwesen und zeigt weitere Entwicklungsmöglichkeiten auf.

2 Produktdatenmodell

Am Lebenszyklus eines Produktes ist eine Vielzahl von Akteuren beteiligt. Jeder dieser Akteure verwendet unterschiedliche Softwareprodukte, die an die jeweiligen Bedürfnisse seiner Anwendung angepasst sind, zur Darstellung und Bearbeitung von Informationen. Wegen der unterschiedlichen Strukturierung und Sichtweisen auf die Informationen gestaltet sich der digitale Informationsaustausch zwischen den verschiedenen Disziplinen schwierig.

Solch ein Problem besteht insbesondere beim sehr interdisziplinären Bauwesen. Laut einer Studie der Universität München wird „im Laufe der Planung bis zur Ausführung und Abrechnung heute jedes Bauteil mindesten sechsmal neu in eine Datenverarbeitung eingegeben“ [IAI1]. Jeder Beteiligte entfernt für ihn nicht relevante Informationen, passt die verbliebenen Informationen an seine Bedürfnisse an, fügt als Ergebnis seiner Arbeit neue Informationen hinzu und leitet gegebenenfalls den Teil der geometrisch darstellbaren Informationen an den nächsten Beteiligten weiter [BITT]. Auf diese Weise entsteht während der Bearbeitung eines Bauprojektes eine Vielzahl von häufig redundanten, inkonsistenten Informationen, was nicht zuletzt ein großes Fehlerrisiko birgt und Informationsverlust mit sich bringt.

Als Lösungsansatz ist nur das Produktmodell anzusehen. Es enthält nicht nur die Daten zum Design und zur Gestaltung eines Gebäudes, sondern vielmehr alle Informationen, die während des Entstehungsprozesses und des weiteren Lebenslaufs anfallen. Dadurch können die relevanten Informationen allen Beteiligten zur Verfügung gestellt werden. Einen Vergleich zwischen dem konventionellen Informationsaustausch und dem Produktmodell ist in Bild 2.1 dargestellt.

Grundsätzlich unterscheidet die Informatik zwischen den Begriffen „Produktmodell“ und „Produktdatenmodell“. Ein **Produktmodell** ist die formale Beschreibung aller Eigenschaften zu einem Produkt (z.B. Bauwerk) über alle Phasen des Lebenszyklus (Planung, Entstehung, Nutzung, Rückbau, Entsorgung) hinweg. Ein **Produktdatenmodell** dient dabei als Schema, durch das festgelegt wird, wie die Daten des zu beschreibenden Produkts aufgebaut sein müssen und welche Beziehung sie zueinander haben sollen. Es beschreibt den Aufbau der Datenstruktur. Wird ein Produktdatenmodell mit den Daten eines konkreten Bauwerks (Produkt) gefüllt, so entsteht das Produktmodell. Das Produktmodell kann damit als eine Instanz des Produktdatenmodells angesehen werden [RANK][NEUB]. Im Allgemeinen wird in der Praxis auf die Unterscheidung verzichtet.

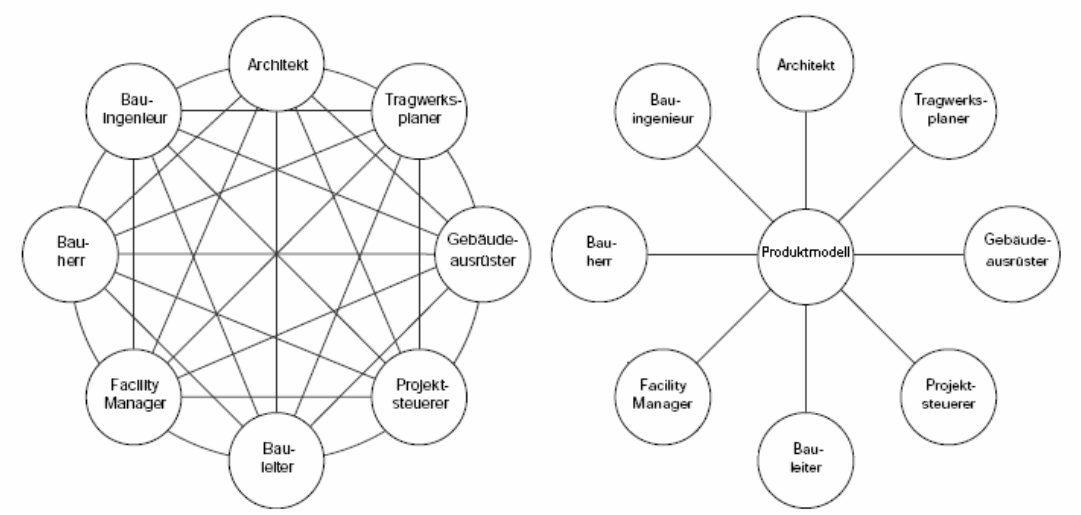


Bild 2.1: Vergleich von konventionellem Informationsaustausch und Produktmodell

Im Bauwesen gibt es zwei Initiativen, STEP (Standard for the Exchange of Product Model Data) und IAI (International Alliance for Interoperability), die den Datenaustausch und die Datenintegration zwischen unterschiedlichen Datenverarbeitungssystemen mit Hilfe des Produktdatenmodells nachhaltig verbessern und auf die heute eingesetzten Systeme bringen sollen. Beide Initiativen sind international organisiert und haben bezogen auf das Bauwesen praktisch identische Ziele [HAAS].

Vor der Diskussion der Produktdatenmodelle im Bauwesen sollen die wichtigsten geometrischen Modelle, die sich im Zuge der Produktmodellierung wiederfinden, im Folgenden kurz erläutert werden.

2.1 Geometrische Modellierung

CAD Programme dienen in erster Linie der interaktiven Konstruktion, Manipulation und der bildlichen Darstellung dreidimensionaler geometrischer Objekte für Anwendungen wie z.B. der Bauwerksgeometrie. Es ist nicht trivial, ein geometrisches Gebilde durch eine Datenstruktur in einem Rechner digital abzubilden. Die Entscheidung für ein Geometriemodell wird entscheidend durch den intendierten Verwendungszweck des Modells mitbestimmt.

Im Folgenden wird ein kurzer Überblick über die wichtigsten Arten möglicher Datenstrukturen zur geometrischen Modellierung gegeben.

2.1.1 Drahtmodelle (Wireframe Models)

Das Drahtmodell (wire frame) ist das bekannteste und zugleich älteste computergestützte Modell zur Repräsentation der Geometrie. Ein Drahtmodell besteht aus einer Liste von Kanten, die jeweils durch zwei Punkte im kartesischen Koordinatensystem beschrieben werden.

Es wird oft in zweidimensionalen Zeichnungen angewandt. Aufgrund fehlender Flächeninformationen sind bestimmte Aussagen und Berechnungen, wie die Sichtbarkeit von Körperkanten oder die Verschneidung von Körpern nicht möglich. Das Drahtmodell benötigt ein geringes Datenvolumen und beansprucht nur geringe Rechnerleistung. Allerdings ist es wegen der Beschränkungen nicht allein für einen Einsatz in CAD-Systemen tauglich [NEUB].

2.1.2 Flächenmodelle (Surface Models)

Das Flächenmodell (surface model) zeigt ausschließlich die Konstruktionsmerkmale der Körperoberfläche. Dabei werden Objekte durch approximierte oder analytische Flächen, z.B. durch eine Liste von miteinander verbundenen Polygonen, repräsentiert. Ein solches Polygon wird durch seine Eckpunkte beschrieben, die durch Kanten verbunden sind (siehe Bild 2.2). Ein Flächenmodell erweitert das Drahtmodell. Gegenüber dem Drahtmodell steigt die Rechenzeit. Ein Flächenmodell beschreibt Oberflächenstrukturen, aber keine Körper.

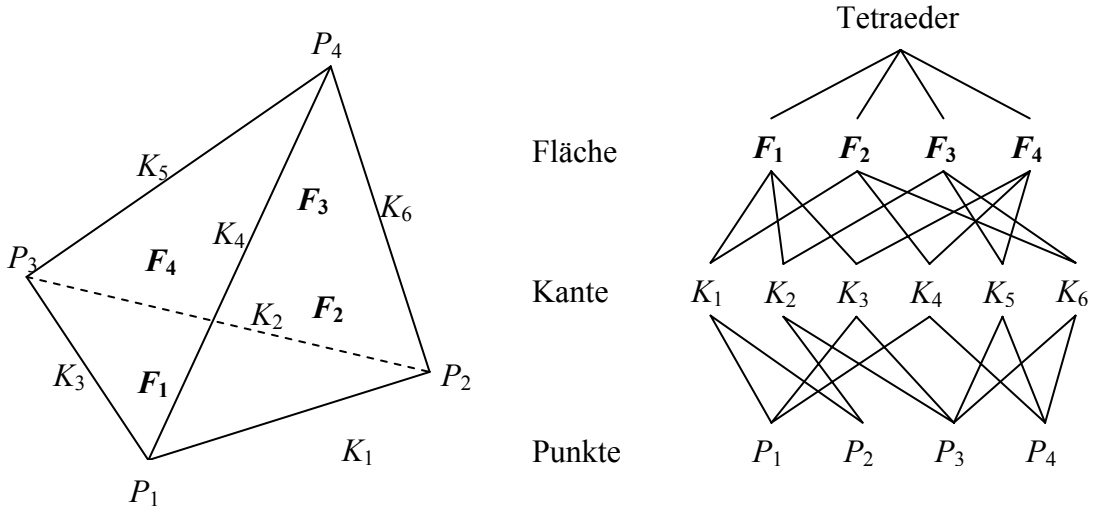


Bild 2.2: Datenstruktur eines Flächenmodells

2.1.3 Volumenmodelle (Solid Models)

Volumenmodell *(solid model):* Hiermit wird eine vollständige Beschreibung der Volumina durchgeführt. Erst mit einem Volumenmodell kann eine CIM (Computer Integrated Manufacturing) Lösung, wie die NC- oder robotergestützte Fertigung realisiert werden. Bei dem Volumenmodell existieren drei Möglichkeiten:

- Das flächenorientierte Volumenmodell - Boundary Representation (**B-Rep**): Beim B-Rep-Modell werden die Oberflächen definiert. Die Oberflächen sind miteinander verknüpft und umschließen das Volumen (Bild 2.3). Im Gegensatz zum Flächenmodell ist der von den Flächen umhüllte Raum materialerfüllt. Die Grundobjekte, Ecken (verti-

ces), Kanten (edges) und Facetten (faces), sowie ihre Beziehungen werden im B-Rep-Modell gespeichert.

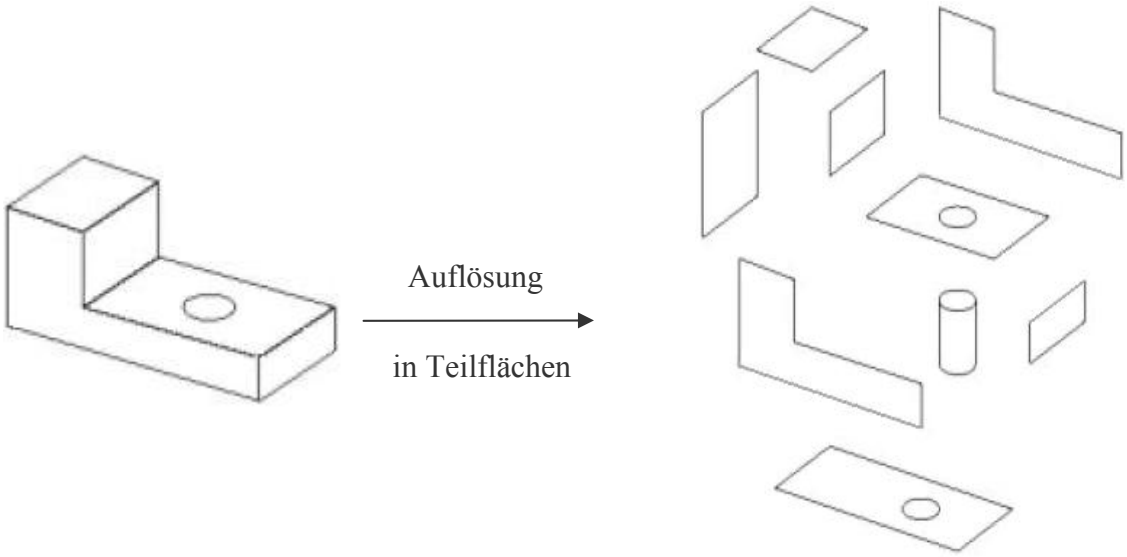


Bild 2.3: Beispiel eines B-Rep-Modells

- **Sweep**-Körper Modell: Sweep-Elemente entstehen, wenn ein Punkt, eine Kurve oder eine Oberfläche (bezeichnet als „generator“) entlang eines bestimmten Pfades verschoben oder um eine Achse gedreht wird. Zur Ausführung des Sweeps ist weiterhin eine analytisch beschreibbare Bahn erforderlich. Diese wird als „director“ bezeichnet. 3D-Sweeps werden verwendet, um z.B. einen Wandkörper zu bilden.
- Das körperorientierte Volumenmodell – Constructive Solid Geometry(**CSG**): Beim CSG-Modell wird ein Objekt durch Erzeugung von Primitivkörpern und Anwendung boolescher Mengenoperation repräsentiert. Zu den Primitivkörpern gehören Quader, Kugel, Prisma, Zylinder usw. Die booleschen Mengenoperationen sind Vereinigung (union), Differenz (subtraction) und Durchschnitt (common). Der Konstruktionsvorgang, der zur Herstellung einer komplexen Struktur führt, kann im CSG-Modell in eine Reihe von Konstruktionsvorgängen, bei denen nur je zwei Teilstrukturen beteiligt sind, zerlegt werden. Damit ist die Darstellung einer Struktur in einem CSG-Modell ein Binärbaum. Die Wurzel stellt das Bauteil dar, die Knoten repräsentieren die Mengenoperationen und die Blätter die Primitivvolumina. Ein CSG-Modell kann stets in ein B-Rep-Modell konvertiert werden. Daneben existieren auch einige Probleme beim CSG-Modell. Wesentlich sind die Mehrdeutigkeiten (siehe Bild 2.4) und die Konstruktionsbeschränkung durch die Verwendung von Grundkörpern und Booleschen Operationen. Außerdem kann die Ermittlung der Informationen über Randflächen, deren Kanten (edges) und Eckpunkte (vertices) relativ aufwändig sein.

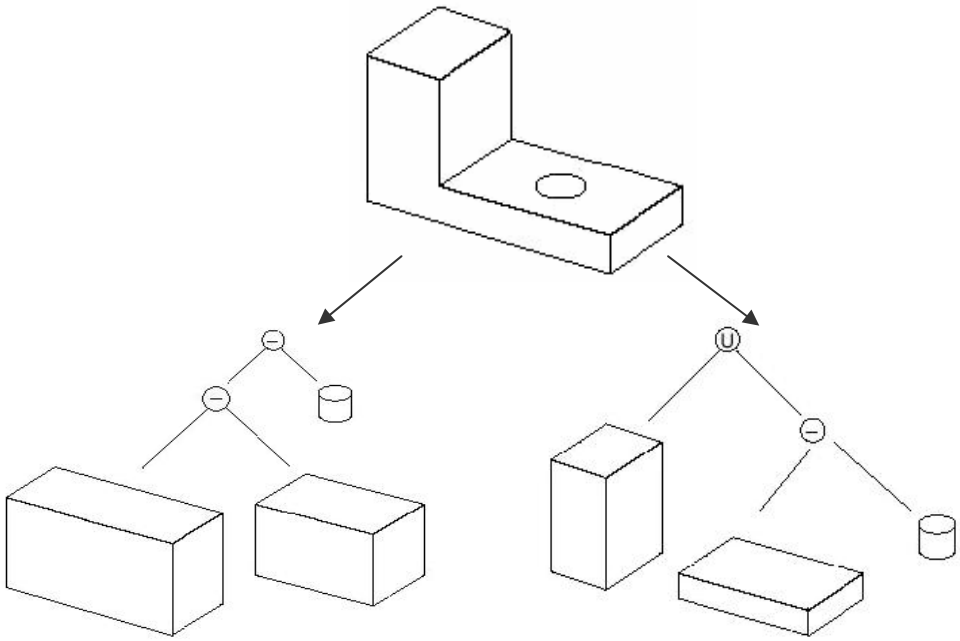


Bild 2.4: Beispiel eines CSG-Modells

2.2 STEP - ein integriertes Produktdatenmodell

STEP ist die informelle Bezeichnung der ISO 10303 Familie internationaler Standards für den Produktdatenaustausch. Es ist eine rechner-interpretierbare Definition von physikalischen und funktionalen Merkmalen eines Produktes während seines kompletten Lebenszyklus. Neben den Geometriedaten werden alle relevanten Daten eines Produktes, wie z.B. die Materialien, Zusammenbaustrukturen, administrative Daten usw. in einem integrierten Produktdatenmodell beschrieben. Damit ermöglicht STEP einen redundanzfreien und zuverlässigen Austausch und die gemeinsame Nutzung von Daten zwischen allen während des Produktlebenszyklusses beteiligten Partnern. STEP hat sich in vielen Anwendungen durchgesetzt wie beispielsweise in CAD, Prozessplanung, CAM, Produktdatenmanagement (PDM), Systems Engineering usw. [PROS]. Unter Verwendung von STEP-Grundbausteinen, den Modellen zur Beschreibung von Produktdaten (Integrated Information Resource), wurden viele so genannte „Applikationsprotokolle“ (Application Protocols – APs) nach definierten und genormten Methoden (Beschreibungsmethoden, Implementierungsmethoden usw.) entwickelt. Die Applikationsprotokolle decken viele Branchen wie den Maschinenbau, den Schiffsbau, die Luft- und Raumfahrtindustrie, den Automobilbau, den Anlagenbau und eben auch das Bauwesen ab. Für das Bauwesen ist vor allem das Applikationsprotokoll AP225 mit dem Titel „Building Elements Using Explicit Shape Representation“ maßgebend. Es erfasst sämtliche Informationen zur vollständigen Darstellung komplexer räumlicher Gebäudemodelle vom Rohbau bis zum schlüsselfertigen Gebäude. Darüber hinaus bietet STEP drei weitere Protokolle speziell für

den Stahlbau (AP230), für Heizung, Lüftung und Klima (AP228) und für die Fabrikplanung (AP227) [ANDL].

Wegen des langwierigen zeitaufwendigen Normierungsprozesses und starker Ausrichtung auf den Maschinenbau und die Automobilindustrie werden aber für die oben genannten STEP-Protokolle im Gegensatz zu IFC fast keine Prozessoren von den großen CAAD-Anbietern zur Verfügung gestellt (vgl. [RANK]). Das einzige zurzeit in der Praxis angewandte Format der ISO 10303 für das Bauwesen ist STEP-CDS, das bei der Fabrikplanung in der Automobilindustrie eingesetzt wird. Es beschränkt sich allerdings bislang auf reine 2D-Datenverarbeitung.

2.3 IFC-Produktdatenmodell der IAI

Um einen problemlosen Datenaustausch zwischen allen Beteiligten am Bauprozess (z.B. Architekten, Ingenieuren, Baufirmen, Bauherren, …) über die Grenzen der Fachdisziplin hinweg zu ermöglichen, wurde von der International Alliance for Interoperability (IAI) ein Produktdatenmodell für das Bauwesen, das IFC-Produktdatenmodell (Industry Foundation Classes), definiert. Die IAI wurde 1995 in den USA gegründet und entwickelte sich zu einer weltweiten Organisation mit heute über 446 Mitgliedern in elf Gruppen (so genannte „Chapters“) aus 24 Mitgliedsländern. Für den Datenaustausch wird eine ähnliche Syntax wie bei STEP verwendet und die Objekte werden ebenfalls mit der EXPRESS-Sprache beschrieben. EXPRESS spezifiziert Informationsobjekte durch so genannte „Entitäten“ (Entities). Entitäten sind Klassen von Objekten mit gleichen Eigenschaften, die durch Attribute und Bedingungen definiert werden. Im Gegensatz zu STEP ist IFC nicht für eine Norm vorgesehen, was sich positiv auf die Entwicklungszyklen auswirkt und direkt auf die Anforderung der Industrie zielt. Inzwischen ist IFC auch als internationaler Standard unter ISO/PAS 16739 von der ISO angenommen. Die aktuelle Modell-Version ist die Ausgabe IFC-2x3 von 2006.

Die zentrale Anforderung an die Entwicklung des IFC Produktdatenmodells war, eine gemeinsame Nutzung von Informationen für verschiedene Fachbereiche zu ermöglichen. Wegen der inhomogenen Struktur des Bauwesens hat das IFC Objektmodell eine modulare und einfach erweiterbare Architektur, die in vier als Layer bezeichnete Ebenen gegliedert ist, deren wesentliches Merkmal eine hierarchische Klassenstruktur darstellt (Bild 2.5). Eine Klasse darf nur Klassen derselben oder einer untergeordneten Ebene referenzieren. Referenzen auf eine höhere Ebene sind nicht zulässig. Um die Modularität zu gewährleisten, sind Referenzen innerhalb einer Ebene sorgfältig zu prüfen [GEIG][IFC2].

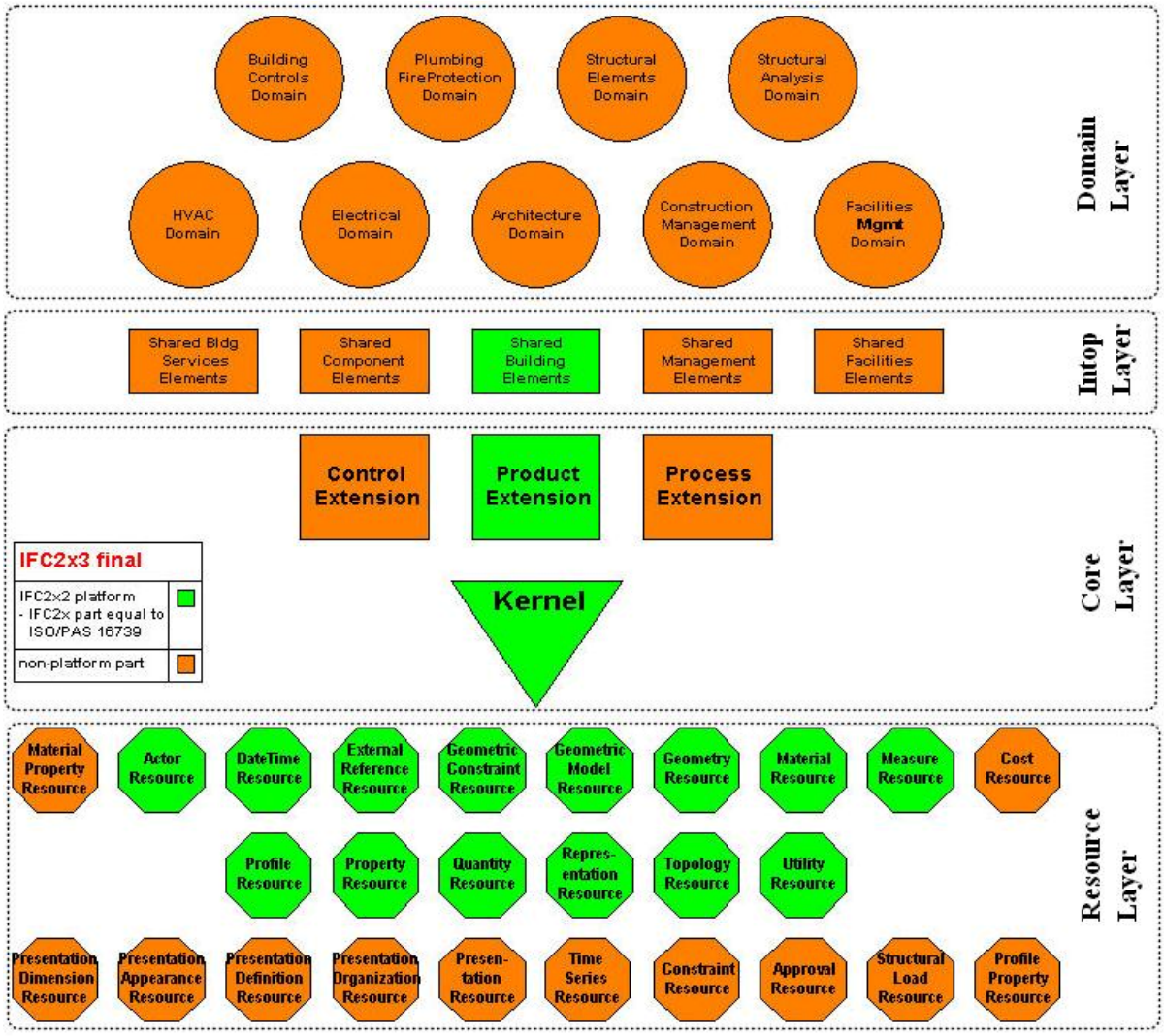


Bild 2.5: Architektur des IFC-Modells [IFC1]

Resource Layer

Die unterste Ebene des IFC-Modells kann von Klassen aller anderen Ebenen referenziert werden. Sie beinhaltet allgemeingültige Klassen, die ein in sich abgeschlossenes Konzept repräsentieren und unabhängig von den anderen Klassen des IFC-Modells sind. Ausnahmen stellen lediglich die Klassen Utility und Measure dar, die von anderen Klassen des Resource Layers referenziert werden. Eine der am häufigsten verwendeten Ressourcen ist die Geometrie, in der alle implizite und explizite Geometrie zusammengefasst ist. Bevor ein Objekt Geometrie definiert werden kann, muss es zuerst aus dem Domain Layer heraus definiert werden [IFC3].

Core Layer

Die nächst höhere Ebene bildet der Core Layer. Er definiert die grundlegende Struktur des IFC-Modells und beinhaltet zumeist abstrakte Konzepte, die in den darüberliegenden Ebenen spezialisiert werden. Der Core Layer hat zwei Abstraktionsebenen, den Kernel und die Core

Extension. Im Kernel werden die grundlegenden Konzepte über die Verfügbarkeit von Objekten, deren Beziehungen, Typendefinitionen und Attribute definiert. Er dient als Plattform für Modellerweiterungen, ist aber nicht bauwesensspezifisch. Die Core Extensions stellen Spezialisierungen der im Kernel definierten Klassen dar und passen die Elemente an die Erfordernisse des Bauwesens an. Die „Product Extension“ beinhaltet beispielsweise Rohbauelemente wie Decken, Wände, Stützen oder Balken [IFC3].

Interoperability Layer

Im „Interoperability Layer“ werden Konzepte bzw. Objekte definiert, die mehreren Domain Modellen zugeordnet werden können. Zusätzlich werden hier die abstrakten Konzepte des Core Layers spezialisiert. Diese Ebene verfügt weiterhin noch über so genannte Adapter. Sie erweitern die Konzepte des Core Layers und bilden damit die Grundlage für die speziellen Domain Models. So werden beispielsweise die grundlegenden Eigenschaften der Wände in den „Shared Building Elements“ definiert [IFC3].

Domain Layer

Die oberste Ebene ist speziell auf die Bedürfnisse eines Fachbereiches angepasst. Der Domain Layer verfügt über Module für Architektur, Bauingenieurwesen und Facility Management, aber auch für Elektrik, Heizung, Lüftung und Sanitär etc. [IFC3].

Bild 2.6 zeigt ein informelles UML-Modell, das die Grundzüge des IFC-Gebäudemodells darstellt und sich auf die in diesem Zusammenhang interessierenden IFC-Objekte beschränkt. Ein IFC-Modell wird durch ein Projekt (*IfcProject*) repräsentiert, das durch Raumstrukturelemente (*IfcSpatialStructureElement*) wie Bauvorhabenstandorte (*IfcSite*), Gebäude (*IfcBuilding*), Stockwerke (*IfcBuildingStorey*) und Räume (*IfcSpace*) gebildet wird. Alle die räumlichen Elemente sind IFC-Produkte (*IfcProduct*). Jedem Element kann so eine Geometrie (*IfcShapeRepresentation*) und ein Ort (*IfcLocalPlacement*) zugewiesen werden. Darüber hinaus können den Elementen der räumlichen Struktur die eigentlichen Bauelemente (*IfcBuildingElement*) zugeordnet werden. Solche Bauelemente sind beispielsweise Wände (*IfcWall*), Türen (*IfcDoor*), Fenster (*IfcWindow*), etc. Wände können Öffnungen (*IfcOpeningElement*) haben, in welche Fenster und Türen eingesetzt werden. Versorgungsanschlüsse wie beispielsweise elektrische und sanitäre Anschlüsse werden durch den Objekttyp „*IfcFlowTerminal*“ modelliert, der einem räumlichen Strukturelement zugeordnet wird [BENN].

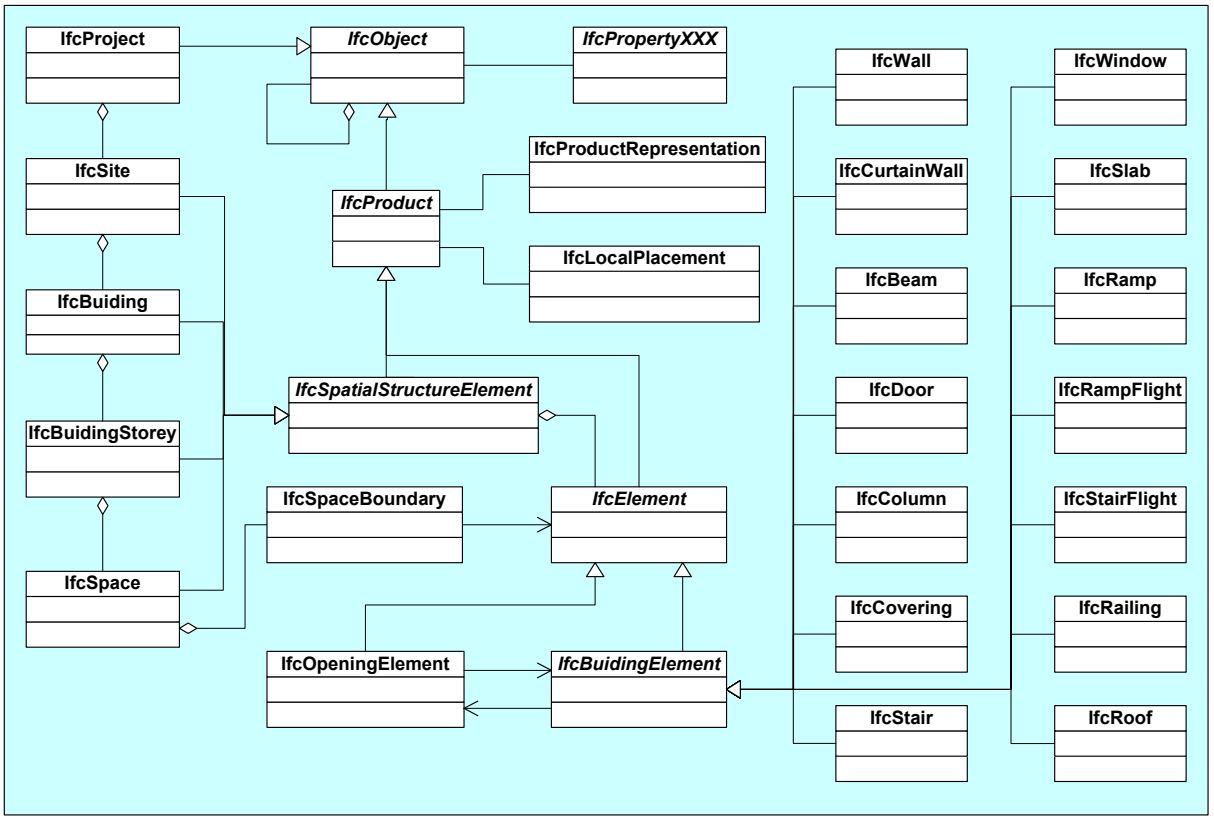


Bild 2.6: Informelles IFC-Gebäudemodell in UML-Notation [BENN]

2.4 Analyse geometrischer Repräsentationen der Objekte in IFC

In der Arbeit sind ausschließlich die Geometrie von Wänden, Wandschrägen und Wandöffnungen sowie ihre topologischen Beziehungen interessant.

Alle physikalischen Objekte sind abgeleitete Unterklassen vom Typ „*IfcProduct*“, der eine einzige (z.B. eine Boundingbox) oder mehrere geometrische Repräsentationen (z.B. B-Rep-Modell) hat. Die geometrischen Repräsentationen werden in demselben Koordinatensystem lokalisiert, welches durch die IFC-Entität „*IfcLocalPlacement*“ oder „*IfcGridPlacement*“ dargestellt wird. Wegen der Vererbung ist die Methode der geometrischen Repräsentationen für alle Wände, Stürze, Öffnungen usw. gleich. Die in der Praxis kaum verwendete Entität „*IfcGridPlacement*“ definiert ein Koordinatensystem mit Hilfe eines Konstruktionsnetzes (design grid). Da in der Implementierung der Methodik zur Manipulation und Extraktion von IFC-Daten (siehe Kapitel 4) immer die Entität „*IfcLocalPlacement*“ und deren Transformationen von zentraler Bedeutung sind, wird sie hier genauer analysiert.

2.4.1 Lagebeschreibung

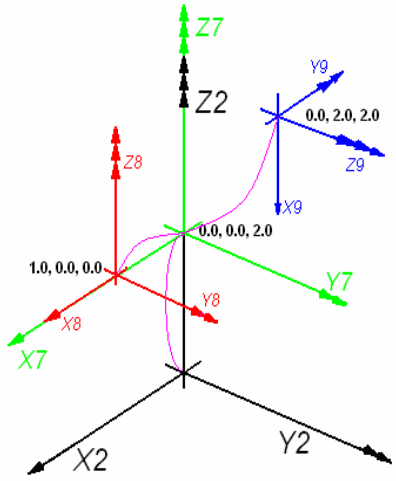
Mit der Entität „*IfcLocalPlacement*“ kann entweder die relative Lage eines Objektes (z.B. Wand, Öffnung) zu einem anderen oder die absolute Lage bezüglich des globalen Koordina-

tensystems, das mit der Entität „*IfcGeometricRepresentationContext*“ angegeben wird, beschrieben werden. Eine durch „*PlacementRelTo*“ referenzierte Entität „*IfcProduct*“ ist innerhalb des lokalen Koordinatensystems einer anderen Entität „*IfcProduct*“ angeordnet.

Für die relative Platzierung sind folgende Vereinbarungen getroffen worden [IFC3]:

- „*IfcSite*“ wird absolut innerhalb des Weltkoordinatensystems platziert.
- „*IfcBuilding*“ wird im Verhältnis zum Koordinatensystem vom „*IfcSite*“ angeordnet,
- „*IfcBuildingStorey*“ wird im Verhältnis zum Koordinatensystem des Typs „*IfcBuilding*“ angeordnet.
- „*IfcElement*“ wird angeordnet relativ zum Koordinatensystem entweder seiner Container-Entität (*IfcSite*, *IfcBuilding*, *IfcBuildingStorey*) oder der Entität „*IfcElement*“, an die es gebunden ist (*IfcRelVoidsElement*, *IfcRelFillsElement*, *IfcRelCoversBldgElements*, *IfcRelAssemblesElements*).

Jedes lokale Placement kann 2D (*IfcAxis2Placement2D*) oder 3D (*IfcAxis2Placement3D*) sein. Es wird durch den Ursprung des lokalen Koordinatensystems, die Z-Achse (die Y-Achse bei 2D) und eine Richtung auf der XZ-Ebene (3D) oder die X-Achsen Richtung (2D) definiert. In Tabelle 2.1 wird anhand eines Beispiels die Definition vom Typ „*IfcLocalPlacement*“ erläutert. Links ist die graphische Darstellung und rechts ist die Beschreibung im IFC.




```
#1=IFCGEOMETRICREPRESENTATIONCONTEXT($,,3Dmodel',3,1.0E-
005, #2, $);
#2=IFCAXIS2PLACEMENT3D(#3, $, $);
#3=IFCCARTESIANPOINT((0.0, 0.0, 0.0));

#4=IFCLOCALPLACEMENT($, #7);
#7=IFCAXIS2PLACEMENT3D(#10, $, $);
#10=IFCCARTESIANPOINT((0.0, 0.0, 2.0));

#5=IFCLOCALPLACEMENT(#4, #8);
#8=IFCAXIS2PLACEMENT3D(#11, $, $);
#11=IFCCARTESIANPOINT((1.0, 0.0, 0.0));

#6=IFCLOCALPLACEMENT(#4, #9);
#9=IFCAXIS2PLACEMENT3D(#12, #13, #14);
#12=IFCCARTESIANPOINT((0.0, 2.0, 2.0));
#13=IFCDIRECTION((0.0, 1.0, 0.0));
#14=IFCDIRECTION((0.0, 0.0, -1.0));
```

Tabelle 2.1: Beispiel der Entität „*IfcLocalPlacement*“

Die Zeilen mit Nummern 1, 2, 3 in Tabelle 2.1 definieren gemeinsam das einzige globale Koordinatensystem eines Bauprojektes $\boldsymbol{K_2}$ (der Index drückt hier die Nummerierung der Achsen in der Grafik aus.), auf das das $\boldsymbol{K_7}$ basiert. Die Koordinatensysteme $\boldsymbol{K_8}$ und $\boldsymbol{K_9}$ sind wiederum bezüglich $\boldsymbol{K_7}$ lokalisiert.

2.4.2 Beschreibung der Wand

Eine Wand ist in IFC als Instanz vom Typ „*IfcWall*“ oder „*IfcWallStandardCase*“ beschrieben. Die Entität „*IfcWallStandardCase*“ (Tabelle 2.2) bildet die Grundlage dieser Arbeit. Sie ist ein Untertyp der Entität „*IfcWall*“ und dient der Beschreibung von Wänden, die entlang ihrer Achse eine konstante Wandstärke besitzen. Die Entität „*IfcWall*“ beschreibt die anderen Fälle und nutzt für die Darstellung der Geometrie das B-Rep-Modell.

ENTITY IfcWallStandardCase;
ENTITY IfcRoot;
GlobalId : IfcGloballyUniqueId;
OwnerHistory : IfcOwnerHistory;
Name : **OPTIONAL** IfcLabel;
Description : **OPTIONAL** IfcText;
ENTITY IfcObject;
ObjectType : **OPTIONAL** IfcLabel;
INVERSE
IsDefinedBy : **SET OF** IfcRelDefines **FOR** RelatedObjects;
HasAssociations : **SET OF** IfcRelAssociates **FOR** RelatedObjects;
HasAssignments : **SET OF** IfcRelAssigns **FOR** RelatedObjects;
Decomposes : **SET OF** IfcRelDecomposes **FOR** RelatedObjects;
IsDecomposedBy : **SET** [0:1] **OF** IfcRelDecomposes **FOR** RelatingObject;
ENTITY IfcProduct;
ObjectPlacement : **OPTIONAL** IfcObjectPlacement;
Representation : **OPTIONAL** IfcProductRepresentation;
INVERSE
ReferencedBy : **SET OF** IfcRelAssignsToProduct **FOR** RelatingProduct;
ENTITY IfcElement;
Tag : **OPTIONAL** IfcIdentifier ;
INVERSE
ConnectedTo : **SET OF** IfcRelConnectsElements **FOR** RelatingElement;
ConnectedFrom : **SET OF** IfcRelConnectsElements **FOR** RelatedElement;
ContainedInStructure : **SET** [0:1] **OF** IfcRelContainedInSpatialStructure **FOR** RelatedElements;
ENTITY IfcBuildingElement;
INVERSE
ProvidesBoundaries : **SET OF** IfcRelSpaceBoundary **FOR** RelatedBuildingElement;
HasOpenings : **SET OF** IfcRelVoidsElement **FOR** RelatingBuildingElement;
FillsVoids : **SET OF** IfcRelFillsElement **FOR** RelatedBuildingElement;
ENTITY IfcWall;
ENTITY IfcWallStandardCase;
END_ENTITY;

Tabelle 2.2: Klassenhierarchie der Entität „*IfcWallStandardCase*“ [IFC1]

Die geometrische Darstellung einer Wand vom Typ „*IfcWallStandardCase*“ hat zwei Bestandteile: eine Führungslinie (*WallAxis*) und einen Wandkörper (*WallBody*). Je nach CAAD-System wird die Geometrie der Boundingbox ebenfalls mit in die IFC-Datei geschrieben. Die Führungslinie wird aus einer zweidimensionalen Kurve (*IfcBoundedCurve*) gebildet. Die Kurve kann zum Beispiel ein Polygon oder ein Kreisbogen sein. Der Wandkörper muss ein Extrusionskörper oder ein CSG-Modell für den Fall mit Wandschrägen (vgl. Abs. 2.4.4) sein. Folgende geometrische Parameter sollen jeder Entität „*IfcWallStandardCase*“ gegeben werden: die Wandhöhe (Extrusionslänge) und das Grundrissprofil, die Wandführungslinie, die Wandstärke und ein Wandversatz bezüglich der Führungslinie aus der Entität „*IfcMateria-*

LayerSetUsage“. Bild 2.7 zeigt die geometrische Repräsentation der Wandachse und des Wandkörpers einer geraden Wand.

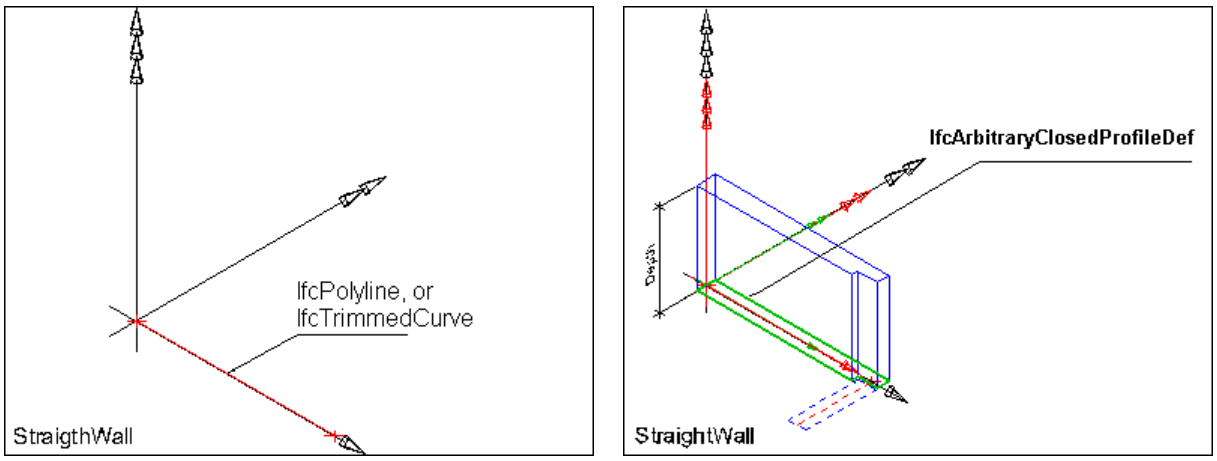


Bild 2.7: Geometrische Repräsentation der Wand (links: Wandachse, rechts: Wandkörper) [IFC3]

2.4.3 Beschreibung der Wandverbindung

In der Entität „*IfcRelConnectsPathElements*“ (Tabelle 2.3.) werden die Informationen der Wandverbindung beschrieben. Dabei werden die beiden verbundenen Wände (*RelatingElement* und *RelatedElement*) und die Verbindungsstelle entlang einer Wand (z.B. *AtStart*, *AtPath*, *AtEnd* oder *NotDefined*) genannt. Eine Geometrie für die Verbindung ist optional, wobei es sich um eine Polylinie handelt, die entlang der Berührungspunkte zweier Wände im Grundriss verläuft.

ENTITY IfcRelConnectsPathElements;	
ENTITY IfcRoot;	
GlobalId	: IfcGloballyUniqueId;
OwnerHistory	: IfcOwnerHistory;
Name	: **OPTIONAL** IfcLabel;
Description	: **OPTIONAL** IfcText;
ENTITY IfcRelConnectsElements;	
ConnectionGeometry	: **OPTIONAL** IfcConnectionGeometry;
RelatingElement	: IfcElement;
RelatedElement	: IfcElement;
ENTITY IfcRelConnectsPathElements;	
RelatingPriorities	: **LIST OF INTEGER**;
RelatedPriorities	: **LIST OF INTEGER**;
RelatedConnectionType	: IfcConnectionTypeEnum;
RelatingConnectionType	: IfcConnectionTypeEnum;
DERIVE	
RelatedLayerCount	: **INTEGER** := IfcNoOfLayers(SELF\IfcRelConnectsElements.RelatedElement);
RelatingLayerCount	: **INTEGER** := IfcNoOfLayers(SELF\IfcRelConnectsElements.RelatingElement);
END_ENTITY;	

Tabelle 2.3: Klassenhierarchie der Entität „*IfcRelConnectsPathElements*“ [IFC1]

In Tabelle 2.4 werden zwei Beispiele der Wandverbindung dargestellt. Das linke Beispiel definiert mit dem IFC-Text #120 eine Verbindung zweier Wände #59 und #111 sowie die Verbindungsstelle entlang der Wände: Ende der Wand #59 (*AtEnd*) und Anfang der Wand #111 (*AtStart*).

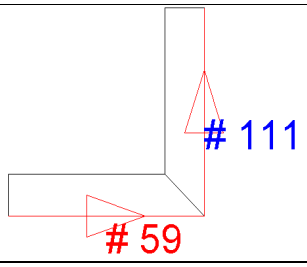
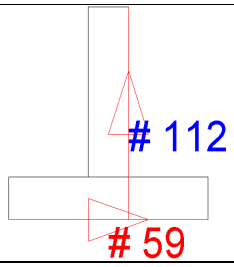
	
#120=IFCRELCONNECTSPATHELEMENTS('3sy9abyIPBPeJ0pjBDIsUF',#6,$,$,#119,#111,#59,(),().ATEND.,.ATSTART.);	#118=IFCRELCONNECTSPATHELEMENTS('1fge0DsNj4GhWUpEqoac54',#6,$,$,#117,#112,#59,(),().ATPATH.,.ATSTART.);

Tabelle 2.4: Wandverbindung im IFC

2.4.4 Beschreibung der Wandschrägen

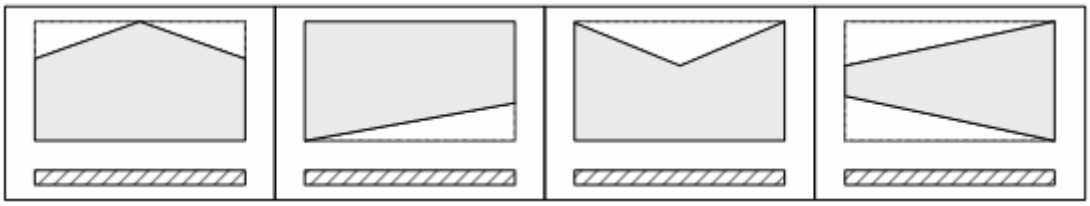


Bild 2.8: Wand mit Schräge (Vorderansicht und Grundriss, Quelle: [IFC1])

Die in Bild 2.8 dargestellten Wände mit Schrägen finden beispielsweise bei den Giebelwänden Anwendung. Die beschnittene Wand wird mit der Entität „*IfcBooleanClippingResult*“ (Tabelle 2.5) beschrieben, die ein CSG-Modell geometrisch repräsentiert. Der „*FirstOperand*“ ist ein Solidmodell und verweist auf das Wandelement, der „*SecondOperand*“ verweist auf die „*IfcHalfSpaceSolid*“, welche einen Halbraum an einer Seite der Verschneidungsebene definiert. Der boolesche Operator ist immer DIFFERENCE. Ein Wandkörper wird vollständig mit einer Ebene verschnitten. Es wird angegeben, welche Seite der Schnittebene ausgeblendet wird. Eine Wand darf durch die Entität „*IfcBooleanClippingResult*“ nicht in ihrer Länge gekürzt werden.

```
ENTITY IfcBooleanClippingResult;
  ENTITY IfcRepresentationItem;
  ENTITY IfcGeometricRepresentationItem;
  ENTITY IfcBooleanResult;
    Operator                : IfcBooleanOperator;
    FirstOperand            : IfcBooleanOperand;
    SecondOperand           : IfcBooleanOperand;
  DERIVE
    Dim                     : IfcDimensionCount := FirstOperand.Dim;
  ENTITY IfcBooleanClippingResult;
END_ENTITY;
```

Tabelle 2.5: Klassenhierarchie der Entität „*IfcBooleanClippingResult*“ [IFC1]

Im Folgenden wird anhand eines Beispiels (Bild 2.9) erläutet, wie eine Wand mit einer Schräge in IFC definiert ist.

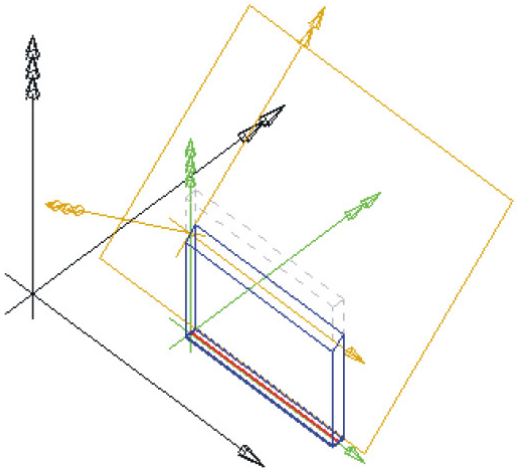


Bild 2.9: Beispiel einer Wandschräge im IFC

Zuerst wird die Wand vom Typ „*IfcWallStandardCase*“ durch eine Placementdefinition (#3) und eine geometrische Repräsentation (#4) festgelegt. Die geometrische Repräsentation (#4) beinhaltet eine Führungslinie (#11) und einen Wandkörper (#13).

```
#1=IFCWALLSTANDARDCASE('abcdefghijklmnopqrst01', #2, $, $, $, #3, #4, $);
#3=IFCLOCALPLACEMENT($, #10);
#4=IFCPRODUCTDEFINITIONSHAPE($, $, (#11, #13));
#10=IFCAXIS2PLACEMENT3D(#16, $, $);
#16=IFCCARTESIANPOINT((2.0E+00, 1.0E+00, 0.0E+00));
#12=IFCGEOMETRICREPRESENTATIONCONTEXT($, $, 3, $, #14, $);
#14=IFCAXIS2PLACEMENT3D(#15, $, $);
#15=IFCCARTESIANPOINT((0.0E+00, 0.0E+00, 0.0E+00));
/* geometrische Repräsentation der Wandachse */
#11=IFCSHAPEREPRESENTATION(#12, 'Axis', 'Curve2D', (#18));
#18=IFCTRIMMEDCURVE(#19,(#20),(#21),.T.,.CARTESIAN.);
#19=IFCLINE(#30, #31);
#30=IFCCARTESIANPOINT((0.0E+00, 0.0E+00));
#31=IFCVECTOR(#32,2.8E+00);
#32=IFCDIRECTION((1.0E+00, 0.0E+00));
#20=IFCCARTESIANPOINT((0.0E+00, 0.0E+00));
#21=IFCCARTESIANPOINT((2.80E+00, 0.0E+00));
/* geometrische Repräsentation des abgeschnitten Körpers */
#13=IFCSHAPEREPRESENTATION(#12, 'Body', 'Clipping', (#50));
#50=IFCBOOLEANCLIPPINGRESULT(.DIFFERENCE., #22, #51);
```

Zeile #13 verweist darauf, dass der Wandkörper durch eine „Clipping“ abgeschnitten wird. Zeile #50 fasst die boolesche Operation und die beiden Operatoren – Extrusionskörper und Schräge zusammen. Von Zeile #22 bis #44 wird der erste Operator (der Extrusionskörper) definiert. Das Teil (#51-56) definiert den zweiten Operator (den Halbraum). In Zeile #51 wird festgelegt, auf welcher Seite der Ebene sich der Halbraum befindet.

```
/* geometrische Repräsentation des Extrusionskörpers */
#22=IFCEXTRUDEDAREASOLID(#23, #26, #29, 2.80E+00);
#26=IFCAXIS2PLACEMENT3D(#28, $, $);
#28=IFCCARTESIANPOINT((0.0E+00, 0.0E+00, 0.0E+00));
#29=IFCDIRECTION((0.0E+00, 0.0E+00, 1.0E+00));
#23=IFCARBITRARYCLOSEDPROFILEDEF(.AREA., $, #40);
#40=IFCPOLYLINE((#41,#42,#43,#44,#41));
#41=IFCCARTESIANPOINT((0.0E+00, 1.0E-01));
#42=IFCCARTESIANPOINT((2.8E+00, 1.0E-01));
#43=IFCCARTESIANPOINT((2.8E+00, -1.0E-01));
```

```
#44=IFCCARTESIANPOINT((0.0E+00, -1.0E-01));
/* geometrische Repräsentation der Schräge */
#51=IFCHALFSPACESOLID(#52, .F.);
#52=IFCPLANE(#53);
#53=IFCAXIS2PLACEMENT3D(#54, #55, #56);
#54=IFCCARTESIANPOINT((0.0E+00, 0.0E+00, 2.0E+00));
#55=IFCDIRECTION((0.0E+00, -0.7070106E+00, 0.7070106E+00));
```

2.4.5 Beschreibung der Wandöffnungen

Der Typ „*IfcOpeningElement*“ (Tabelle 2.6) steht für alle Arten von Öffnungen und Nischen. Der Unterschied zwischen Öffnungen und Nischen besteht darin, dass die Tiefe der Öffnung größer oder gleich der Wandstärke ist, während die Nische kleiner als die Wanddicke sein muss. Ob es sich um eine Öffnung oder eine Nische handelt, entscheidet der „*ObjectType*“. Der Text „*Opening*“ oder kein angegebener Wert steht für eine Öffnung. Der Text „*Recess*“ steht für eine Nische.

```
ENTITY IfcOpeningElement;
  ENTITY IfcRoot;
    GlobalId                : IfcGloballyUniqueId;
    OwnerHistory            : IfcOwnerHistory;
    Name                    : OPTIONAL IfcLabel;
    Description             : OPTIONAL IfcText;
  ENTITY IfcObject;
    ObjectType              : OPTIONAL IfcLabel;
  INVERSE
    IsDefinedBy             : SET OF IfcRelDefines FOR RelatedObjects;
    HasAssociations         : SET OF IfcRelAssociates FOR RelatedObjects;
    HasAssignments          : SET OF IfcRelAssigns FOR RelatedObjects;
    Decomposes              : SET OF IfcRelDecomposes FOR RelatedObjects;
    IsDecomposedBy          : SET [0:1] OF IfcRelDecomposes FOR RelatingObject;
  ENTITY IfcProduct;
    ObjectPlacement         : OPTIONAL IfcObjectPlacement;
    Representation          : OPTIONAL IfcProductRepresentation;
  INVERSE
    ReferencedBy            : SET OF IfcRelAssignsToProduct FOR RelatingProduct;
  ENTITY IfcElement;
    Tag                     : OPTIONAL IfcIdentifier ;
  INVERSE
    ConnectedTo             : SET OF IfcRelConnectsElements FOR RelatingElement;
    ConnectedFrom           : SET OF IfcRelConnectsElements FOR RelatedElement;
    ContainedInStructure    : SET [0:1] OF IfcRelContainedInSpatialStructure FOR RelatedElements;
  ENTITY IfcOpeningElement;
  INVERSE
    VoidsElements           : IfcRelVoidsElement FOR RelatedOpeningElement;
    HasFillings             : SET OF IfcRelFillsElement FOR RelatingOpeningElement;
END_ENTITY;
```

Tabelle 2.6: Klassenhierarchie der Entität „*IfcOpeningElement*“ [IFC1]

Die Beziehung zwischen den Entitäten „*IfcWallStandardCase*“ und „*IfcOpeningElement*“ definiert die Entität „*IfcRelVoidsElements*“. Sie basiert auf einer booleschen Operation, bei der die Öffnungsgeometrie von der Wandgeometrie abgezogen wird. Eine Öffnung kann durch eine Tür oder ein Fenster gefüllt werden. Beides wird durch „*IfcRelFillsElement*“ referenziert.

Das lokale Koordinatensystem einer Öffnung ist bezüglich des Koordinatensystems seines Containers angegeben, der in dem Fall die Wand ist. Der Körper, der die Öffnung ausschneidet, hat sein eigenes lokales Koordinatensystem (Extrusionskoordinatensystem), welches innerhalb des lokalen Koordinatensystems der Öffnung positioniert ist. Die Z-Achse des Extrusionskoordinatensystems ist immer die Richtung, die in die Wand hinein zeigt. Der Körper ist ein Extrusionskörper, der durch ein 2D-Grundprofil auf der lokalen XY-Ebene in die lokale Z-Achse (beide bezüglich des Extrusionskoordinatensystems) extrudiert wurde. Das 2D-Grundprofil ist mit einem 2D-Placement positioniert, das dem Extrusionskoordinatensystems zugeordnet wird. Für eine Öffnung gibt es also drei lokale Koordinatensysteme, nämlich das Koordinatensystem für die Öffnung selbst, für den Extrusionskörper und für das 2D-Grundprofil.

Das Prinzip wird mit Hilfe der folgenden Beispiele mit einem rechteckigen Fenster (Bild 2.10) erläutert.

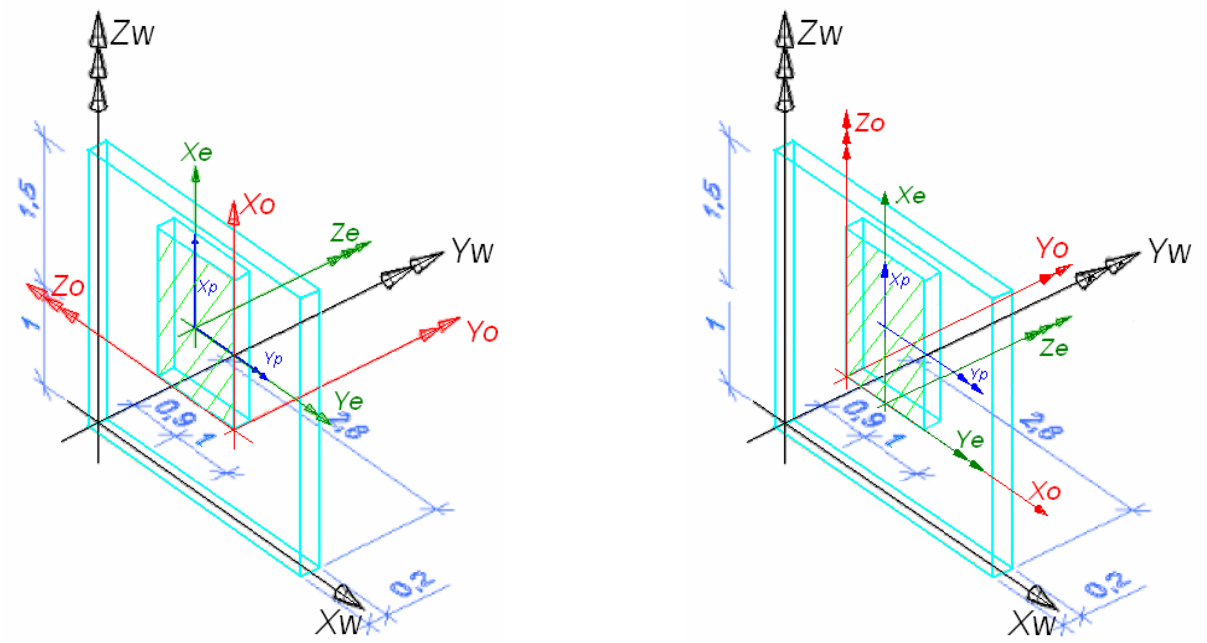


Bild 2.10: Beispiel einer Öffnung in einer Wand

Das lokale Koordinatensystem der Öffnung kann irgendwo definiert werden. Bei dem linken Beispiel liegt das Öffnungskoordinatensystem im rechten unteren Eckpunkt der Öffnung mit gedrehter Orientierung. Das lokale Koordinatensystem des 2D-Profils und des Extrusionskörpers haben die gleiche Orientierung (X-, Y-Achse) und den gleichen Ursprung, der im Mittelpunkt des Rechtecks liegt. Bei dem rechten Beispiel liegt das lokale Öffnungskoordinatensystem ungedreht im linken unteren Eckpunkt der Öffnung. Während das lokale Koordinatensys-

tem des 2D-Profils im Mittelpunkt des Rechtecks positioniert ist, liegt das Extrusionskörperkoordinatensystem im Mittelpunkt der Unterkante des Rechtecks.

```
/* Repräsentation der Wand */
#1=IFCWALLSTANDARDCASE('abcdefghijklmnopqrst01', #2, $, $, $, #3, #4, $);  /* #2 ist das IfcOwnerHistory */
#3=IFCLOCALPLACEMENT($, #10);
#10=IFCAXIS2PLACEMENT3D(#16, $, $);
#16=IFCCARTESIANPOINT((2., 1., 0.));
#4=IFCPRODUCTDEFINITIONSHAPE($, $, (#11, #13, #211));                    /* Geometrie der Wand */

/* Für das linke Beispiel in Bild 2.10 */
/* Beziehung und lokales Koordinatensystem der Öffnung relative zur Wand */
#81=IFCOPENINGELEMENT('2DVz9Ik7nDE8UYM00UgOTP', #2, $, '', $, #80, #75, $);
#82=IFCRELVOIDSELEMENT('0CFN5X3K519evNPGT30ZIp', #2, $, $, #1, #81);
#80=IFCLOCALPLACEMENT(#3, #79);                                           /* Placement der Öffnung */
#76=IFCCARTESIANPOINT((1.9, -0.1, 1.));
#77=IFCDIRECTION((-1., 0., 0.));
#78=IFCDIRECTION((0., 0., 1.));
#79=IFCAXIS2PLACEMENT3D(#76, #77, #78);
#75=IFCPRODUCTDEFINITIONSHAPE($, $, (#74));

/* Geometrische Repräsentation der Öffnung unter dem lokalen Extrusionskoordinatensystem */
#74=IFCSHAPEREPRESENTATION(#111, 'Body', 'SweptSolid', (#72));
#72=IFCEXTRUDEDAREASOLID(#65, #70, #71, 0.2);
#65=IFCRECTANGLEPROFILEDEF(.AREA. , $, #64, 1.5, 1.);
#64=IFCAXIS2PLACEMENT2D(#62, #63);                                       /* Placement des 2D-Profils */
#62=IFCCARTESIANPOINT((0., 0.));
#63=IFCDIRECTION((1., 0.));
#70=IFCAXIS2PLACEMENT3D(#67, #68, #69);                                  /* Placement des Extrusionskörpers */
#67=IFCCARTESIANPOINT((0.75, 0., 0.5));
#68=IFCDIRECTION((0., 1., 0.));
#69=IFCDIRECTION((1., 0., 0.));
#71=IFCDIRECTION((0., 0., 1.));                                          /* Extrusionsrichtung */

/* für das rechte Beispiel in Bild 2.10 */
/* Beziehung und lokales Koordinatensystem der Öffnung relative zur Wand  */
#81=IFCOPENINGELEMENT('2VDz9Ik7nDE8UMY00TPgOUg', #2, $, '', $, #80, #75, $);
#82=IFCRELVOIDSELEMENT('0FC5NXK3519NPGTevIpp03', #2, $, $, #1, #81);
#80=IFCLOCALPLACEMENT(#3, #79);                                           /* Placement der Öffnung */
#76=IFCCARTESIANPOINT((0.9, -0.1, 1.));
#77=IFCDIRECTION((0., 0., 1.));
#78=IFCDIRECTION((1., 0., 0.));
#79=IFCAXIS2PLACEMENT3D(#76, #77, #78);
#75=IFCPRODUCTDEFINITIONSHAPE($, $, (#74));

/* Geometrische Repräsentation der Öffnung unter dem lokalen Extrusionskoordinatensystem */
#74=IFCSHAPEREPRESENTATION(#111, 'Body', 'SweptSolid', (#72));
#72=IFCEXTRUDEDAREASOLID(#65, #70, #71, 0.2);
#65=IFCRECTANGLEPROFILEDEF(.AREA. , $, #64, 1.5, 1.);
#64=IFCAXIS2PLACEMENT2D(#62, #63);                                       /* Placement des 2D-Profils */
#62=IFCCARTESIANPOINT((0.75, 0.));
#63=IFCDIRECTION((1., 0.));
#70=IFCAXIS2PLACEMENT3D(#67, #68, #69);                                  /* Placement des Extrusionskörpers */
#67=IFCCARTESIANPOINT((0.5, 0., 0.));
#68=IFCDIRECTION((0., 1., 0.));
#69=IFCDIRECTION((0., 0., 1.));
#71=IFCDIRECTION((0., 0., 1.));                                          /* Extrusionsrichtung */
```

Tabelle 2.7: Auszug des IFC-Quellcodes einer Öffnung in der Wand

3 Konzeption und Definition eines neutralen Datenaustauschformats –BauXML

3.1 Ausgangssituation und Ziel

Das IFC Produktdatenmodell hat riesige Mengen an Informationen. Wie im Kapitel 2.2 vorgestellt, enthält das IFC Produktdatenmodell neben den Geometriedaten auch topologische und technologische Informationen. Darüber hinaus ist das IFC mehr auf das Gebiet der Architektur fokussiert. So sind nicht alle Informationen davon für die Vorfertigung nützlich und erforderlich. In IFC hat z.B. eine Wand eine gesamte Geometriedarstellung und alle Öffnungen sind bezüglich der zugehörigen Wand positioniert. Es gibt aber keinen Datentyp für den Stein. Für die Vorfertigung steht allerdings nur der Stein in Blickpunkt. Wenn die Wand in einzelne Steine aufgeteilt wird, dann muss die Wandgeometrie in einzelne Steingeometrien umgewandelt und angepasst werden. Für die Aufgaben einer Vorfertigungsfabrik sollten außer den geometrischen Informationen von Mauerwerksteinen auch die administrativen Informationen wie beispielsweise die Informationen des Auftrags- und Produktionsmanagements gespeichert werden können. Alle diese Anforderungen können mit dem jetzigen IFC-Standard nicht erfüllt werden. Eine Erweiterung von IFC ist sehr zeitintensiv. Deshalb wird in dem vorliegenden Kapitel ein neues Datenaustauschformat für die Vorfertigung eingeführt. Dabei werden die Idee und der derzeitige Entwicklungsstand eines neutralen Datenaustauschformats zwischen der Hausbauplanung und der Vorfertigung vorgestellt.

3.2 Anforderung an ein neutrales Datenaustauschformat

Die Entwicklung eines neutralen Datenformats zur Speicherung aller von der Vorfertigung benötigten Informationen ist eine aufwändige und anspruchsvolle Aufgabe, weil das zu entwickelnde Datenformat mächtig genug sein muss, sämtliche nötigen Informationen zu erfassen. Anderseits soll es sowohl für einfache als auch komplexe Wandsteine praktikabel sein.

Im Folgenden werden grundlegende Anforderungen umrissen:

- Die Hierarchie des Gebäudes muss konkret erkennbar sein.
- Jeder Stein muss korrekt abgebildet werden können. Dazu gehören sowohl die geometrischen Informationen wie z.B. die Geometrie des Steines und die daran geknüpften Fertigungsfeatures (Abs. 3.3), als auch die administrativen Informationen wie der Bearbeitungsstatus des Steins und der Features.

- Objektorientierte Konzepte müssen unterstützt werden. Ein objektorientiertes Konzept ist vorteilhaft und hat sich deshalb in der Praxis bereits etabliert.
- Das neue Datenformat muss möglichst an den Bedürfnissen und Anforderungen der Fertigung ausgerichtet sein, aber nicht konkret eine Maschine bedienen sondern möglichst neutral allen einsetzbaren Fertigungsmaschinen zur Verfügung stehen.
- Die Möglichkeit, die Informationen für die Auftrags- und Produktionsverwaltung zu speichern, muss zur Verfügung stehen.
- Das neue Datenformat muss skalierbar und leicht erweiterbar sein.
- Mit dem Gedanken, dass in der Praxis unterschiedliche Leitsysteme verwendet werden, sollte das neue Datenformat möglichst plattformunabhängig sein.

3.3 Feature basierendes Metamodell – BauXML

3.3.1 Fertigungsfeature (Machining Feature)

Ein CAD-System liefert gewöhnlich die Topologie und Geometrieinformationen eines Werkstücks, die aber für die Fertigung des Werkstücks manchmal kritisch sind. Die Geometrie eines Werkstücks kann in viele kleine geometrische Entitäten, die so genannten „Features“ (Bild 3.1), unterteilt werden. Features stellen eine Ansammlung von Entitäten in einer intelligenten Weise, so wie die Ingenieure denken, dar. Für die Integration von CAD/CAM erlangt das Verwenden der Features immer mehr Aufmerksamkeit in der Forschung und Praxis [VDI9][MASS][WEBE][TÖNS]. Zunehmende Integration erfordert ein Kommunikationsmittel zwischen Design und Fertigung. Features sind in beiden Anwendungen verwendbar. Heutzutage ist der Einsatz der Featuretechnik, die bereits vielfach untersucht worden ist und inzwischen als gesicherte Vorgehensweise zur Planung im CAD/CAM Bereich betrachtet werden kann [EVER], ein Leistungsmerkmal für NC-Programmiersysteme.

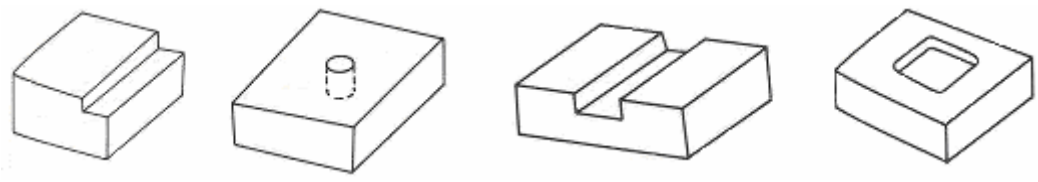


Bild 3.1: Feature [BERN]

Es ist wichtig zu erkennen, dass die Features, die in der Entwurfsphase benutzt werden, manchmal für die Fertigung nicht notwendig oder geeignet sind. Das Werkstück in Bild 3.2(a) kann z.B. im Design so konstruiert werden, dass eine Rippe auf einen Grundblock addiert

wird (Bild 3.2(b)), während in der Fertigung tatsächlich zwei Stufen vom Grundwerkstück abgezogen werden (Bild 3.2(c)).

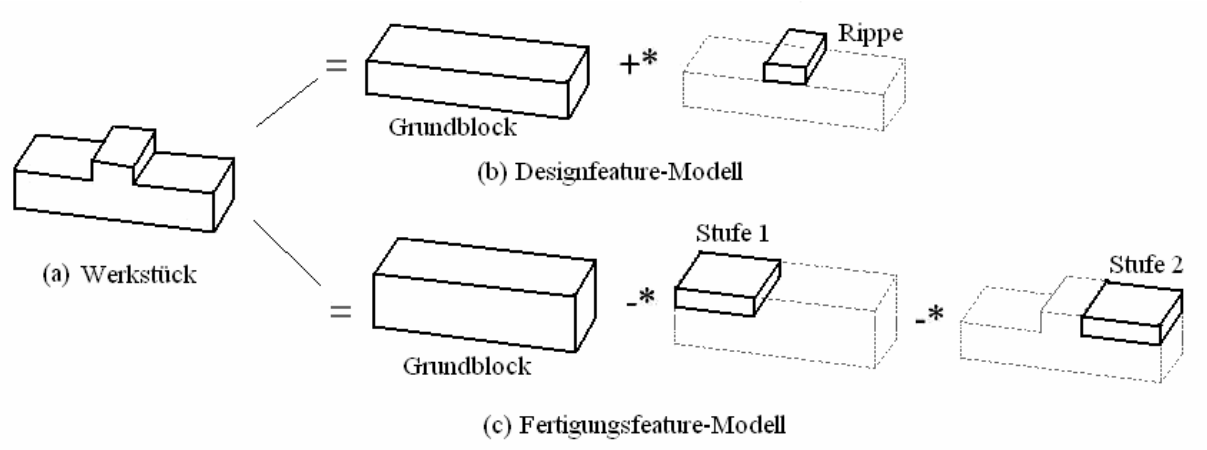


Bild 3.2: Designfeatures und Fertigungsfeatures [HANP]

So werden die Features mit Bezug zur Fertigung im Folgenden als Fertigungsfeatures bezeichnet. Nach ISO 10303-24-STEP [ISO1] wird ein Fertigungsfeature als das Volumen von Material definiert, das zum Erhalten der endgültigen Geometrie vom Grundwerkstück abgenommen werden soll. Die endgültige Gesamtgestalt wird also durch Subtraktion der Features von der Grundform gebildet.

3.3.2 XML (Extensible Markup Language) als Beschreibungssprache

Zur Definition der BauXML-Metamodellelemente wurde XML [W3C] als Beschreibungssprache gewählt. XML steht für Extensible Markup Language. Sie ist eine Metasprache für alle Anwender, um eigene Markups zu schreiben. Inzwischen ist XML ein standardisierter Mechanismus für den Datenaustausch zwischen unterschiedlichen Applikationen. Außer der verbreiteten Anwendung im Internet spielt XML eine immer größere Rolle auch in der Automatisierung. Auch dort ist es notwendig, strukturierte Daten bereitzustellen, um Bereichen mit anderen Applikationen die Manipulation der Information zu ermöglichen.

Das BauXML-Modell basiert auf dem XML-Schema. Die Austauschdokumente zwischen Bauplanung und Vorfertigung dürfen nicht aus beliebigem, regellosem Text bestehen, sondern müssen einer gemeinsamen Form oder Struktur folgen, die im gegebenen Problembereich vereinbart wurde und gültig ist. Eine gewünschte Folge der gemeinsamen Richtlinien ist, dass ein Hausmodell gemäß der Richtlinie in einzelnen fertigungsfreundlichen Steinformen zur Verfügung gestellt werden kann. Darüber hinaus kann eine Vorfertigungsfabrik ihre Software so umstellen, dass die nach dieser BauXML-Richtlinie erstellten Datensätze auch in die Software eingelesen und von ihr verarbeitet und interpretiert werden können. Zur Über-

prüfung eines BauXML-Dokuments bietet das BauXML-Schema außer der so genannten Wohlgeformtsheitprüfung auch eine „Validierung“. Ein BauXML-Dokument ist wohlgeformt, wenn es zu jedem Start-Tag einen End-Tag gibt, Elemente korrekt verschachtelt sind und keine Zeichen fehlen oder an der falschen Stelle stehen. Eine Validierung ist die Gültigkeitprüfung, ob ein BauXML-Dokument von Struktur und Inhalt mit der Schema-Spezifikation übereinstimmt. Damit wird eine wichtige Datenquelle der Fertigung gesichert. Die Hauptmotivation hinter der Nutzung des XML-Schemas ist: Der Datenaustausch erfolgt durch ein Instanz-Dokument und wird durch das Schema überprüft.

Ein XML-Schema besteht hauptsächlich aus benannten Typendefinitionen und Elementdeklarationen. Es unterstützt zwei verschiedene Typenkategorien: komplexe Typen und einfache Typen. Der einfache Typ (*simpleType*) repräsentiert nur den Typ als String ohne Elemente oder Attribute und dient zur Beschreibung der Kinder von Attributen und Nur-Text-Elementen. In BauXML werden vier einfache Typen definiert, und zwar *AngleUnitType*, *LengthUnitType*, *OrderStatus* (Abs. 3.4.1) und *GuidType* (Abs. 3.4.2). Die ersten beiden Typen sind die Aufzählungen der möglichen Einheiten (mm, cm, m oder deg, grad, usw.). Der komplexe Typ (*complexType*) spezifiziert die Bedingungen für die Kinder und Attribute eines gegebenen Typs von Informationselement. Das Inhaltsmodell der Kinder eines Elementes ist aus Partikeln zusammengesetzt. Ein Partikel ist eine lokale Elementdeklaration oder eine Referenz auf eine globale Elementdeklaration (*element*), eine Komposition (*sequence*, *choice*, oder *all*), eine Referenz auf eine benannte Inhaltsmodellgruppe (*group*) oder ein Element-Wildcard (*any*). Eine komplexe Typdefinition besteht aus einer Sequenz von Partikeln, die angeben, welche Elemente nach welcher Reihenfolge in Instanzen des Typs auftauchen können. Die in BauXML verwendeten Partikel sind ausschließlich Kompositionen. Es ist auch möglich anzugeben, wieviele Male Elemente auftauchen können, die zu einem bestimmten Partikel gehören. Solche Eigenschaften werden mit Hilfe des *minOccurs*–Attributs und des *maxOccurs*–Attributs beschrieben. Das *minOccurs*–Attribut beschreibt die Mindestanzahl der Vorkommen als nicht-negative Integerzahl. Das *maxOccurs*–Attribut beschreibt die maximal erwartete Anzahl der Vorkommen als nicht-negative Integerzahl oder mit dem Wert *unbounded*, der angibt, dass es keine obere Beschränkung gibt [ESSE].

Im Folgenden sind die BauXML-Modellelemente zur besseren Lesbarkeit sowohl in der zugrunde liegenden textuellen Notation als auch in der grafischen Notation nach Altova XMLSpy [ALTO] dargestellt. Die Bedeutungen der gezeichneten Symbole sind in Tabelle 3.1 gezeigt.

	*sequence*
	*choice*
	*all*
	*minOccurs* = 0 *maxOccurs* = *unbounded*

Tabelle 3.1: Notationssymbole

3.3.3 Allgemeines zur BauXML-Struktur

Für jedes Bauvorhaben wird eine Projektstruktur aufgestellt. Das Projekt ist als Wurzelelement in der BauXML-Baumstruktur (siehe Bild 3.3) der oberste Datencontainer aller Informationen. In einer BauXML-Datei gibt es nur eine Instanz vom Projekttyp.

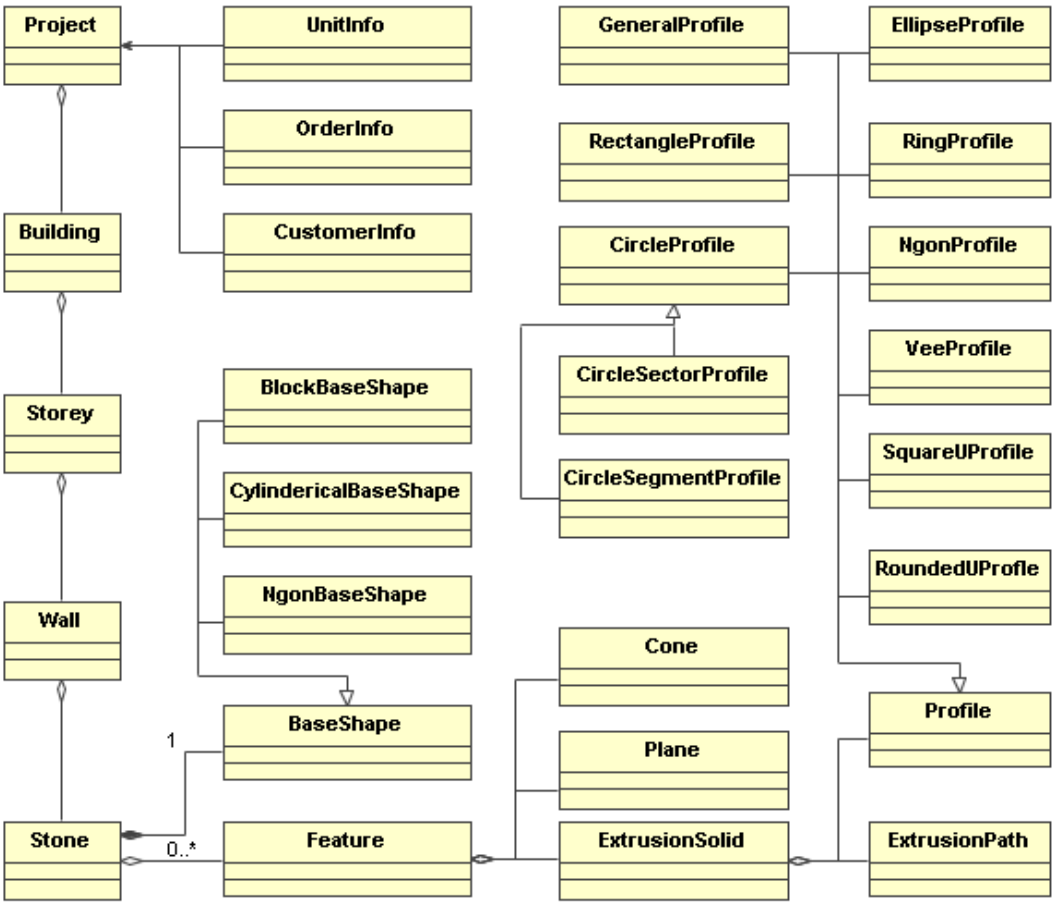


Bild 3.3: Datenstruktur des BauXML-Modells in UML-Notation

Ein Haus wird in eine Gebäude-Stockwerk-Wand-Stein-Feature-Hierarchie strukturiert. Ein Gebäude kann mehrere Stockwerke haben, während ein Stockwerk aus mehreren Wänden entsteht. Eine Wand kann aus mehreren Steinen bestehen. Jeder Stein hat eine Grundformabmessung, Features sind jedoch optional: Ein Stein kann mehrere Features oder auch überhaupt keine Features haben. Die sich für ein Projekt ergebende Struktur ist hierarchisch und stellt

den Untergliederungen eindeutige Oberbegriffe voran. Die Hierarchiestruktur ermöglicht eine objektorientierte Interoperabilität zwischen Stockwerk, Wänden und Steinen. Die Grundstruktur ist für alle Projekte gleich gehalten. Einzelne Untergliederungen können entfallen.

3.4 Datensatzbeschreibung

3.4.1 Administrative Informationen in BauXML

Die administrativen Informationen werden unter dem Projekt zusammengefasst. Dazu gehören Vertrags- und Kundendaten, verwendete Einheiten und die in der Segmentierung verwendeten Parameter. Sie werden einmalig global festgelegt. Zur internen Verwaltung werden die nötige Auftragsdaten und Kundendaten anders als die Einheitendaten optional aufgeschrieben.

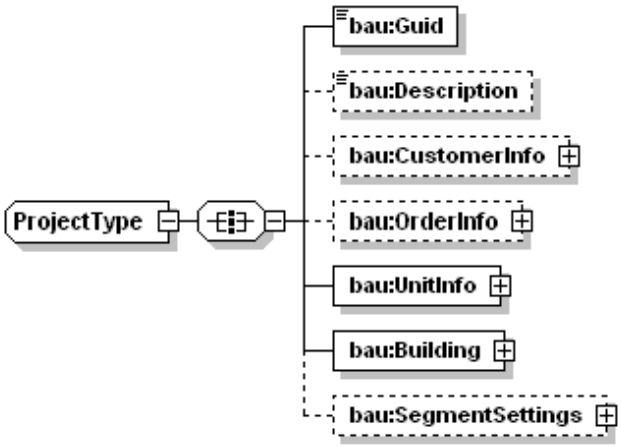
Diagramm	
Quellcode	<xs:complexType name="ProjectType"> <xs:all> <xs:element name="Guid" type="bau:GuidType"/> <xs:element name="Description" type="xs:string" minOccurs="0"/> <xs:element name="CustomerInfo" type="bau:AddressType" minOccurs="0"/> <xs:element name="OrderInfo" type="bau:OrderInfoType" minOccurs="0"/> <xs:element name="UnitInfo" type="bau:UnitInfoType"/> <xs:element name="Building" type="bau:BuildingType"/> <xs:element name="SegmentSettings" type="bau:SegmentSettingsType" minOccurs="0"/> </xs:all> </xs:complexType>

Tabelle 3.2: Projektdefinition in BauXML

Auftrag: Zu den Auftraginformationen, die jederzeit leicht erweiterbar sind, sind derzeit sechs Elemente anzugeben.

- *OrderNumber*: Die interne Auftragsnummer.
- *InitiatedDate*: Das Auftragserstellungsdatum.
- *DeliveryDate*: Der Auslieferungstermin.
- *QuantityOrdered*: Die Auftragsmenge.
- *Creator*: Der Auftragsbearbeiter.

- *OrderStatus*: Vier Auftragszustände sind vordefiniert und können immer wieder aktualisiert werden: „*Created not yet manufactured*“, „*In Manufacturing*“, „*Manufactured not yet transported*“, „*Transported*“.

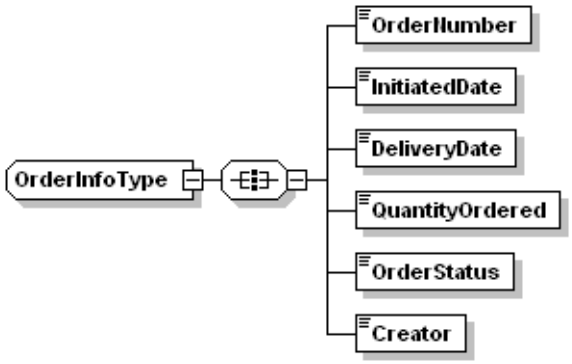
Diagramm	
Quellcode	<xs:complexType name="OrderInfoType"> <xs:all> <xs:element name="OrderNumber" type="xs:string"/> <xs:element name="InitiatedDate" type="xs:string"/> <xs:element name="DeliveryDate" type="xs:string"/> <xs:element name="QuantityOrdered" type="xs:nonNegativeInteger"/> <xs:element name="OrderStatus" type="bau:OrderStatus"/> <xs:element name="Creator" type="xs:string"/> </xs:all> </xs:complexType>

Tabelle 3.3: Auftragsinformationen in BauXML

Kunden: Das Kundenelement wird definiert mit „*AdressType*“. Es beinhaltet derzeit nur die Informationen, die eigentlich für die Lieferung der Fertigsteinteile gedacht sind.

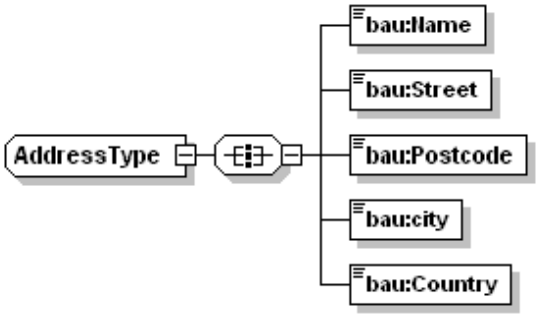
Diagramm	
Quellcode	<xs:complexType name="AddressType"> <xs:all> <xs:element name="Name" type="xs:string"/> <xs:element name="Street" type="xs:string"/> <xs:element name="Postcode" type="xs:string"/> <xs:element name="city" type="xs:string"/> <xs:element name="Country" type="xs:string"/> </xs:all> </xs:complexType>

Tabelle 3.4: Kundeninformationen in BauXML

3.4.2 Verwendete Grundbegriffe

Point: Ein Punkt wird mit seinen drei als Attribute definierten Koordinaten als „*CartesianPoint3D*“ im dreidimensionalen kartesischen Koordinatensystem definiert (Tabelle 3.5).

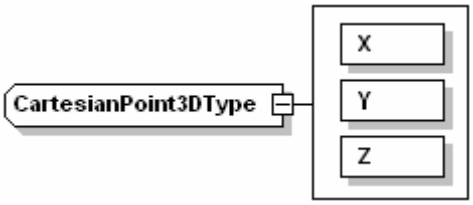
Diagramm	
Quellcode	<xs:complexType name="CartesianPoint3DType"> <xs:attribute name="X" type="xs:double" use="required"/> <xs:attribute name="Y" type="xs:double" use="required"/> <xs:attribute name="Z" type="xs:double" use="required"/> </xs:complexType>

Tabelle 3.5: Punktdefinition in BauXML

Direction: Der Typ definiert durch seine drei als Attribute definierten Komponenten eine Richtung im dreidimensionalen Raum (Tabelle 3.6). Die Komponenten sind die normalisierten Projektoren des Richtungsvektors auf die drei Achsen des Koordinatensystems: *ratioX* (auf die X-Achse), *ratioY* (auf die Y-Achse) und *ratioZ* (auf die Z-Achse).

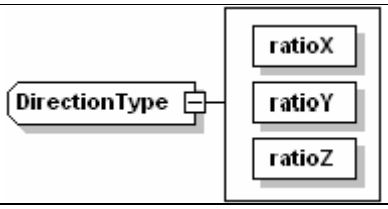
Diagramm	
Quellcode	<xs:complexType name="DirectionType"> <xs:attribute name="ratioX" type="xs:double" use="required"/> <xs:attribute name="ratioY" type="xs:double" use="required"/> <xs:attribute name="ratioZ" type="xs:double" use="required"/> </xs:complexType>

Tabelle 3.6: Richtungsdefinition in BauXML

Vector: Der Vektortyp wird in der Featurebeschreibung für den „*ExtrusionsPath*“ verwendet. Zwei Attribute sind dafür notwendig: die Richtung (*Orientation*) und der Betrag (*Magnitude*).

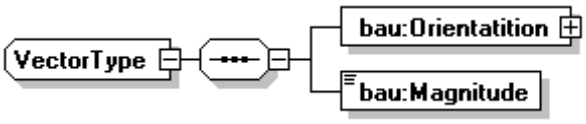
Diagramm	
Quellcode	<xs:complexType name="VectorType"> <xs:sequence> <xs:element name="Orientatition" type="bau:DirectionType"/> <xs:element name="Magnitude" type="xs:double"/> </xs:sequence> </xs:complexType>

Tabelle 3.7: Vektordefinition in BauXML

Guid: Der „Globally Unique Identifier“ ist ein 22-stelliges Stringelement, das einen Bauteil eindeutig identifiziert (Tabelle 3.8). Verwendung findet er bei den Elementen wie Gebäude, Stockwerke, Wände, Steine und Features. Für Gebäude, Stockwerke und Wände werden die „*Guids*“ aus den IFC-Daten übernommen. Für die in IFC nicht definierten Typen „Stone“ und „Feature“ wird jeweils ein eindeutiger „*Guid*“ angegeben.

Quellcode	<xs:simpleType name="GuidType"> <xs:restriction base="xs:normalizedString"> <xs:minLength value="22"/> <xs:maxLength value="22"/> </xs:restriction> </xs:simpleType>

Tabelle 3.8: Bauteil-Identifikation in BauXML

BaseShape: Das Element „*BaseShape*“ spezifiziert die Ausgangsform eines Steines bevor die Features abgenommen werden. Hierfür stehen drei Formen zur Verfügung: Quader, Zylinder und Prisma (Tabelle 3.9).

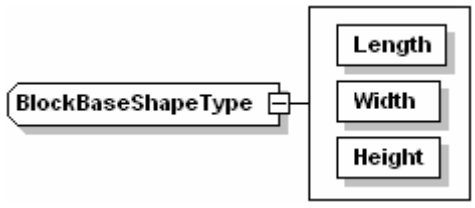
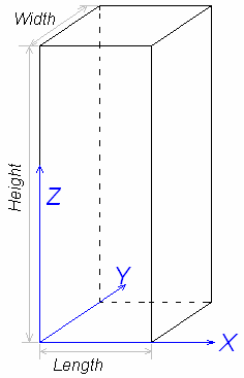
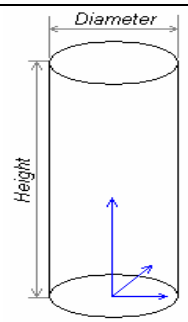
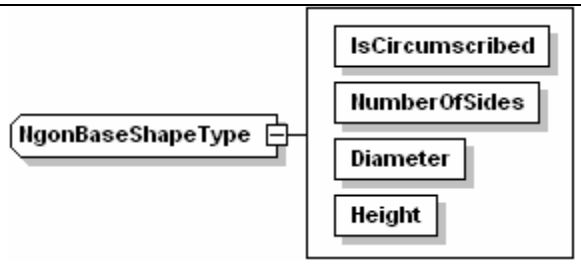
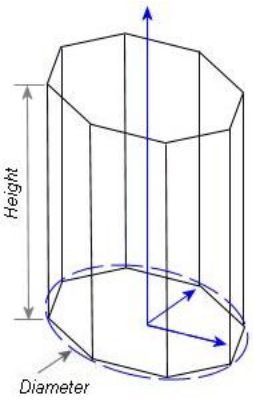
Quader	 <xs:complexType name="BlockBaseShapeType"> <xs:attribute name="Length" type="xs:double" use="required"/> <xs:attribute name="Width" type="xs:double" use="required"/> <xs:attribute name="Height" type="xs:double" use="required"/> </xs:complexType>	
Zylinder	 <xs:complexType name="CylindricalBaseShapeType"> <xs:attribute name="Diameter" type="xs:double" use="required"/> <xs:attribute name="Height" type="xs:double" use="required"/> </xs:complexType>	
Prisma	 <xs:complexType name="NgonBaseShapeType"> <xs:attribute name="IsCircumscribed" type="xs:boolean" use="required"/> <xs:attribute name="NumberOfSides" type="xs:positiveInteger" use="required"/> <xs:attribute name="Diameter" type="xs:double" use="required"/> <xs:attribute name="Height" type="xs:double" use="required"/> </xs:complexType>	

Tabelle 3.9: Steingrundformdefinition in BauXML

Quader als Grundform ist für Steine am häufigsten. Zur Festlegung eines Quaders werden eine Länge, eine Dicke und eine Höhe gebraucht. Steine mit zylindrischer oder prismatischer Form finden beispielsweise bei Stürzen, die in IFC als „*IfcColumn*“ definiert sind, Anwendungen. Während ein Zylinder mit dem Durchmesser der Kreisfläche und der Zylinderhöhe

definiert wird, wird ein Prisma durch die Kantenanzahl, die Höhe, den Durchmesser des Grundkreises sowie eine boolesche Kennung festgelegt. Ob das Vieleck innerhalb oder außerhalb des Grundkreises liegt, entscheidet die boolesche Kennung „*IsCircumscribed*“.

3.4.3 Koordinatensysteme

Auf der Ebene der BauXML-Struktur unterhalb des Stockwerks, z.B. eine Wand, ein Stein, ein Feature oder ein 2D-Grundprofil, wird jeweils ein eigenes lokales, rechtsdrehendes Koordinatensystem definiert. Es ist zur Visualisierung und zur Erstellung einer Werkplanung notwendig und sinnvoll.

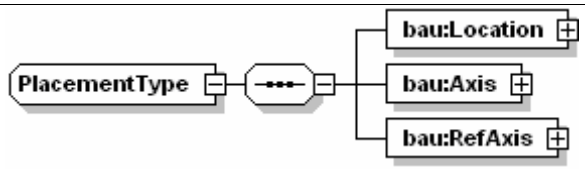
Diagramm	
Quellcode	<xs:complexType name="PlacementType"> <xs:sequence> <xs:element name="Location" type="bau:CartesianPoint3DType"/> <xs:element name="Axis" type="bau:DirectionType"/> <xs:element name="RefAxis" type="bau:DirectionType"/> </xs:sequence> </xs:complexType>

Tabelle 3.10: Definition des Placements in BauXML

Ein lokales Koordinatensystem sollte so positioniert werden, dass der Ursprung ein logischer Fixpunkt (z.B. Ecke vorne unten links bei der Wand und beim Stein, Mittelpunkt eines Kreises) ist, der zur Positionierung der Objekte in einem CAD-System und zur Generierung des Fertigungsprogramms dienen kann. Im Einzelnen sind die Nullpunktlage und die Ausrichtung des Koordinatensystems objektspezifisch festzulegen. Für die Wand und den Stein sollte die positive Z-Achse nach „oben“ und die X-Achse in die Wandführungsrichtung zeigen. Für das Feature sollte die positive Z-Achse immer in das Werkstück hinein zeigen. Innerhalb des Steinkoordinatensystems sind die Punkte definiert, auf die die Features positioniert werden. Die Lagen und Richtungen der Features werden durch lokale Koordinatensysteme festgelegt. Die lokalen Koordinatensysteme werden definiert durch ihren Ursprungspunkt und zwei normierte Raumvektoren (*Axis* und *RefAxis*), welche die Lage der Z- und X-Achse im Steinkoordinatensystem angeben. Der Raumvektor für die lokale Y-Achse ergibt sich folgerichtig aus dem Vektorprodukt des Z-Vektors mit dem X-Vektor.

3.4.4 2D-Grundprofile

Ein 2D-Profil kann als eine Berandungslinie einer Translationsfläche, als Mantellinie eines Translationskörpers oder als selbständiges Geometrieelement zur Darstellung drahtförmiger

Geometrie verwendet werden. Neben einer Reihe von Parametern (z.B. Durchmesser oder eine Liste von Punkten), die das Profil beschreiben, hat jedes Profil zwei weitere Elemente:

- *Placement*: Mit dem Placement wird das 2D-Profil im Steinkoordinatensystem und damit später in der Fertigung ein Werkzeug schnell positioniert.
- *IsArea*: Mit dieser Flächenkennung unterscheidet ein Profil, ob es sich um eine Fläche oder eine drahtförmige Geometrie handelt. Die Unterscheidung beeinflusst die späteren Interpolationen der Fertigungsmakros. In der Voreinstellung ist mit dem Wert „true“ eine Fläche definiert.

Jedes Grundprofil wird aus einer Profilklasse abgeleitet und hat immer ein geerbtes Placement und eine Flächenkennung. Im Bauwesen treten das Rechteckprofil und das Kreisprofil häufig auf. Ein Extrusionskörper aus einer Kombination der beiden Profile kann z.B. eine Steckdosenöffnung und einen Installationsschlitz verkörpern. Die anderen Profile sind für eventuell mögliche Erweiterungen vorbereitet. Im Folgenden werden die wichtigsten Profile vorgestellt.

Rechteckprofil: Die Installationsschlitze und Aussparungen, z.B. für den Elektroverteiler, werden als Extrusionskörper vom Typ „*RectangleProfile*“ definiert. Das Placement liegt im Mittelpunkt des Rechtecks mit der X-Achse parallel zu einer Seitenlinie. Das Element „*Length*“ ist immer in X-Richtung definiert.

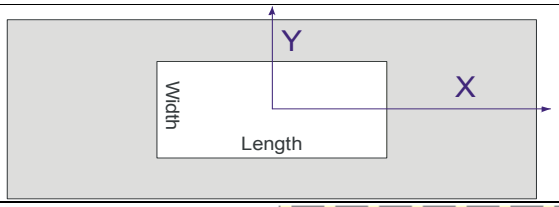
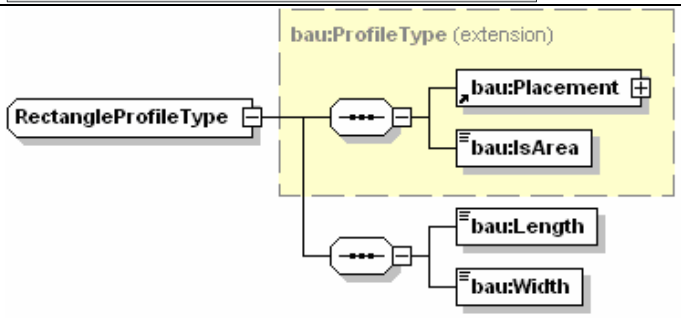
Grafik	
Diagramm	
Quellcode	<xs:complexType name="RectangleProfileType"> <xs:complexContent> <xs:extension base="bau:ProfileType"> <xs:sequence> <xs:element name="Length" type="xs:double"/> <xs:element name="Width" type="xs:double"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Tabelle 3.11: Rechteckprofil

Kreisprofil: Steckdosen, Dübellöcher oder sonstige kreisförmige Öffnungen verwenden das „*CircleProfile*“. Das Placement ist im Mittelpunkt des Kreises positioniert. In Tabelle 3.12 ist das Kreisprofil zusammen mit dem Quellcode anschaulich dargestellt.

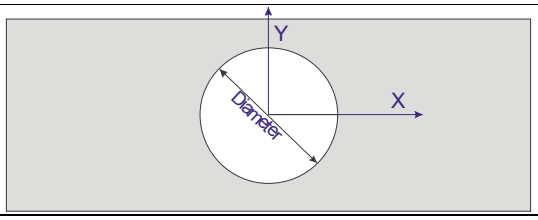
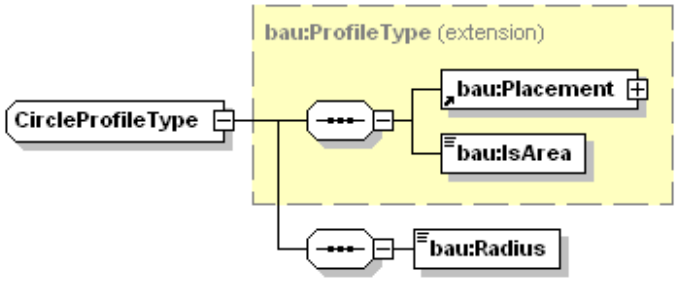
Grafik	
Diagramm	
Quellcode	<xs:complexType name="CircleProfileType"> <xs:complexContent> <xs:extension base="bau:ProfileType"> <xs:sequence> <xs:element name="Radius" type="xs:double"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Tabelle 3.12: Kreisprofil

Ellipsenprofil: Hier wird eine Ellipse definiert, deren Mittelpunkt den Ursprung des Koordinatensystems definiert und deren Hauptachse mit der X-Achse zusammenfällt (Tabelle 3.13).

Es gilt folgende Gleichung für die Koordinaten der Punkte einer solchen Ellipse:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{3.1}$$

Hieran ist erkennbar, dass die Hauptachse a (*SemAxis1* in Tabelle 3.13) und die Nebenachse b (*SemAxis2* in Tabelle 3.13) nötig sind, um eine Ellipse zu definieren.

In fertigungsfreundlicher Parameterform werden die Koordinaten folgendermaßen angegeben:

$$\begin{cases} x = a\cos t \\ y = b\cos t \end{cases}, \quad 0 \le t \le 2\pi \tag{3.2}$$

Der Quellcode der Definition ist mit einer erläuternden Veranschaulichung in Tabelle 3.13 zu finden.

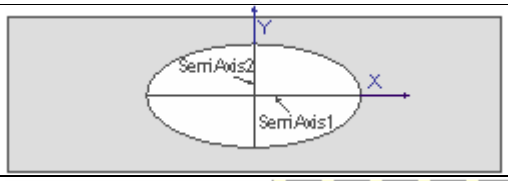
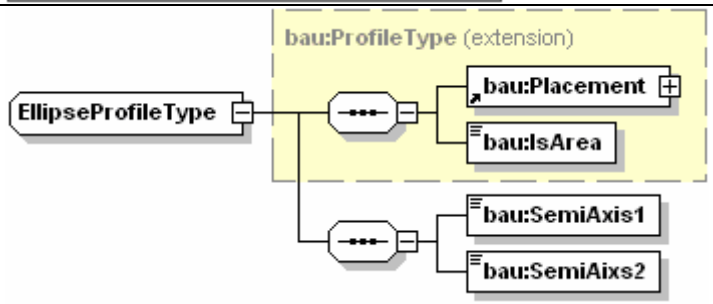
Grafik	
Diagramm	
Quellcode	<xs:complexType name="EllipseProfileType"> <xs:complexContent> <xs:extension base="bau:ProfileType"> <xs:sequence> <xs:element name="SemiAxis1" type="xs:double"/> <xs:element name="SemiAixs2" type="xs:double"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Tabelle 3.13: Ellipsenprofil

Kreissektorprofil: Ein Kreissektor ist ein Kreisausschnitt, der von einem Kreisbogen und zwei Kreisradien begrenzt wird. Das Placement liegt im Mittelpunkt. Nötige Parameter sind der Profil- und der Neigungswinkel (gegen Uhrzeigersinn), sowie der Radius (Tabelle 3.14).

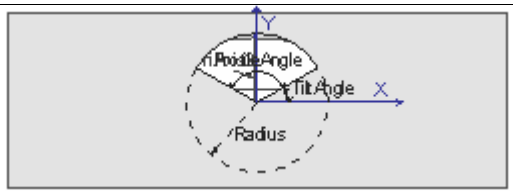
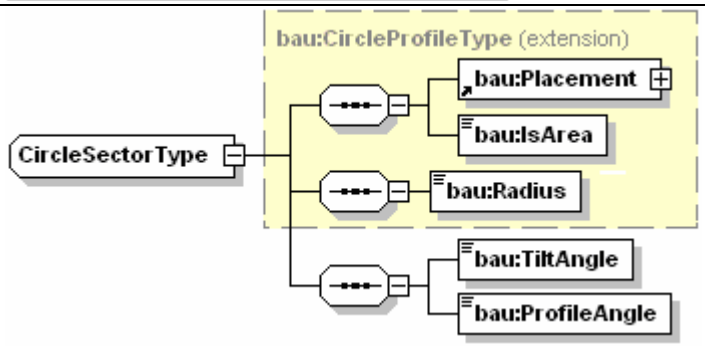
Grafik	
Diagramm	
Quellcode	<xs:complexType name="CircleSectorType"> <xs:complexContent> <xs:extension base="bau:CircleProfileType"> <xs:sequence> <xs:element name="TiltAngle" type="xs:double"/> <xs:element name="ProfileAngle" type="xs:double"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Tabelle 3.14: Kreissektorprofil

Kreissegmentprofil: Ein Kreissegment ist ein Abschnitt einer Kreisfläche, die von einem Kreisbogen und einer Kreissehne eingeschlossen wird. Das Profil kann im Zusammenhang mit Fensteröffnungen benutzt werden, die manchmal aus einem Rechteck und einem Kreissegment zusammengesetzt werden können. Das Placement liegt im Mittelpunkt des Kreises. Analog wie das Kreissektorprofil wird das Kreissegmentprofil durch einen Profilwinkel, der hier allerdings durch zwei „virtuelle“ Radien begrenzt wird, und den Neigungswinkel der „Anfangslinie“ zur X-Achse (siehe Tabelle 3.15) beschrieben.

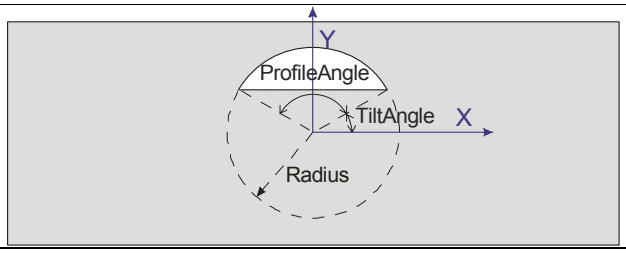
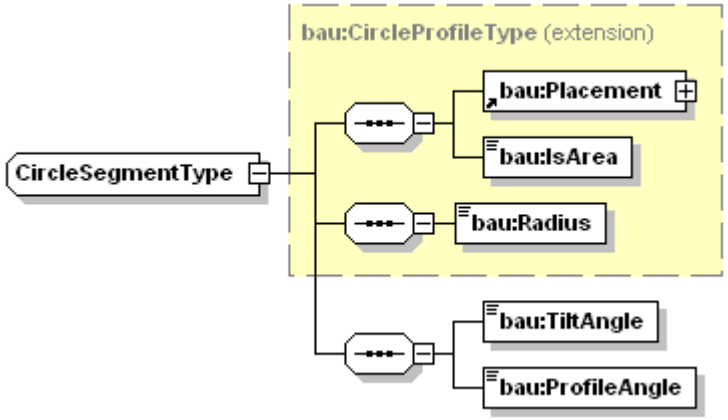
Grafik	
Diagramm	
Quellcode	<xs:complexType name="CircleSegmentType"> <xs:complexContent> <xs:extension base="bau:CircleProfileType"> <xs:sequence> <xs:element name="TiltAngle" type="xs:double"/> <xs:element name="ProfileAngle" type="xs:double"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Tabelle 3.15: Kreissegmentprofil

Ringprofil: Als Ringprofil wird die Fläche bezeichnet, die zwischen zwei Kreisbögen mit gemeinsamem Mittelpunkt und durch zwei Kreisradien begrenzt wird. Im Kreismittelpunkt wird das Placement definiert. Außer den zwei Parametern (Profilwinkel und Neigungswinkel) wie beim Kreissektorprofil, sind noch zwei Radien der beiden Kreisbögen anzugeben (siehe Tabelle 3.16).

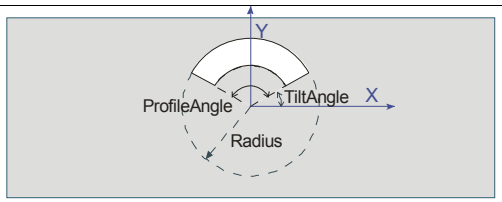
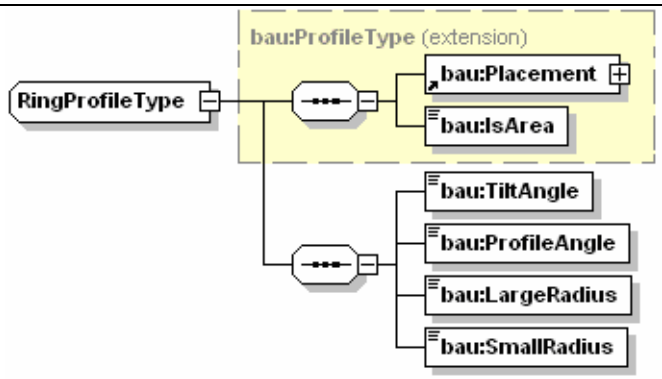
Grafik	
Diagramm	
Quellcode	<xs:complexType name="RingProfileType"> <xs:complexContent> <xs:extension base="bau:ProfileType"> <xs:sequence> <xs:element name="TiltAngle" type="xs:double"/> <xs:element name="ProfileAngle" type="xs:double"/> <xs:element name="LargeRadius" type="xs:double"/> <xs:element name="SmallRadius" type="xs:double"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Tabelle 3.16: Ringprofil

Allgemeines Profil: Das „*GeneralProfile*“ ist für allgemeine Konturen mit mindestens drei Eckpunkten gedacht. Das Placement wird irgendwo im Steinkoordinatensystem definiert.

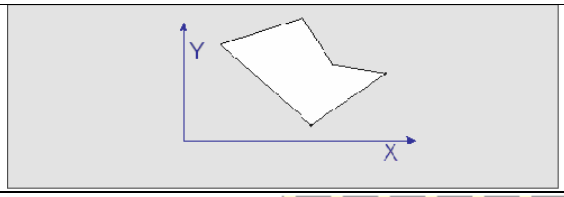
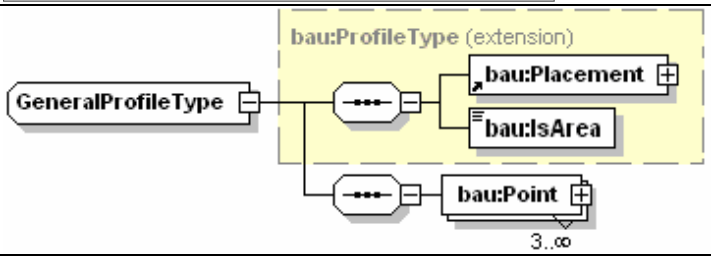
Grafik	
Diagramm	
Quellcode	<xs:complexType name="GeneralProfileType"> <xs:complexContent> <xs:extension base="bau:ProfileType"> <xs:sequence> <xs:element name="Point" type="bau:CartesianPoint3DType" minOccurs="3" maxOc curs="unbounded"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Tabelle 3.17: Allgemeines Profil

Polygonprofil: Ein Polygonprofil repräsentiert ein Vieleck. Das Placement liegt am Mittelpunkt des Profils. Ob das Vieleck innerhalb oder außerhalb eines Kreises liegt, entscheidet der Parameter „*IsCircumscribed*“. Falls TRUE, liegt das Vieleck innerhalb des Kreises.

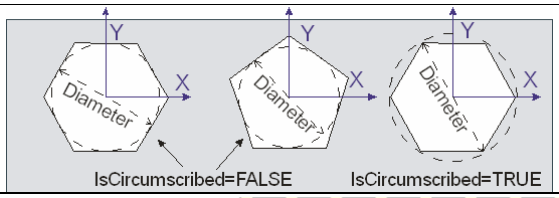
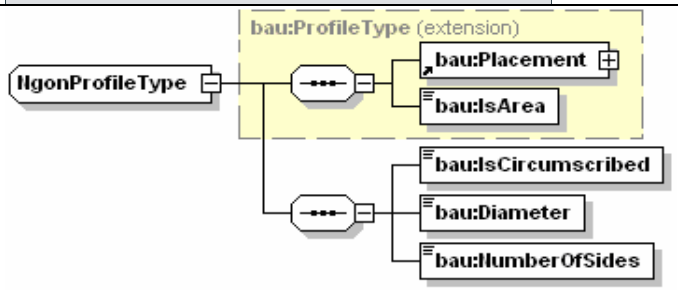
Grafik	
Diagramm	
Quellcode	<xs:complexType name="NgonProfileType"> <xs:complexContent> <xs:extension base="bau:ProfileType"> <xs:sequence> <xs:element name="IsCircumscribed" type="xs:boolean" default="true"/> <xs:element name="Diameter" type="xs:double"/> <xs:element name="NumberOfSides" type="xs:positiveInteger"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Tabelle 3.18: Polygonprofil

Rundes U-Profil: Es wird von zwei parallelen Linien und einem Halbkreis einseitig offen begrenzt. Die beiden Linien tangieren den Halbkreis an seinen Endpunkten. Das Placement ist definiert im Mittelpunkt. Die Öffnung des Profils zeigt in Richtung der positiven X-Achse.

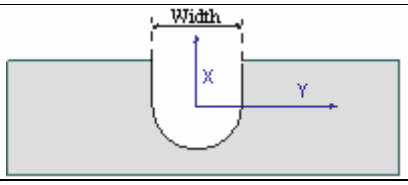
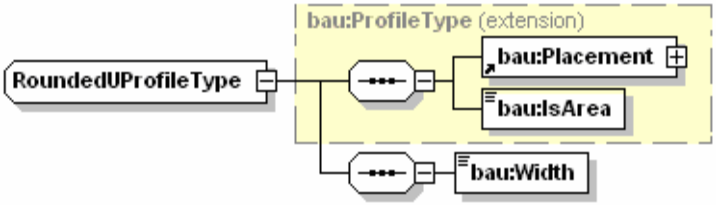
Grafik	
Diagramm	
Quellcode	<xs:complexType name="RoundedUProfileType"> <xs:complexContent> <xs:extension base="bau:ProfileType"> <xs:sequence> <xs:element name="Width" type="xs:double"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Tabelle 3.19: Rundes U-Profil

Trapezförmiges U-Profil: Ein trapezförmiges U-Profil ist durch drei Linien abgegrenzt. Eine ist die Grundlinie mit bestimmter Länge. Die anderen zwei Linien fangen von den Enden der Grundlinien an und verlängern sich unendlich mit einem Winkel zur Grundlinie zwischen 0° und 180°. Das trapezförmige U-Profil wird im Mittelpunkt der Grundlinie positioniert. Die X-Achse steht senkrecht auf der Grundlinie und zeigt aus der Öffnung heraus (siehe Tabelle 3.20). Als notwendige Parameter sind die Länge der Grundlinie (*BaseWidth*) und die zwei Neigungswinkel zu sehen.

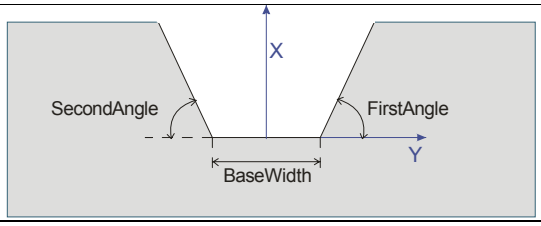
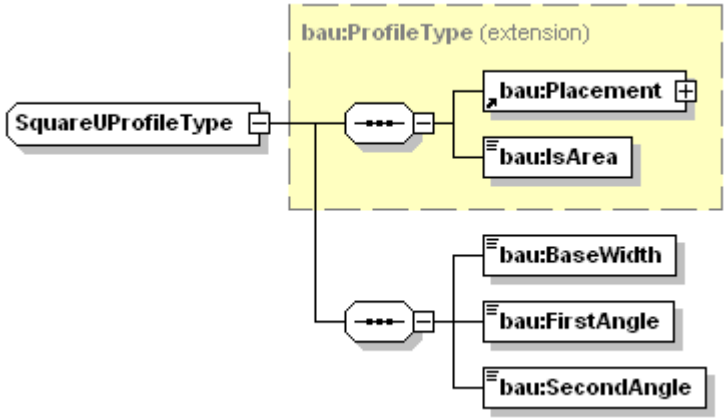
Grafik	
Diagramm	
Quellcode	<xs:complexType name="SquareUProfileType"> <xs:complexContent> <xs:extension base="bau:ProfileType"> <xs:sequence> <xs:element name="BaseWidth" type="xs:double"/> <xs:element name="FirstAngle" type="xs:double"/> <xs:element name="SecondAngle" type="xs:double"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Tabelle 3.20: Trapezförmiges U-Profil

Anwendungen eines trapezförmigen U-Profils sind z.B. Aussparungen für Balken, Dachpfetten oder Dachsparren.

V-Profil: Ein V-Profile wird von zwei Linien erzeugt, die an einem Punkt verbunden sind. Das Placement ist an dem Verbindungspunkt definiert. Ein V-Profil hat zwei Elemente: einen Profilwinkel zwischen den zwei Linien und einen Neigungswinkel (siehe Tabelle 3.21).

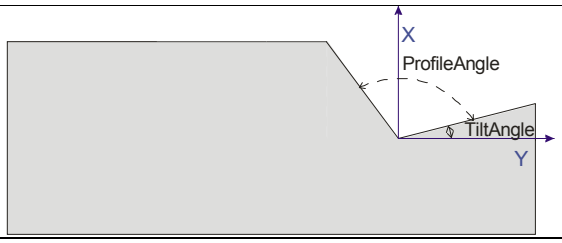
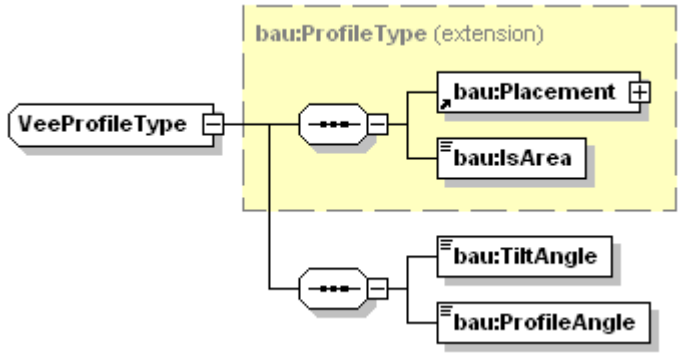
Grafik	
Diagramm	
Quellcode	<xs:complexType name="VeeProfileType"> <xs:complexContent> <xs:extension base="bau:ProfileType"> <xs:sequence> <xs:element name="TiltAngle" type="xs:double"/> <xs:element name="ProfileAngle" type="xs:double"/> </xs:sequence> </xs:extension> </xs:complexContent> </xs:complexType>

Tabelle 3.21: V-Profil

3.4.5 Allgemeine Featuredefinition in BauXML

Als Feature sind Extrusionskörper, Kegel und Ebenen zu unterscheiden, die nachfolgend kurz beschrieben werden. Außer einer optionalen Beschreibung „*bau:Description*“ hat jedes Feature die beiden Attribute „*Guid*“ und „*IsFabricated*“ (siehe Bild 3.4). Das boolesche Attribut dient der Kennzeichnung des Fertigungszustands des Features. Als Voreinstellung ist „*false*“ definiert. Nach der Bearbeitung wird „*IsFabricated*“ auf „true“ gesetzt.

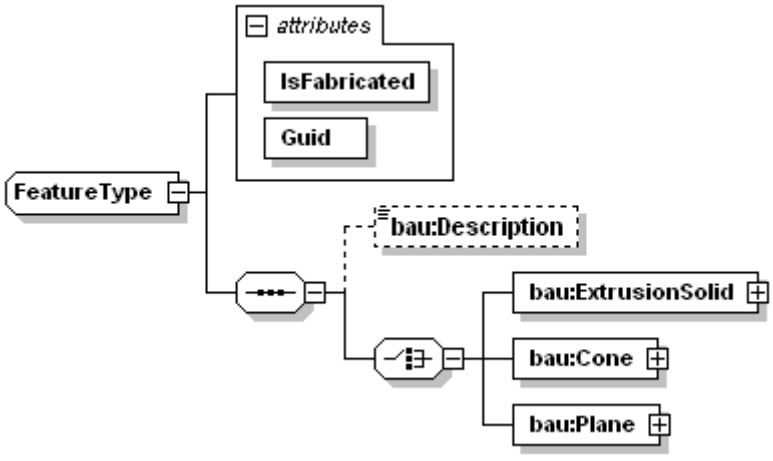


Bild 3.4: Featuredefinition

3.4.6 Extrusionskörper-Feature

Der überwiegende Teil der Fertigungsfeatures, die gefertigt werden müssen, entsprechen einem Extrusionskörper. Ein Extrusionskörper entsteht aus einem 2D-Profil und einem Vektor, entlang dessen das 2D-Profil extrudiert wird (siehe Abs. 2.1.3 „Sweep-Körper Modell“). Der Datensatz in BauXML besitzt neben der Angabe der Extrusionsrichtung durch einen normierten Vektor die Angabe der Extrusionslänge als Flächenabstand der beiden zueinander parallelen Deckflächen. Als 2D-Profil kann irgendein der in BauXML vordefinierten Grundprofile genommen werden.

3.4.7 Kegel-Feature

Das Placement ist im Mittelpunk des Kreises positioniert, das durch den „*BeginRadius*“ definiert ist. Der Anfangsradius kann auch kleiner als der Endradius sein. Ein Radius kann zur Beschreibung eines Spitzkegels die Größe 0,0 annehmen.

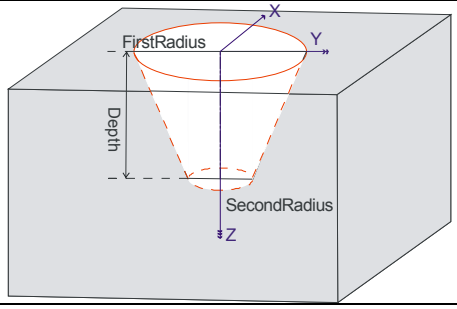
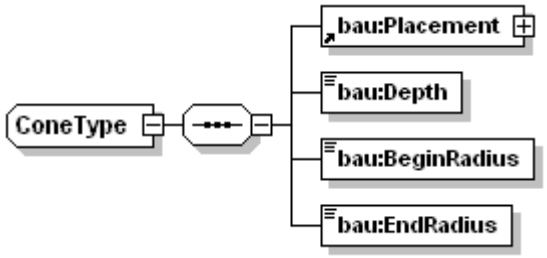
Grafik	
Diagramm	
Quellcode	<xs:complexType name="ConeType"> <xs:sequence> <xs:element ref="bau:Placement"/> <xs:element name="Depth" type="xs:double"/> <xs:element name="BeginRadius" type="xs:double"/> <xs:element name="EndRadius" type="xs:double"/> </xs:sequence> </xs:complexType>

Tabelle 3.22: Kegel-Feature

3.4.8 Ebene-Feature

Ebenen, die in der 3D-Computergraphik oft als „Clipping planes“ bezeichnet werden, definieren Schnittebenen durch Körper. Daher werden aus dem Ebene-Featuretyp meist Sägeaufträge abgeleitet. Eine Ebene ist eigentlich eine unbegrenzte Fläche mit einem konstanten Norma-

lenvektor. Sie ist definiert durch einen Punkt auf der Ebene und eine Normalenrichtung der Ebene. Die mathematische Beschreibung einer Ebene in Punkt-Normal-Form lautet:

$$\vec{n} \cdot (M - P) = 0, \tag{3.3}$$

wobei $\vec{n} \neq \vec{0}$ ein Normalenvektor der Ebene und P ein Punkt auf der Ebene ist (Bild 3.5). Eine Ebene entsteht dann aus allen Punkten M, die diese Bedingung erfüllen.

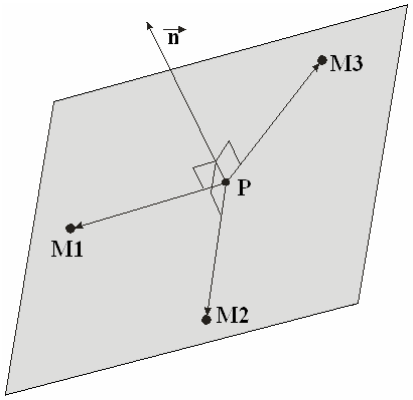


Bild 3.5: Ebenenbeschreibung im euklidischen Raum

Eine Ebene in BauXML wird durch drei Parameter festgelegt: Ein beliebiger Punkt auf der Ebene (*Point*), die Normalenrichtung der Ebene (*Normal*) und irgendeinen Richtungsvektor auf der Ebene (*RefDirection*). Eine Ebenenberandungslinie ist optional.

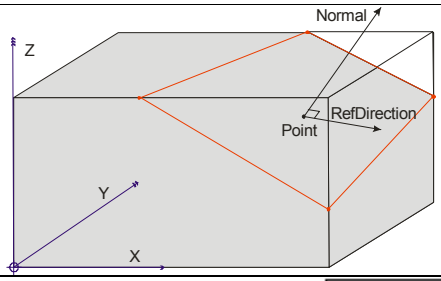
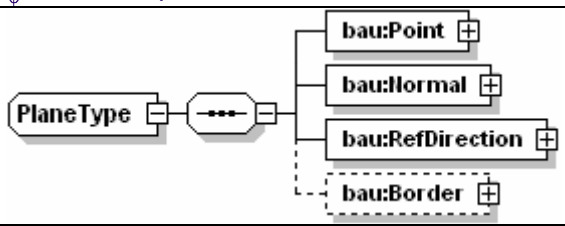
Grafik	
Diagramm	
Quellcode	<xs:complexType name="PlaneType"> <xs:sequence> <xs:element name="Point" type="bau:CartesianPoint3DType"/> <xs:element name="Normal" type="bau:DirectionType"/> <xs:element name="RefDirection" type="bau:DirectionType"/> <xs:element name="Border" minOccurs="0"> <xs:complexType> <xs:sequence> <xs:element name="Point" type="bau:CartesianPoint3DType" minOccurs="3" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType>

Tabelle 3.23: Ebene-Feature

3.5 Fazit

In der Arbeitsvorbereitung der Fertigung wurden bisher meistens 2D-Zeichnungen als „Eingangsinformationen“ benutzt. Der Arbeitsvorbereitungs-Fachmann muss die Bearbeitungsprogramme für alle Mauerwerksteine festlegen. Änderungen eines Bearbeitungsobjektes, müssen durch Anpassung der betroffenen Werkzeuge und der NC-Programme nachgeführt werden, das heißt der NC-Quelltext muss in der Arbeitsvorbereitung geändert werden. Eine solche Lücke im Informationsfluss behindert ein durchgängiges Engineering, führt zu einem nicht notwendigen und kostspieligen Pflegebedarf und sollte durch ein digitalisiertes Produktmodell ersetzt werden. Das IFC-Produktmodell hat zwar zahlreiche Informationen, viele sind jedoch für die Fertigung uninteressant und irrelevant. Fehlende Steinobjekte beispielsweise, sowie eine zeitaufwändige Erweiterung und Zertifizierung fordert ein neues Datenformat, um die Wünsche hinsichtlich der Fertigung zu befriedigen.

In der Arbeit wurde ein Feature-basiertes neutrales Datenformat BauXML mit Hinsicht auf die Fertigung in der XML-Beschreibungssprache konzipiert und neu entwickelt. Das BauXML-Modell, das modular aufgebaut und skalierbar ist, ermöglicht den Informationsaustausch von Steinelementen zwischen der Planung und der Fertigung. Es erlaubt die Anwendung, sowohl für ganz einfache Mauerwerksteine als auch für Mauerwerksteine mit relativ komplexer Form. Das Informationsmodell BauXML ist sehr strikt objektorientiert aufgebaut, das heißt, es besitzt ein Typ-Instanzenkonzept. Die einmal vordefinierten Features können mehrfach referenziert werden. Mit seinen Hierarchiekonzepten erlaubt es die Abbildung tiefer funktionaler Hierarchien und aus der Sicht der Softwareentwicklung eine Klassenhierarchie. Das Informationsmodell BauXML beinhaltet bewusst keine Informationen, weder über die in der Fertigung einzusetzenden konkreten Werkzeuge oder Maschinen (CNC-Maschinen oder ein Industriefertigungsroboter), noch über die Fertigungsreihenfolge. BauXML ist von den Fertigungsausrüstungen unabhängig und fokussiert auf fertigungsfreundliche CAD-Daten.

Das Konzept von BauXML erweitert quantitativ und qualitativ die Nutzung von Informationen, Wissen und Erfahrung. Damit wird die innerbetriebliche Kommunikation zwischen Konstruktion, Arbeitsplanung und Fertigung stark unterstützt.

4 Extraktion und Manipulation der IFC-Daten sowie Generierung des BauXML-Modells

4.1 Grundlagen der 3D-Computergraphik

Hier werden einige Grundbegriffe und ihre Notation, die in der Arbeit häufig benutzt wurden, zusammenfassend dargestellt.

4.1.1 Koordinatensystem

Zur Beschreibung der Objektlage wird in IFC und BauXML wie üblich das orthonormale kartesische rechtshändige Koordinatensystem benutzt. Die Basisvektoren haben die Länge Eins und stehen paarweise senkrecht aufeinander.

4.1.2 Vektoren und Ortsvektoren

Wird beispielsweise ein Körper durch ein 2D-Extrusionsprofil dargestellt, so müssen sowohl der Betrag der Translation als Maßzahl als auch die Richtung, in die sich das 2D-Profil bewegt, angegeben werden. In BauXML wird der Typ „*VectorType*“ (siehe Abs. 3.4.2) definiert, welcher eine Richtung (Orientation) und einen Betrag (Magnitude) beinhaltet. In IFC ist die Entität „*IfcVector*“ analog definiert. Diese beiden Angaben können in einem Vektor zusammengefasst werden.

Ein Vektor wird durch einen Spaltenvektor festgelegt. In der Spalte werden die senkrechten Projektionen auf die jeweiligen Achsen des Koordinatensystems angeordnet.

$$\mathbf{V} = \begin{pmatrix} \mathbf{V}_x \\ \mathbf{V}_y \\ \mathbf{V}_z \end{pmatrix} \quad \text{mit} \quad \mathbf{V}_x, \mathbf{V}_y, \mathbf{V}_z \in \mathbb{R} \tag{4.1}$$

Spezielle Richtungsvektoren sind die Basisvektoren eines Koordinatensystems.

$$\mathbf{e_x} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad \mathbf{e_y} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad \mathbf{e_z} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \text{ und } \mathbf{e_z} = \mathbf{e_x} \times \mathbf{e_y} \tag{4.2}$$

Ein Punkt **P** ist in einem kartesischen Koordinatensystem durch den Ortsvektor eindeutig bestimmt:

$$\mathbf{P} = \mathbf{P}_x\mathbf{e}_\mathbf{x} + \mathbf{P}_y\mathbf{e}_\mathbf{y} + \mathbf{P}_z\mathbf{e}_\mathbf{z} = \begin{pmatrix} \mathbf{P}_x \\ \mathbf{P}_y \\ \mathbf{P}_z \end{pmatrix} \text{ mit } \mathbf{P}_x, \mathbf{P}_y, \mathbf{P}_z \in \mathbb{R} \tag{4.3}$$

4.1.3 Dreidimensionale Transformationen

Bei der Implementierung ist neben den rein geometrischen Attributen auch die Manipulation einer Darstellung von großer Bedeutung. Damit sind nicht nur die Visualisierung und Animation von BauXML-Objekten gemeint, bei denen grafische Objekte kontinuierlich in ihrer Lage oder Größe verändert werden, sondern auch die Manipulation der IFC-Daten und die Erzeugung des BauXML-Modells, bei denen die Objekte an eine andere Stelle verdreht oder verschoben werden. Dabei spielt die Transformation eine sehr große Rolle.

4.1.3.1 Affine Transformation

Eine affine Transformation ist eine Abbildung (Transformation) zwischen zwei Vektorräumen, die Kollinearitäten und Abstandsverhältnisse bewahrt [WEBE]. Bewahrung der Kollinearität bedeutet eine lineare Transformation. Das Abstandsverhältnis entspricht einer Translation. Eine affine Transformation setzt sich also aus einer linearen Transformation und einer Translation zusammen. Wird die lineare Transformation als Matrix-Vektor Produkt beschrieben, so ergibt sich die affine Transformation

$$\Phi(\mathbf{V}) = \mathbf{MV} + \mathbf{T} \tag{4.4}$$

Die affinen Transformationen umfassen alle linearen Transformationen (z.B. Rotation, Skalierung, Scherung) mit $\mathbf{T} = 0$ und ergänzen diese um die Translation mit $\mathbf{T} \neq 0$.

Lineare Transformation

Eine lineare Transformation hat die Eigenschaft, dass Geraden vor der Transformation nach der Transformation auch Geraden bleiben. Für eine lineare Abbildung gilt:

$$\Phi : \mathbb{R}^3 \to \mathbb{R}^3; \mathbf{V} \to \mathbf{V'} \tag{4.5}$$

mit

$$\forall \mathbf{V_1}, \mathbf{V_2} \in \mathbb{R}^3, \alpha, \beta \in \mathbb{R} \tag{4.6}$$

Hierfür gilt die Beziehung

$$\Phi(\alpha\mathbf{V}_1 + \beta\mathbf{V_2}) = \alpha\Phi(\mathbf{V}_1) + \beta\Phi(\mathbf{V_2}) = \alpha\mathbf{V_1'} + \beta\mathbf{V_2'}. \tag{4.7}$$

Das heißt: es soll egal sein, ob die Vektoren vor oder nach der Transformation gekürzt oder addiert werden.

Wird eine Matrix auf der linken Seite eines Vektors multipliziert:

$$\Phi(\mathbf{V}) = \mathbf{MV} \text{ mit } \mathbf{V} = \alpha\mathbf{V}_1 + \beta\mathbf{V}_2 \text{ und } \mathbf{M}_{3\times3} \tag{4.8}$$

ergibt sich:

$$\begin{aligned} \Phi(\alpha\mathbf{V}_1 + \beta\mathbf{V}_2) &= \mathbf{M}(\alpha\mathbf{V}_1 + \beta\mathbf{V}_2) \\ &= \mathbf{M}\alpha\mathbf{V}_1 + \mathbf{M}\beta\mathbf{V}_2 \\ &= \alpha\Phi(\mathbf{V}_1) + \beta\Phi(\mathbf{V}_2) \end{aligned} \tag{4.9}$$

Diese Eigenschaft ist bei der Transformation der Geometrie von besonderer Bedeutung. Die Geometrie eines Objekts ist immer durch eine Reihe von Definitionspunkten und Richtungen zusammengefasst. Die Geometrietransformation ist also genau das Ergebnis einer Transformation aller Definitionsobjekte. Durch die Transformation darf die Linearität der Geometrie auf keinen Fall zerstört werden.

Translation

Zur Transformation eines Punkts **P** gehören eine lineare Transformation und eine Translation. Sie wird wie folgt beschrieben

$$\mathbf{P}' = \mathbf{MP} + \mathbf{T}, \text{ mit } \mathbf{M}_{3\times3}, \ \mathbf{T}_{3\times1} \tag{4.10}$$

Nach einer zweiten Transformation ergibt sich dann eine etwas komplizierte Form

$$\begin{aligned} \mathbf{P}' &= \mathbf{M}_2(\mathbf{M}_1\mathbf{P} + \mathbf{T}_1) + \mathbf{T}_2 \\ &= \mathbf{M}_2\mathbf{M}_1\mathbf{P} + \mathbf{M}_2\mathbf{T}_1 + \mathbf{T}_2 \end{aligned} \tag{4.11}$$

Hieran ist zu sehen, dass eine Translation nicht mit anderen Transformationen (Rotation, Skalierung, Scherung) durch eine einfache Matrizenmultiplikation kombiniert werden kann. Eine solche Kombinierung ist aber wünschenswert, um mehrere hintereinander angeordnete affine Transformationen effizient in einer einheitlichen Matrixform durchführen zu können. So werden die so genannten **homogenen Koordinaten** eingeführt. Punkte und Vektoren werden durch vier statt drei Koordinaten spezifiziert. Dabei werden 4x4- statt 3x3-Matrizen benutzt. Eine 4x4 Transformationsmatrix **F** wird dann auf der Basis von der 3x3 Rotationsmatrix **M** und der 3D Translationsmatrix **T** aufgebaut:

$$\mathbf{F} = \begin{pmatrix} \mathbf{M} & \mathbf{T} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} & \mathbf{M}_{13} & \mathbf{T}_x \\ \mathbf{M}_{21} & \mathbf{M}_{22} & \mathbf{M}_{23} & \mathbf{T}_y \\ \mathbf{M}_{31} & \mathbf{M}_{32} & \mathbf{M}_{33} & \mathbf{T}_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.12}$$

Mit der affinen Transformation kann also die Formel (4.4):

$$\Phi(\mathbf{V}) = \mathbf{MV} + \mathbf{T}$$

mit einer einheitlichen Transformationsmatrix F wie folgt zusammengefasst werden:

$$\Phi(\mathbf{V}) = \mathbf{FV} \tag{4.13}$$

Damit werden komplexe Transformationen aus mehreren einfachen Transformationen durch die Matrixmultiplikation zusammengesetzt.

4.1.3.2 Geometrietransformationen

Die wesentlichen Transformationen werden in Bild 4.1 zusammengestellt.

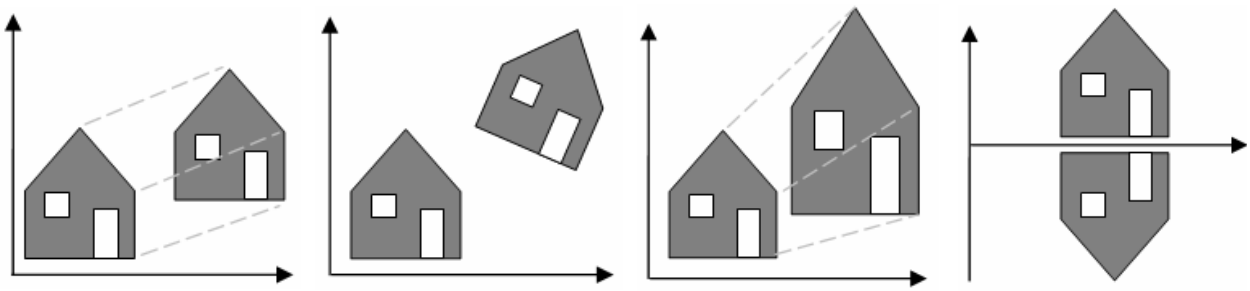


Bild 4.1: Translation, Rotation, Skalierung und Spiegelung

Entsprechend gilt die jeweilige Transformationsmatrix:

Translation um den Vektor $\left(\mathbf{T}_x, \mathbf{T}_y, \mathbf{T}_z\right)^T$: $\mathbf{F} = \begin{pmatrix} 1 & 0 & 0 & \mathbf{T}_x \\ 0 & 1 & 0 & \mathbf{T}_y \\ 0 & 0 & 1 & \mathbf{T}_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$ (4.14)

Skalierung mit den Faktoren $\left(\mathbf{S}_x, \mathbf{S}_y, \mathbf{S}_z\right)$: $\mathbf{F} = \begin{pmatrix} \mathbf{S}_x & 0 & 0 & 0 \\ 0 & \mathbf{S}_y & 0 & 0 \\ 0 & 0 & \mathbf{S}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ (4.15)

Rotation um die x-Achse mit Winkel φ: $\mathbf{F}_x(\varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) & 0 \\ 0 & \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ (4.16)

Rotation um die y-Achse Achse mit Winkel φ: $\mathbf{F}_y(\varphi) = \begin{pmatrix} \cos(\varphi) & 0 & \sin(\varphi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\varphi) & 0 & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ (4.17)

Rotation um die z-Achse Achse mit Winkel φ: $\mathbf{F}_z(\varphi) = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) & 0 & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ (4.18)

Spiegelung: $\mathbf{F}_{xy} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $\mathbf{F}_{xz} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, $\mathbf{F}_{yz} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ (4.19)

4.1.3.3 Framearithmethik und Transformation

Eine der am häufigsten verwendeten Transformationen ist die Änderung von Koordinatensystemen. Wird z.B. eine komplexe Szene betrachtet, so ist es sehr aufwendig, die Objekte in einem Weltkoordinatensystem zu definieren. Die Lösung des Problems liegt in der Verwendung eines so genannten *Modellkoordinatensystems*, in dem das jeweilige Objekt definiert wird. Die Relation zwischen zwei Koordinatensystemen, also die Lage eines Koordinatensystems bezogen auf ein anderes wird oft als Frame bezeichnet [WÖRN] (siehe. Bild 4.2). Durch das Frame wird das Objekt in das Bezugs- oder Weltkoordinatensystem transformiert.

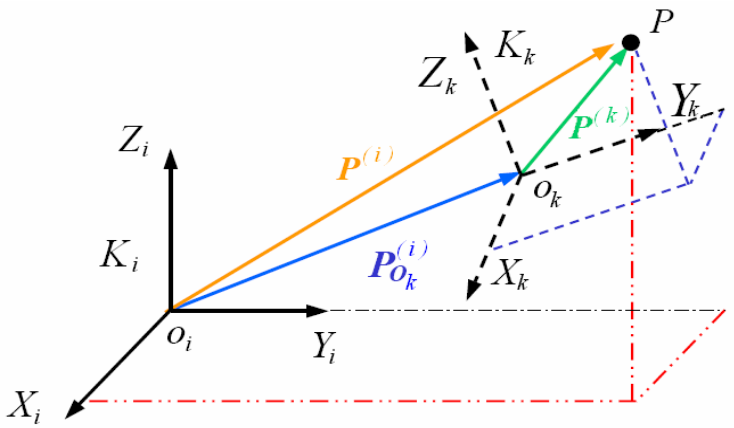


Bild 4.2: Relative Lage zweier Koordinatensysteme zueinander

Die Lage eines Koordinatensystems K_k im Raum ist allein durch die Angabe des Ortsvektors $\mathbf{P}_{O_k}^{(i)}$ [1] vom Ursprung des Bezugskoordinatensystems $\mathbf{O}_i$ zum eigenen Ursprung $\mathbf{O}_k$ nicht eindeutig bestimmt. Es muss noch angegeben werden, wie das Koordinatensystem K_k im Raum ausgerichtet ist - die Orientierung. Durch die Angabe der Basisvektoren $\mathbf{X}_k, \mathbf{Y}_k, \mathbf{Z}_k$ in Koordinaten des ruhenden Bezugssystems K_i wird definiert, wie K_k gegen K_i gedreht ist. Eine affine Transformation kann hier in einer homogenen 4x4-Matrix zusammengefasst werden:

$$\mathbf{F}_i^k = \begin{pmatrix} \mathbf{X}_k^{(i)} & \mathbf{Y}_k^{(i)} & \mathbf{Z}_k^{(i)} & \mathbf{P}_{ik}^{(i)} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.20}$$

Diese homogene 4x4-Matrix $\mathbf{F}_i^k$ wird oft als **Frame** benannt. Er repräsentiert die Lage eines Koordinatensystems K_k bezogen auf ein anderes Koordinatensystems K_i. Hier sind die Basisvektoren von K_k und der Ortsvektor vom Ursprung K_i zum Ursprung K_k in Koordinaten von K_i angegeben. Damit ist die Lage von K_k in Bezug auf K_i vollständig beschrieben. Nun gilt:

$$\mathbf{P}^{(i)} = \mathbf{F}_i^k \mathbf{P}^{(k)} \tag{4.21}$$

also

$$\begin{pmatrix} P_x^{(i)} \\ P_y^{(i)} \\ P_z^{(i)} \\ 1 \end{pmatrix} = \begin{pmatrix} X_{xk}^{(i)} & Y_{xk}^{(i)} & Z_{xk}^{(i)} & P_{xO_k}^{(i)} \\ X_{yk}^{(i)} & Y_{yk}^{(i)} & Z_{yk}^{(i)} & P_{yO_k}^{(i)} \\ X_{zk}^{(i)} & Y_{zk}^{(i)} & Z_{zk}^{(i)} & P_{zO_k}^{(i)} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x^{(k)} \\ P_y^{(k)} \\ P_z^{(k)} \\ 1 \end{pmatrix} \tag{4.22}$$

Ist ein Vektor $\mathbf{P}^{(i)}$ in K_i gegeben und der entsprechende Vektor $\mathbf{P}^{(k)}$ im Koordinatensystem K_k gesucht, gilt:

$$\mathbf{P}^{(k)} = [\mathbf{F}_i^k]^{-1} \mathbf{P}^{(i)} = \mathbf{F}_k^i \mathbf{P}^{(i)} = \begin{pmatrix} \mathbf{X}_i^{(k)} & \mathbf{Y}_i^{(k)} & \mathbf{Z}_i^{(k)} & \mathbf{P}_{ik}^{(k)} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.23}$$

Sind weitere Koordinatensysteme zwischen K_i und K_k definiert, gilt entsprechend:

[1] Der in Klammer gesetzte hochgestellte Index soll hier und im Folgenden ausdrücken, in welchem Koordinatensystem ein Vektor beschrieben ist.

$$\mathbf{F}_i^k = \mathbf{F}_i^{i+1}\mathbf{F}_{i+1}^{i+2}.....\mathbf{F}_{k-2}^{k-1}\mathbf{F}_{k-1}^{k} \tag{4.24}$$

und für die inverse Transformation

$$\mathbf{F}_k^i = [\mathbf{F}_i^k]^{-1} = [\mathbf{F}_{k-1}^k]^{-1}[\mathbf{F}_{k-2}^{k-1}]^{-1}.....[\mathbf{F}_{i+1}^{i+2}]^{-1}[\mathbf{F}_i^{i+1}]^{-1} = \mathbf{F}_k^{k-1}\mathbf{F}_{k-1}^{k-2}.....\mathbf{F}_{i+2}^{i+1}\mathbf{F}_{i+1}^{i} \tag{4.25}$$

Die Frametransformation wird in der Implementierung sehr häufig benutzt. Wie im Kapitel 2 vorgestellt, wird die Geometrie jeder IFC-Entität im lokalen Koordinatensystem definiert. Die Entität kennt die Transformation zum Koordinatensystem ihres Container-Objekts durch das „*IfcLocalPlacement*“ (siehe Abs. 2.4.1). Nach der IFC-Spezifikation besteht das „*IfcLocalPlacement*“ aus zwei Vektoren (*Axis*, *RefDirection*) und einen Punkt (*Location*). In Bild 4.3 wird das „*IfcLocalPlacement*“ einer 3D-Darstellung anschaulich dargestellt.

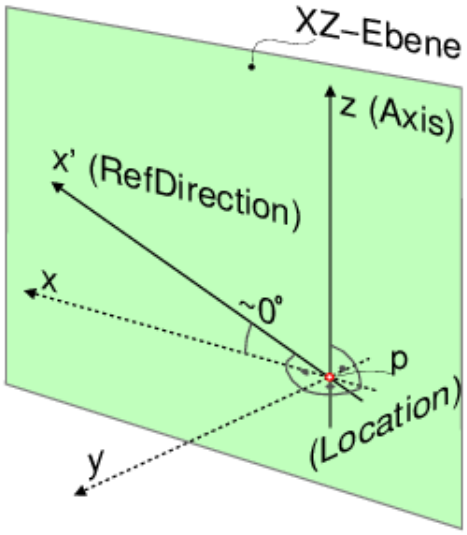


Bild 4.3: „*IfcLocalPlacement*“ in 3D-Darstellung

Der Punkt **P** (*Location*) definiert die Stelle, wo der Ursprung des lokalen Koordinatensystems innerhalb des Koordinatensystems ihres Container-Objekts liegt. Die *Axis* entspricht exakt der Z-Achse. Die *RefDirection* ist aber nicht unbedingt die X-Achse sondern nur eine auf der XZ-Ebene liegende Richtung. Die genauen X- und Y-Achsen müssen deswegen berechnet werden. In Vektorschreibweise werden der Punkt und die beiden Richtungen im 3D-Raum wie folgt beschrieben:

Location: $\mathbf{P} = (\mathbf{P}_x, \mathbf{P}_y, \mathbf{P}_z)^T$

RefDirection: $\mathbf{X}' = (\mathbf{X}'_x, \mathbf{X}'_y, \mathbf{X}'_z)^T$ mit $\mathbf{X}'_x, \mathbf{X}'_y, \mathbf{X}'_z \in \mathbb{R}$,

Axis: $\mathbf{Z} = (\mathbf{Z}_x, \mathbf{Z}_y, \mathbf{Z}_z)^T$ mit $\mathbf{Z}_x, \mathbf{Z}_y, \mathbf{Z}_z \in \mathbb{R}$.

Die Richtung der Y-Achse $\mathbf{Y} = (\mathbf{Y}_x, \mathbf{Y}_y, \mathbf{Y}_z)^T$ mit $\mathbf{Y}_x, \mathbf{Y}_y, \mathbf{Y}_z \in \mathbb{R}$ berechnet sich aus dem Kreuzprodukt von *Axis* und *RefDirection*: $\mathbf{Y} = \mathbf{Z} \times \mathbf{X}'$. Die Richtung der X-Achse berechnet sich dann aus dem Kreuzprodukt von der neu berechneten Y-Achse und *Axis*: $\mathbf{X} = \mathbf{Y} \times \mathbf{Z}$. Nun sind die drei genauen Achsen $\mathbf{X}$, $\mathbf{Y}$ und $\mathbf{Z}$ sowie der Ursprung durch die *Location* berechnet. Das „*IfcLocalPlacement*“ repräsentiert dann eine Frametransformation.

$$\mathbf{F} = \begin{pmatrix} \mathbf{M} & \mathbf{P} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{X}_x & \mathbf{Y}_x & \mathbf{Z}_x & \mathbf{P}_x \\ \mathbf{X}_y & \mathbf{Y}_y & \mathbf{Z}_y & \mathbf{P}_y \\ \mathbf{X}_z & \mathbf{Y}_z & \mathbf{Z}_z & \mathbf{P}_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.26}$$

Die Lagebeziehungen werden in Bild 4.4 veranschaulicht. Eine Wand hat ein „*IfcLokalPlacement*“ $\mathbf{L}_{\text{Wall}}$, das die Transformation zu ihrem „*IfcBuildingStorey*“ Objekt definiert. Das „*IfcBuildingStorey*“ kennt wiederum seine Transformation $\mathbf{L}_{\text{Storey}}$ zu seinem Container „*IfcBuilding*“.

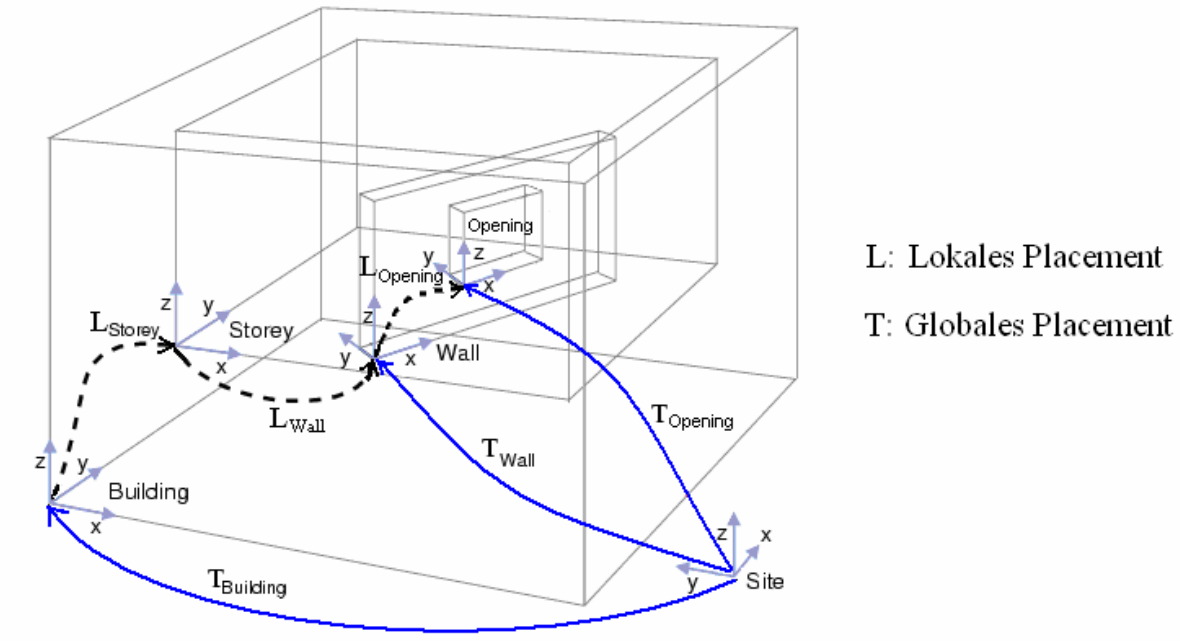


Bild 4.4: „*IfcLocalPlacements*“ und die „eigene“ daraus errechnete globale Transformation

Ein ständiges Navigieren durch den recht komplexen IFC-Entitäten-Baum ist softwaretechnisch schlecht handhabbar und kostet viel Speicher. Obwohl die relativen Transformationen als „*IfcLocalPlacement*“ in IFC schon vorhanden sind, werden deshalb die verketteten Transformationen jeder Entität beim Einlesen der IFC-Datei und noch vor der Übergabe an die Applikation mit einer rekursiven Funktion zu einer Matrix aufmultipliziert. Der Zusammenhang lässt sich am Beispiel von Bild 4.4 verdeutlichen:

$$\mathbf{T}_{wall} = \mathbf{T}_{Building}\mathbf{L}_{Storey}\mathbf{L}_{wall} \tag{4.27}$$

und ebenfalls gilt:

$$\mathbf{T}_{Opening} = \mathbf{T}_{wall}\mathbf{L}_{Opening} = \mathbf{T}_{Building}\mathbf{L}_{Storey}\mathbf{L}_{Wall}\mathbf{L}_{Opening} \tag{4.28}$$

Weil es sich hier nur um einen Bauplatz (*IfcSite*) handelt, ist das Koordinatensystem des Bauplatzes (*IfcSite*) absolut im globalen Koordinatensystem des ganzen Projekts definiert und normalerweise sind die beiden identisch. Deshalb wird das Placement bezogen auf den Bauplatz als globales Placement bezeichnet. Hier steht $\mathbf{T}$ für eine globale Beschreibung und $\mathbf{L}$ für eine lokale Beschreibung.

Somit ergibt sich eine Transformation vom lokalen Koordinatensystem jeder IFC-Entität in das globale Koordinatensystem, z.B. $\mathbf{T}_{Wall}$ und $\mathbf{T}_{Building}$.

4.2 Konzeption des Arbeitsprozesses

Ein Hausmodell wird aus einem CAAD-Projekt im IFC-Format übertragen. Nach dem Einlesen und der Visualisierung des Modells wird die Umsetzung der Arbeit in vier Schritte unterteilt:

1) Extraktion arbeitsrelevanter Elemente: Alle Wandelemente, Wandöffnungen und an die Wände angehefteten interessierenden Objekte werden aus dem virtuellen Hausmodell herausgefiltert. Alle für den weiteren Verlauf nicht benötigten Daten werden dabei bewusst übergangen.

2) Bereinigung der Wandgeometrie: Die Wandverschnitte (Stumpfstoß oder Gehrungsstoß) werden fertigungs- und montagegerecht bearbeitet und korrigiert.

3) Segmentierung der Wandelemente: Wände werden unter bestimmten Randbedingungen in einzelne Steine aufgeteilt. Als sinnvolle Randbedingungen sind die Umsetzung der statischen und montagetechnischen Problemstellungen sowie die Fabriksvorgaben anzusehen.

4) Generierung des BauXML-Modells: Die Wandgeometrie wird auf die einzelnen Steine übertragen. Die zu entfernenden Teile werden als Fertigungsfeature generiert und zugeordnet. Die Steininformationen zusammen mit generellen Informationen über das Projekt, das Gebäude, die Stockwerke und Wände werden im BauXML-Modell zusammengefasst.

Der ganze Prozess wird anhand des schematischen Bildes 4.5 deutlicher.

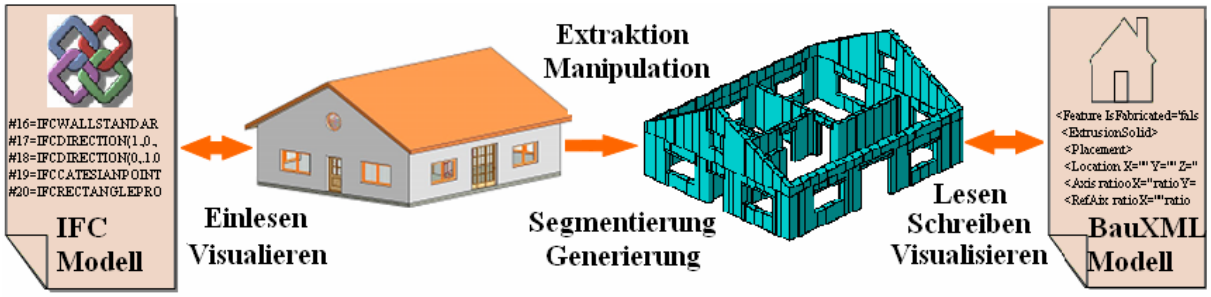


Bild 4.5: Arbeitsprozess vom IFC zum BauXML-Modell

4.3 Manipulation der Wandverschnitte in IFC

4.3.1 Problemstellung

Als Wandverschnitt wird in den CAAD-Systemen der Übergang der untereinander verbundenen Wandelemente bezeichnet. In der Bautechnik spricht man hierbei von einem Wandanschluss. Generell werden zwei Arten unterschieden: **Stumpfstoß** und **Gehrungsstoß**.

Die Ausführung von Wandanschlüssen ist abhängig von der Bauweise, dem Baustoff und den Vorgaben der Statik. Bei der Verwendung von kleinformatigen Steinen wird das Mauerwerk bei einer Eckverbindung im Verbund ausgeführt. Für die Darstellung dieses vollständigen Verbunds ist hier ein Gehrungsstoß richtig. Mit geschosshohen Steinen ist nur einfaches Aneinanderstoßen möglich. Solche Wände müssen dann durch andere Mechanismen verbunden werden, z.B. vollflächiges Verkleben, kraftschlüssige Verbindung. Bei der Holzständerbauweise werden die Wände stumpf aneinander gestoßen und verschraubt.

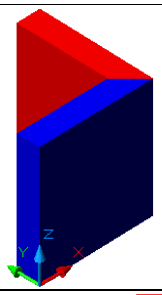
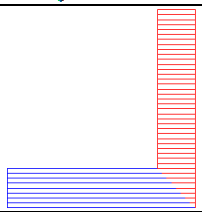
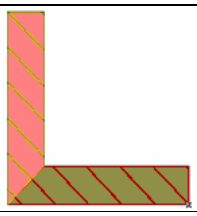
3D Ansicht		
2D Ansicht		

Tabelle 4.1: Gehrungsstoß im originalen IFC-Format

Die unterschiedlichen Anforderungen an verschiedene Bauweisen werden standardmäßig von CAAD-Systemen aber nicht berücksichtigt. Sind dort die Wände untereinander verbunden, so werden sie geometrisch mit einer Gehrung in die IFC-Daten übernommen (siehe Tabelle 4.1).

Auf dem Bildschirm sieht es so aus, als ob die Wandelemente miteinander verschmelzen. Ein Gehrungsstoß ist allerdings in der Praxis eher selten. Besonders unpraktikabel ist ein Gehrungsverschnitt bei einem Übergang mit Materialwechsel. Bei dem Fall werden die Wände nicht so ausgeführt, wie sie die CAAD-Systeme wiedergeben.

Aus konstruktiver, fertigungs- und montagetechnischer Sicht müssen die Wandverschnitte vor allen weiteren Schritten hin zum BauXML-Modell bearbeitet und korrigiert werden.

4.3.2 Vorüberlegungen und Vereinbarungen

Im Hinblick auf eine effektive Vorfertigung sollten die Anzahl der Schnittsteine, die Anzahl der Sägeschnitte pro Stein und die Schnittfugenlänge gering gehalten werden. Deswegen werden die folgenden Vereinbarungen getroffen [GEIG]:

1. Alle Wandverschnitte sollen prinzipiell als Stumpfstöße interpretiert werden.
2. Liegt der Verbindungswinkel zwischen 0 und 90 Grad, so ist ein Stumpfstoß die praktikablere Lösung. Im Fall der Verbindung zwischen 90 und 180 Grad kann ein Gehrungsschnitt die sinnvollere Alternative sein, um die Schnittfugenlänge zu reduzieren. Die Diagonalen der schraffierten Parallelogramme in Bild 4.6 stehen für einen Gehrungsschnitt. Es ist deutlich zu erkennen, wie bei dem linken Beispiel die Schnittkanten eines Stumpfstoßes immer länger werden, je größer der Winkel wird. Im rechten Beispiel ist der Winkel zwischen 0-45 Grad. Hier ist der Gehrungsschnitt zu beachten. Je kleiner der Winkel wird, desto länger wird die Schnittkante für eine Gehrung. Dieser als „kritischer Winkel“ bezeichnete Parameter kann je nach der praktischen Situation im Programm voreingestellt werden.

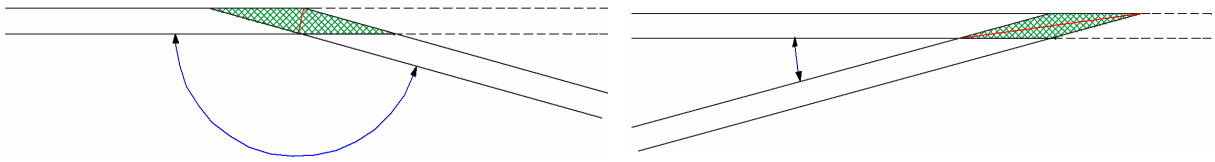


Bild 4.6: Verschneidungswinkel [GEIG]

3. Wände mit unterschiedlichen Längen/Stärken: Bei der Verschneidung zweier Wände mit unterschiedlicher Stärke hat die dickere Wand den Vorrang vor der dünneren Wand. Die dünnere Wand wird also verkürzt. Dies entspricht den baupraktischen Gegebenheiten, da eine dickere Wand tragende oder aussteifende Funktionen hat und damit einer Trennwand im Normalfall vorgezogen wird. Sind zwei Wände mit unterschiedlichen Längen vorhanden, dann werden die ursprünglichen Längen der beiden nach der Errechnung der Schnittpunkte miteinander verglichen. Dabei wird die Stärke

des Verschnittes von der längeren Wand subtrahiert. Tritt in beiden Verfahren der Fall auf, dass zwei gleiche Werte vorliegen, so wird entsprechend der Reihenfolge aus den Verbindungsinformationen der IFC-Daten der Vorrang ermittelt.

4. Weiterhin ist fertigungstechnisch die Länge der einzelnen Steinelemente zu berücksichtigen. Von Bedeutung sind Verschnitte mit sehr kurzen Wandelementen. Dabei muss vermieden werden, dass ein kurzes Wandstück durch einen Verschnitt noch zusätzlich gekürzt wird.

4.3.3 Verschneidungs-Algorithmen und Vorgehensweise

Aus den oben genannten Randbedingungen und Überlegungen stehen insgesamt vier Verschneidungs-Algorithmen zur Auswahl:

1. Verschneidung durch Vergleich der Wandlänge
2. Verschneidung durch Kontrolle der Wandstärke
3. Verschneidung unter Zuhilfenahme der Materialien
4. Verschneidung unter Zuhilfenahme der Wandbezeichnungen.

Die vier Algorithmen greifen auf die gleiche Methode zur Bestimmung der Schnittpunkte zurück. Die ersten beiden Algorithmen beziehen sich auf die Geometrie und basieren auf der dritten Vereinbahrung. In der Voreinstellung besteht die Möglichkeit, Prioritäten sowohl für die Materialienliste, wie auch für die Liste der Namen zu vergeben. Dies geschieht durch Ändern der Reihenfolge der Einträge, wobei der oberste Eintrag auch die höchste Priorität besitzt. Bei diesen beiden Verschneidungs-Algorithmen werden zwei zu verschneidende Wände mit der Reihenfolge der Prioritätenliste verglichen und daraus die Wand bestimmt, die den Vorrang hat.

Schritt 1: Koordinatentransformation in gemeinsames Koordinatensystem

Um nun die Verschneidung zweier Wandelemente miteinander zu berechnen, müssen die Koordinaten der beiden Wandelemente in ein gemeinsames Koordinatensystem transformiert werden, damit die Schnittpunkte errechnet werden können. Anschließend müssen die Schnittpunkte in die lokalen Koordinatensysteme der beiden Wände zurück gerechnet werden, damit die Polylinie der jeweiligen Wand neu erstellt werden kann.

Schritt 2: Überprüfung der Schnittwinkel

Die Durchführung der Verschneidung erfolgt immer zwischen zwei Wänden. Hierzu werden als erstes die Führungslinien betrachtet. Aus den Führungslinien wird der Schnittwinkel zwischen den beiden Wänden errechnet. Dieser Winkel ist bei der Berechnung eines Stumpfstoßes für die Verarbeitung des Grenzwinkels notwendig. Abhängig vom errechneten Schnittwinkel bzw. den Verschneidungsvoreinstellungen wird ein Stumpfstoß oder ein Gehrungsstoß aus den Schnittpunktkoordinaten gebildet (vgl. Randbedingung und Vorüberlegung 2).

Schritt 3: Verschneidungsberechnung

Die prinzipielle Vorgehensweise bei der Verschneidungsberechnung ist in der Tabelle 4.2 schematisch dargestellt. Aus den vier Eckpunkten des Grundrisses einer Wand werden zwei Strahlen gebildet, die in Richtung der Führungslinie verlaufen. Diese Strahlen werden mit den Strahlen einer zweiten Wand zum Schnitt gebracht. Dadurch entsteht das Verschneidungsviereck.

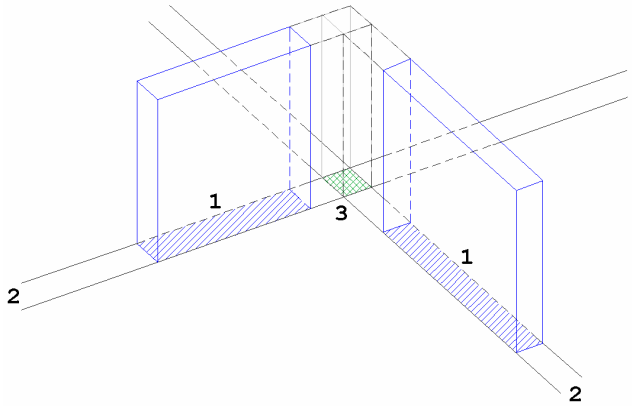
	1. Grundriss eines Wandelementes mit den vier Eckpunkten der geschlossenen Polylinie 2. Strahlen, die durch jeweils zwei Eckpunkte verlaufen 3. Verschneidungsviereck aus den vier Schnittpunkten der Strahlen

Tabelle 4.2: Algorithmus der Verschneidungsberechnung

Schritt 4: Bestimmung und Anpassung des neuen Wandkörpers

Aus dem in Schritt 3 entstandenen Verschneidungsviereck werden die neuen Eckpunkte der Wandverbindung bestimmt, mit denen der neue Wandkörper hergestellt wird. Aus welchem Wandelement die Verschneidungsgeometrie subtrahiert wird, wird von dem angewandten Algorithmus entschieden.

Schritt 5: Bestimmung und Anpassung der neuen Führungslinie

Als letztes wird die Führungslinie der modifizierten Wand neu bestimmt.

4.4 Wandsegmentierung

4.4.1 Randbedingungen

Als Randbedingungen gelten die verwendeten Baustoffe, die in der Fabrik vorhandenen Steinmaße und die durch Wandöffnungen entstehenden Abhängigkeiten.

1) Material: Als Material wurden hier die Stahlbetonfertigteile und geschosshohe Porenbetonsteine berücksichtigt. Es wurde angenommen, dass bei Stahlbetonfertigteilen keine Stoßfugen durch Öffnungen gehen sollen. Das Fertigteil mit Öffnung wird als Ganzes vergossen. Aus statischer Sicht ist dafür der zulässige Mindestabstand zu den Wandöffnungen interessant. Bei geschosshohen Porenbetonsteinen sitzen Stoßfugen auf beiden Öffnungslaibungen. Die Angabe der Mindestabstände ist hier überflüssig.
2) Steinmaße: Durch die Vorgaben von minimaler und maximaler Segmentbreite werden die in der Fabrik vorhandenen Steinmaße genannt.
3) Öffnungen: Die Länge der Wand und die Wandöffnungen (z.B. Fenster, Türe) gelten als die geometrischen Randbedingungen für die Segmentierung.

4.4.2 Geometrische Beziehung der Öffnungen zur Wand

Die relative geometrische Beziehung einer IFC-Entität zu einer anderen wird mit Hilfe ihrer globalen Transformationsmatrizen ermittelt.

Eine Wand mit globalem Placement $\mathbf{T}_{Wall}$ hat zum Beispiel eine Öffnung mit globalem Placement $\mathbf{T}_{Opening}$. Die Beziehung kann wie folgend beschrieben werden:

$$\mathbf{T}_{Opening} = \mathbf{T}_{Wall}\mathbf{L}_{Opening2Wall} \tag{4.29}$$

Daraus ergibt sich die relative Lage der Öffnung zur Wand (Bild 4.7).

$$\mathbf{L}_{Opening2Wall} = \mathbf{T}_{Wall}{}^{-1}\mathbf{T}_{Opening} \tag{4.30}$$

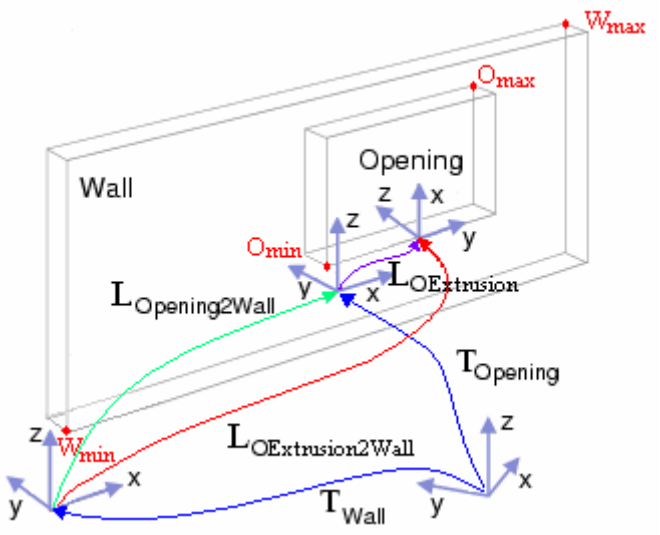


Bild 4.7: Lagebeziehung zwischen einer Öffnung und einer Wand

In Abs. 2.4.5 wurde ausgeführt, dass eine IFC-Öffnung immer ein Extrusionskörper ist. Sie besitzt drei Placements, nämlich das Placement der Öffnung selbst, das Placement des Extrusionskörpers und das Placement des 2D-Profils. Die Extrusionsrichtung und das Placement des 2D-Profils sind bezogen auf das Extrusionskoordinatensystem. Das Placement der Öffnung ist wie bei einer Wand irgendwo frei definierbar: es ist nicht immer streng an seinen Extrusionskörper gebunden (Bild 4.7).

Daher kann hier $\mathbf{L}_{Opening2Wall}$ eine falsche geometrische Beziehung zwischen dem Öffnungskörper und dem Wandkörper ergeben. Aus diesem Grund soll die Geometrie der Öffnung und der Wand in einem gemeinsamen Koordinatensystem abgebildet werden. Es ist also eine Koordinatensystemtransformation von der Extrusion der Öffnung zur Wand notwendig und zu bestimmen.

$$\mathbf{L}_{OExtrusion2Wall} = \mathbf{T}_{Opening2Wall}\mathbf{T}_{OExtrusion} \tag{4.31}$$

4.4.3 Algorithmus und Vorgehensweise

Bei der Segmentierung wird angenommen, dass die Stärke der Wand über ihre Länge und Höhe konstant bleibt. Dies entspricht dem Typ „*IfcWallStandardCase*“ in IFC. Da die Stoßfugen von Fertigteilen lediglich vertikal und orthogonal zur Wandführungslinie verlaufen, konnte die Ausgangsgeometrie einer dreidimensionalen Wand auf eine Dimension reduziert werden – die Wand also als Linie betrachtet werden. Der Algorithmus ist in Bild 4.8 veranschaulicht dargestellt.

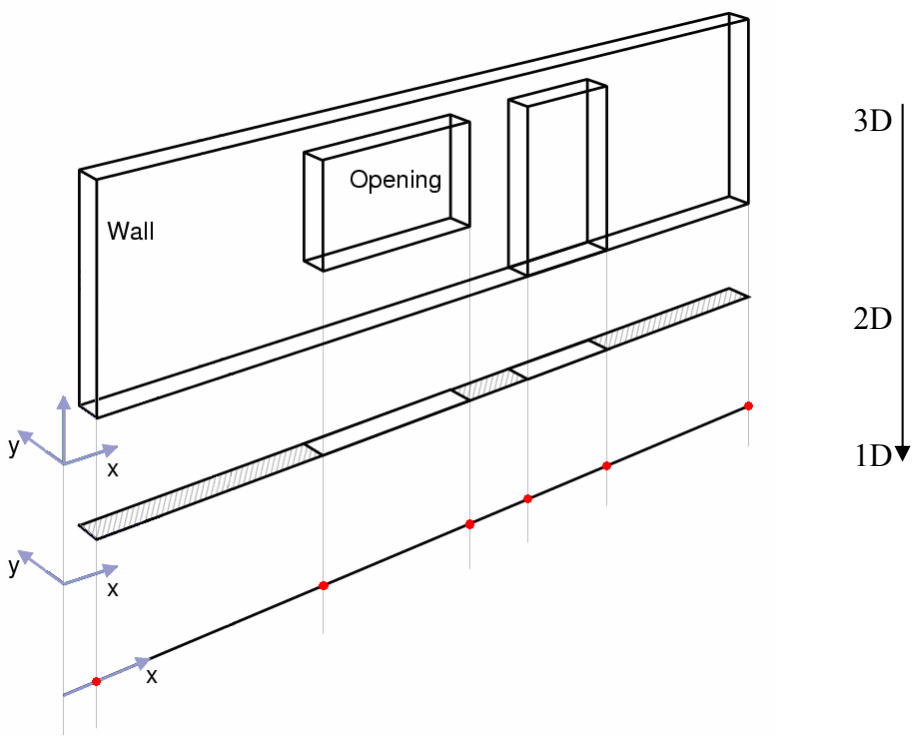


Bild 4.8: Algorithmus zur Reduktion der Grafikdimension

Zu jeder Wand werden alle Öffnungen, die dieser Wand zugeordnet sind, extrahiert. Dabei werden die Boundingboxen der Öffnungen ermittelt. Dadurch werden zwei Eckpunkte (links unten $O_{\min}$ und rechts oben $O_{\max}$) erhalten. Sie sind allerdings bezüglich des Extrusionskoordinatensystems der Öffnung angegeben. Mit $\mathbf{L}_{OExtrusion2Wall}$ werden die Eckpunkte der Öffnung in das Wandkoordinatensystem transformiert. Die zwei Eckpunkte der Wand ($W_{\min}$ und $W_{\max}$) werden ebenfalls in das Wandkoordinatensystem transformiert. Aus den Original-IFC-Daten sind nun lediglich die X-Koordinaten der Wand-Enden und der Öffnungseckpunkte zu betrachten.

Während der Extraktion aller Wandöffnungen wird die Größe der jeweiligen Boundingbox mit der vorgegebenen minimalen Segmentlänge verglichen. Die Öffnungen werden entsprechend in zwei Gruppen unterteilt: Öffnungen größer als das Segmentminimum und Öffnungen kleiner als das Segmentminimum. Die Unterteilung ist für die Segmentierung sinnvoll. Nach der IFC-Spezifikation sind alle Aussparungen in einer Wand als *IfcOpeningElement* definiert. *IfcOpeningElemente* sind nicht nur die Tür- und Fensteröffnungen sondern auch kleine Aussparungen wie beispielsweise eine Aussparung an einer Giebelwand, die als Balkenauflage für eine Dachpfette dient. Eine kleine Öffnung kann in einem Steinsegment beinhaltet werden und dient hier nicht als Randbedingung zur Segmentierung. Die kleinen Öff-

nungen werden später bei der Generierung der Steingeometrie als einfache Öffnungen erzeugt.

Aus dem verwendeten Baustoff und in Abhängigkeit der größeren Wandöffnungen ergeben sich die so genannten Nicht-Schneide-Bereiche, in denen keine Segment-Fugen gesetzt werden dürfen, und die Stellen, an denen bevorzugt geschnitten werden sollen, so genannte „PreferCuts“. Bei dem geschosshohen Porenbetonstein sind die „PreferCuts“ die Stoßfugen auf beiden Öffnungs-Laibungen während bei Stahlbetonfertigteilen die Stoßfugen einen bestimmten Mindestabstand zur Laibung einhalten müssen. Alle Nicht-Schneide-Bereiche und bevorzugte Schnitte werden aufsteigend entlang der Wandführungslinie sortiert.

Der Segmentierungsalgorithmus arbeitet rekursiv. Als geometrische Angabe bekommt die Funktion den zu segmentierenden Bereich, den Start- und End-Wert in X-Richtung. Es wird nach Nicht-Schneide-Bereichen gesucht, die innerhalb des zu segmentieren Bereichs liegen. Wenn ein Nicht-Schneide-Bereich gefunden wurde, wird die Segmentierung für die beiden verbleibenden Bereiche vor und nach dem gefundenen Nicht-Schneide-Bereich neu aufgerufen. Die automatische Segmentierung findet statt zwischen bevorzugten Schnitten und Grenzen von Nicht-Schneide-Bereichen. Ist ein Segment kürzer als die gewünschte (oder zulässige) minimale Segmentlänge, wird der zuletzt errechnete Schnitt verworfen. Die Vereinigung von letztem und vorletztem Segment wird halbiert. Die Auflagen für Stürze, die Segmentierung der Brüstungen und der Übermauerungen oberhalb der Stürze wurden hier nicht berücksichtigt. Sie werden erst später bei der Generierung der Steingeometrie zum BauXML-Modell berücksichtigt.

4.5 Generierung des BauXML-Modells

4.5.1 Generierung administrativer Informationen

Die meisten administrativen Informationen über Projekt, Gebäude, Stockwerke und Wände können aus IFC-Daten extrahiert und übernommen werden, z.B. die GUID (Global Unique Identification) und IFC-OID (Object Identification). Die in IFC fehlenden aber vom BauXML vorgesehenen Daten können nach der BauXML-Definition frei gelassen oder neu angegeben werden, weil sie in BauXML meisten als optional definiert werden.

Projekt

Die GUID und die im Projekt verwendeten Längen- und Winkel-Einheiten werden direkt aus den IFC-Daten übernommen oder standardmäßig geschrieben. Eine Beschreibung zum Projekt, die Kunden- und Auftragsdaten können optional später dazu angegeben werden.

Gebäude

Die GUID wird von den IFC-Daten übernommen. Eine Beschreibung des Gebäudes inklusive dem Namen des Bauherrn und der Adresse des Gebäudes kann aus den IFC-Daten übernommen werden, falls die Daten im IFC-Modell vorhanden sind.

Stockwerke

Für das Stockwerk werden eine GUID, ein Name und eine ID-Nummer als Attribute gefordert. Die ID-Nummer beginnt bei Null. Sie beeinflusst die ID-Nummern der zugehörigen Wände und weiterhin die ID-Nummern der Steine. Eine Stein-ID ist für die spätere Identifikation z.B. Barcode-Erzeugung vorgesehen. Eine Stockwerks-ID mit einer Null am Anfang weist auf einen Keller.

Die IFC-Entität „*IfcBuildingStorey*“ beschreibt ein Stockwerk. Allerdings ist diese Entität nicht immer in der IFC-Struktur enthalten bzw. es sind Wände enthalten, die direkt am Gebäude hängen. Falls Stockwerke in IFC vorhanden sind, werden die GUID, OID mit (oder ohne) den Namen aus dem IFC-Modell übernommen. Falls nicht, dann werden die Stockwerke hier erzeugt. In dem Fall werden die Zuordnung der zugehörigen Wände und die Reihfolge allen Stockwerke so bestimmt, dass rekursiv die Z-Koordinaten der Ursprünge aller Wandkoordinatensysteme verglichen werden. Die GUID, die ID-Nummer und ein Name werden hier erzeugt.

Wände

In der Vorfertigung gibt es eigentlich keine echten Wandobjekte sondern nur Steine. Eine Wand ist nur eine „virtuelle“ Zusammenfassung von Steinen. Zur Lokalisierung wird ein globales Placement zu jeder Wand angebracht. Die relativen Lagen ihrer Steine werden bezogen auf die Wand definiert. Damit werden alle Steine zu ihren zugehörigen Wänden und weiterhin im Raum eindeutig positioniert. Damit wird eine Visualisierung des ganzen BauXML-Modells im 3D-Raum erleichtert. Darüber hinaus ermöglicht es auch, am Ende der Steinvorfertigung mehrere Steine zu einer Wand zusammenzukleben, um größere Einheiten zur Baustelle zu schaffen. Eine GUID, das Wandmaß und eine ID werden an das Wandobjekt geschrieben. Die ID für die achte Wand im Erdgeschoß kann z.B. wie „1-8“ aussehen.

Für Steine gibt es keine entsprechende Entität in der IFC-Struktur. D.h. alle Informationen wie GUID, ID-Nummer, Geometrie und Topologie sind hier zu generieren. Ein Stein mit der ID-Nummer in Form „2-8-12“ bedeutet der zwölfte Stein der achten Wand im ersten Oberge-

schoß. Das lokale Placement jedes Steines wird immer am Eckpunkt vorne links unten definiert (Bild 4.8). Alle an den Stein gehefteten Features sind bezüglich des Steins dargestellt. Im Folgenden wird erläutert, wie die Informationen (z.B. Grundform und Features auf dem Stein) über die Steine ins BauXML-Modell generiert werden.

4.5.2 Generierung der Grundform des Steins

In der Arbeit werden die Wände behandelt, deren Wandstärke über die Länge und Höhe der Wand konstant bleibt. Solche Wände können in IFC entweder als „*IfcWallStandardCase*“ oder als „*IfcWall*“ definiert werden. Eine Wand mit beliebigen Konturen als B-Rep-Modell gibt es im Wohnungsbau selten, deshalb werden sie in dieser Arbeit nicht berücksichtigt.

Daher ist ein Quader als Grundform für die Steine geeignet (Bild 4.9). Alle Aussparungen, wie z.B. Öffnungen, Schlitze usw., sind als Features zu definieren und werden später in der Fertigung von der Grundform herausgenommen. Die Grundformen Zylinder und Vielkantprisma sind ausschließlich für Stützen gedacht, die in IFC als „*IfcColumn*“ definiert sind.

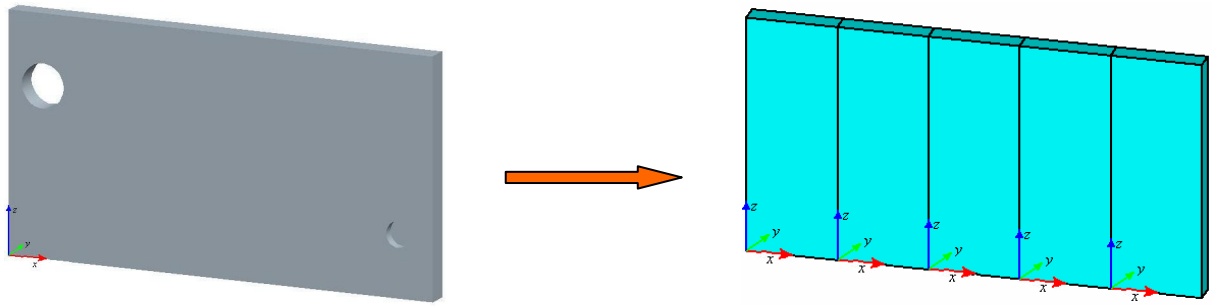


Bild 4.9: Erzeugen der Grundformen aller Steinblöcke

Nach der Segmentierung werden die X-Werte aller Schnittpunkte zwischen dem Anfangspunkt und dem Endpunkt einer Wand aufsteigend sortiert. Von dem Anfangspunkt an wird durch je zwei X-Werte ein rechteckiger Steinkörper erzeugt. Jeder Steinkörper hat die gleiche Stärke wie die Wand. Die Steinhöhe ist je nach der Steinposition entweder die Wandhöhe, die Brüstungshöhe oder die Übermauerungshöhe (vgl. Bild 4.12). Dadurch entstehen die Grundformen aller Steinblöcke ohne irgendein Feature.

4.5.3 Generierung der aus dem Grundriss resultierenden Features

Wie oben erwähnt, wird eine Wand mit rechteckiger Form angenommen. d.h. der Grundriss ist ein zweidimensionales Rechteck. Bei einer Wand, die durchgehende Aussparungen über die gesamte Wandhöhe hat, ist der Grundriss nicht mehr rechteckig, sondern irgendein Polygon. Ein Beispiel ist eine Wand mit Gehrungsstößen. Durchgehende Aussparungen können

auch als „*IfcOpeningElement*“ in IFC definiert werden. Ist dies der Fall, dann werden sie nicht hier, sondern als Features vom Typ Öffnungen (Abs. 4.5.6) behandelt.

Das wirkliche Grundriss-Profil wird mit einem Rechteck verglichen, das durch die Länge und die Stärke der Wand entsteht. Die Abweichungsprofile werden zu der Wand zusammengefasst und registriert. Nach der Segmentierung muss jedes Abweichungsprofil der Wand geometrisch überprüft werden, in welches Steingrundprofil oder in welche Steingrundprofile es fällt. Das Grundprofil jedes Steinsegments wird also mit allen Abweichungsprofilen rekursiv verglichen. Die Überlappungskontur wird ermittelt, egal ob ein Abweichungsprofil total oder nur teilweise in das Grundprofil des Steinsegments fällt. Durch Extrusion dieser Überlappungskontur werden Fertigungsfeatures im Stein erzeugt. Dabei werden die Koordinaten der Überlappungskontur vom Wandkoordinatensystem ins Steinkoordinatensystem umgewandelt. Bild 4.10 zeigt schematisch den Vorgang.

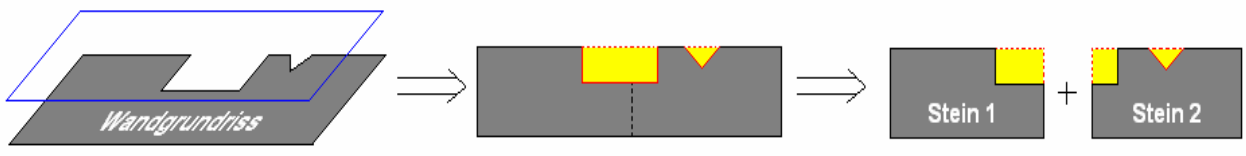


Bild 4.10: Herausfilterung und Unterteilung Abweichungsprofile im Grundriss einer Wand

4.5.4 Generierung der von haustechnischen Installationen resultierenden Features

Haustechnik ist ein umfangsreicher Bereich. In IFC sind eine ganze Menge Typen dafür definiert. Ein Heizkörper wird z.B. als „*IfcEnergyConversionDevice*“ definiert und ein Ventilator kann als „*IfcFlowMovingDevice*“ definiert werden. Der Typ „*IfcFlowTerminal*“ definiert das Auftreten eines dauerhaft angebrachten Elements, das als Beginn eines Verteilungssystems dient (z.B. Wachbecken, Steckdose, Schalter, usw.). Der Anschluss ist gewöhnlich ein Punkt, an dem ein Element angebracht wird. Seine Art wird durch „*RelDefByType*“ mit einer Unterklasse von „*IfcFlowTerminalType*“ bestimmt. Unter dem Typ „*IfcSanitaryTerminalType*“ werden z.B. die sanitären Objekte definiert. Elektrische Auslässe wie Steckdosen, Schalter, Telefon werden mit dem Typ „*IfcOutletTyp*“ bestimmt. Als „*IfcFlowSegment*“ kann eine Leitung definiert werden.

Zwar hat die IFC 2x2-Spezifikation diese Entitäten definiert, aber eine ausgereifte Implementierung im IFC-Format stellen die meisten CAAD-Systeme dazu noch nicht zur Verfügung. Es ist klar, dass sich ein Architekt in seinem Hausentwurf meistens keine Gedanken macht, wie die Kabel verlaufen und an welcher Position genau ein Waschbecken oder eine Steckdose

an der Wand angebracht werden sollen. Diese Sachen sollen vom Haustechniker festgelegt und ins IFC-Modell hinzufügt werden. Eine Aussparung für die Kabel soll nach der Spezifikation als „*IfcOpeningElement*“ definiert werden. Im Moment sind leider noch keine solchen IFC-Dateien zu bekommen. Sofern die Informationen vorhanden sind, können sie als kleine Wandöffnungen (Abs. 4.5.6) behandelt werden. Gegenwärtig wird es so gemacht, dass die Position durch das Placement der Entität „*IfcFlowTerminal*“ ermittelt wird. Die geometrische Beschreibung der Aussparungen wird in programminternen Makros definiert.

4.5.5 Generierung der von der Dachschräge resultierenden Features

Im Abs. 2.4.4 wurde ausgeführt, dass eine beschnittene Wand mit der IFC-Entität „*IfcBooleanClippingResult*“ beschrieben wird, die ein CSG-Modell geometrisch repräsentiert. Der „*FirstOperand*“ ist ein Solidmodell und verweist auf das Wandelement und der „*SecondOperand*“ auf die Verschneidungsebene. Der boolesche Operator ist immer DIFFERENCE. Ein Wandkörper wird vollständig mit einer Ebene verschnitten. Das „*IfcLocalPlacement*“ einer Schräge stellt einen Frame zur Wand dar. Es wird durch Angaben der Normalenrichtung, einer Hilfsrichtung und eines Punkts auf der Ebene festgelegt. Eine Außenkontur kann nach der IFC-Spezifikation optional zu der Verschneidungsebene angegeben werden. Selbst wenn eine Außenkontur angegeben ist, handelt sie sich allerdings nicht immer genau um die richtige Verschneidungskontur mit dem Wandkörper sondern nur grob [IFC1].

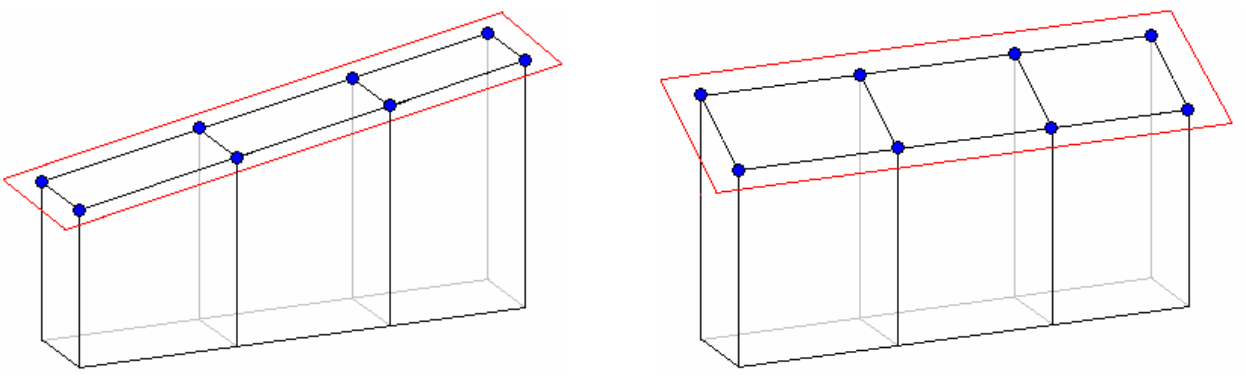


Bild 4.11: Übertragen der Dachschrägekontur auf einzelnen Stein

Nach der Segmentierung wird die schräge Ebene auf den einzelnen Stein übertragen. Die richtige Verschneidungskontur der Ebene mit dem einzelnen Stein wird berechnet und ins lokale Steinkoordinatensystem transformiert (siehe Bild 4.11). Die Normalenrichtung und die Hilfsrichtung auf der Ebene werden unverändert im Steinkoordinatensystem übernommen, weil die Orientierung der Ebene unverändert bleibt. Der Ursprung des Ebenenkoordinatensystems wird an einem Punkt der Verschneidungskontur positioniert. So wird eine Ebene als Fertigungsfeature auf jedem Steinsegment erzeugt.

4.5.6 Generierung der von Wandöffnungen resultierenden Features

Wie bei der Segmentierung erläutert, werden alle Öffnungen („*IfcOpeningElement*“) aus einer Wand herausgenommen und der Größe nach in zwei Gruppen aufgeteilt: Öffnungen größer als das Segmentminimum und Öffnungen kleiner als das Segmentminimum (siehe Abs. 4.4.3). Die größeren Öffnungen dienen als geometrische Randbedingung der Segmentierung und die kleineren Öffnungen werden intern gespeichert.

1) Genau so wie der Grundriss werden die kleinen Öffnungen geometrisch überprüft, in welchem Stein oder in welchen Steinen sie liegen. Dann wird die Überlappungskontur berechnet und weiter ins lokale Steinkoordinatensystem transformiert. Ein Fertigungsfeature entsteht als ein Extrusionskörper durch die Überlappungskontur. Ob die Öffnung durch die Wand durchgeht oder nur mit einer bestimmten Tiefe aus der Wand herausgeschnitten ist, entscheidet die Definition in IFC. Die Entität „*IfcOpeningElement*“ mit einem Attribut „*RECESS*“ bedeutet eine blinde Öffnung, sonst ist die Öffnung durchgehend.

2) Die größeren Wandöffnungen sind nach der Segmentierung „verschwunden“. Statt der Öffnungen werden die Schnittpunkte (die aufsteigenden X-Werte) geliefert. Jeder Schnittpunkt wird gefragt, was für eine geometrische Relation zu einer ihm nahe liegenden Öffnung er besitzt:

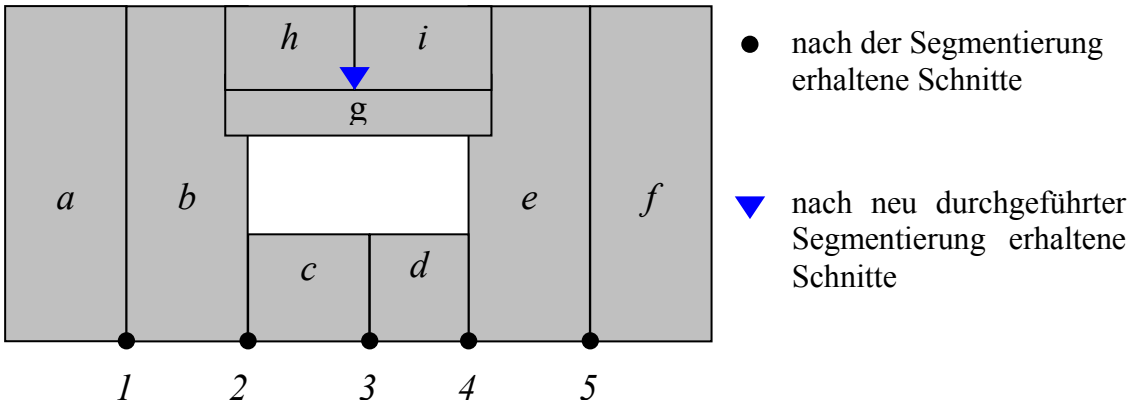


Bild 4.12: Überprüfung der Schnitte

- Schließt eine linke Öffnungslaibung an, dann wird die Aussparung für die linke Sturzauflage als ein Feature erzeugt (z.B. Stein b in Bild 4.12). Sturzauflager sind als rechteckige Extrusionskörper oder falls speziell gefordert auch durch eine andere Geometrie definiert. Die Extrusionsrichtung zeigt entlang der Y-Achse des Steinkoordinatensystems durch die Wand. Positioniert wird das Feature in der Höhe der Öffnungsoberkante.

- Fällt der Schnittpunkt auf eine rechte Öffnungslaibung (Position 4 in Bild 4.12), handelt es sich um ein Segment im Bereich unterhalb der Öffnung. Darüber hinaus wird die Höhe der entsprechenden Öffnungsunterkante berechnet. Ist sie gleich Null, dann handelt sich um eine Türöffnung. Es werden dann keine Brüstungssteine vorgesehen. Falls die Höhe nicht gleich Null ist, handelt es sich um die Brüstung einer Fensteröffnung.
- Der X-Wert liegt außerhalb der Öffnung (z.B. Position 1 und Position 5 in Bild 4.12). Es kann ein Steinsegment sein, das total außerhalb einer Öffnung liegt (z.B. Stein a). Es kann aber auch das Steinsegment sein, das an der Rechtlaibung der Öffnung steht (z.B. Stein e). Dafür wird die geometrische Beziehung des Anfangspunkts (X-Wert) des Steins mit der Öffnung ermittelt. Liegt der Anfangspunkt außerhalb der Öffnung, steht das Segment total außerhalb der Öffnung (Stein a). In dem Fall werden keine Features in Hinsicht auf die Öffnung erzeugt. Falls der Anfangspunkt an der Rechtslaibung der Öffnung liegt, dann steht das Steinsegment rechts unmittelbar neben der Öffnung, z.B. der Stein e in Bild 4.12. Eine Aussparung für die rechte Sturzauflage wird ebenso wie bei dem links stehenden Segment als ein Feature erzeugt. Die Situation, dass der Anfangspunkt eines Segments an der linken Laibung einer Öffnung liegt während der Endpunkt außerhalb der Öffnung liegt, kann nicht vorkommen: Die beiden Laibungstellen werden als „bevorzugte Schnitte“ behandelt und alle Schnittpunkte sind aufsteigend sortiert, d.h. der Endwert eines Segments ist immer größer als der Anfangswert.

4.5.7 Generierung der Steine oberhalb einer Wandöffnung

Aus statischen Gründen soll die Kraft von oben nicht direkt an das Fenster oder die Tür übertragen werden. Bautechnisch wird das Problem normalerweise durch einen Sturz gelöst. Der Sturz verteilt die anfallende Last in die beiden neben der Öffnung stehenden Steine. In diesem Arbeitsschritt werden der Sturz und die Übermauerungssteine erzeugt. In den Katalogen von Beton- und Porenbetonanbietern gibt es eine große Auswahl an Stürzen mit unterschiedlichen Dimensionen. Die Parameter wie die Höhe und die Dicke können vor der Generierung des BauXML-Modells angegeben werden. Die Länge des Sturzes wird aus der Öffnungslänge und den beiden Auflagerlängen bestimmt. Der Bereich für die Übermauerung wird auch segmentiert. Die neu entstehenden Segmente wiederholen den ganzen Ablauf der Feature-Erzeugung (kleine Öffnungen, Installationen und Dachschräge).

4.6 Softwaretechnische Umsetzung

4.6.1 Projektübersicht

Die zuvor dargestellten Arbeiten sind im Programm „IfcWallModificator“ implementiert. Das Programm „IfcWallModificator“ liest eine IFC-Datei ein und visualisiert das ganze Modell. Das Programm ermöglicht es dem Benutzer, komfortabel auf das Gebäudemodell zuzugreifen, die relevanten Gebäudeelemente gezielt zu extrahieren, zu bearbeiten und zu visualisieren. In „IfcWallModificator“ werden alle nötigen Informationen über die Wandelemente der eingelesenen Datei dargestellt, z.B. das Material, die Abmessung, die Lage im globalen Koordinatensystem und die Beziehungen des Wandelementes mit seinem Wandöffnungen und anderen Wänden.

Die Korrektur der Wandverschnitte ist in diesem Programm nach dem Einlesen durchzuführen. Das Ergebnis wird in einer neuen IFC-Datei abgelegt und kann jederzeit in einem CAAD-System veranschaulicht und weiterverarbeitet werden. Nach der Segmentierung wird das BauXML-Modell generiert (ohne Segmentierung wird nur ein Stein für die ganze Wand generiert) und anschaulich dargestellt. Die 3D-Darstellung des BauXML-Modells dient der Überprüfung, ob das Originalmodell aus den erzeugten Steinen regeneriert werden kann.

Das Programm „IfcWallModificator“ ist auf Basis der MFC (Microsoft Foundation Classes) unter der Entwicklungsumgebung Microsoft Visual Studio 6 implementiert. Zur 3D-Visualisierung und Darstellung kommt die OpenGL-Technik zum Einsatz.

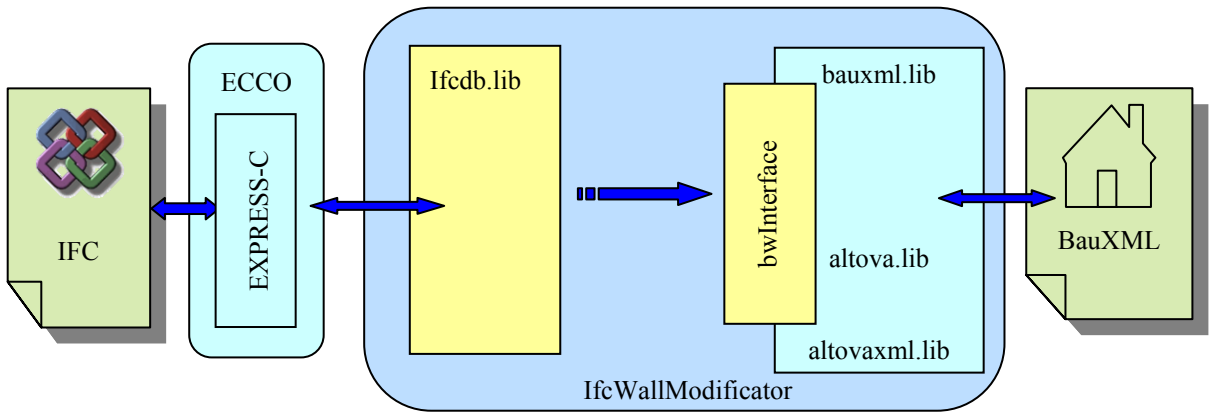


Bild 4.13: Datenfluss und Struktur des Programms „IfcWallModificator“

Zum Lesen, Schreiben und Bearbeiten von IFC-Daten wird das ECCO-Toolkit [ECCO] verwendet. Aus den selbst mit EXPRESS-C, eine speziell Sprache für den Zugriff auf STEP-Daten, definierten Schemata wird eine DLL (Dynamic Link Library) vom ECCO-Toolkit zu-

sammen mit dem IFC-Schema erstellt. Diese DLL wird in die C++-Applikation dynamisch eingebunden.

Mit Hilfe von Altova XMLSpy [ALTO] werden aus dem entwickelten BauXML-Schema ganze Klassen in einer Bibliothek - bauxml.lib erfasst. Zusammen mit den zwei Firmen-Bibliotheken altova.lib und altovaxml.lib kann eine BauXML spezifische Datei geschrieben, eingelesen oder gezielt manipuliert werden. Darüber hinaus kann eine XML-Datei anhand des BauXML-Schemas validiert werden – alle erzeugten BauXML-Dateien müssen valid sein, sowohl das Vokabular als auch die Grammatik, sonst akzeptiert der Interpreter der Produktionssteuerung die Datei nicht.

Sowohl die vom ECCO-Toolkit als auch die vom XMLSpy erstellten Bibliotheken (.dll) bieten von außen per C++ recht umständliche Zugriffsmöglichkeiten. Als Adapter ist eine eigene Datenhaltung sinnvoll. Auf der IFC-Seite wird dafür die Ifcdb.lib neu implementiert. Diese Datenhaltung wird von der ECCO-Seite mit Kopien der bereits in ECCO vorhandenen Objekte inklusive der Geometrie gefüllt. Auf der BauXML-Seite wird eine Reihe von Wrapper-Klassen in „bwInterface“ implementiert. Von der Applikationsseite her können dann die Daten aus den Datenhaltungen per C++ komfortabel geholt und benutzt werden. Der vollständige Datenfluss und die Struktur des entwickelten Programms werden in Bild 4.13 anschaulich dargestellt.

4.6.2 Datenhaltung zum IFC – IFCDB

Die Bibliothek ifcdb.lib ist eine Kopplungsschicht vom ECCO-Toolkit zur Applikation. Bestandteile sind die „Population“, die Geometrieklassen sowie die Interfaceklassen zum ECCO. Diese Bibliothek wird statisch an die Applikation gelinkt.

IFC Klassen bilden eine recht komplexe Klassenhierarchie. Jede Art von reinem Bauteil, Gebäude, Stockwerk, Wand, Öffnung, Decke, Stütze usw., wird als eine eigene Klasse abgebildet. Darüber hinaus existieren noch eine ganze Menge Klassen für alle Objekte, die in einem Gebäude vorkommen können. Dadurch werden Gebäude mit IFC detailliert abbildet. In der ifcdb.lib wurde aber ganz auf die Nachbildung der IFC-Klassenhierarchie in C++ verzichtet, was Wände und Öffnungen und sonstige Bauteile angeht. Dadurch wird die Komplexität der IFC-Datenstruktur entsprechend den eigenen Bedürfnissen reduziert. In der eigenen Datenhaltung sind alle Bauteile von der Klasse „*IfcEntity*“ instanziiert. Bild 4.14 zeigt den Auszug der Klasse „*IfcEntity*“ in UML-Notation mit einigen „get“-Methoden und ohne Attribute.

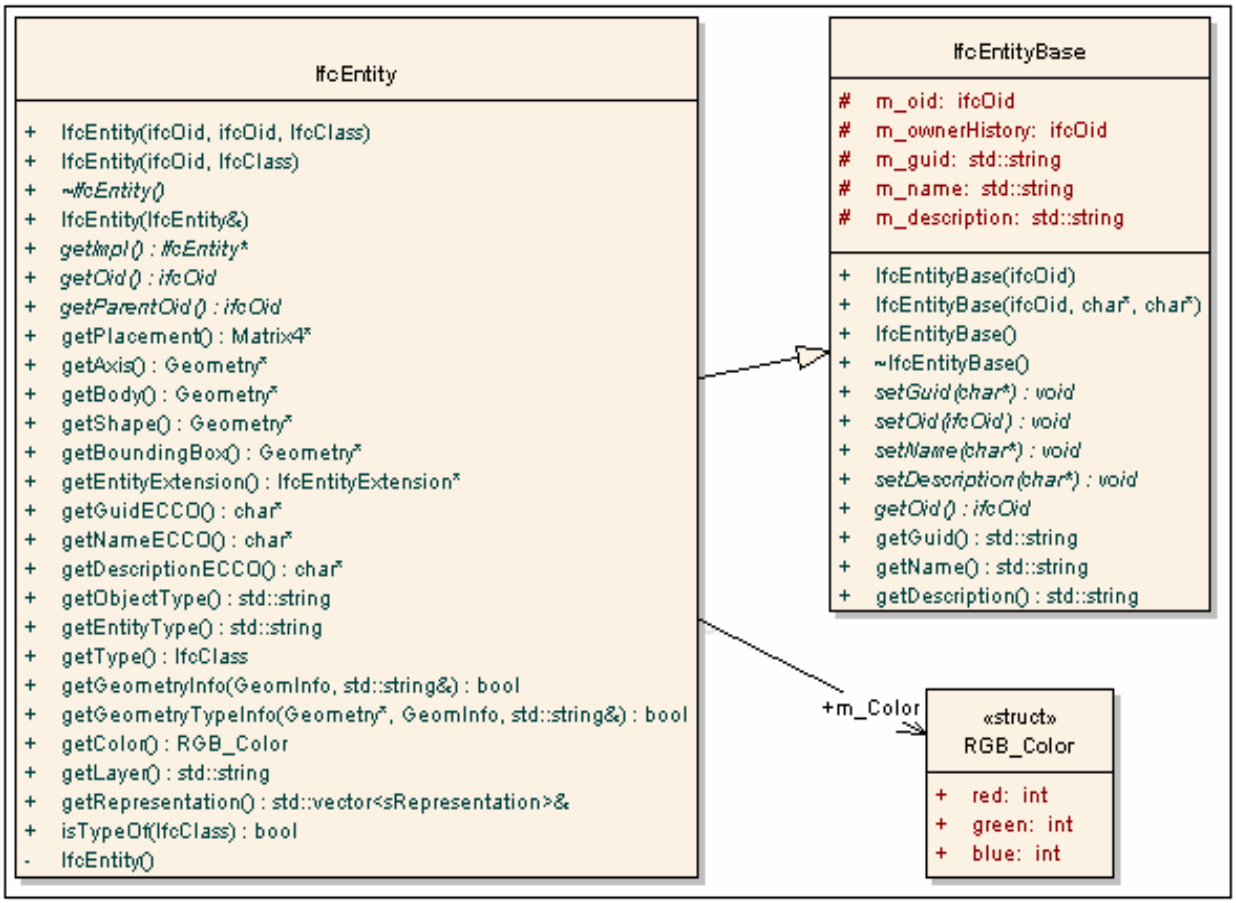


Bild 4.14: Auszug des UML-Diagramms der Klasse „IfcEntity“

Es ist jederzeit möglich, mit der Methode „*getEntityType*“ einen Bauteil-Typ abzufragen. Das Placement und die geometrische Repräsentation können ebenso durch „*getPlacement*“ und „*getBody*“ erhalten werden.

In der Klasse von „Population“ werden „sämtliche“ Entitäten und deren Beziehungen gespeichert und verwaltet. Sie kann als eine Art Datenbank angesehen werden. Die „1 zu n“ Relationen wie z.B. „Wand zu berührenden Wänden“ oder „Wand zu Öffnungen“, werden im Typ *std::map<unsigned int, std::vector<unsigned int>>* zusammengefasst. Gespeichert werden die Objekt-IDs (IfcOid), wobei die Container-IfcOid als ein Schlüssel dient. Mit der Methode *Populationi::getChildren(IfcClass,IfcEntityList&, ifcOid)* werden alle Kinderelemente mit der gewünschten IFC-Klasse erhalten. Die Methoden der Klasse von „Population“ werden im Bild 4.15 als UML-Diagramm dargestellt.

Bild 4.15: Auszug des UML-Diagramms der Klasse „Populationi“

In der Applikation wird die „Population“ in der Dokumentklasse „*CIfcWallModificatorDoc*“ instanziiert. Die an die „Population“ übergegebenen Parameter sind der Dateiname der einzulesenden IFC-Datei, ein Zeiger auf *DefaultFactory* oder eine von ihr abgeleitete Klasse. Mit der Methode „*CIfcWallModificator::getPopulation()*“ steht die ganze Datenbank den anderen Klassen z.B. bei der Wandverschnittmanipulation und der Implementierung des BauXML-Modells immer zur Verfügung.

Innerhalb der selbt erstellten 2D/3D-Geometrieklassen (siehe Billd 4.16) sind zahlreiche notwendige mathematische und computergraphische Algorithmen und Methoden in der Arbeit implementiert und neu entwickelt worden. Sie umfassen Matrizenalgebra (z.B. Matrizenrepräsentation affiner Transformationen, Skalar-/Kreuzprodukt von Matrizen, Matrizenaddition/-

multiplikation, Inversion/Transposition einer Matrix), Objektdarstellung geometrischer 2D/3D-Primitive (Punkt, Linie, Polygon, Kreis, Ebene, Körper), Verschneidungen, Überlappungen geometrischer Primitive und Positionsüberprüfungen (z.B. Liegt ein Punkt innerhalb eines Polygons oder auf einer Linie?) usw. [BERG][LENG][SCNE].

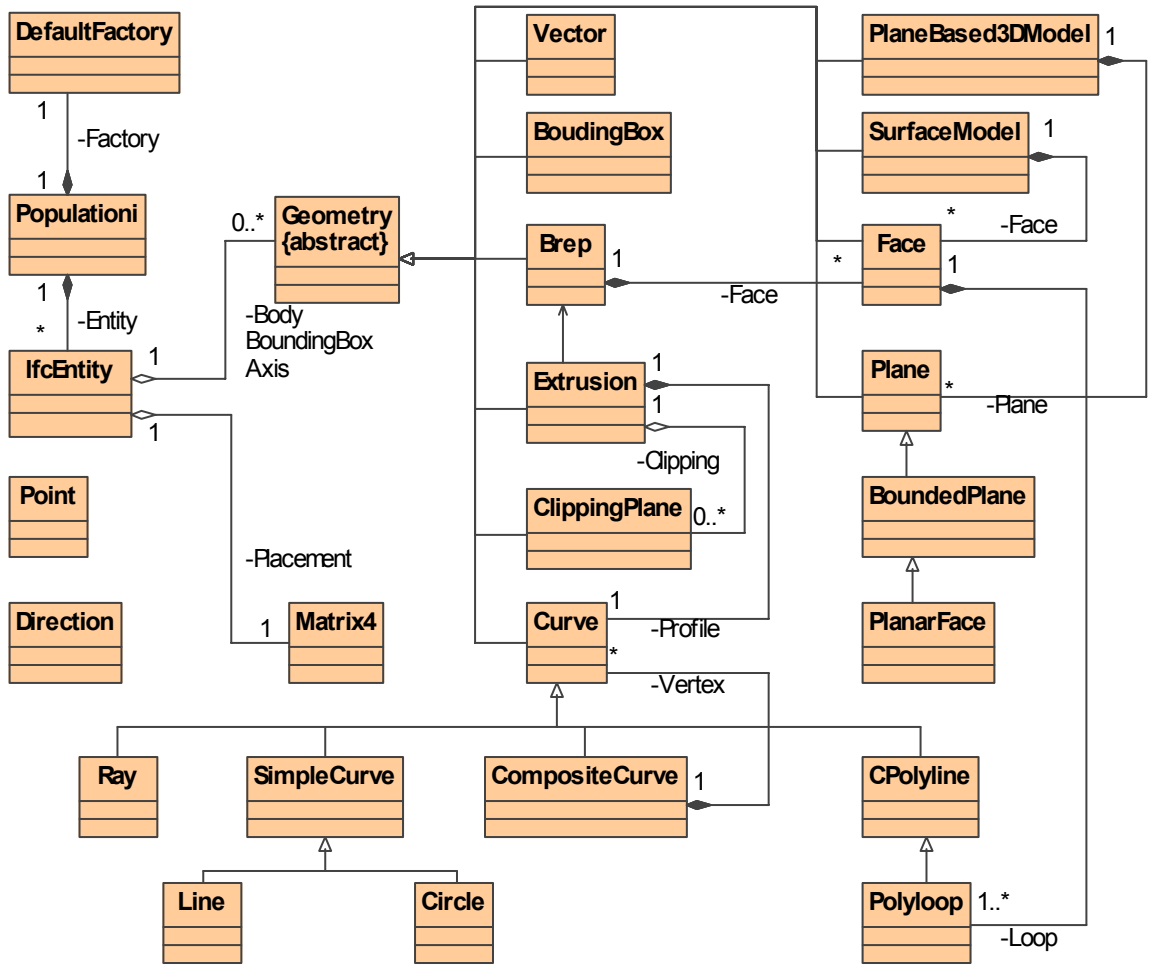


Bild 4.16: Auszug des UML-Diagramms der Geometrieklassen in Ifcdb

4.6.3 Datenhaltung zum BauXML - bwInterface

Die von Altova XMLSpy erstellte Bibliothek bauxml.lib bildet alle BauXML-Klassen eins zu eins ab. Es ist aber umständlich, in der Applikation von außen auf die BauXML-Daten per C++ zuzugreifen. So werden die Wrapper-Klassen in der Datei bwInterface erfasst. Dazu gehören die Klassen *bwProject*, *bwBuilding*, *bwStorey*, *bwWall*, *bwStone* und *bwFeature* sowie andere Hilfsklassen (Bild 4.17). In der Klasse *bwProject* wird eine Member-Variable m_*root* vom Typ *bau::CProjectType* deklariert. Sie ist das Wurzelelement des BauXML-Modells, an dem alle XML-Daten angehängt sind. Die Eltern-Kinder-Relation wird mit dem Datentyp *std::vector<>* realisiert. In *bwBuilding* werden z.B. die Zeiger von *bwStorey*, in *bwStorey* die Zeiger von *bwWall* gespeichert und verwaltet. Dadurch werden alle erzeugten Daten in der

Klasse *bwProject*, bzw. um genau zu sein in dem Wurzelelement *m_root* der Klasse *bwProject*, gespeichert.

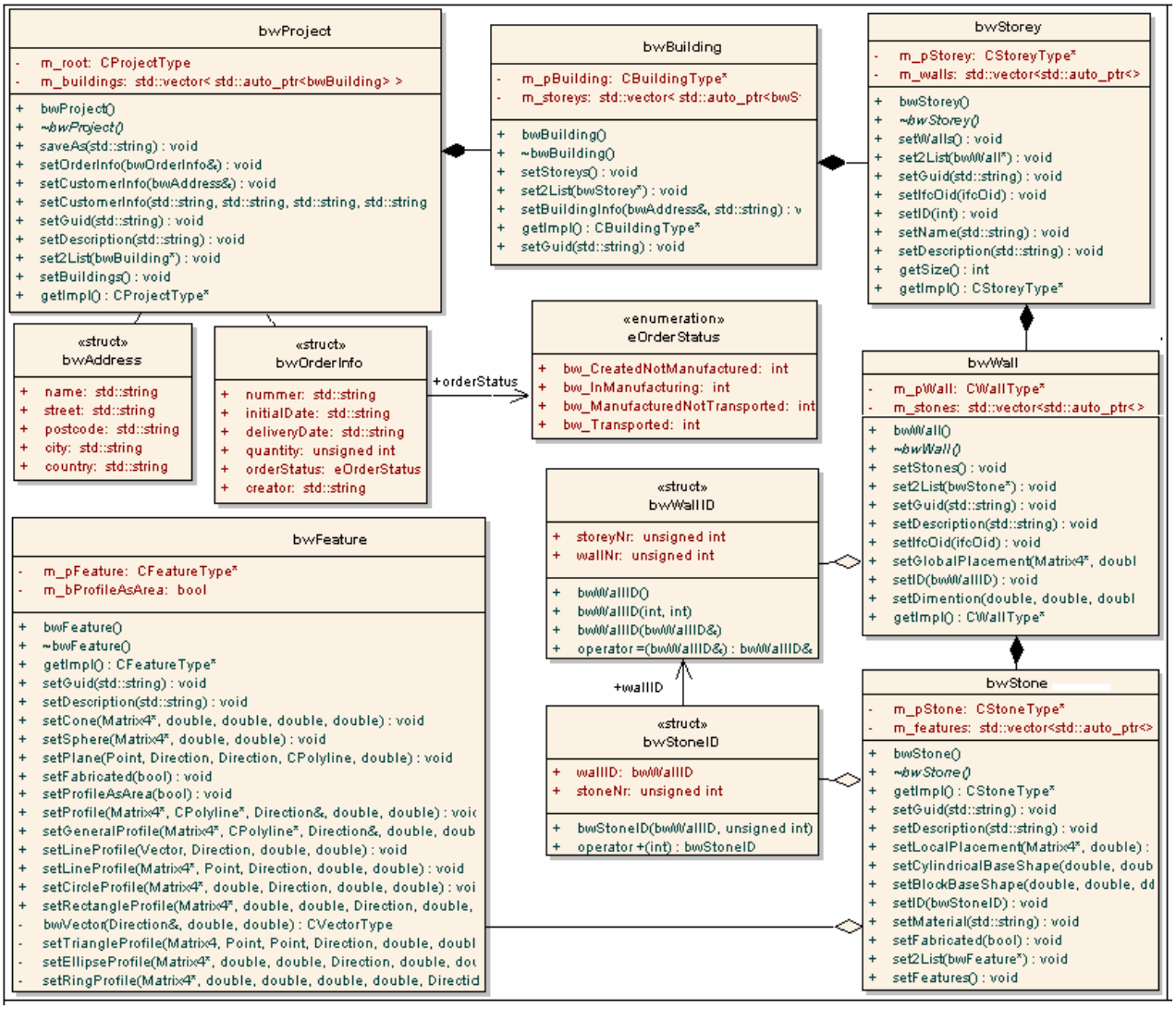


Bild 4.17: Auszug des UML-Diagramms der Klassen „bwInterface“ zum BauXML

4.6.4 Klasse zur Manipulation der Wandverschnitte

Zur Manipulation der Wandverschnitte werden vier Klassen implementiert (Bild 4.18). Die Hauptklasse „*WallTrimmer*“ steuert den ganzen Ablauf, von der Aufnahme der zu manipulierenden Wände bis zur Bereitstellung der neuen Wandverschnitte. Die geometrische Berechnung der Wandverschnitte wird in der Klasse „*TrimElement*“ durchgeführt, dazu gehört die Berechnung und Transformation der Schnittkontur sowie die Generierung der Geometrie des Wandkörpers. Die Klasse „*MergeElements*“ überprüft, ob zwei Wände kollinear sind, damit wird entschieden, ob die beide sinnvoll zusammenführt werden können. Die Klasse „*ElementConnection*“ behandelt den Fall, dass zwar die IFC-Entität „*IfcRelConnectsPathEle-*

ments“ (Abs. 2.4.3) jedoch keine Angabe der Position (*AtStart*, *AtPath*, oder *AtEnd*) vorhanden ist. Gegebenfalls wird hier eine geometrische Berechnung dafür durchgeführt.

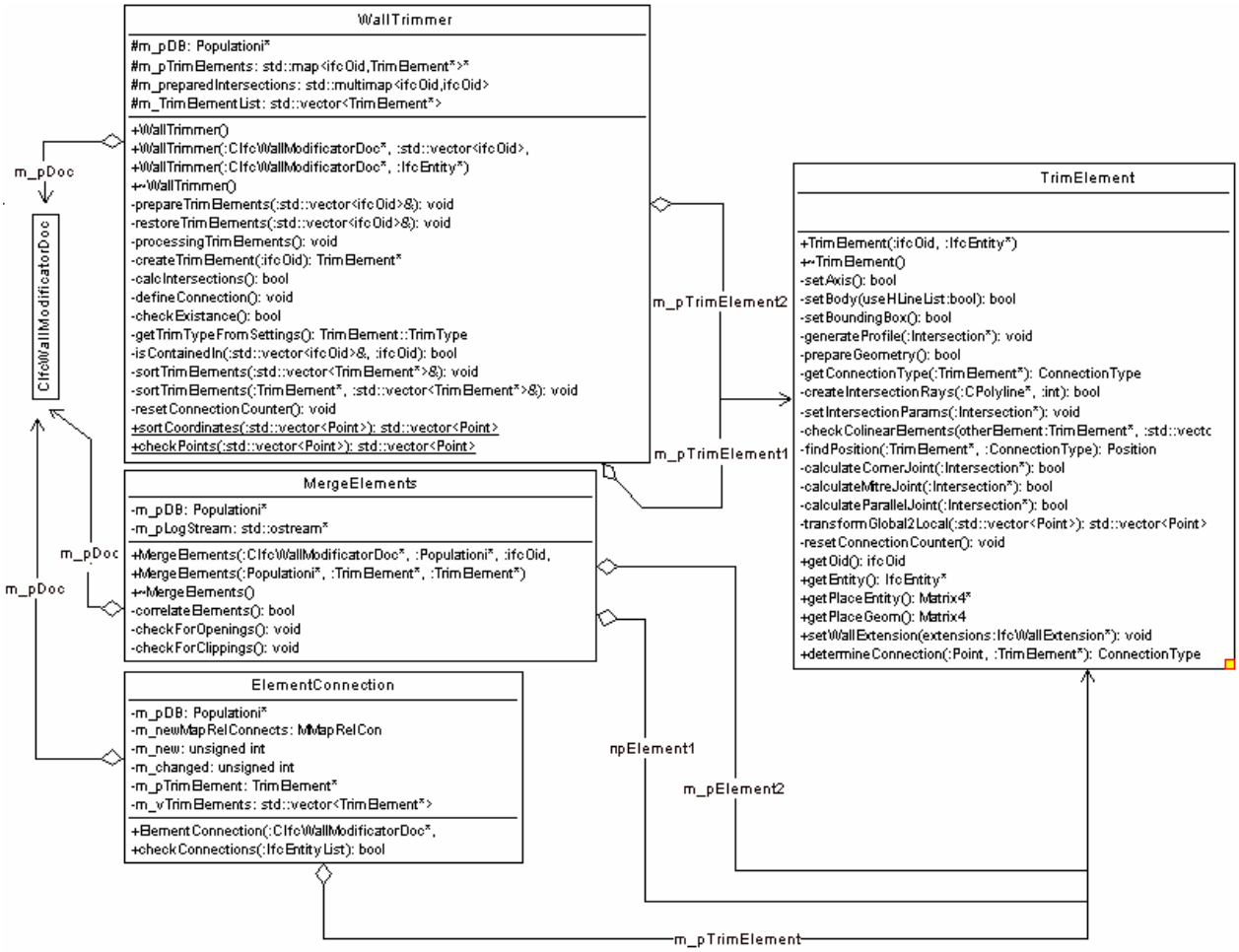


Bild 4.18: UML-Diagramm der Klassen „WallTrimmer“ zur Wandverschnittkorrektur

4.6.5 Klasse zur Wandsegmentierung

Die Funktionen zur Segmentierung werden in der Klasse „*WallSegmenter*“ erfasst (siehe Bild 4.19). Hilfsattribute sind die minimale- und maximale Segmentlänge, die Mindestdistanz der Schnitte zur Öffnungslaibung sowie das Wandmaterial. Sie werden in der Applikation global voreingestellt. In der Applikation kann ausgewählt werden, ob alle Wände oder nur gezielte ausgesuchte Wände segmentiert werden sollen. Für jede der zu segmentierenden Wände wird die Klasse „*WallSegmenter*“ rekursiv aufgerufen. Dabei werden die Anfangs- und Endposition der Wand berechnet. Die Öffnungen werden durch die Methode *registerOpenings()* herausgefiltert werden. Anschließend werden die Nicht-Schneide-Bereiche und die bevorzugten Schnittpunkte abhängig von dem verwendeten Steinmaterial bestimmt (verg. Abs. 4.4.3). Das Material wird als Aufzählungstyp (Enumeration) definiert: Porenbeton (*LW_CONCRETE*) oder Ortbeton (*CONCRETE*).

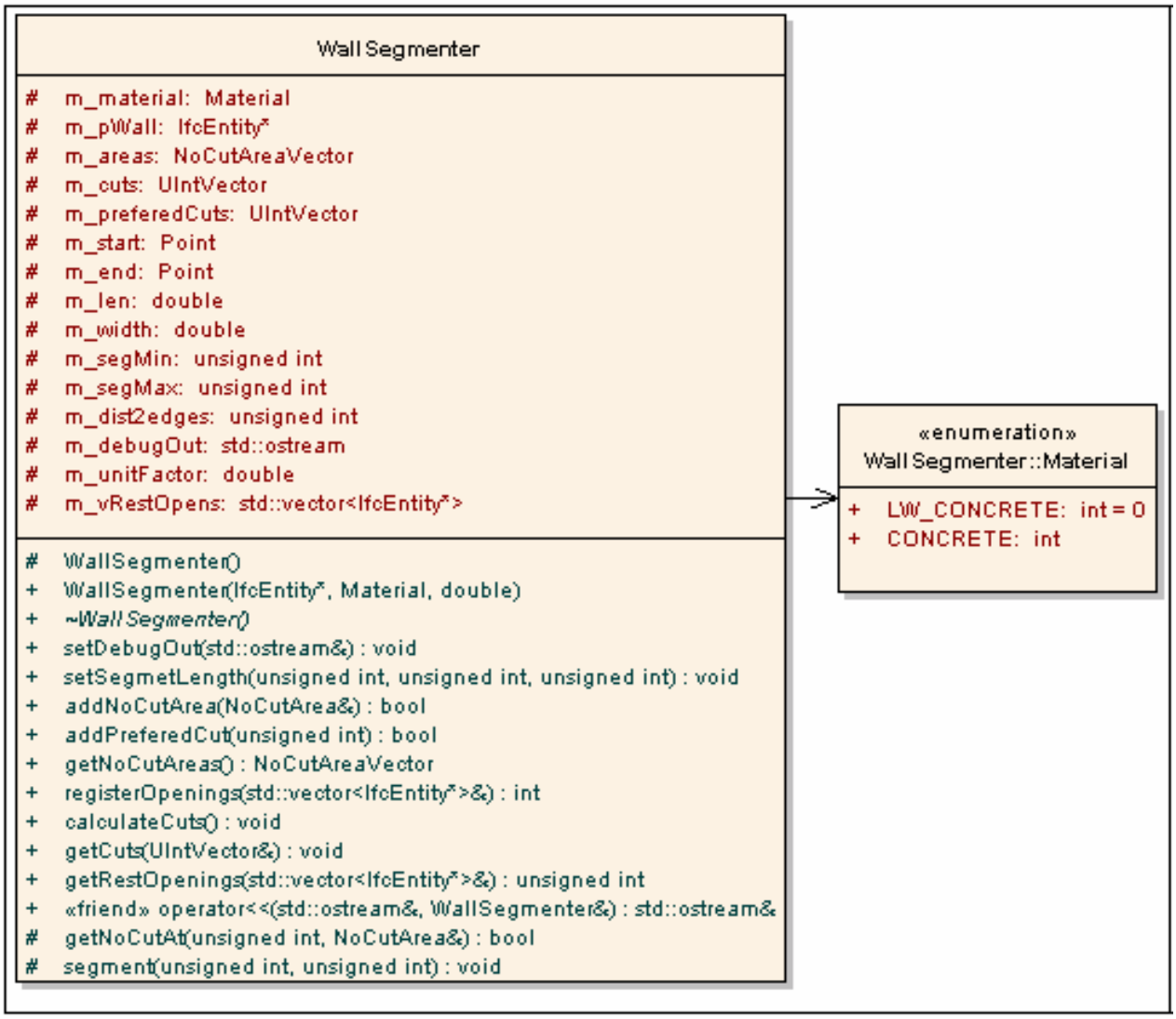


Bild 4.19: Auszug des UML-Diagramms der Klasse „WallSegmenter“ zur Segmentierung

Innerhalb der Methode *calculateCuts()* werden die zu segmentierenden Bereiche an die *segment(unsigned int left,unsigned int right)* abgegeben. Dort wird der Segmentierungsalgorithmus durchgeführt. Als Ergebnis werden die gewünschten aufwärts sortierten Schnittwerten erhalten. Sie sind in einem Vektor *std::vector<>* gespeichert und können mit der Methode *getCuts()* abgerufen werden. Falls es in der Wand noch Öffnungen gibt, deren Länge kleiner als die minimale Segmentlänge ist, die als „SmallOpenings“ bezeichnet sind (vgl. Abs. 4.4.3), werden sie durch *getRestOpenings()* zurück geliefert und in der folgende Klasse „*cBauXMLGenerator*“ mit der Methode *createSmallOpenings()* als Öffnungen generiert.

4.6.6 Klasse zur Implementierung des BauXML-Modells

In der Klasse „*cBauXMLGenerator*“ wird ein BauXML-Modell generiert (siehe Bild 4.20).

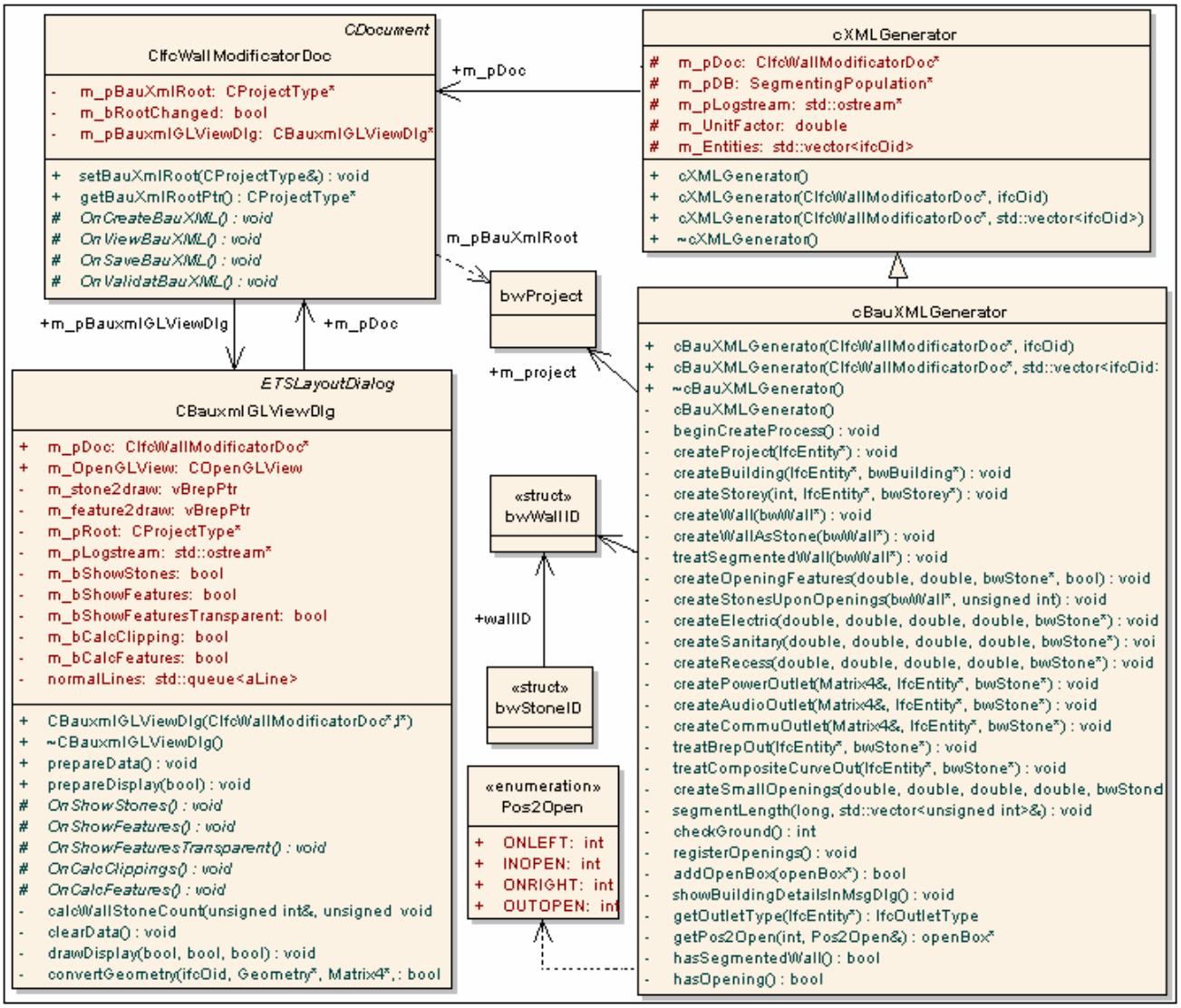


Bild 4.20: UML-Klassendiagram zur Generierung und Visualisierung des BauXML-Modells

Die Klasse „*cBauXMLGenerator*“ ist von der Klasse „*cXMLGenerator*“ abgeleitet, die in der Applikation für die Implementierung allgemeiner XML-basierter Dokumenten benutzt werden, wie z.B. dem GML-Dokument (Geography Markup Languge). In der „*cXMLGenerator*“ stehen allen abgeleiteten Kinderklassen die Zeiger der Dokumentklasse der Applikation *m_pDoc* und die Zeiger der Datenbankklasse *m_pDB:Populationi** zur Verfügung. In der Methode *beginCreateProcess()* werden die Generierung des Projektes, des Gebäudes, des Stockwerks, der Wand, des Steins und eventuell der Features rekursiv durchgeführt. Im Programm wird geprüft, ob eine Wand segmentiert wurde. Wenn nicht, dann wird die Wand als ein Stein mit der Methode *createWallAsStone()* erzeugt, sonst wird die Funktion *treatSegmentedWall()* aufgerufen, in der eine Wand segmentiert wird und Steine mit entsprechender Features erzeugt werden. Vor dem Generierungsprozess kann entschieden werden, wie der obere Teil der Öffnung aussehen soll, ob also ein Sturzelement existieren soll und ob über dem

Sturz eine Übermauerung stehen soll. Dafür zuständig sind die Funktionen wie *registerOpenings()*, *getPos2Open()* und *createStoneUponOpenings()* usw.

Nach der Implementierung des BauXML-Modells wird das Wurzelelement *m_project* an die Dokumentklasse geliefert. Dort kann das ganze BauXML-Modell visualisiert, gespeichert und validiert werden. Zuständig für die Visualisierung ist die Klasse „*cBauxmlGLViewDlg*“. Ausgehend von den generierten BauXML-Daten wird die Geometrie und Topologie aller Steine konvertiert, interpretiert und mit OpenGL-Technik 3D dargestellt.

4.6.7 IfcWallModificator – Programmbedienung

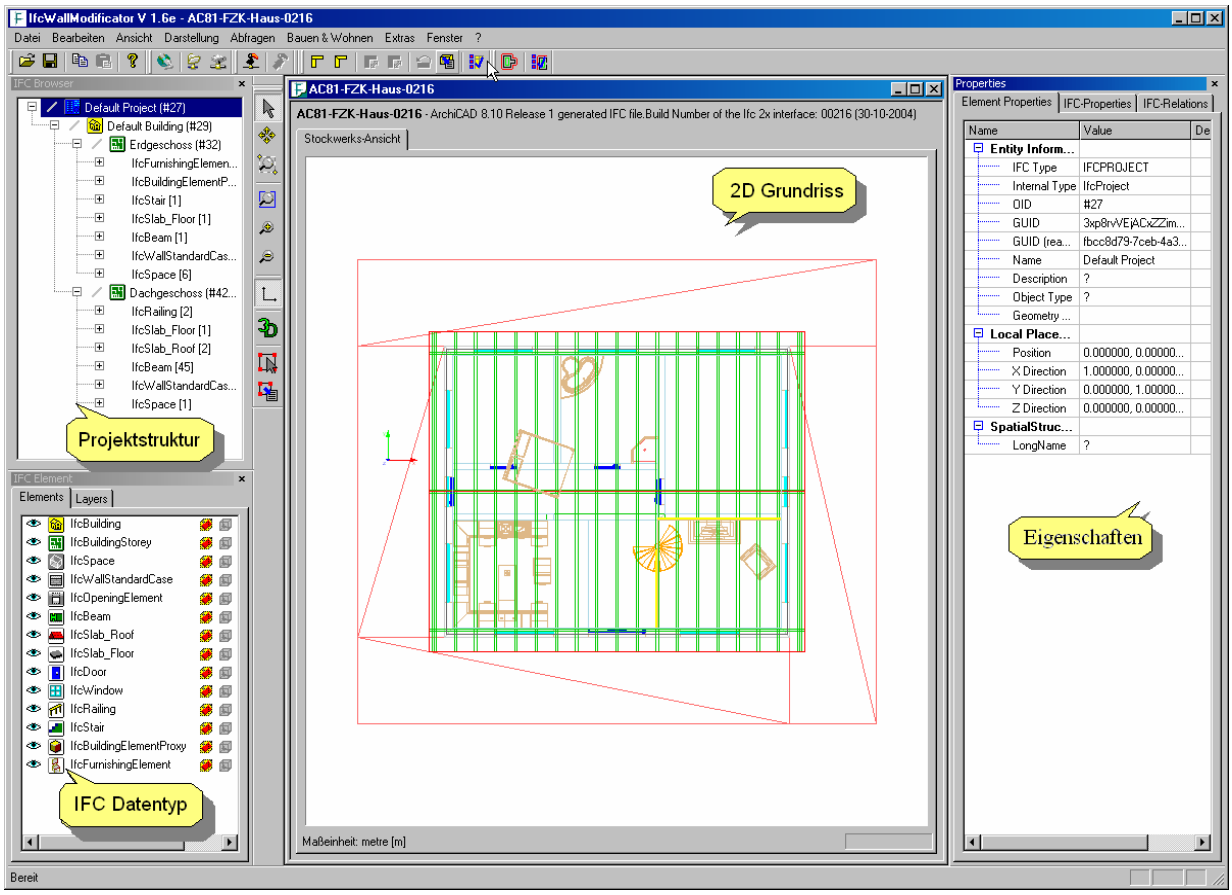


Bild 4.21: Benutzeroberfläche der Applikation „IfcfWallModificator“

Die Benutzeroberfläche des Programms „IfcWallModificator“ gliedert sich in vier Bereiche (Bild 4.21). Das Hauptfenster ist eine 2D-Darstellung des Grundrisses. Alle enthaltenen Objekte werden hier auf den Grundriss projiziert und gezeichnet. Im Fenster oben links befindet sich die Darstellung der Projektstruktur der eingelesenen IFC-Datei. Ausgehend vom Projekt, dem Gebäude bis hin zu den einzelnen Stockwerken können die darin enthaltenen Bauelemente angezeigt werden. Links unten befindet sich eine Übersicht aller enthaltenen IFC-Entitäten. Darin können durch Klick mit der Maus Elemente eines bestimmtem Typs ein- oder ausgeschaltet werden, damit sie visualisiert werden sollen oder nicht. Darüber hinaus können

die Objekt-IDs (OID) und die Führungslinien der Wände angezeigt werden. Auf dem rechten Teil der Benutzeroberfläche befindet sich die Darstellung der Eigenschaften des ausgewählten Elementes. Dazu gehören der IFC-Datentyp, OID, GUID, das Placement sowie eventuell andere Eigenschaften. Außer den grundlegenden Windowsfunktionen wie öffnen, speichern, kopieren und einfügen stellt die Oberfläche zahlreiche Menüs und Icons zu verschiedenen Zwecken bereit.

Hier werden nur die in der Arbeit relevanten Bedienungen vorgestellt, also die Trimmung, die Segmentierung und die Generierung des BauXML-Modells. Sie sind alle unter dem Menu „“Bauen & Wohnen“ erfasst (Bild 4.22).

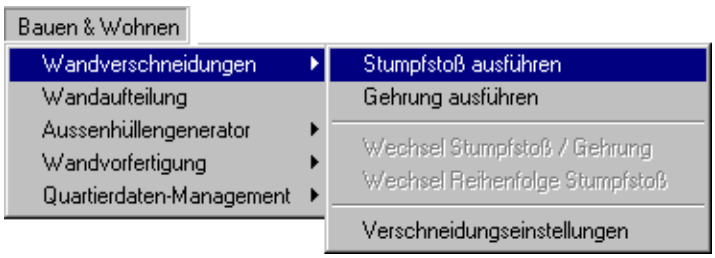


Bild 4.22: Menü „Bauen & Wohnen“

Die Funktionen für das BauXML-Modell werden unter dem Menü „Wandvorfertigung“ zusammengefasst (Bild 4.23). Hier kann das BauXML-Modell generiert, visualisiert, gespeichert und validiert werden. Logischerweise können die Menüpunkte zur Visualisierung und Speicherung erst nach der Generierung des Modells aktiviert werden. Eine XML-Validierung erfolgt immer nach der Speicherung des BauXML-Modells.

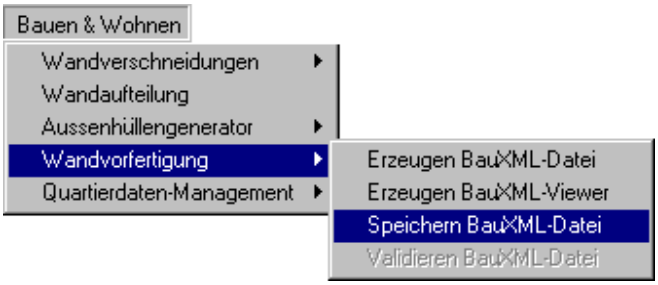


Bild 4.23: Menüpunkte rund um das BauXML-Modell

Wenn der Algorithmus zur Generierung des BauXML-Modells Öffnungen entdeckt, wird ein Fenster geöffnet, in dem entschieden wird, ob Sturzelemente über den Öffnungen existieren und wie sie aussehen (Bild 4.24).

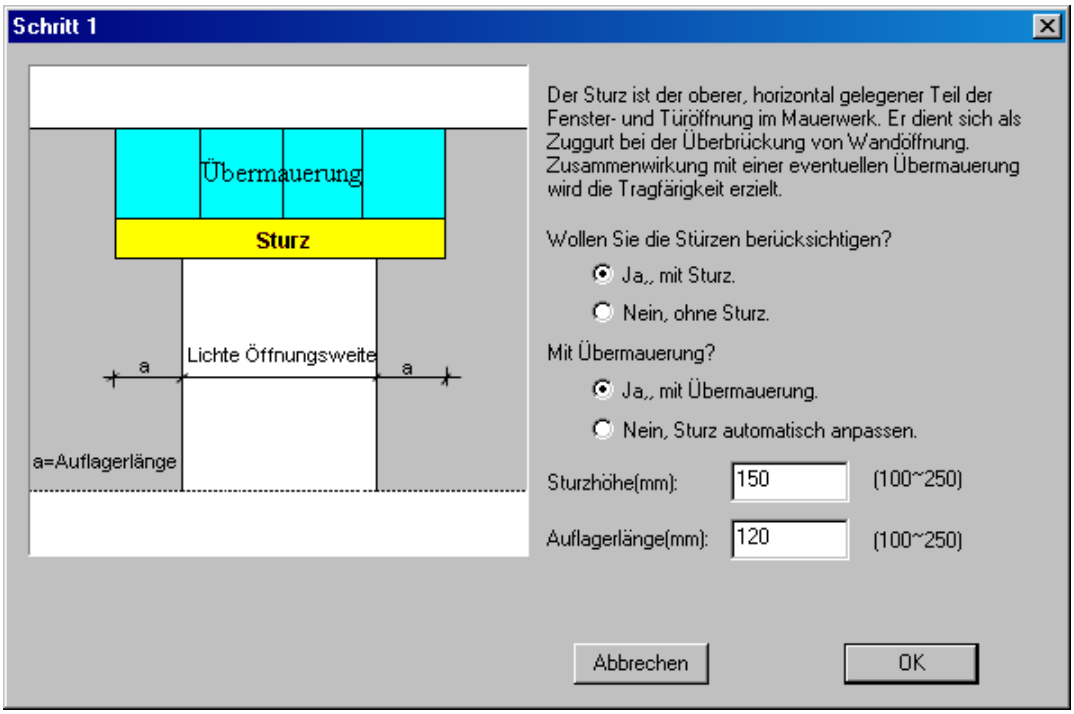


Bild 4.24: Eingabefenster für das Sturzelement

Die für die Wandverschnittkorrektur und Segmentierung notwendigen Parameter werden im Programm global definiert. Sie werden in einem Registerkartenfenster (Bild 4.25), das durch Klick des Menüpunktes „Optionen“ unter „Extras“ geöffnet wird, angegeben.

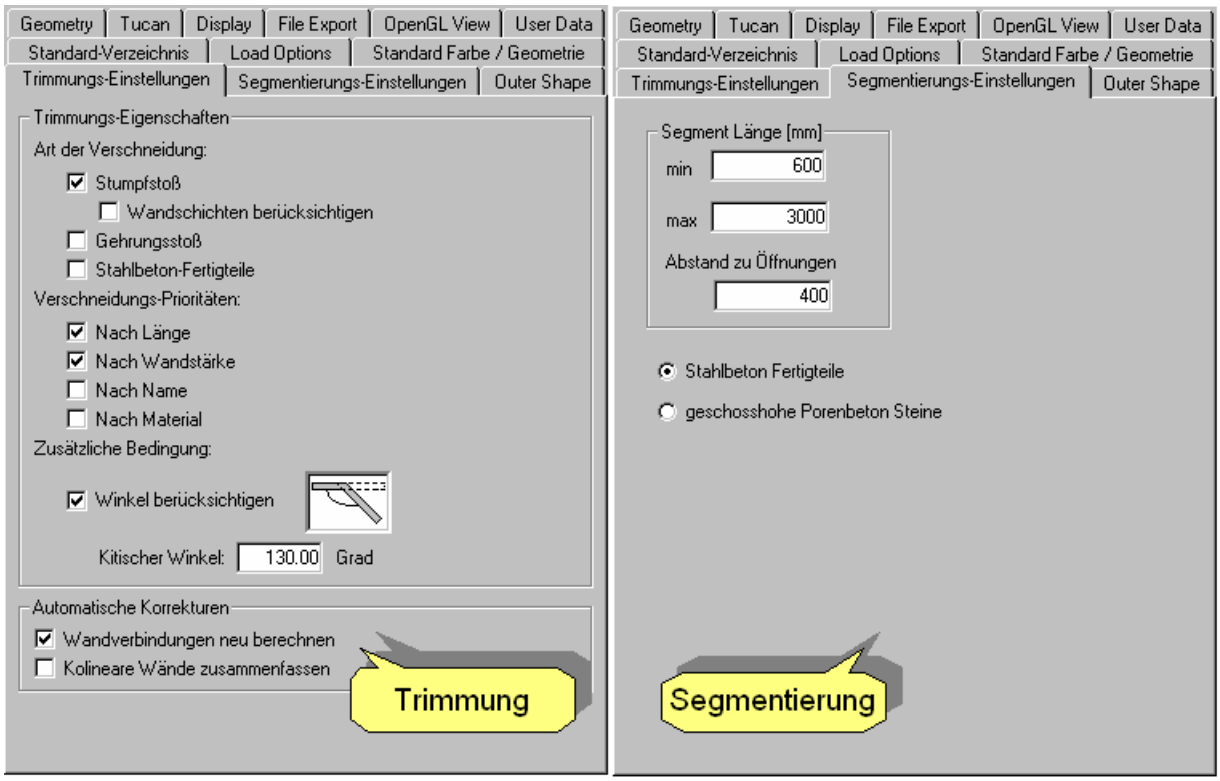


Bild 4.25: Registerkarten für globale Programmeinstellungen

In der Registerkarte „Trimmungs-Einstellungen“ kann der Benutzer die Art der Verschneidung, die Verschneidungs-Prioritäten usw. regeln. In den „“Segmentierungs-Einstellungen“ wird das Wandmaterial ausgewählt. Damit wird im Eingabenfeld der Abstand zu Öffnungen aktiviert oder deaktiviert. Weiterhin können die minimale und die maximale Segmentlänge angegeben werden.

4.7 Evaluierung des erarbeiteten Verfahrens mit Beispielmodellen

4.7.1 FZK-Haus

Das Gebäudemodell FZK-Haus wurde im Institut für Angewandte Informatik (IAI) modelliert, um den IFC-Datenaustausch zu testen. Es besteht aus zwei Stockwerken mit 13 Wänden. Insgesamt 16 Wandöffnungen werden durch elf Fenster und fünf Türen gefüllt. Einige Einrichtungsgegenstände wie Möbel, Badewanne, Kamin usw. wurden im Modell modelliert. Bild 4.26 stellt zwei unterschiedlichen Ansichten dar. Die IFC-Datei ist 2218 KB groß.

Bild 4.26: Modellansicht des FZK-Hauses

Auswertung:

1) Korrektur der Wandverschnitte

Das Modell enthält insgesamt 16 Verbindungsinformationen der Wände untereinander. Die Ergebnisse zu diesem Modell sind in Tabelle 4.3 aufgeführt. Die linke Spalte ist die originale CAD-Darstellung jedes Geschosses, die Spalte in der Mitte zeigt den Wandgrundriss vor der Korrektur der Wandverschnitte (mit originalen IFC-Daten) und in der rechten Spalte wird der Wandgrundriss nach der Korrektur dargestellt. Hieran ist zu sehen, dass die Wandverschnitte vom Gehrungsstoß in Stumpfstoß neu berechnet und korrigiert.

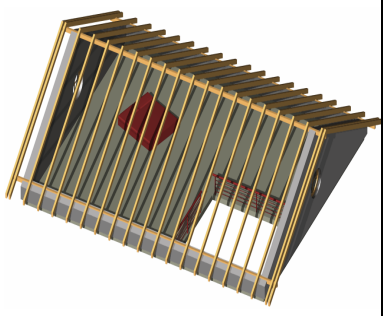
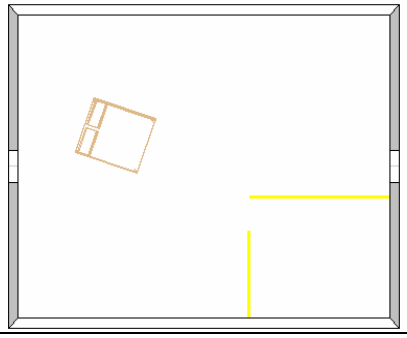
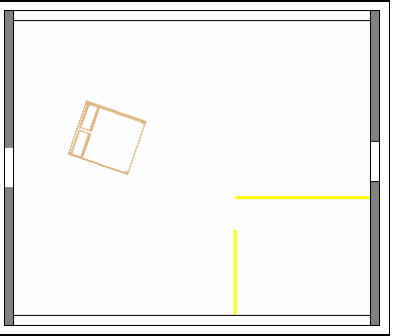
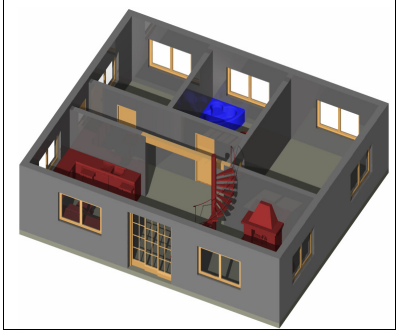
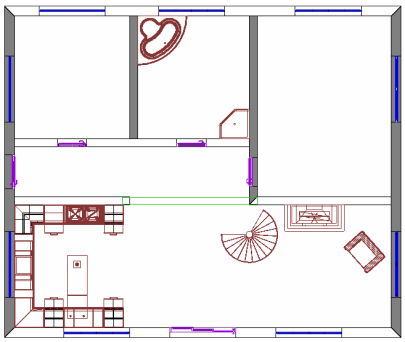
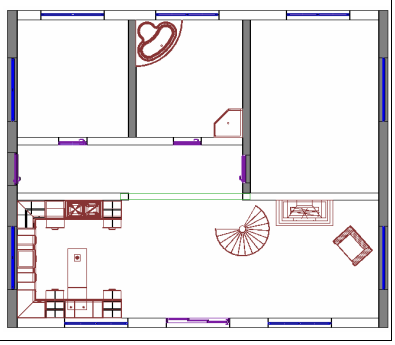
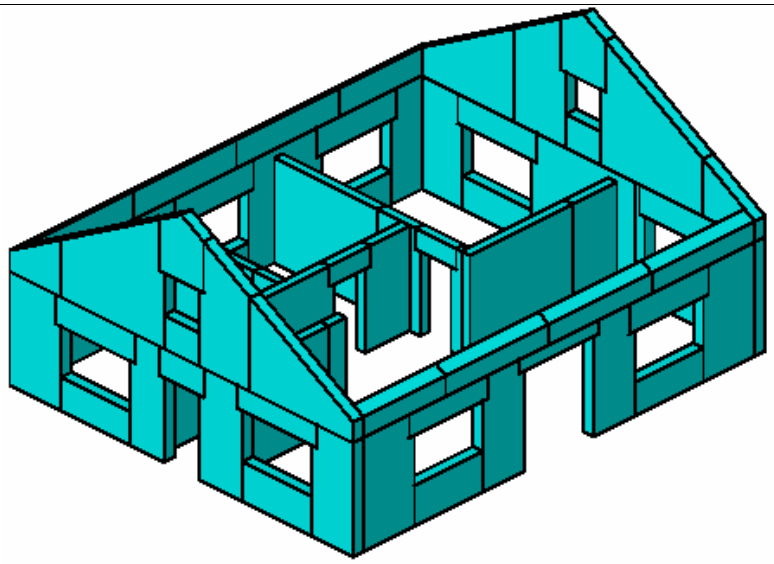
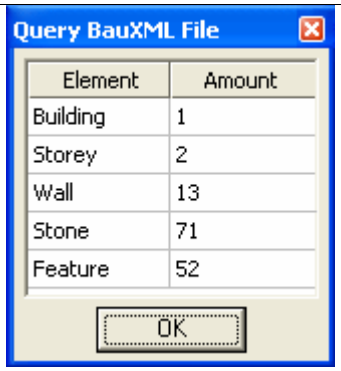
Geschoss	Vor Korrektur	Nach Korrektur
		
		

Tabelle 4.3: Ergebnis der Korrektur der Wandverschnitte des FZK-Hauses

2) Segmentierung und Generierung des BauXML-Modells.

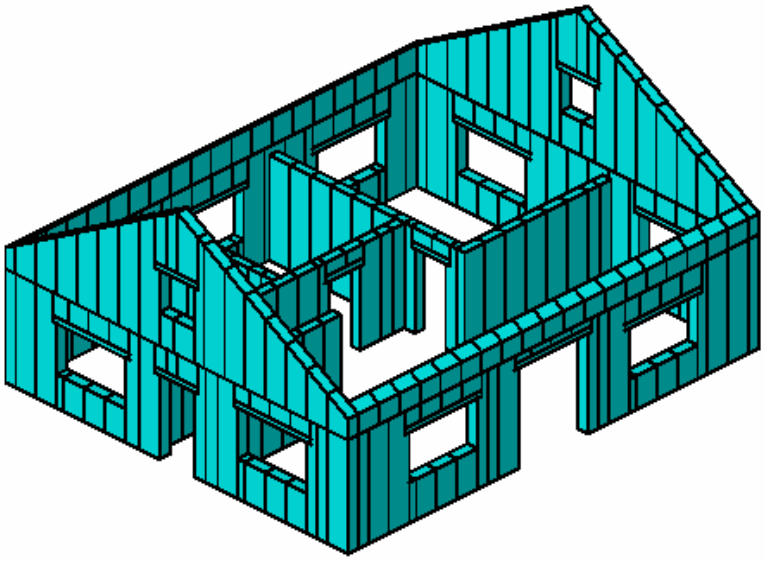
Durch unterschiedliche frei definierbare Einstellungen der Segmentierungsparameter und der Parameter über Stürze werden unterschiedliche BauXML-Modelle generiert. Tabelle 4.4 und 4.5 zeigen zwei BauXML-Modelle bei unterschiedlichen Einstellungen.

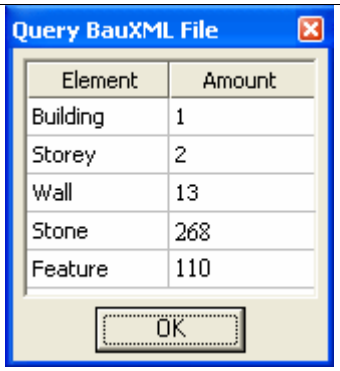




Stein-Minimum: 200 mm
Stein-Maximum: 3000 mm
Sturzhöhe automatisch anpassen
Auflagelänge: 150 mm
Dateigröße: 52 KB

Tabelle 4.4: BauXML-Modell des FZK-Hauses bei Parametersatz (1)





Stein-Minimum: 200 mm
Stein-Maximum: 600 mm
Mit Sturz: Ja
Sturzhöhe: 150 mm
Auflagelänge: 120 mm
Dateigröße: 144 KB

Tabelle 4.5: BauXML-Modell des FZK-Hauses bei Parametersatz (2)

4.7.2 „Niedrigenergiehaus“

Das Haus besteht aus vier Stockwerken, 19 Wänden und 16 Wandöffnungen (Bild 4.27). Die Größe der IFC-Datei beträgt 975 KB.

Bild 4.27: Modellansicht des Niedrigenergiehauses

Auswertung:

1) Korrektur der Wandverschnitte

Das Modell enthält insgesamt 22 Verbindungsinformationen der Wände untereinander. Die Ergebnisse zu diesem Modell sind in Tabelle 4.6 aufgeführt.

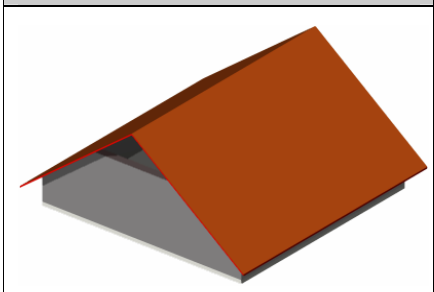
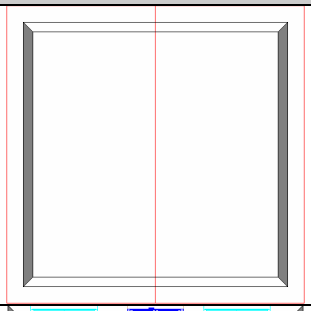
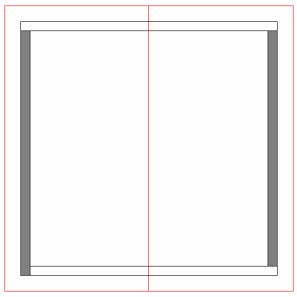
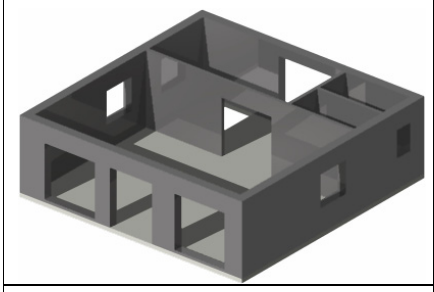
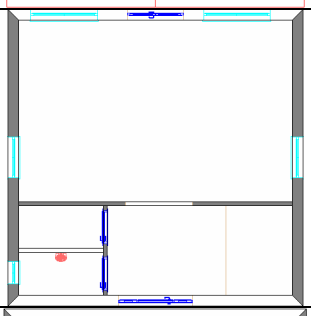
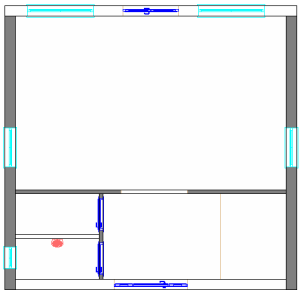
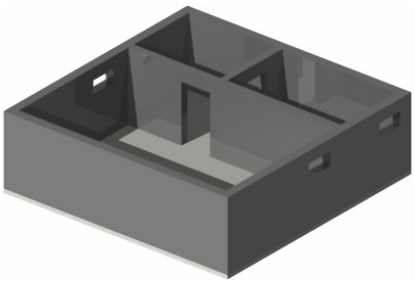
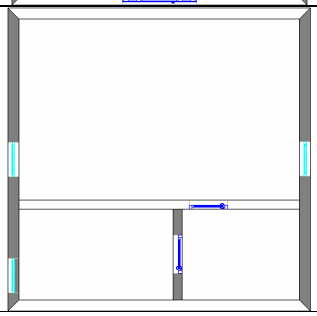
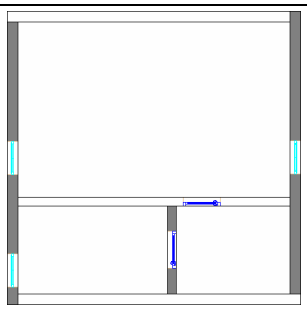
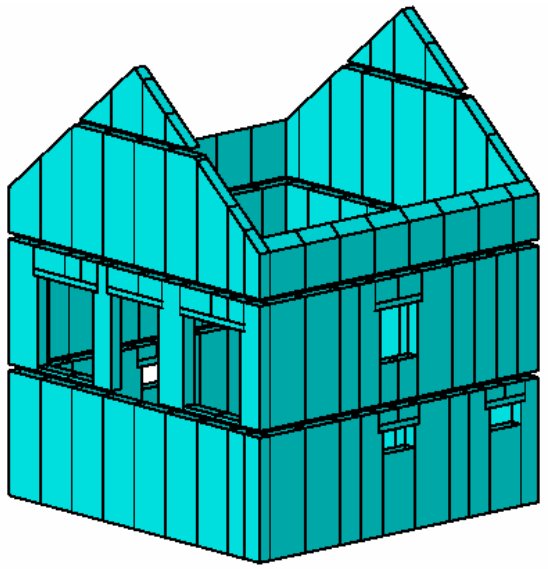
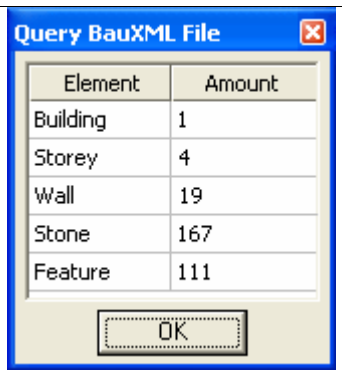
Geschoss	Vor Korrektur	Nach Korrektur
		
		
		

Tabelle 4.6: Ergebnis der Korrektur der Wandverschnitte des Hauses

Das oberste Stockwerk, der Spitzboden, besteht aus zwei gegenüber stehenden Wänden und dem Boden. Es gibt also keine Wandverbindung.

2) Segmentierung und Generierung des BauXML-Modells





Stein-Minimum: 600 mm
Stein-Maximum: 1000 mm
Mit Sturz: Ja
Sturzhöhe: 150 mm
Auflagelänge: 120 mm
Dateigröße: 111 KB

Tabelle 4.7: BauXML-Modell des Niedrigenergiehauses

4.7.3 „Nova-Haus“

Das Nova-Haus-Modell (Bild 4.28) wurde im Institut in Anlehnung an die Grundrisse der Firma Hanse-Haus neu modelliert [NOVA]. Es besteht aus vier Stockwerken und 31 Wänden. Die Größe der IFC-Datei beträgt 7651 KB.

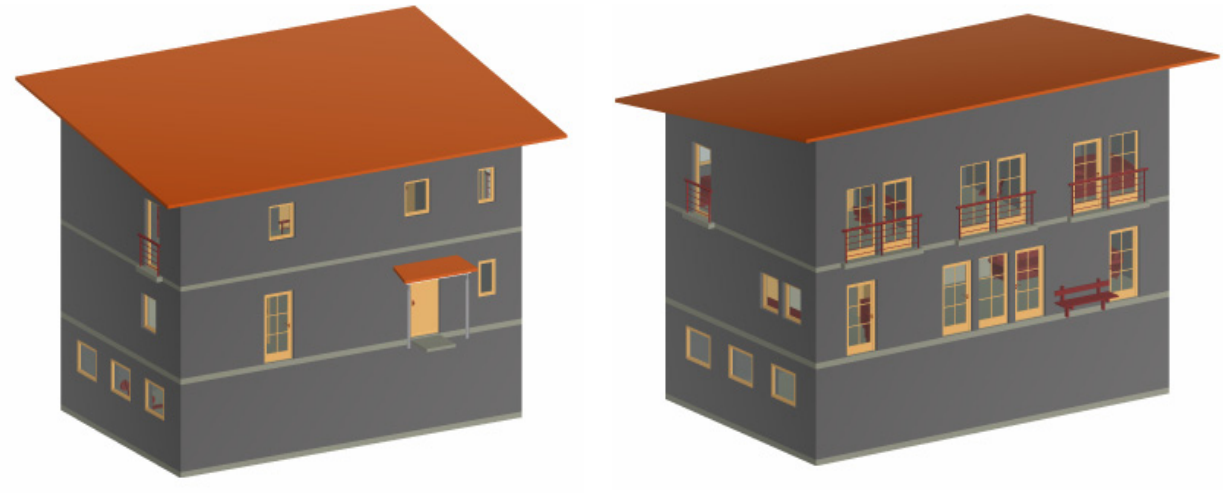
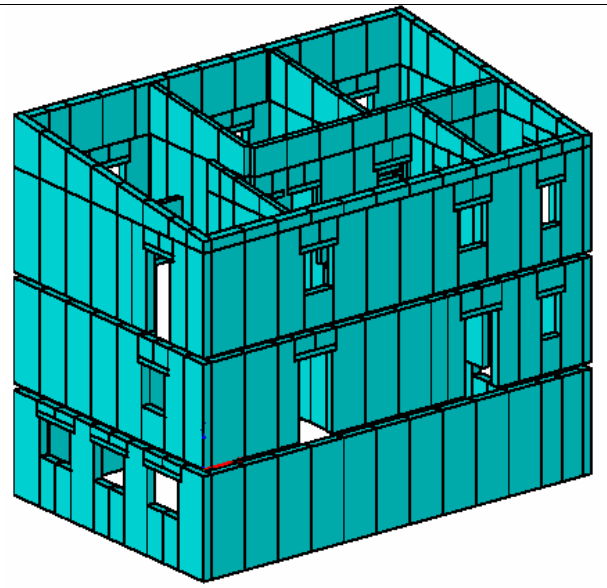
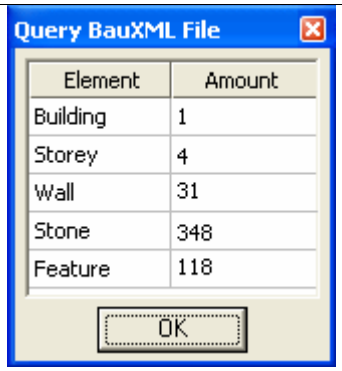


Bild 4.28: Modellansicht des Nova-Hauses

Das Ergebnis des Nova-Modells wird unten nur kurz in der Tabelle 4.8 dargestellt.





Stein-Minimum: 600 mm
Stein-Maximum: 1000 mm
Mit Sturz: Ja
Sturzhöhe: 150 mm
Auflagelänge: 120 mm
Dateigröße: 181 KB

Tabelle 4.8: BauXML-Modell des Nova-Hauses

An den oben vorgestellten Modellen ist deutlich zu sehen, dass die erzeugten Fertigungsfeatures an dem richtigen Stein und an der richtigen Stelle positioniert sind. Es ist dann sicher, dass die Originalmodelle durch die erzeugten BauXML-Daten vollständig regeneriert werden können. Diese Daten gehen in der Wandvorfertigungsfabrik ein und steuern dort die Fertigung.

5 Konzeption und Realisierung einer prototypischen CIM-Wandvorfertigungsfabrik

5.1 Konzept einer Wandvorfertigungsfabrik

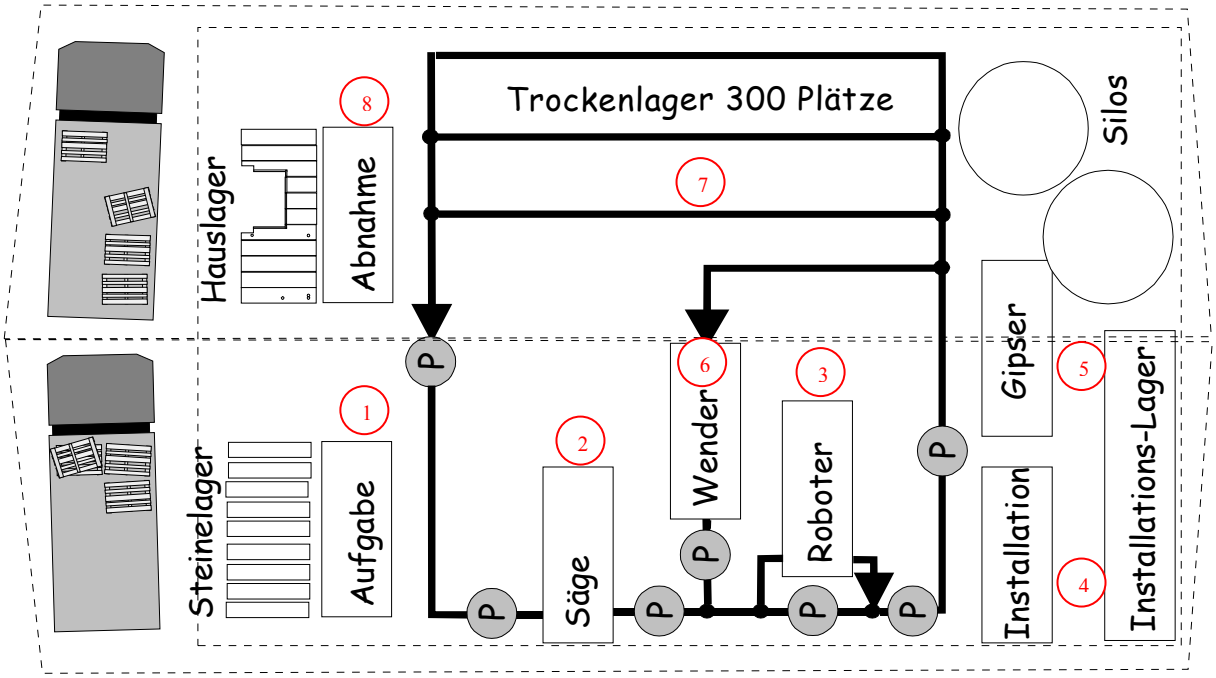


Bild 5.1: Vorfertigungsfabrik der Wandelemente [BRET]

Bild 5.1 zeigt ein grobes Layout der designierten Fabrik zur Vorfertigung der raumhohen Wandelemente. Die Produktionskette beinhaltet von der Aufgabe der Steine bis zum Abstellen im Lager alle nötigen Schritte und Arbeitsstationen. Die Vorfertigung gliedert sich im Wesentlichen in:

1) Aufgabe der Steine auf eine Transportpalette
2) Säge zur Herstellung benötigter Steinmaße
3) Fräsroboter zum Fräsen der Installationen
4) Installation einbauen mitsamt dem dazugehörenden Installationslager
5) Gips auftragen mit Silos für Putzmaterial
6) Wandelement wenden. Bei Bedarf erneuter Durchlauf durch die Stationen, damit die weite Seite bearbeitet werden kann.
7) Trockenlager für ca. 300 Wandelemente.
8) Hauslager bzw. Verladeplatz für LKW

Zwischen den einzelnen Fertigungsstationen sind so genannte „Pufferstationen (P)“ eingerichtet. Sie sind wichtig für den ungestörten Produktionsablauf. Sie haben die Aufgabe unterschiedliche Arbeitsgeschwindigkeiten der einzelnen Arbeitsstationen auszugleichen und damit den steten Materialfluss zu sichern.

Im Folgenden werden die einzelnen Arbeitsstationen in der geplanten Vorfertigungsfabrik kurz vorgestellt. Die erste Arbeitsstation ist die Sägeeinrichtung (Bild 5.2), in der der Stein auf die benötige Steinmaße (Außenkontur) geschnitten wird.

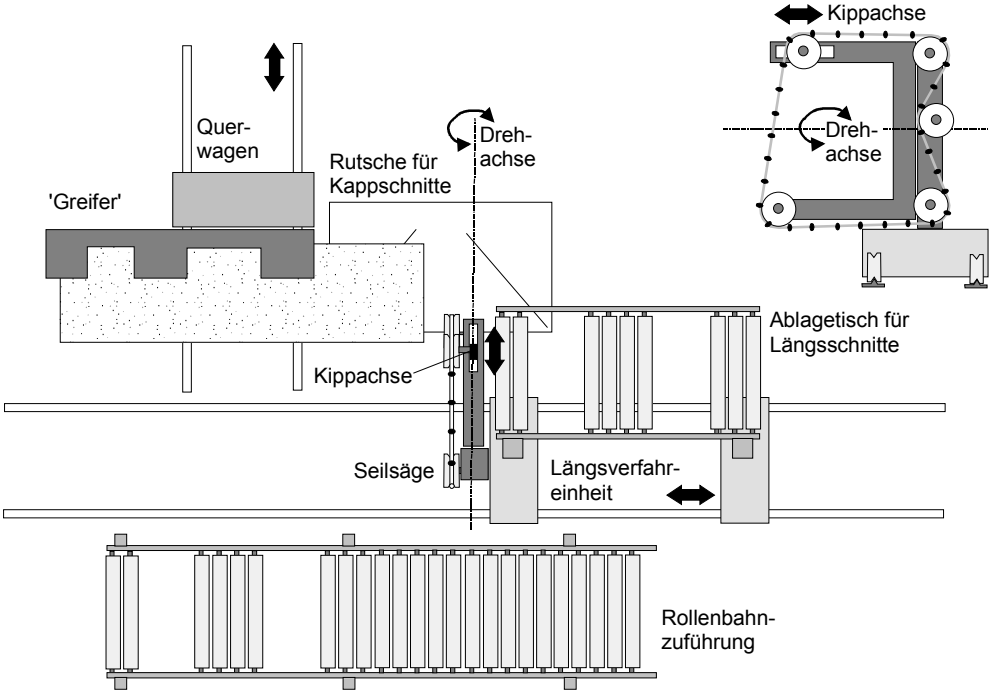


Bild 5.2: Sägeeinrichtung [ISEL]

Nach der Säge gelangt das Einzelelement auf einer Transportpalette zur weiteren Arbeitsstation „Fräsroboter“. In der Roboterzelle (Bild 5.3) werden die Installationsschlitze und Öffnungen von einem Industrieroboter in den Stein gefräst. Dabei ist eventuell ein Werkzeugwechsel erforderlich. Hier können zukünftig auch mehrere Roboter miteinander verknüpft werden, um verschiedene Aufgaben zu übernehmen.

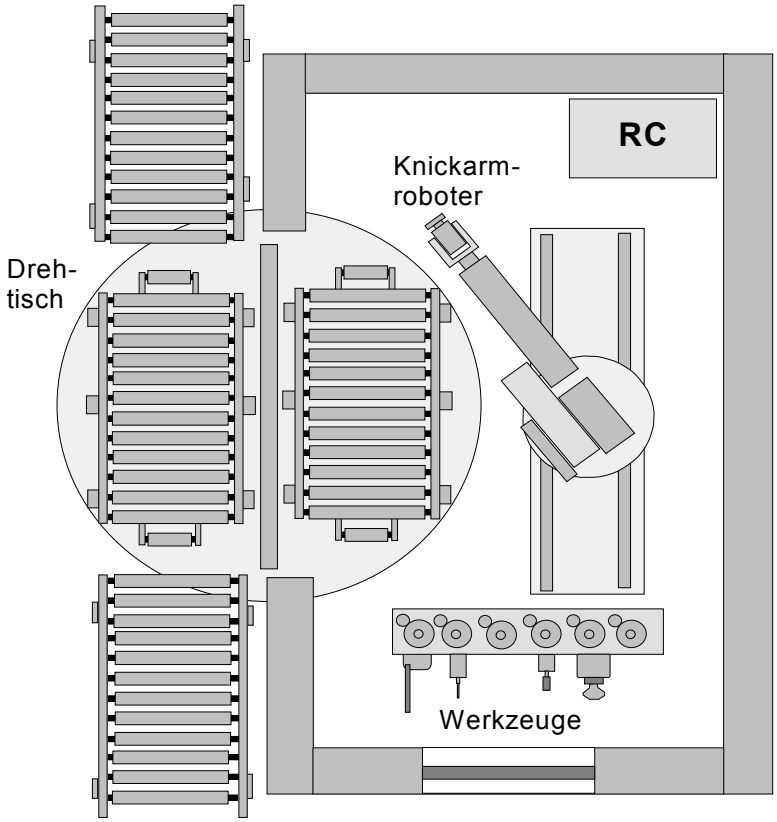


Bild 5.3: Roboterzelle [ISEL]

In die gefrästen Aussparungen werden von einem Arbeiter alle Wandinstallationen eingelegt. Im Weiteren wird der erste Putz auf die Wand aufgebracht und damit auch die Installationen fixiert (Bild 5.4).

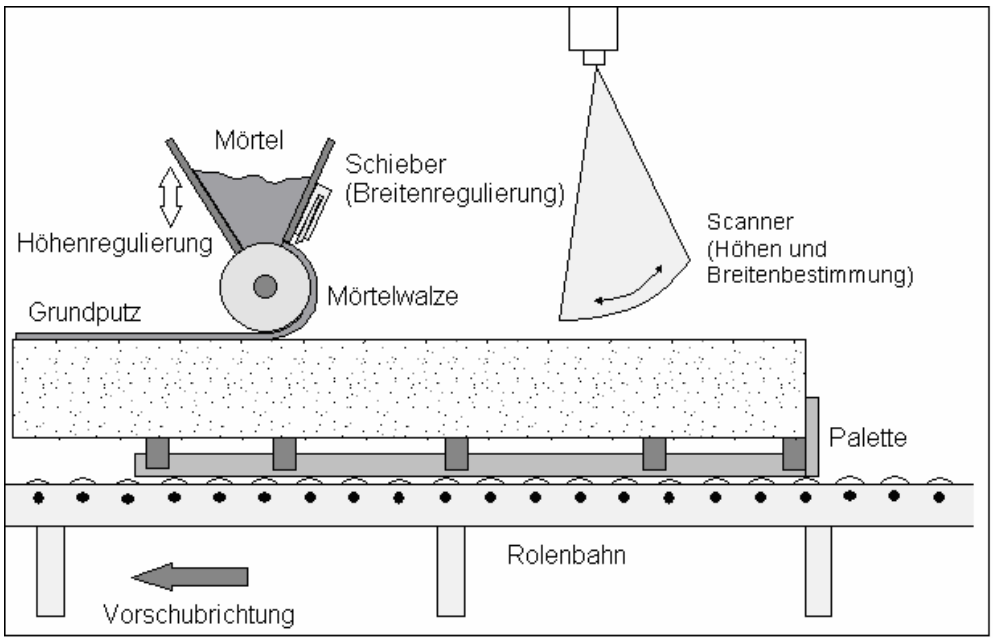


Bild 5.4: Putzeinrichtung [ISEL]

Wenn eine Bearbeitung der anderen Seite gefordert ist, dann kommen die Einzelsteine in den „Wender“ (Bild 5.5).

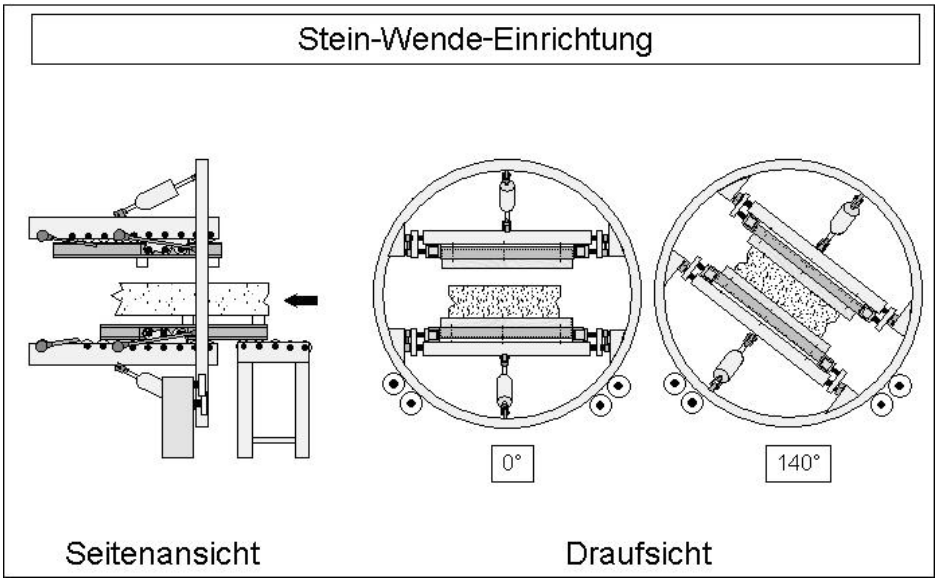


Bild 5.5: Stein-Wende-Einrichtung [ISEL]

Nach dem oben genannten Verlauf wird der Stein von der Transportpalette heruntergenommen und in der Steinlagerpalette abgelegt (Bild 5.6) oder direkt auf den LKW verladen.

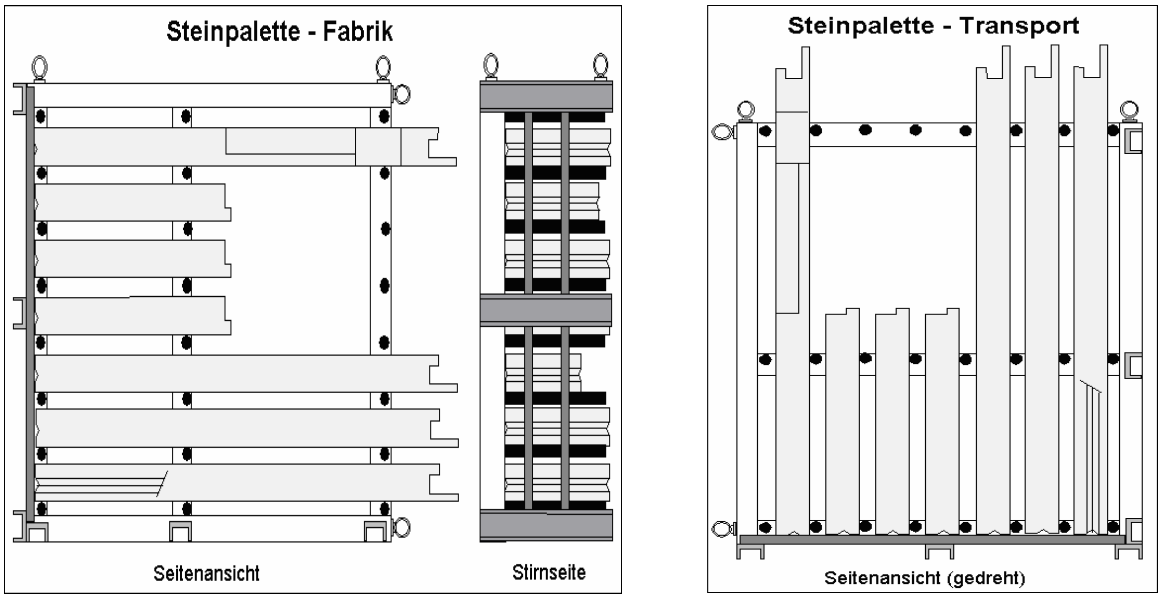


Bild 5.6: Steinpalette [ISEL]

5.2 Prototyp einer automatisierten Roboterzelle zur Wandvorfertigung.

Im Forschungszentrum Karlsruhe wurde eine kleine prototypische Wandvorfertigungsfabrik eingerichtet. Aus Kostengründen wurde statt separater Sägestation und Frässtation nur eine Roboterzelle aufgebaut (Bild 5.7), in der ein 6-Achs KUKA Knickarmroboter eingesetzt wird. Auf einen Drehtisch wurde verzichtet.

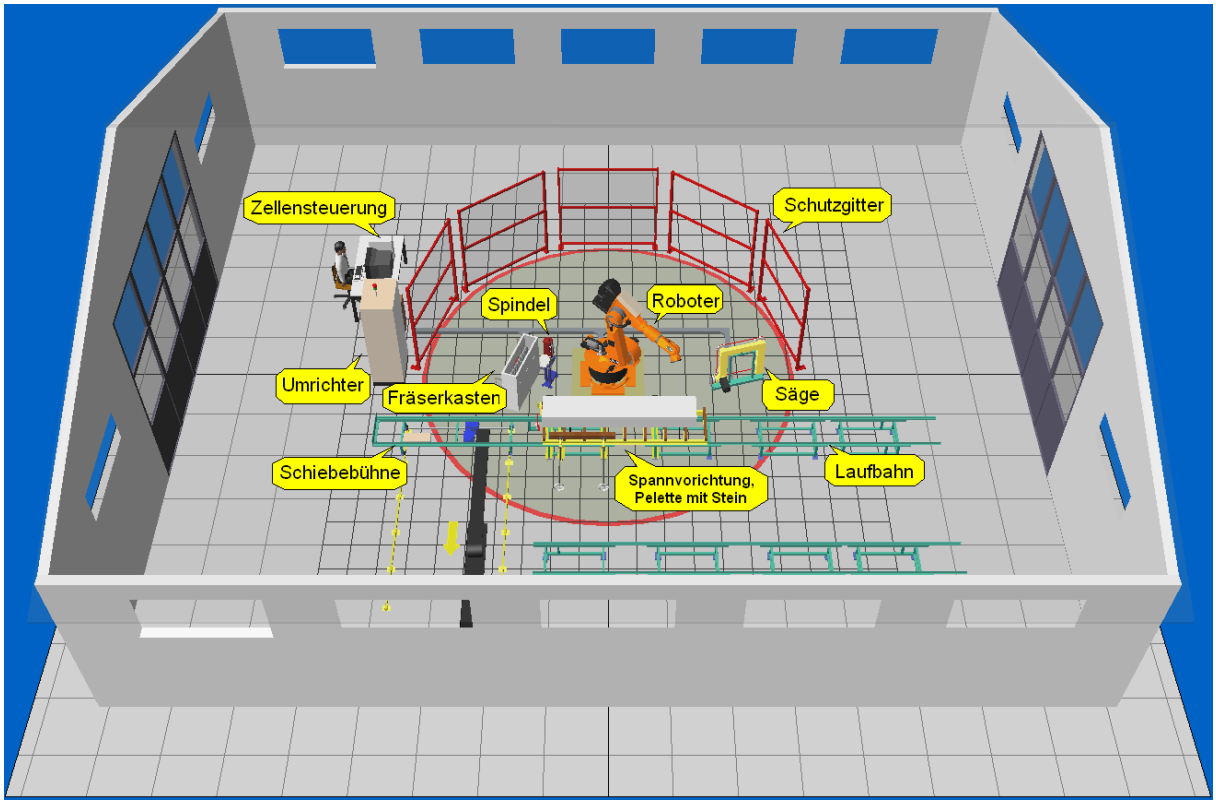


Bild 5.7: Prototyp einer durch Feldbus und OPC vernetzten Roboterzelle

Der Roboter kann eine Seilsäge oder eine Frässpindel aufnehmen, um damit die Steine zu bearbeiten. Mit der Frässpindel kann der Roboter unterschiedliche Fräser aufnehmen. Alle Peripherie wie der Werkzeugspeicher, die Säge, die Frässpindel (siehe Bild 5.8) und die dazu gehörigen verschiedenen Sensoren/Aktoren werden durch den Feldbus DeviceNet in die Robotersteuerung integriert, sodass der Wechsel zwischen der Fräsarbeit und der Sägearbeit, sowie ein Fräserwechsel realisiert werden können.

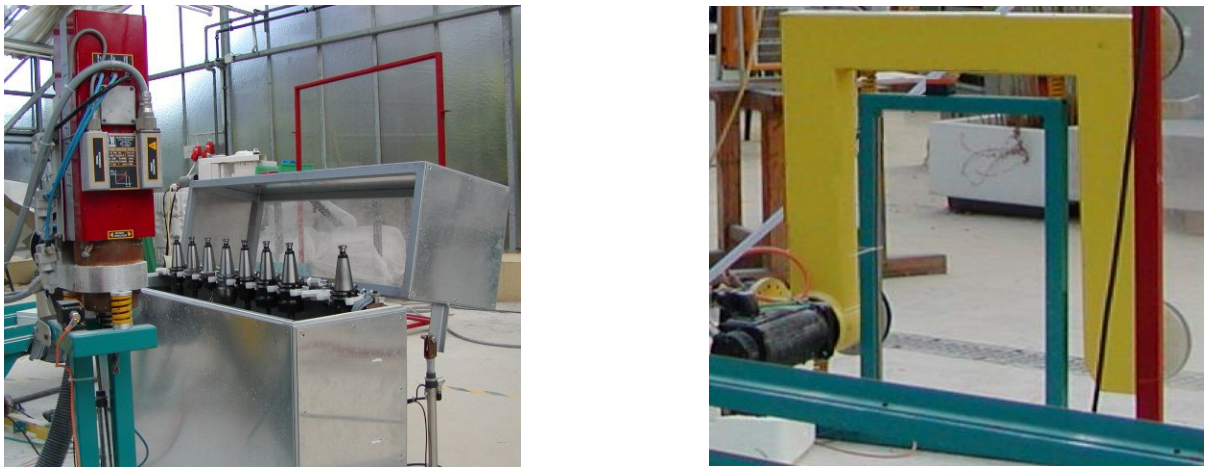


Bild 5.8: Frässpindel und Fräserkasten (links), Seilsäge (rechts)

Ein Zellenrechner gewährleistet die Zusammenarbeit der Robotersteuerung und der Fließbandsteuerung. Beide werden durch eine SPS (Speicherprogrammierbare Steuerung) gesteuert

und kommunizieren mit der Zellensteuerung via OPC (OLE for Process Control) über Ethernet. Dafür wurden zwei OPC-Server im Zellenrechner installiert und konfiguriert.

5.2.1 Fertigung mit einem KUKA Industrieroboter

In der Fertigung wird ein 6-achsiger KUKA Roboter [KUKA] eingesetzt. Bild 5.9 zeigt die gesamte Systemarchitektur der Steuerung KRC1. Kern der Steuerung ist aus Sicht der Hardware ein PC-Motherboard mit Intel Pentium Prozessor und 32MB RAM. Auf dieser Ein-Prozessor-Lösung laufen gleichzeitig zwei Betriebssysteme, nämlich WINDOWS 95 und VxWorks, die über das TCP/IP-Protokoll Daten (z.B. Variablenwerte, Kommandos, Up/Download von Roboterprogrammen) miteinander austauschen [SCHN].

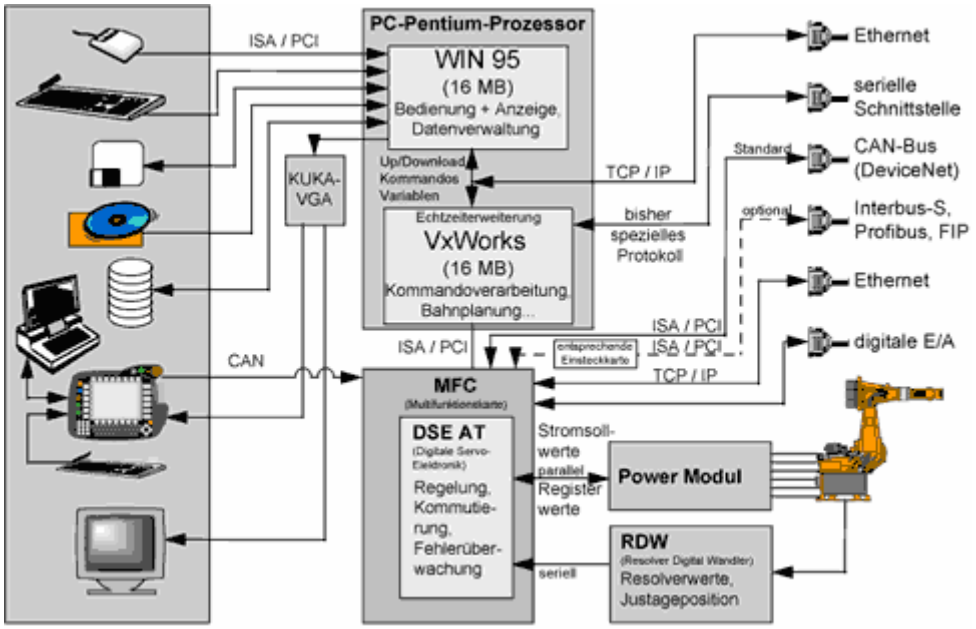


Bild 5.9: Hardwarekonzept der Robotersteuerung KRC1 (Werkbild KUKA)

WINDOWS 95 übernimmt die Bedienung, Programmerstellung, -korrektur, -archivierung, Anzeige und Datenverwaltung usw., also alle Bedienhandlungen des Grundsystems. VxWorks ist ein Echtzeitbetriebssystem und übernimmt alle echtzeitrelevanten Aufgaben wie die Kommandoverarbeitung, die Bahnsteuerung und die Sensor-Integrationen, also die komplette Roboteransteuerung.

Zur Ankopplung von Bussystemen wie DeviceNet, Interbus oder Profibus etc. stehen Schnittstellen zur Verfügung. Über Ethernet und andere Schnittstellen kann die Robotersteuerung mit übergeordneten Steuerungen kommunizieren. Neben der Programmerstellung mit dem Handprogrammiergerät KCP (KUKA Control Panel) ist eine Off-line-Programmierung in der KRL (KUKA Robot Language) möglich.

Auf der Robersteuerung befinden sich zwei Netzwerkkarten jeweils für Windows 95 und VxWorks. Um die benötigten Fertigungsparameter an die Programme zu übertragen, muss die Kommunikation mit der Echtzeitsteuerung unter VxWorks gelingen. Deshalb wurde eine KUKA Soft-SPS mit OPC-Schnittstelle für die Datenübertragung in den Echtzeit-Kern als Konzept gewählt.

5.2.2 DeviceNet integrierte Peripheriesteuerung mit dem Roboter

Die Peripherie wie der Werkzeugspeicher, die Säge, die Frässpindel und die verschiedenen Sensoren/Aktoren werden mit der Robotersteuerung durch CAN-Bus integriert. CAN (Controller Area Network) verbreitet sich immer mehr als Feldbus in Industriebetrieben, wo schnelle Datenübertragungsaufgaben bei meist geringen Datenmengen erledigt werden müssen. Es gibt verschiedene herstellerabhängige CAN Protokollvarianten z.B. CANopen, DeviceNet [KUKA]. Die KRC1-Steuerung von KUKA verwendet ausschließlich das DeviceNet Protokoll. Auf dem DeviceNet arbeitet die Steuerung KRC1 als Master und weist die eindeutigen Bus-Zugriffsprioritäten an die Teilnehmer (Slave) zu.

Um den angeschlossenen CAN-Bus ansprechen zu können, ist eine Projektierung der Buskoppler in der Robotersteuerung KRC1 nötig. In der Datei DEVNET.INI werden die im Netz vorhandenen Buskoppler mit MAC-ID projektiert. Die MAC-ID der Steuerung KRC1 hat als Vorgabe „0“ und braucht nicht in die Datei eingetragen zu werden. Im Moment sind zwei DeviceNet-Koppler in Betrieb, und zwar ein Koppler im Werkzeugkasten und ein Koppler am Roboterarm. Der Koppler im Werkzeugkasten verwaltet alle Sensoren/Aktoren, die für das Schließen/Öffnen der Werkzeugkasten, für die Kennung des herausgenommenen Fräsers, die Frässpindel und die Seilsäge zuständig sind. Der am Roboterarm angebrachte DeviceNet-Koppler steuert die Sensoren und Aktoren, die für die Werkzeugüberwachung am Roboter und alle nötigen Funktionen rund um den Werkzeugwechsel (Entriegeln, Luft blasen, Verriegeln) zuständig sind. Die Datei DEVNET.INI sieht folgendermaßen aus (Tabelle 5.1):

```
DEVNET.INI
[1]
MACID=1
[2]
MACID=2
```

Tabelle 5.1: Projektierung der DeviceNet-Koppler in Robotersteuerung

Neben der Grundkonfiguration und der Vergabe der MAC-IDs muss noch das Eingangs-/Ausgangsabbild für die Module erstellt werden. Diese Zuweisung erfolgt in der Datei IO-SYS.INI. Erst damit erhalten alle Modul-Schnittstellen eine eindeutige und von der KRC1-

Steuerung adressierbare Adresse. Der Block [DEVNET] in der Datei IOSYS.INI ist in Tabelle 5.2 gezeigt.

```
[DEVNET]
INB40=0          ; $IN[321-328]
OUTB40=0         ; $OUT[321-328]
INB41=2          ; $IN[329-336]
INB42=3          ; $IN[337-344]
INB43=4          ; $IN[345-352]
OUTB41=1         ; $OUT[329-336]
```

Tabelle 5.2: Eingangs/Ausgangsabbildung in Robotersteuerung

Die aktuelle Ein-/Ausgangsbelegung in der Robotersteuerung wird in folgender Tabelle 5.3 zusammengefasst.

Eingang		Ausgang	
Nr.	Bedeutung	Nr.	Bedeutung
$IN(321)	Druckwächter auf dem Roboter	$OUT(321)	Fräser entriegeln
$IN(322)	Kein Werkzeug auf dem Roboter	$OUT(322)	Kegel abblasen
$IN(323)		$OUT(323)	Dauerblasen auf dem Lager
$IN(324)		$OUT(324)	
$IN(325)		$OUT(325)	Werkzeugwechsel schließen
$IN(326)		$OUT(326)	Werkzeugwechsel öffnen
$IN(327)		$OUT(327)	
$IN(328)		$OUT(328)	
$IN(329)	Fräserplatz 1 belegt	$OUT(329)	Fräserkasten schließen
$IN(330)	Fräserplatz 2 belegt	$OUT(330)	Fräserkasten öffnen
$IN(331)	Fräserplatz 3 belegt	$OUT(331)	Klappe der Frässpindel öffnen
$IN(332)	Fräserplatz 4 belegt	$OUT(332)	Klappe der Seilsäge öffnen
$IN(333)	Fräserplatz 5 belegt	$OUT(333)	
$IN(334)	Fräserplatz 6 belegt	$OUT(334)	
$IN(335)	Fräserplatz 7 belegt	$OUT(335)	
$IN(336)	Fräserplatz 8 belegt	$OUT(336)	
$IN(337)	Fräserplatz 9 belegt	$OUT(337)	
$IN(338)	Fräserplatz 10 belegt	$OUT(338)	
$IN(339)	Fräserkasten ist zu	$OUT(339)	
$IN(340)	Fräserkasten ist offen	$OUT(340)	
$IN(341)	Frässpindel ist auf der Ablage	$OUT(341)	
$IN(342)	Klappe der Frässpindel ist zu	$OUT(342)	
$IN(343)	Klappe der Frässpindel ist offen	$OUT(343)	
$IN(344)	Seilsäge ist auf der Ablage	$OUT(344)	
$IN(345)	Klappe der Seilsäge ist zu	$OUT(345)	
$IN(346)	Klappe der Seilsäge ist offen	$OUT(346)	

Tabelle 5.3: Adressierung der Sensoren/Aktoren in der Robotersteuerung

Dadurch ist es möglich, von der Robotersteuerung aus auf die an DeviceNet angeschlossenen Sensoren und Aktoren zuzugreifen. Bei der Programmierung können dann die Eingänge abgelesen und die Ausgangssignale gesetzt werden.

In der Roboterzelle kann der Industrieroboter eine Frässpindel und eine Seilsäge aufnehmen. Beide Maschinen sind mit je einem Umrichter ausgestattet. Im Steuerschrank wurden zwei elektrische Einheiten - Signalbox und Leistungsbox eingebaut. Dadurch werden die Leitungsverkabelung und die Signalleitungen für die Seilsäge und die Frässpindel sicher getrennt. Die Frässpindel kann ihren Fräser wie eine Werkzeugmaschine auswechseln. Weil zu unterschiedlichen Fräsern unterschiedliche Drehzahlen einzustellen sind, muss eine Verbindung zwischen der Robotersteuerung und dem Frequenzumrichter hergestellt werden. Durch diese Verbindung kann die Drehzahl von der Robotersteuerung geändert werden.

5.2.3 Drehzahlsteuerung der Frässpindel

Für die Drehzahlsteuerung der Frässpindel wird ein Frequenzumrichter (VLT 5016) von der Firma Danfoss benutzt. Der Umrichter wird benötigt, um den Motor der Frässpindel zu steuern. Bei diesem Motor lässt sich die Drehzahl nur über die Verstellung der Frequenz verändern. Die Kommunikation zwischen dem Frequenzumrichter und der Robotersteuerung wird über eine serielle Schnittstelle realisiert.

Einstellung der Parameter auf dem Frequenzumrichter:

Um eine Kommunikation über die serielle Schnittstelle herzustellen, müssen einige notwendige Parameter am Bedienfeld auf dem Frequenzumrichter eingestellt werden. Sie sind in Tabelle 5.4 aufgelistet.

Parameter		Einstellung
201	Ausgangsfrequenz niedrig	Hier wird die minimale Frequenz gewählt, mit der der Motor angesteuert wird. Hier 0 Hz
202	Ausgangsfrequenz hoch	Hier wird die maximale Frequenz gewählt, mit der der Motor angesteuert wird. Hier 333 Hz
500	Adresse	Hier wird die Adresse des Frequenzumrichters eingestellt. Hier Werkseinstellung 1
501	Baudrate	Hier Werkseinstellung 9600 BAUD
502-508		Hier muss immer „Bus oder Klemme“ eingestellt werden, damit der Motor entweder von der Handbedienung oder der seriellen Schnittstelle aus gesteuert werden kann.
512	Telegrammprofil	Hier wird die Werkseinstellung FC Drive gewählt. Mit dieser Einstellung wird das Steuerwort und Zustandswort beeinflusst.

Tabelle 5.4: Einstellungen auf dem Frequenzumrichter

Einstellung in der Robotersteuerung:

Um den Frequenzumrichter durch die serielle Schnittstelle anzusprechen, wird der COM2-Port in der Robotersteuerung KRC1 benutzt. Hier muss COM2 auf ENABLE eingestellt werden (siehe Tabelle 5.5). COM1 ist für die Mauskommunikation in Windows 95 reserviert.

```
[SERIAL]
COM1=DISABLE      ; Win95 mouse
COM2=ENABLE       ; VxWorks
COM3=DISABLE      ; not implemented
COM4=DISABLE      ; not implemented
```

Tabelle 5.5: Quellcode zur Aktivierung der COM2-Schnittstelle in Robotersteuerung

Des Weiteren muss in der Datei SERIAL.INI die Schnittstelle beschrieben werden. Für COM 2 wird das Protokoll XON/XOFF ausgewählt. Da der Frequenzumrichter das XON und XOFF Zeichen nicht benötigt, wird unter [XONXOFF] XON_VAL = 0 und XOFF_VAL= 0 gesetzt. Zusätzlich ist unter [COM2] die Baudrate (BAUD=9600), die Anzahl Datenbits (CHAR_LEN = 8) und eine gerade Parität (PARITY=2) einzustellen (siehe Tabelle 5.6).

```
; Configuration of the serial ports and theirs procedures
[COM2]
BAUD=9000
CHAR_LEN=8
STOP_BIT=1
PARITY=2
PROC=4

[XONXOFF]
CHAR_TIMEOUT=20          ; msec Timeout after last received character
                         ; to recognize the end of telegram
MAX_TX_BUFFER=2 ; 1..5
MAX_RX_BUFFER=2 ; 1..20
SIZE_RX_BUFFER=100       ; 1..2048 expected telegram length +15 characters
XON_VAL=0                ; 0..255  XON  character (decimal)
XOFF_VAL=0               ; 0..255  XOFF character (decimal)
                         ; if XON_VAL=0 and XOFF_VAL=0, XON/XOFF protocol
                         ; is disabled (pure communication)
```

Tabelle 5.6: Einstellung der Kommunikationsparameter in Robotersteuerung

5.2.4 Drehzahlsteuerung der Seilsäge

Für die Seilsäge, die durch einen SEW-Motor angetrieben wird, wird ein Antriebsumrichter der Firma SEW für die Drehzahlsteuerung benutzt. Der Umrichter wurde über den KRC1 Peripheriestecker X11 direkt an die Robotersteuerung angeschlossen. Dafür wurden die Ausgänge 9~14 und die Spannungsversorgung am Umrichter benutzt, um die Verkabelung mit der Robotersteuerung zu realisieren (Verkabelung siehe Anhang 8.1).

Da das Sägeseil nicht automatisch ausgetauscht wird, ist eine kontinuierliche Drehzahlverstellung des Sägeantriebes nicht erforderlich. Deshalb wird die Drehzahl vom Antriebsumrichter nur als Festsollwert vorgegeben. Dafür müssen die Signale an den Klemmen DIØØ…DIØØ5 so anliegen wie die Tabelle 5.7 zeigt.

Funktion	DIØØ /Regelsperre	DIØ2 Links/Halt	DIØ3 Freigabe/Stopp	DIØ4 n11/n21	DIØ5 n12/n22
Regelsperre	0	x	x	x	x
Schnellstopp	1	x	0	x	x
Freigabe und Halt	1	0	1	x	x
Linkslauf mit 50% n	1	1	1	1	0
Linkslauf mit 100% n	1	1	1	0	1

Tabelle 5.7: Signalbelegung für den Antrieb der Seilsäge mit den Festsollwerten

Durch Setzen der entsprechenden Ausgänge ist es möglich, die Seilsäge zu steuern. Das Programm zum Einschalten der Säge ist in Tabelle 5.8 zu sehen:

```
1.  DEF SaegeEin(DREHZAHL)
2.  INT DREHZAHL                    ; Einheit mit 1/min
3.  INI
4.  PTP SAEGEHOME Vel= 100% DEFAULT
5.  $OUT[9] = true                  ; RegelSperre
6.  $OUT[10] = true                 ; Freigabe
7.  $OUT[11] = true                 ; Links
8.  IF DREHZAL == 2000 THEN
9.  $OUT[12] = true
10. ELSE
11. IF DREHZAHL == 4500 THEN
12. $OUT[13] = true
13. ENDIF
14. ENDIF
15. PTP SAEGEHOME Vel= 100% DEFAULT
```

Tabelle 5.8: Roboterprogramm zum Einschalten der Säge mit bestimmter Drehzahl

Es wurden drei Drehzahlstufen programmiert. Wenn die Ausgänge $OUT[12] und $OUT[13] nicht gesetzt werden, dann läuft die Säge mit 500 1/min. Wenn $OUT[12] gesetzt wird aber $OUT[13] nicht, ist die Drehzahl 2000 1/min. Die Säge läuft mit 4500 1/min wenn $OUT[13] gesetzt wird aber $OUT[12] nicht.

Das Programm zum Ausschalten der Säge ist recht einfach (Tabelle 5.9):

```
1.  DEF SaegeAus()
2.  $OUT[9]  = FALSE
3.  $OUT[10] = FALSE
4.  $OUT[11] = FALSE
```

Tabelle 5.9: Roboterprogramm zum Ausschalten der Säge

Nun ist es möglich, in der Robotersteuerung durch Aufruf des entsprechend vorbereiteten Programms (ohne oder mit Parameter) die Seilsäge und die Frässpindel zu bedienen und zu steuern.

5.2.5 Dezentrale SPS-Steuerung des Fördersystems

Um die Fertigungsprozeduren zu testen, wurde in der prototypischen Fertigungsfabrik ein Fördersystem aus einem Schienensystem und einer Schiebebühne eingebaut. Ein Stein wird auf eine Palette gelegt und auf der Laufbahn an die Bearbeitungsposition transportiert und fest gespannt. Nach der Bearbeitung wird die Palette zusammen mit dem Stein auf die Schiebebühne weiterlaufen. Die Schiebebühne fährt die Palette mit dem Stein zur nächsten Station. In Bild 5.10 werden ein Foto (links) und eine grobe Skizze (rechts) des Fördersystems gezeigt (vgl. Bild 5.7).

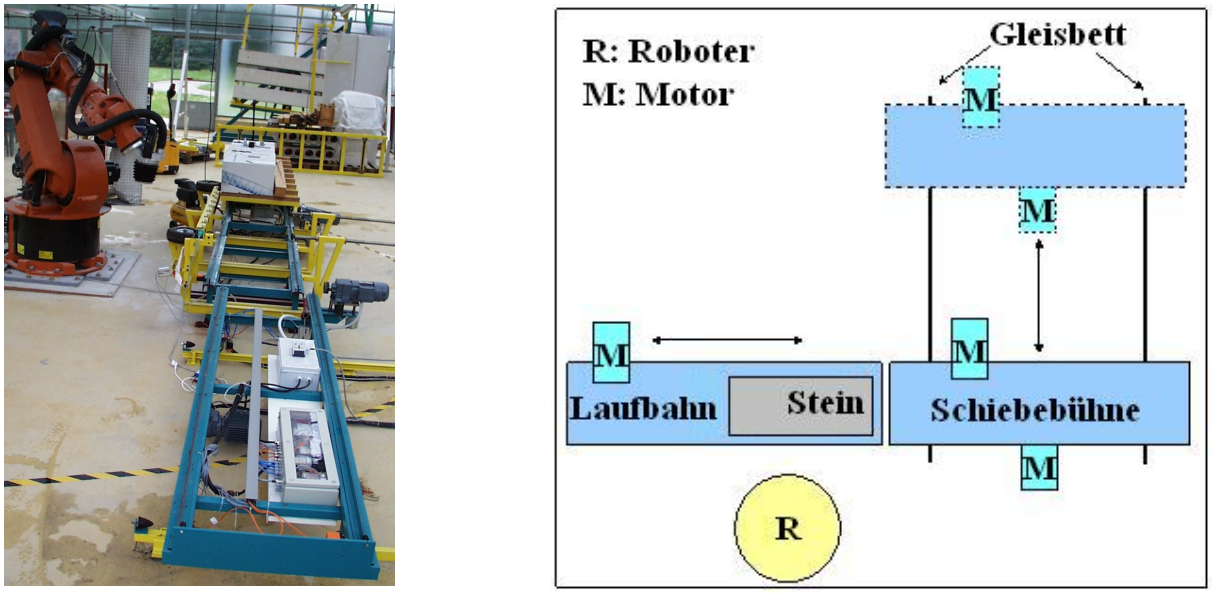


Bild 5.10: Fördersystems

Für die Steuerungen beider Bahnen wurden zwei programmierbare Ethernet-Controller und eine Reihe von I/O-Klemmen der Firma WAGO [WAGO] verwendet (siehe Anhang 8.6). Die Controller kombinieren die ETHERNET TCP/IP-Funktionalität mit der Speicherprogrammierbaren Steuerung (SPS). Sämtliche Eingangssignale der Sensoren werden im Controller zusammengeführt. Entsprechend der IEC 61131-3-Programmierung (Anhang 8.2) erfolgt die Bearbeitung der Prozessdaten vor Ort in dem Controller. Die daraus erzeugten Verknüpfungsergebnisse können direkt an die Aktoren ausgegeben oder über den Feldbus an die übergeordnete Steuerung übertragen werden. Die SPS-Programmierung des Controllers wurde im Programm „WAGO-IO-Pro 32“ mit der grafischen orientierten SPS-Programmiersprache Funktionsplan (FUP) realisiert.

5.2.5.1 SPS-Steuerung der Laufbahn

Eine Palette wird über einen Motor mit zwei Geschwindigkeiten und Richtungen transportiert. Der Antriebmotor ist kippbar am Schienensystem angebracht und wird mittels eines Pneuma-

tikzylinders angehoben, damit ein hoher Anpressdruck zwischen Antriebrad des Motors und der Palette resultiert. Zum Stoppen und Spannen der Palette ist ein Stopper am Ende des Schienensystems eingerichtet. Der Stopper wird mittels eines Pneumatikzylinders gekippt.

I/O-Belegungen:

Zusammen mit dem Ethernet-Controller WAGO 750-842 werden sechs Eingangsmodule 750-410, drei Ausgangsmodule 750-517 (Anhang 8.6) und ein Pneumatikmodul benutzt. Bild 5.11 und Tabelle 5.10 zeigen die jetzigen I/O-Belegungen.

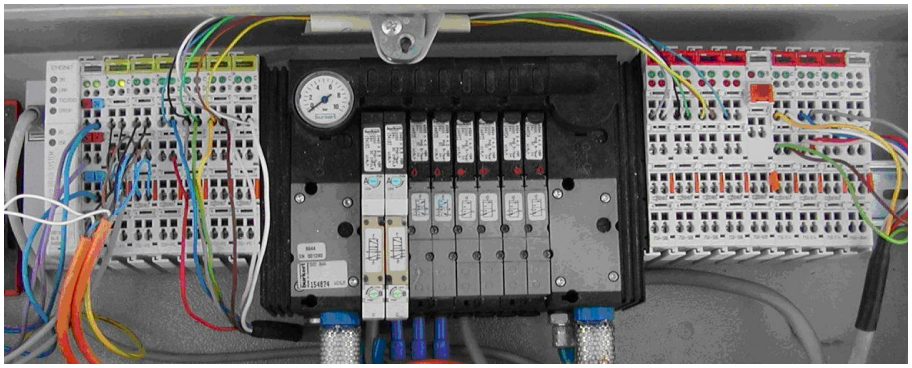


Bild 5.11: Steuereinheit der Laufbahn

	Variable	Adresse	Kommentare
Eingänge	IIS6	%IX0.0	Sensor: Stopper eingefahren, aktiviert.
	IIS5	%IX0.1	Sensor: Stopper ausgefahren, deaktiviert.
	SStopper	%IX0.2	Sensor: Stein am Stoppe
	SLangsam	%IX0.3	Sensor: Palette kurz vor dem Stopper → langsam
	IPV1	%IX0.5	Schalter: Motor anpressen
	IM1SV	%IX0.6	Schalter: Motor schnell nach vorne fahren
	IM1SZ	%IX0.7	Schalter: Motor schnell zurück fahren
	IM1LV	%IX0.8	Schalter: Motor langsam nach vorne fahren
	IM1LZ	%IX0.9	Schalter: Motor langsam zurück fahren
	ISTV	%IX0.10	Schalter: Stopper nach vorne fahren
	ISTZ	%IX0.11	Schalter: Stopper zurück fahren
Ausgänge	QSV	%QX0.0	Ankopplungszylinder verriegeln
	QSZ	%QX0.1	Ankopplungszylinder entriegeln
	QPV1	%QX0.6	Palettenverriegelungszylinder entriegeln
	QM1SV	%QX0.26	Palettenverriegelungszylinder verriegeln
	QM1LV	%QX0.27	Zylinder für den Transportmotor anpressen
	QM1SZ	%QX0.28	Zylinder für den Transportmotor lösen
	QM1LZ	%QX0.29	Antriebsmotor nach vorne fahren

Tabelle 5.10: I/O-Belegung der Steuerung der Laufbahn

Die Quellcodes der SPS-Programmierung sind im Anhang 8.3 dargestellt.

5.2.5.2 SPS-Steuerung der Schiebebühne

Die Schiebebühne pendelt zwischen dem Schienensystem der Bearbeitungsstation und dem parallelen Schienensystem, um den gefertigten Stein weiter zu transportieren. Eingesetzt werden zwei Motoren. Ein Motor, der Antriebsmotor, treibt die Schiebebühne auf dem Gleisbett an. Der andere Motor, eine Transportmotor, ist für den Antrieb der Palette auf der Schiebebühne zuständig (vgl. Bild 5.10).

I/O-Belegungen:

Zusammen mit dem Ethernet-Controller WAGO 750-842 werden sieben Eingangsmodule 750-410, drei Ausgangsmodule 750-517 und ein Pneumatikmodul mit vier 5/2-Ventile benutzt. Tabelle 5.11 und Bild 5.12 zeigen die I/O-Belegungen.

	Variable	Adresse	Kommentare
Eingänge	SZ1O	%IX0.0	Sensor: Ankopplungszylinder OBEN -> verriegelt
	SZ1U	%IX0.1	Sensor: Ankopplungszylinder UNTEN-> entriegelt
	SZ2O	%IX0.2	Sensor: Verriegelungszylinder OBEN -> entriegelt
	SZ2U	%IX0.3	Sensor: Verriegelungszylinder UNTEN --> verriegelt
	SZ3O	%IX0.4	Sensor: Anpresszylinder für Transportmotor OBEN → keine Palette im Übergabebereich
	SBP1	%IX0.5	Sensor: Schiebebühne beim Roboter
	SBP2	%IX0.6	Sensor: Schiebebühne bei der Wendeeinrichtung
	SPP	%IX0.7	Sensor: Palette auf der Schiebebühne ist in Position.
	BZ1	%IX0.8	Schalter: Ankopplung aktivieren/deaktivieren
	BZ2	%IX0.9	Schalter: Palettenverriegelung aktivieren/deaktivieren
	BMAV	%IX0.10	Taster: Antriebsmotor nach vorne fahren
	BMAZ	%IX0.11	Taster: Antriebsmotor zurück fahren
	BMTLV	%IX0.12	Taster: Transportmotor langsam nach vorne fahren
	BMTLZ	%IX0.13	Taster: Transportmotor langsam zurück fahren
Ausgänge	Z1_AUF	%QX0.0	Ankopplungszylinder verriegeln
	Z1_AB	%QX0.1	Ankopplungszylinder entriegeln
	Z2_AUF	%QX0.2	Palettenverriegelungszylinder entriegeln
	Z2_AB	%QX0.3	Palettenverriegelungszylinder verriegeln
	Z3_AUF	%QX0.4	Anpresszylinder für den Transportmotor anpressen
	Z3_AB	%QX0.5	Anpresszylinder für den Transportmotor lösen
	MA_V	%QX0.20	Antriebsmotor nach vorne fahren
	MA_Z	%QX0.21	Antriebsmotor zurück fahren
	MT_LV	%QX0.22	Transportmotor langsam nach vorne fahren
	MT_LZ	%QX0.23	Transportmotor langsam zurück fahren
	MT_SV	%QX0.24	Transportmotor schnell nach vorne fahren
	MZ_SZ	%QX0.25	Transportmotor schnell zurück fahren

Tabelle 5.11: I/O-Belegung der Steuerung der Schiebebühne

Anhang 8.4 stellt die Quellcodes der SPS-Steuerungen für die Schiebebühne.

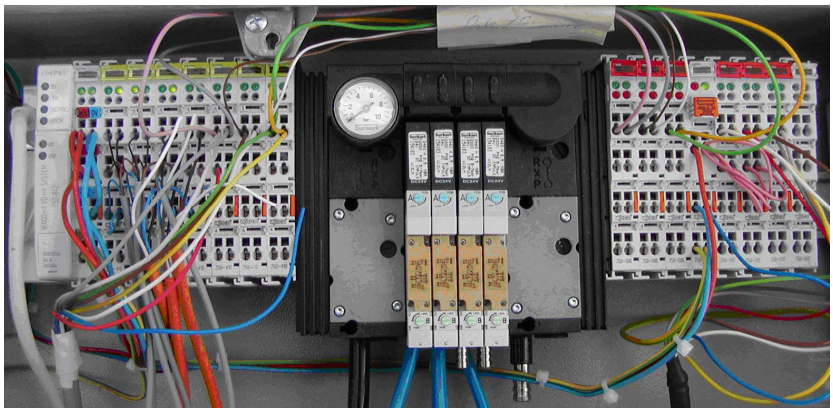


Bild 5.12: Steuereinheit der Schiebebühne

5.2.6 Übergeordnete Zellensteuerung via OPC (OLE for Process Control)-Technologie

Der Einsatz von Software spielt in der Automatisierung eine immer bedeutendere Rolle. Ob zur Bedienung, Visualisierung, Archivierung oder Steuerung, der Trend zu reinen, meist PC basierten Softwarelösungen ist unübersehbar. Längst werden diese Softwarelösungen nicht mehr als monolithische Blöcke entwickelt, sondern setzen sich vielmehr aus einzelnen Softwarekomponenten zusammen. Für jede neue Kombination von Softwaremodulen müssen Zeit und Kosten für die Anpassung der Kommunikationsschnittstelle investiert werden. So gibt es zum Beispiel für Prozessleit- oder Visualisierungssysteme hundert Softwaretreiber, um mit den Peripheriegeräten zu kommunizieren [SOFT]. Aus diesem Grunde haben sich vor einigen Jahren einige namhafte und marktführende Hersteller zusammengeschlossen, um einen neuen Standard für die Anbindung von Automatisierungssoftware und –hardware zu etablieren. So entstand die OPC-Foundation.

OPC steht für „OLE for Process Control“ und ist ein offener Schnittstellen-Standard. Aufbauend auf der Windows-basierten Technologie von OLE (Object Linking and Embedding), COM (Component Object Model) und DCOM (Distributed COM) ermöglicht er einen einfachen standardisierten Datenaustausch zwischen Automatisierungs-/Steuerungsanwendungen, Feldgeräten und Büroanwendungen (Überwachung, Planung). OPC vereinfacht die Anbindung von Automatisierungskomponenten unterschiedlicher Hersteller an PC-Applikationen wie z.B. Prozessleit- und Visualisierungssysteme. Mit der Einführung dieser standardisierten Schnittstelle zwischen Windows-Programmen reduziert sich für Hardwarehersteller die Zahl der Treiberentwicklungen für ihre Komponenten auf eine einzige, den OPC-Server. Ebenso ist für den Softwarehersteller nur noch eine einzige Treiberanbindung erforderlich, die OPC-Client Schnittstelle.

Ein **OPC-Server** besteht aus drei hierarchisch abgestuften Objekten: Server, Gruppen und Items. Der standardisierte Datenaustausch erfolgt über frei definierbare „OPC-Items“, die auf Prozessgrößen abgebildet werden. Diese Items sind innerhalb des OPC-Servers eindeutig. Sie können nach logischen und dynamischen Gesichtspunkten in einer oder mehreren so genannten „OPC-Groups“ zusammengefasst werden. OPC-Items und OPC-Groups laufen innerhalb einer einzigen Software-Komponente, dem OPC-Server. Jedes Item enthält eine Typbeschreibung, einen aktuellen Wert, einen Zeitstempel, eine Variable für die Schreibberechtigung durch den Client und eine Qualitätsangabe. Items erhalten vom Server ihre aktuellen Werte. Eine Aktualisierung der Werte für den Client geschieht immer dann, wenn die in der Aktualisierungsrate angegebene Zeit abgelaufen ist, allerdings nur dann, wenn auch schon neue Werte vom Server bereitstehen. Umgekehrt kann jedes Item auch einen neuen Wert vom Client bekommen und im Server überschrieben werden.

Es ist möglich, dass mehr als nur ein Client gleichzeitig auf den OPC-Server zugreift. Ein OPC Client kann über die verschiedenen Interfaces, die die Server bereitstellen, Informationen über den Namensraum der Servers erhalten und so alle bereitgestellten Items ermitteln.

Um Daten von einer anderen Applikation zu erhalten oder an sie zu schicken, öffnet das OPC-Client Programm einen Kanal zum OPC-Server. Dabei werden der Name des Servers, die vom Server bereitgestellten Informationen (z.B. Adresse der SPS/Teilnehmeradresse) über den Namensraum und alle im OPC-Server bereitgestellten Items ermittelt. Existiert ein entsprechender OPC-Server, dann erhält der Client eine positive Rückantwort. Nach dem Kommunikationsaufbau wird dem OPC-Server mitgeteilt, welche Items der Client benötigt. Hat der OPC-Server neue Daten von der SPS geholt, so sendet er an alle Clients, die an dem jeweiligen Item interessiert sind, eine Nachricht, in der ein Handle übergeben wird. Der Handle ermöglicht dem Client, die Daten aus dem Speicher auszulesen. Der Datentransfer vom Client zum OPC-Server läuft in gleicher Weise: Hat der Client die Items mit neuen Daten überschrieben, so sendet er sie an den Server und weitere Interessenten.

In der Arbeit wurden zwei OPC-Server benutzt. Ein OPC-Server ist für den Roboter und der andere ist für die Steuerung des gesamten Fördersystems zuständig. Im Folgenden werden die Arbeiten erläutert, die zur Bereitstellung der OPC-basierten Zellensteuerung notwendig sind.

5.2.7 Bereitstellung des KUKA OPC-Servers

5.2.7.1 Einstellungen der Kommunikationsschnittstelle in KRC1: Automatik Extern

Ist der Betriebsarten-Wahlschalter am KCP (KUKA Control Panel) auf Automatik-Extern eingestellt, kann ein Leitrechner mit der Robotersteuerung über die Schnittstelle „Automatik Extern“ kommunizieren und verschiedene Roboterprozesse auslösen. Ebenso kann die Robotersteuerung Informationen über Betriebszustände und Störungsmeldungen an den Leitrechner übermitteln.

Bei der Schnittstelle „Automatik Extern“ müssen die physikalischen Ein- und Ausgänge der Robotersteuerung den Signalen zugeordnet werden. Bild 5.13 zeigt die aktuelle Konfiguration. Mit dem Eingang 485 wird beispielsweise ein externes Start-Freigabesignal gesetzt. Sind alle Startbedingungen erfüllt, so kann durch das Signal der Systemvariablen $EXT_START das Programm CELL.SRC gestartet werden.

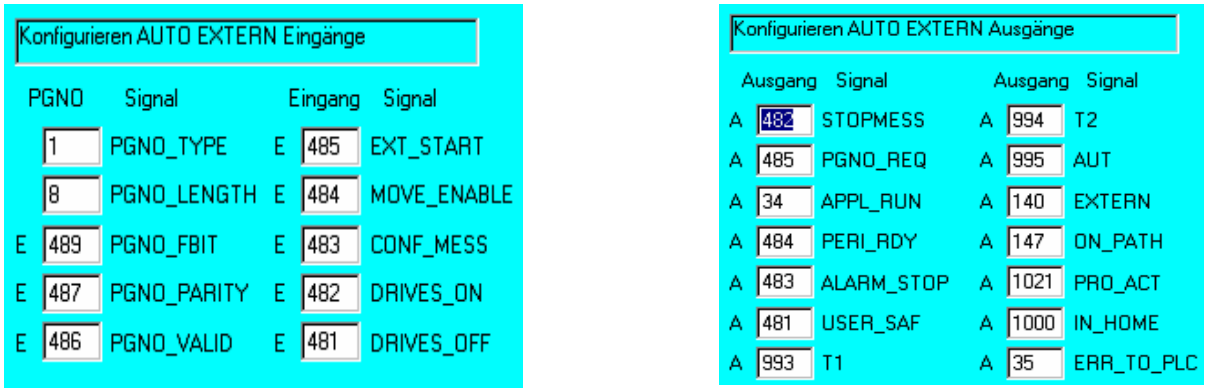


Bild 5.13: I/O-Konfiguration der Schnittstelle „Automatik Extern“ in der KRC1

5.2.7.2 Bereitstellung des Organisationsprogramms CELL.SRC in KRC1

Bei der KRC1 wird die Kommunikation nach außen durch das technologiespezifische Organisationsprogramm CELL.SRC und durch die Funktion des Moduls P00 realisiert. Mit dem Befehl *P00(#EXT_PGNO, #PGNO_GET, DMY[], 0)* wird im Programm CELL.SRC eine Programmnummer beim Leitrechner angefordert. Die anschließend empfangene Programmnummer wird in der Variable PGNO gespeichert. Wenn die Programmnummer korrekt übertragen wurde, versucht das Programm die Nummer einer Prozedur zuzuordnen. Wenn dies gelingt, nimmt die Funktion mit dem Befehl *P00(#EXT_PGNO, #PGNO_ACKN, DMY[], 0)* die Anforderung zurück und teilt dem Leitrechner den korrekten Empfang der Programmnummer mit. Wenn ein Fehler auftritt, z.B. wenn keine Prozedur der Programmnummer zugeordnet ist, wird durch den Befehl *P00(#EXT_PGNO, #PGNO_FAULT, DMY[], 0)* ein Übertragungsfehler im Meldungsfenster angezeigt.

Das einzelne Bearbeitungsprogramm jedes Features und alle Steuerungsprogramme für den Werkzeugwechsel, die Seilsäge, die Frässpindel usw. werden in der KRC1 programmiert und können in CELL.SRC entsprechend aufgerufen werden.

Wie im Abs. 5.2.1 vorgestellt, laufen auf einem Intel Pentium Prozessor in der Steuerung KRC1 gleichzeitig zwei Betriebssysteme, nämlich Windows 95 und das Echtzeitsystem VxWorks, die über das TCP/IP-Protokoll Daten austauschen. Die Peripherie ist durch DeviceNet und serielle Schnittstellen an den Roboter angeschlossen. Um den Zugriff auf die Prozessdaten der Feldebene und als Folge davon die Parameterübertragung von außen direkt an die Robotersteuerung zu ermöglichen, wird ein Konzept verfolgt, das eine Soft-SPS mit OPC-Schnittstelle in die Echtzeitumgebung einfügt.

Das ProConOS (Programmable Controller Operating System) Laufzeitsystem ist eine IEC 61131-3 kompatible Soft-SPS. ProConOS wurde auf der Basis des Echtzeitsystems VxWin/VxWorks implementiert. Es wurde auf der KRC1 installiert und läuft parallel zur Robotersteuerung auf der Echtzeitseite des Steuerungs-PCs. Die Soft-SPS kommuniziert mit der Robotersteuerung über Softwarebibliotheken, mit dem Leit- und Visualisierungssystemen über OPC und mit der I/O-Ebene über alle gängigen Feldbusprotokolle.

Der KUKA OPC Server ermöglicht den externen, netzweiten Zugriff auf System- und Benutzervariablen der Robotersteuerung. Er spezifiziert den Austausch von Positionsdaten, Betriebsmodi, Programmparametern und Steuerungsmeldungen zwischen dem KUKA OPC-Server und dem OPC-Client. Auf einer Seite erhält der OPC Client bei Anforderung vom OPC Server die aktuellen Werte der Variablen, die z.B. zur Visualisierung der laufenden Steuerungsprozesse verwendet werden, auf der anderen Seite schickt der OPC-Client auch über den OPC Server die Variable, die zur Bedienung und Steuerung benötigt werden, an die Soft-SPS (siehe Bild 5.14).

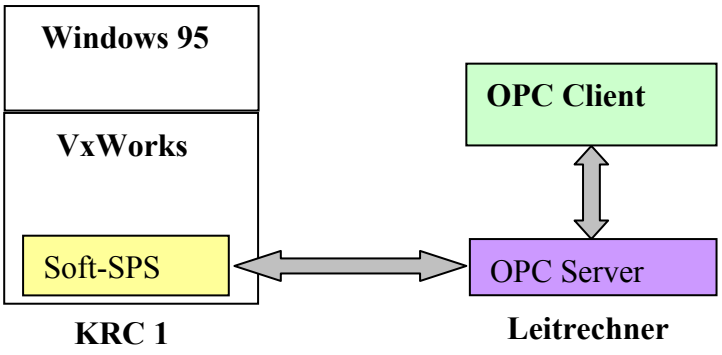


Bild 5.14: Kommunikation zwischen Applikation und Soft-SPS über OPC

Die Soft-SPS Programmierung ist unter der Entwicklungsumgebung MULTIPROG wt entwickelt. Zum Programmieren stehen alle IEC-61131 Programmiersprachen (siehe Anhang 8.2) zur Verfügung: die textuelle Sprache wie strukturierter Text (ST) und Anweisungsliste (AWL), die graphische Sprachen wie der Funktionsplan (FUP), der Kontaktplan (KOP) und die Ablaufsprache (AS).

5.2.7.3 Konfiguration des E/A-Verbindungen zwischen der Soft-SPS und der KRC1

Um dem Soft-SPS Laufzeitsystem ProConOS die Möglichkeit zu bieten, mit der Robotersteuerung KRC1 Daten über nicht physikalisch existierende Ein- und Ausgänge auszutauschen, müssen die E/A-Verbindungen zwischen der Soft-SPS und der KRC1 konfiguriert werden. Hierbei werden in ProConOS projektierte Ausgänge direkt in die Eingänge der KRC1 übertragen. Lesezugriffe auf derart projektierte Eingänge werden auf die Ausgänge der KRC1 umgeleitet.

Die E/A Zuordnung für ProConOS wird in der Multiprog I/O konfiguriert. In dieser Arbeit werden jeweils 24 Byte für ProConOS-Ausgänge und Eingänge benutzt. Der Parameter *DRIVER_PAR1* gibt hierbei an, ab welchem Byte der KRC1-I/Os zugegriffen wird. Die Parameter *VAR_ADR* und *END_VAR_ADR* geben den Beginn und das Ende des E/A-Bereichs in ProConOS an.

```
PROGRAM o_iKRC1 : OUTPUT
(
  VAR_ADR := 0,                    (*Start adress ProConOS output byte 0*)
  END_VAR_ADR := 24                (*End adress ProConOS output byte *)
  DEVICE := DRIVER,
  DRIVER_NAME := 'KRCIODRV',
  DATA_TYPE := BYTE,
  DRIVER_PAR1 := 40,               (*Start address KRC input byte 40($IN[321])*)
  DRIVER_PAR2 := 1,                (*Type of I/O's 0 -> Peripheral I/O's*)
);

PROGRAM i_OKRC1 : INPUT
(
  VAR_ADR := 0,                    (*Start address ProConOS input byte 0*)
  END_VAR_ADR := 24,               (*End address ProConOS input byte *)
  DEVICE := DRIVER,
  DRIVER_NAME := 'KRCIODRV',
  DATA_TYPE := BYTE,
  DRIVER_PAR1 := 40,               (*Start address KRC input byte 40*)
  DRIVER_PAR2 := 1,                (*Type of I/O's 0 → Peripheral I/O's*)
);
```

Zwei Programmorganisationseinheiten (POEs) sind programmiert, nämlich „Automatik“ und „Datenaustausch“. Die POE „Automatik“ stellt den Kanal zum Organisationsprogramm CELL.SRC (vgl. Abs. 5.2.4.2) zur Verfügung. Die POE „Datenaustausch“ schreibt die Werte

an die jeweilige Systemvariable und benutzerdefinierte Variable wie z.B. $SEN_PINT[1…20] und $SEN_PREAL[1…20].

5.2.7.4 POE: „Automatik“

Im Code-Arbeitsblatt unter der POE „Automatik“ wird die graphische Sprache Funktionsplan (FUP) verwendet. Die Soft-SPS bedient mit dieser Funktion die Schnittstelle des Robotersteuerungsmoduls „Automatik Extern“ (vgl. Abs. 5.2.7.1).

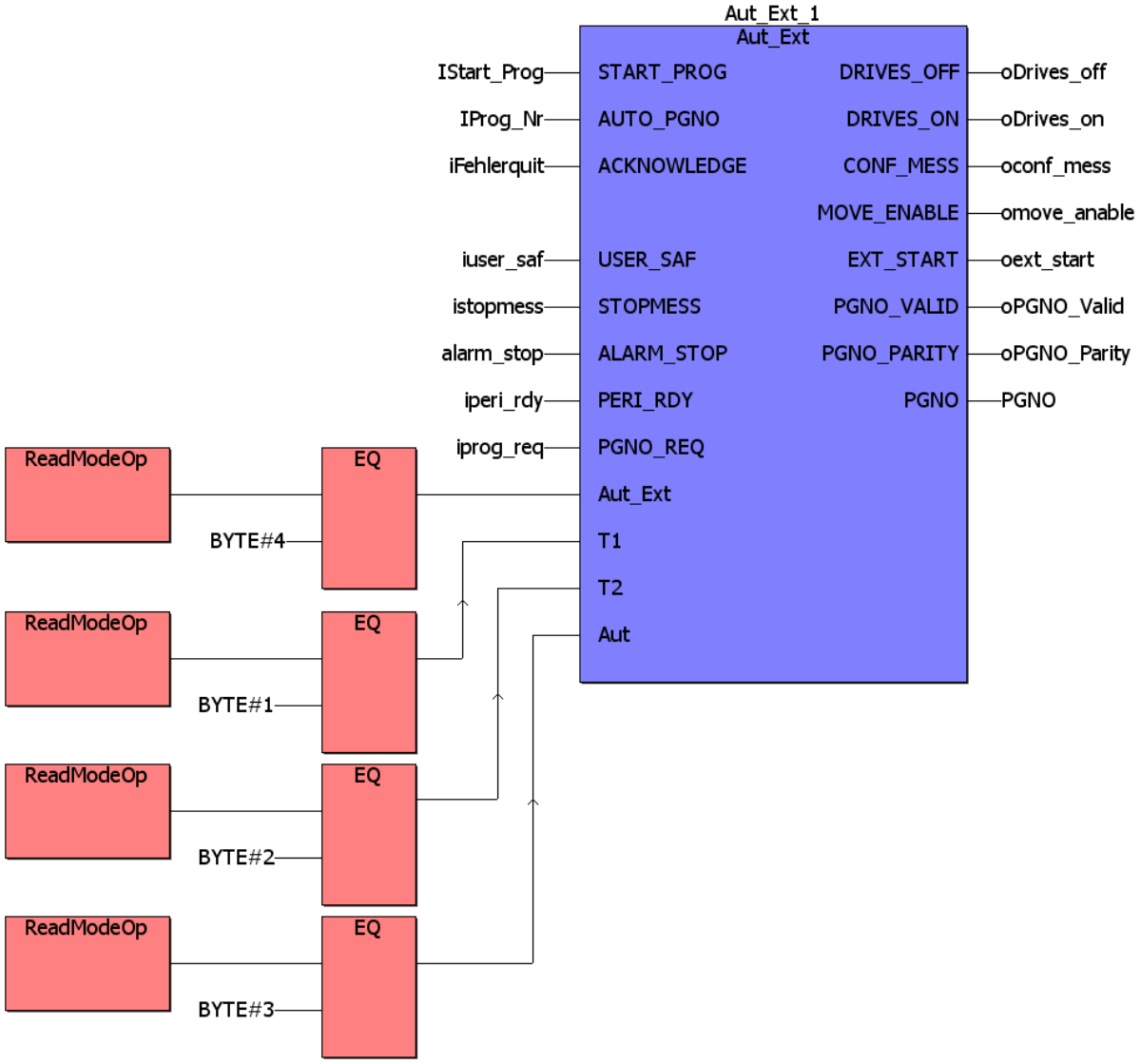


Bild 5.15: Quellcode der POE “Automatik“ in FUP

Am Eingang liest der Eingangsmodul „ReadModeOp“ die jetzige Arbeitsart (Handbetrieb oder „Automatik Extern“) des Roboters. Die POE „Automatik“ übernimmt die komplette Signal-Handhabung für die Übermittlung von Programmnummern über einen Leiterrechner. Sie empfängt die Programmnummer, die über den Leitrechner übermittelt wird, überprüft die Bereitschaft der Peripherie. Mit einer Freigabesignal „IStart_Prog“ wird die übermittelte Pro-

grammnummer an die Robotersteuerung geschickt und der Roboter wird bewegt (vgl. Abs. 5.2.7.1). Die lokalen Variablendeklarationen werden im Anhang 8.8 gezeigt. Hier werden nur die Deklarationen globaler Variablen vorgestellt. Dabei bedeutet *{CSV}*, dass die Variablen in der CSV-Datei gespeichert werden, damit die globalen Resource-Variablen in dem OPC-Server angezeigt und überwacht werden können.

Die für die POE „Automatik“ zuständigen Variablen werden wie folgt deklariert:

```
(*Schnittstelle für Automatik Extern*)
oDrives_off          AT %QX20.0    :BOOL; (* $OUT[481] *)
oDrives_on           AT %QX20.1    :BOOL; (* $OUT[482] *)
oconf_mess           AT %QX20.2    :BOOL; (* $OUT[483] *)
omove_anable         AT %QX20.3    :BOOL; (* $OUT[484] *)
oext_start           AT %QX20.4    :BOOL; (* $OUT[485] *)
oPGNO_Valid          AT %QX20.5    :BOOL; (* $OUT[486] *)
oPGNO_Parity         AT %QX20.6    :BOOL; (* $OUT[487] *)
PGNO                 AT %QB21      :BYTE; (* $OUT[489]~$OUT[496] *)
IProg_NR                           :INT     {CSV};
IStart_Prog                        :BOOL  {CSV};
iFehlerquit                        :BOOL  {CSV};
iuser_saf            AT %IX20.0    :BOOL; (* $IN[481] *)
istopmess            AT %IX20.1    :BOOL; (* $IN[482] *)
alarm_stop           AT %IX20.2    :BOOL; (* $IN[483] *)
iperi_rdy            AT %IX20.3    :BOOL; (* $IN[484] *)
iprog_req            AT %IX20.4    :BOOL; (* $IN[485] *)
```

Dabei definieren “AT %QX20.0…6” und „AT %IX20.0…4“ die physikalische I/O-Adresse in der Soft-SPS. „Q“ steht für Ausgang und „20.0“für den ersten Ausgang im 20ten Ausgangsmodul in der Soft-SPS. Von der Konfiguration der I/Os in der SPS ist bekannt, dass der Parameter DRIVER_PAR1 gleich 40 ist. Die Soft-SPS greift ab dem 40ten Byte der KRC1-I/Os zu. Also entspricht der erste Ausgang des 20ten Ausgangsmoduls in der Soft-SPS dem KRC1-Ausgang $OUT[481]: $(40+20)\times 8+1=481$. Analog dazu steht „I“ für Eingänge und „10..x“ für die Position des Eingangs. Für die entsprechende Zuordnung der I/Os in „Automatik Extern“ in der KRC1 siehe Bild 5.12.

Außerdem werden hier die 40 Variablen vom Datentyp Integer und Real als OPC-Items definiert. Dazu gehören noch einige andere Variablen wie beispielsweise die Variablen für das aktuelle Roboterarbeitsmodul oder für die aktuellen Werte der TCP-Position des Roboters (siehe Anhang 8.8).

5.2.7.5 POE: „Datenaustausch“

In dieser POE wird die KUKA Bibliothek *KrcLib* eingebunden, um Roboterdaten lesen und schreiben zu können. Die Bibliothek enthält folgende Funktionen bzw. Funktionsbausteine.

Funktionsblock	KRC Variable	Bemerkungen
ReadAxisAct	*$AXIS_ACT*	Lesen aktueller Achs-Istwerte der Roboterbahn
ReadBaseAct	*$ACT_BASE*	Lesen aktueller Position des Basisursprungs
ReadOvPro	*$OV_PRO*	Lesen aktuelles Overrides der KRC1
WriteOvPro	*$OV_PRO*	Schreiben aktuelles Overrides der KRC1
ReadPosAct	*$POS_ACT*	Lesen aktueller Positionswerte der Roboterbahn
ReadSenInt	*$SEN_PINT[]*	Integration der Sensorschnittstellen detaillierte Beschreibung siehe unten
WriteSenInt	*$SEN_PINT[]*	
ReadSenReal	*$SEN_PREA[]*	
WriteSenReal	*$SEN_PREA[]*	
ReadToolAct	*$ACT_TOOL*	Lesen aktueller Position des Werkzeugs
ReadModeOp	*$MODE_OP*	Lesen der aktuellen Betriebsart
ReadProState	*$PRO_STATE*	Lesen aktuelles Submit- und Roboterinterpreters
DisplayKCPNotifyMsg		Anzeigen der Quittungsmeldung in GUI
DisplayKCPStatusMsg		Anzeigen der Zustandsmeldungen in GUI
ClearKCPStatusMsg		Quittieren der Zustandsmeldung in GUI
ClearAllKCPStatusMsg		Löschen aller aktiven Zustandsmeldungen

Tabelle 5.12: Verfügbare Funktionsblöcke der KUKA Bibliothek KrcLib

Durch die Methoden *ReadSenInt*, *WriteSenInt*, *ReadSenReal*, *WriteSenReal* in der KrcLib ist es möglich, auf die KRC-Variablen der Datentypen Integer und Real über diese Sensorschnittstelle zuzugreifen. Allerdings ist die Anzahl der Variablen des jeweiligen Datentyps auf 20 beschränkt. Um die benötigen Positions- und Geometriedaten jedes Features von BauXML auf die Robotersteuerung zu übertragen, werden die insgesamt 40 Variablen verwendet. Die POE „Datenaustausch“ wird in strukturiertem Text (ST) programmiert:

```
(*real Variable for objekts*)
WriteSenReal_1(Index:= BYTE#1,Value:=x_wert);
…
WriteSenReal_20(Index:=BYTE#20,Value:=Rvar20);

(*Integer Variable for objekts*)
WriteSenInt_1(Index:=BYTE#1,Value:=DIvar1);
…
WriteSenInt_20(Index:=BYTE#20,Value:=DIvar20);

(*Write/read KRC1 variable $OV_PRO representing the reduction of the current velocity in percent *)
curVelocity:=ReadOvPro();
WriteOvPro_1(OvPro:=VelocityToBe);

(* read the KRC1 variable $MODE_OP containing the current operating mode. *)
Rob_Mode:=ReadModeOp();

(* Read $PPOS_ACT representing the current position actual value of the robot trajectory *)
ReadPosAct_1();
Pos_Valid:=ReadPosAct_1.bValid;
Pos_X:=ReadPosAct_1.X;
Pos_Y:=ReadPosAct_1.Y;
Pos_Z:=ReadPosAct_1.Z;
Pos_A:=ReadPosAct_1.A;
```

```
Pos_B:=ReadPosAct_1.B;
Pos_C:=ReadPosAct_1.C;
```

Die Soft-SPS schickt beispielsweise mit dem Befehl „*WriteSenReal_4*“ den Wert „*Rvar4*“, der z.B. der Durchmesser einer Dose aus BauXML-Daten ist, an die KRC1-Systemvariable *$SEN_PREA[4]*. Der Index „n“ in der Funktion „*WriteSenReal_n*“ oder „*WriteSenInt_n*“ entspricht dem Index „n“ der Systemvariablen *$SEN_PREA[n]* oder *$SEN_PINT[n]*. Zur detaillierten Beschreibung der POE-Einheiten samt den Deklarationen siehe Anhang 8.8.

5.2.7.6 Konfiguration der Kommunikation zur Soft-SPS Steuerung

Die Datenübertragung wird über das Ethernet-Netzwerk durchgeführt. Die notwendigen Einstellungen für die Kommunikation mit der Steuerung sind im Dialog „Resource-Einstellung“ im Programm Multiprog-Arbeitsstation vorzunehmen (Bild 5.16).

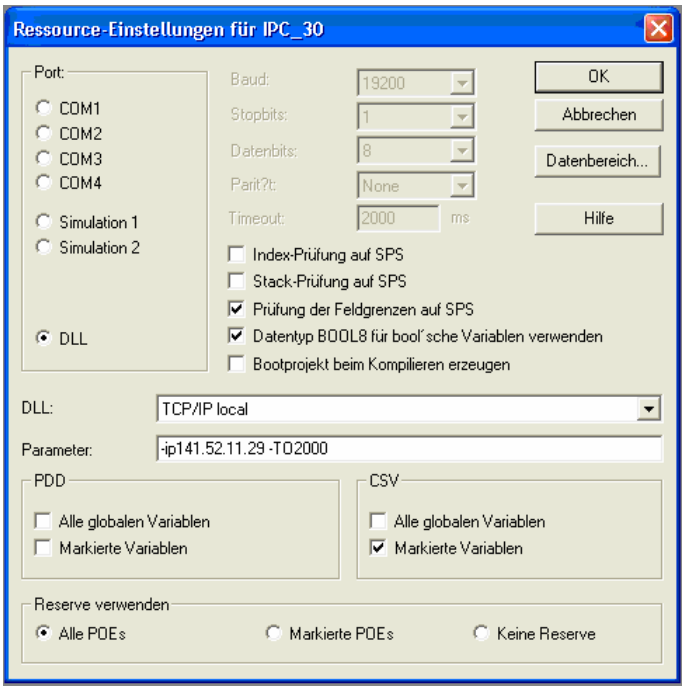


Bild 5.16: Kommunikationsparameter der Soft-SPS

Hierbei sind die „DLL: TCP/IP“ auszuwählen und die Kommunikationsparameter anzugeben. Die IP-Adresse „-ip141.52.11.29“ gehört zur Windows95 Ebene im KRC1-Rechner. „-TO2000“ bedeutet ein „Timeout“ von 2 Sekunden.

Das Steuerungsprojekt wurde im Programm Multiprog gespeichert und im KUKA OPC-Server geladen. Alle im Projekt als CSV-Variablen deklarierten Variablen werden im OPC-Server verwaltet.

Da sich zwei Netzwerkkarten in der KRC1 befinden (vgl. Abs. 5.2.1) und um die Datenpakete, die über die Windowsschnittstelle ankommen, an die Soft-SPS ProConOS weiterzuleiten,

wird ein Router auf dem KRC1 System installiert. Somit ist es möglich, von einem fernen Computer aus auf VxWin über das TCP/IP Protokoll zuzugreifen (Bild 5.17).

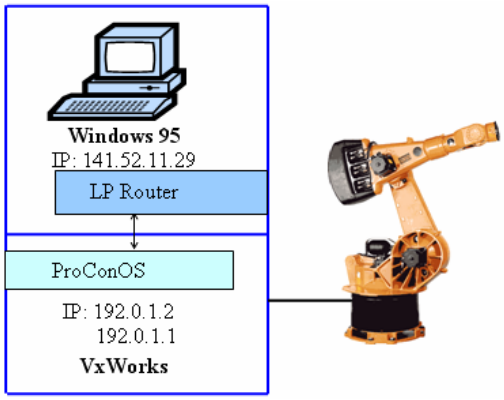


Bild 5.17: Umleitung der TCP/IP-Datenpakete durch einen LP Router

5.2.8 Bereitstellung des OPC-Servers für das Fördersystem

Die Bereitstellung des OPC-Servers für das gesamte Fördersystem erfolgt durch das Programm „Modbus/TCP Konfigurator“ von der Firma WAGO. Konfiguriert werden die physikalische Struktur des Systems (vorhandene Geräte) und die über OPC erreichbaren Prozesswerte. In dieser Arbeit werden zwei Modbus/TCP Controller 750-842 konfiguriert und zwar je ein Controller für das Schienensystem und für die Schiebebahn (Bild 5.18).

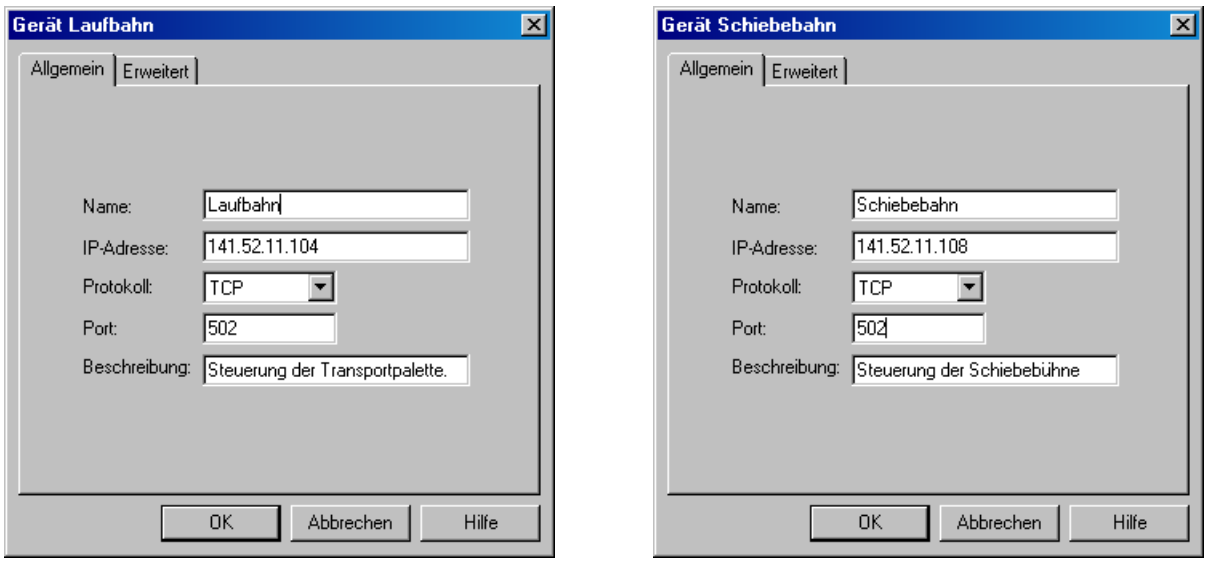


Bild 5.18: Einstellung zweier Ethernet-Controller im OPC-Server

Die Fernsteuerung der Transporteinrichtungen über OPC wurde so konstruiert, dass die Eingangssignale per OPC aus der Ferne auslöst werden, als ob die Tasten auf dem Handbediengerät bedient werden. In der Steuerung sind die Signale der Tasten auf Eingänge gelegt, die durch OPC aber nicht beschrieben werden dürfen. Um Signale an diesen Eingängen zu setzen, werden in der Steuerungsbox noch Ausgangsmodule eingesetzt und mit den Eingangsmodu-

len verkabelt, die ausschließlich für die Tasten zuständig sind. In der übergeordneten Zellensteuerung werden diese „neuen“ Ausgänge als die Tasten auf dem Handbediengerät betrachtet. In beiden Steuerungen sind jeweils drei WAGO 750-506 Digital-Eingangs-Module (Anhang 8.6) eingesetzt. Die über OPC erreichbaren Prozesswerte beider SPS-Steuerungen werden in eine Werte-Liste dargestellt. Dargestellt im WAGO OPC-Server sind die Eingänge mit Sensoren zur Überwachung und die „neuen“ Ausgänge zur Steuerung. Bild 5.19 und 5.20 zeigen jeweils die dem im Geräte-Baum selektiertem Gerät zugeordneten Werte.

Name	Adresse	Datentyp	Beschreibung
S_Langsam	100004	Bit	Sensor:Palette kurz vor Stopper
S_Motorventil	100006	Bit	Sensor: Motor andrueckt auf Palette.
S_NoStop	100001	Bit	Sensor:Stopper ist zurueck.
S_Stein	100003	Bit	Kap. Sensor:Stein an Stopper.
S_Stop	100002	Bit	Sensor:Stopper ist vor.
T_M1LV	000019	Bit	Taste: M1 langsam vor
T_M1LZ	000020	Bit	Taste: M1 langsam zurueck
T_M1SV	000015	Bit	Taste: M1 schnell vor
T_M1SZ	000016	Bit	Taste: M1 schnell zurueck
T_Motorventil	000023	Bit	Taste: Motorventil aktivieren
T_NoStop	000012	Bit	Taste: Stopper zurueck
T_Stop	000011	Bit	Taste: Stopper vor

Bild 5.19: Bereitstellung des OPC-Servers der Laufbahn

Name	Adresse	Datentyp	Beschreibung
S_Bahnankoppelt	100001	Bit	Sensor: Ankopplungszylinder Oben->verriegelt
S_Bahnentkoppelt	100002	Bit	Sensor: Ankopplungszylinder Unten->entriegelt
S_beiRob	100006	Bit	Sensor:Schiebebühne ist bei Roboter.
S_beiWende	100007	Bit	Sensor: Schiebebühne ist bei Wendeeinrichtung.
S_PalAufSchi	100008	Bit	Sensor: Palette ist auf der Schiebebühne in Position.
S_Palettenentriegelt	100003	Bit	Sensor: Palettenentriegelt
S_Palettenverriegelt	100004	Bit	Sensor: Palettenverriegelt
S_tranMotorPress	100005	Bit	Sensor: Transportmotor ist anppresst!
T_BahnAnkopp	000009	Bit	Schalter: Bahnankopplung aktivieren/deaktivieren .
T_BahnVor	000013	Bit	Taster: Antriebsmotor VOR(weg vom Roboter).
T_BahnZur	000014	Bit	Taster: Antriebsmotor zurück.(zum Roboter)
T_PalVer	000010	Bit	Schalter auf Panel: Palette verriegen/entriegeln.
T_PalVor	000017	Bit	Taster: Transportmotor(Palette) langsam VOR(weg vom Roboter).
T_PalZur	000018	Bit	Taster: Transportmotor(Palette) langsam zurück(zum Roboter)

Bild 5.20: Bereitstellung des OPC-Servers der Schiebebühne

In der Werte-Liste sind Wertenamen, Adressen, Datentypen und Beschreibungen dargestellt. Die Adresse wurde als 6-stellige Dezimalzahl eingegeben. Die erste Stelle der Adresse wird von WAGO festgelegt („0“ für Output und „1“ für Input), die weiteren vier bzw. fünf Ziffern sind die mit Eins beginnende Adresse des Werts. Der Datentyp ist immer „Bit“.

5.3 Experimentelle Festlegung der Fertigungsparameter

In der Arbeit wird Porenbeton dank seiner guten Eigenschaften wie der guten Wärmedämmung, der leichten Bearbeitbarkeit aber dennoch hohen Materialfestigkeit benutzt. Bisher liegen jedoch keine nutzbaren Erfahrungen über die Fertigungsparameter für das Sägen und das Fräsen von Porenbeton vor. Es ist für die Fertigung unerlässlich, vernünftige Werte für die Vorschubgeschwindigkeit, die Schnitttiefe und die Schnittgeschwindigkeit bzw. die Fräserdrehzahl vorzugeben. Wenn z.B. der Fräser mit großer Drehzahl aber kleiner Vorschubgeschwindigkeit und Schnitttiefe arbeitet, dann werden Energie und Zeit verschwendet. Im anderen Fall muss jeder Zahn tief ins Werkstück eindringen und einen großen Span lösen. Das verursacht steigenden Widerstand im Stein. Als Konsequenz wird entweder die Spindel wegen Überlastung stoppen oder der Fräser wird abbrechen.

Im Abs. 5.2.4 wurde vorgestellt, dass das Sägeseil der Säge nicht automatisch ausgetauscht wird und die Drehzahl deshalb als Festsollwert vorgegeben wird. Um eine gute Qualität der Schnittfläche zu erreichen, sind die Drehzahl 4500 1/min und die optimale Vorschubgeschwindigkeit 0.5 m/s zu empfehlen. Eine zu hohe Vorschubgeschwindigkeit verursacht wegen der Elastizität des Sägeseils eine unebene schlechte Schnittfläche oder sogar ein Versagen des Seils.

In der prototypischen Roboterzelle werden insgesamt neun Fräser mit unterschiedlichen Durchmessern zur Verfügung gestellt. Sie sind im Fräserkasten mit festen Platznummern positioniert und durch Sensoren überwacht. Die Reihenfolge der Fräser darf nicht vertauscht werden, sonst ist eine richtige Kennung des herausgenommenen Fräsers nicht garantiert. Tabelle 5.13 zeigt die jetzige Festlegung der Fräser und die entsprechende Adressierung in der KRC1 (vgl. Bild 5.8 (l.) und Tabelle 5.3).

Nr. 1	Nr. 2	Nr. 3	Nr. 4	Nr. 5	Nr. 6	Nr. 7	Nr. 8	Nr. 9
$IN(321)	$IN(322)	$IN(323)	$IN(324)	$IN(325)	$IN(326)	$IN(327)	$IN(328)	$IN(329)
								

Tabelle 5.13: Platzfestlegung der verschiedenen Fräser

Nach dem Fräserwechsel müssen die Drehzahl, die Vorschubgeschwindigkeit und die Schnitttiefe von der Robotersteuerung verstellt werden. Als Frässpindel wurde eine Elektrospindel der Firma ELEKTROMECCANICA verwendet. Bild 5.21 zeigt die Leistungskennlinie der Spindel.

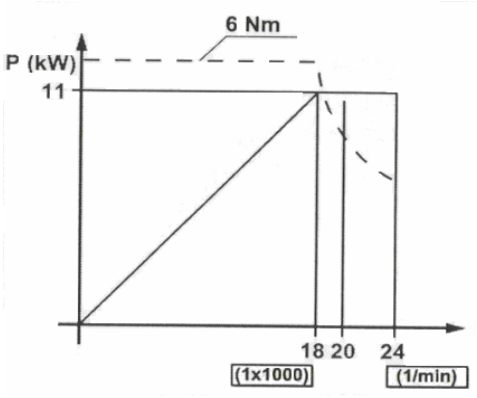


Bild 5.21: Leistungskennlinie der Frässpindel

Die maximale Drehzahl beträgt 24,000 U/min, die maximale abgegebene Leistung ist 11 kW. Die abgegebene Leistung wächst mit steigender Drehzahl linear, bevor sie die Grenzleistung (11kW) erreicht. Bis zu Drehzahl von 18000 U/min bleibt das Motor-Drehmoment konstant bei 6 Nm. Danach fällt es stark ab.

Insgesamt wurden drei oft benutzte Fräser mit einem Durchmesser von 12mm, 20mm und 63mm (Nr. 7, Nr. 8 und Nr. 1 in Tabelle 5.13) in der Arbeit getestet. Hier werden die Ergebnisse der Versuche zusammen dargestellt.

5.3.1 Drehzahl

Die Drehzahl ist entweder durch eine bauartbedingte Drehzahlgrenze oder die zulässige Drehzahl wegen der Wuchtgüte begrenzt. In den Versuchen wurden folgende Grenzen eingehalten.

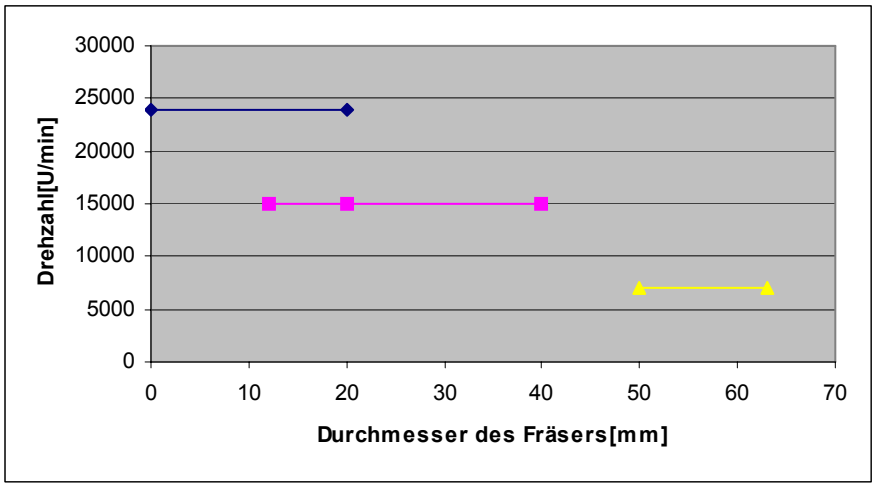


Bild 5.22: Drehzahlgrenze

Die obere Linie in Bild 5.22 ist die maximale Spindeldrehzahl. Die Bauart des Fräsers begrenzt die maximale Drehzahl des jeweiligen Fräsers. Je kleiner der Durchmesser des Fräsers ist, umso größer ist die Grenzdrehzahl mit der ein Fräser drehen kann. Die Drehzahl der Fräser, deren Durchmesser kleiner als 40mm ist, ist maximal auf 15,000 U/min beschränkt (Mittlere Linie in Bild 5.22), weil die Werkzeuge nicht ausgewuchtet sind. Die Fräser mit einem Durchmesser größer als 50 mm arbeiten wegen der Bauart am besten nicht über 7,000 U/min (untere Linie in Bild 5.22). Es sollte immer die zulässige Maximaldrehzahl gewählt werden.

5.3.2 Vorschubgeschwindigkeit und Schnitttiefe

Die beiden Parameter, Vorschubgeschwindigkeit und Schnitttiefe, beeinflussen zusammen die maximale Leistung. Sie begrenzen sich auch gegenseitig, d.h. wenn mit großer Vorschubgeschwindigkeit gefräst wird, dann darf die Schnitttiefe nicht zu groß gewählt werden. Umgekehrt gilt, dass mit großer Schnitttiefe nur langsam gefräst werden kann.

Der Grenzwert der Schnitttiefe ist die Schneidenlänge des jeweiligen Fräsers. Die maximal zulässige Schnitttiefe ist also abhängig von der Bauart des Fräsers. Grundsätzlich sollte mit möglichst großer Vorschubgeschwindigkeit und großer Schnitttiefe gefräst werden, weil mit größerer Vorschubgeschwindigkeit und großer Schnitttiefe die Steine schneller und günstiger bearbeitet werden. Mit steigender Schnitttiefe wird jedoch die Fräsqualität schlechter.

Unter Berücksichtigung der Bearbeitungsqualität werden bei der Auswahl der beiden Fertigungsparameter zwei Fräsarten unterschieden: gerades Fräsen und kreisförmiges Fräsen.

Gerades Fräsen:

Die maximale Geschwindigkeit, die in den Versuchen genau eingestellt werden kann, ist 1,25 m/s, d.h. unter 1,25 m/s kann der Roboter die eingestellte Geschwindigkeit genau erreichen. Obwohl der Roboter noch schneller fahren kann, kann er mit der eingestellten Geschwindigkeit nicht genau übereinstimmen (Beschleunigungs- und Bremsphasen). Unter Berücksichtigung der Qualität und mit dem Kompromiss bezüglich der Schnitttiefe ist als Vorschubgeschwindigkeit für gerades Fräsen 1 m/s zu empfehlen. Die maximal zulässigen Schnitttiefen der in den Versuchen verwendeten Fräser sind in Tabelle 5. 14 aufgelistet.

Durchmesser [mm]	Maximal zulässige Schnitttiefe [mm]
12	40
20	40
63	10

Tabelle 5.14: Maximale zulässige Schnitttiefen beim geraden Fräsen

Kreisförmiges Fräsen

Aus Bild 5.23 geht klar hervor, dass die Dose schon bei über 0,1m/s Vorschubgeschwindigkeit Verformungen hat. Je schneller die Vorschubgeschwindigkeit ist, umso größer ist die Verformung. Beim kreisförmigen Fräsen ist zu empfehlen, die Vorschubgeschwindigkeit 0,1m/s einzustellen.

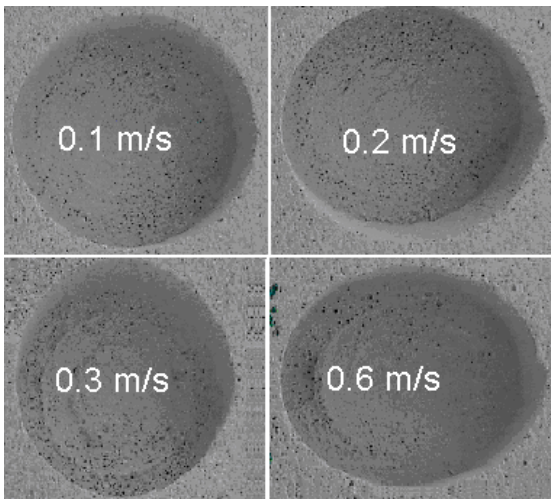


Bild 5.23: Dosenverformung bei unterschiedlichen Vorschubgeschwindigkeiten

Die maximal zulässige Schnitttiefe für die Fräser mit Durchmesser 12 mm und 20 mm ist in Tabelle 5.15 zu finden. Auf Grund der niedrigen Vorschubgeschwindigkeit kann hier der Fräser mit Durchmesser 20mm bis zu 65mm eintauchen. Der Fräser mit Durchmesser 63 mm ist wegen seiner Bauart nicht geeignet für das Dosenfräsen.

Durchmesser [mm]	Maximal zulässige Schnitttiefe [mm]
12	50
20	65

Tabelle 5.15: Maximal zulässige Schnitttiefen beim kreisförmigen Fräsen

5.4 Roboterprogrammierung

Je nach Programmierverfahren lässt sich die Roboterprogrammierung in Online- und Offline-Programmierung unterteilen. Die „Online-Programmierung“ ist die direkte Programmierung, bei der die Programmierung direkt am Einsatzort des Roboters in der Produktionsumgebung durchgeführt wird. Mithilfe des KCP (KUKA Control Panel) wird der Roboter TCP (Tool Center Point) manuell zu Zielstellungen, oder entlang der Bewegungsbahn geführt, die er beim Arbeitsvorgang abfahren soll und die Ist-Positonswerte abgespeichert (sog. Teach-In). Die Arbeiten zum Ablegen/Aufnehmen der Frässpindel und der Seilsäge sind beispielsweise per Teach-In programmiert.

Unter “Off-line-Programmierung” wird eine textuelle Programmierung in einer problemorientierten Programmiersprache in einem Editor verstanden. Dabei werden die Koordinaten von Positionen textuell durch Eingabe von Zahlenwerten spezifiziert. Bild 5.24 zeigt den Programmiereditor von KRL (KUKA Robot Language).

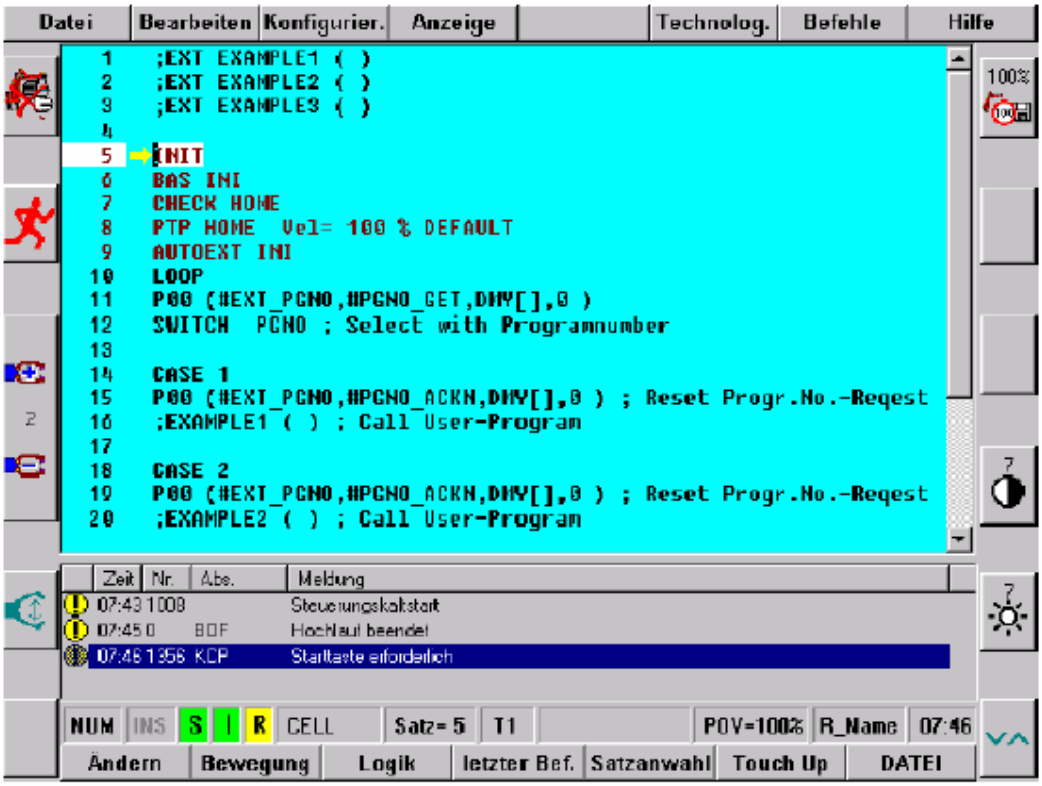


Bild 5.24: Programmierumgebung von KRL

Die Erstellung eines Programms erfolgt durch textuelle Eingabe von Befehlen mit den entsprechenden Parametern. Im Wesentlichen stellt die KRL folgende Befehlsarten zur Verfügung (Tabelle 5.16):

Befehlsarten	Bemerkungen
Bewegungsanweisung	PTP; LIN; CIRC
Ablaufanweisung	WAIT; WAIT FOR; IF...THEN...ELSE...ENDIF; SWITCH...CASE...ENDSWTICH
Peripheriesteuerung	OUT:OUT; OUT:PULSE; HALT
Wiederholstrukturen (Schleife)	FOR...ENDFOR; WHILE...ENDWHILE; REPEAT…UNTIL; LOOP…ENDLOOP
Ein-/Ausgabeanweisung	$IN[*Nr*], $OUT[*Nr*], $ANIN[*Nr*], $ANOUT[*Nr*]

Tabelle 5.16: Befehlsarten der Roboterprogrammierung

Zu den wichtigsten Aufgaben einer Robotersteuerung gehört die Bewegung des Roboters. Der Programmierer steuert dabei die Bewegungen des Industrieroboters mit Hilfe von speziellen Bewegungsanweisungen. Den **PTP** und **LIN** Bewegungsarten ist gemeinsam, dass die Pro-

grammierung von der aktuellen Position in eine neue Position erfolgt. Deshalb ist in einer Bewegungsanweisung im Allgemeinen nur die Angabe der Zielposition notwendig.

Als Bezugspunkt (sog. “XP1“ Punkt in allen KRL Programmen) wurde der Punkt definiert, der 100 mm über der Ecke des Steins neben dem Anschlag liegt. Dieser Punkt wird per „Teach-In“ im Programm definiert und gespeichert. Die Bezugskoordinaten werden in Bild 5.25 verdeutlicht.

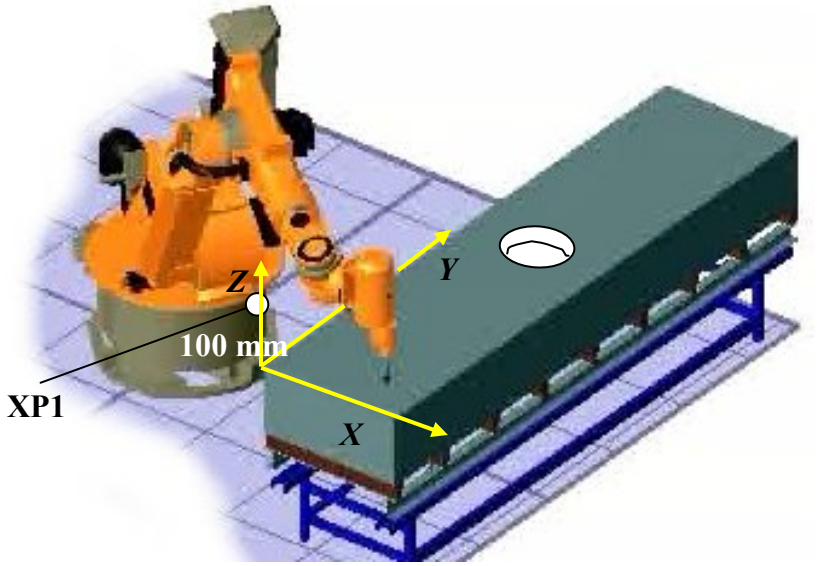


Bild 5.25: Roboterprogrammierung

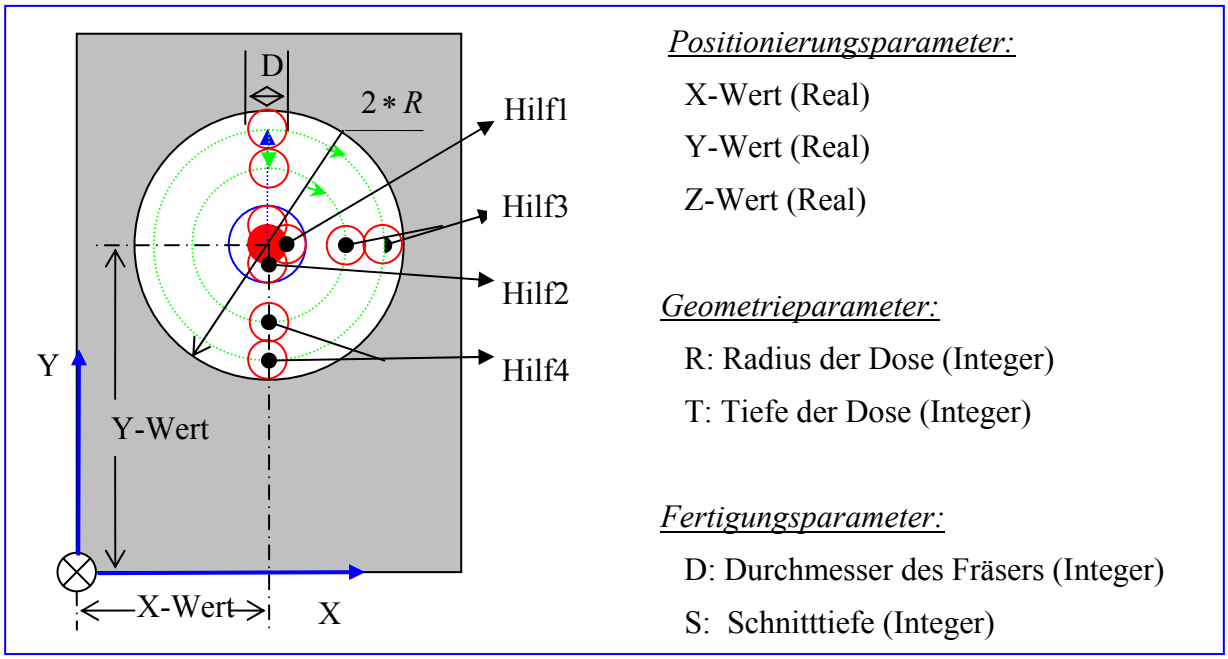


Bild 5.26: Dosenfräsen

Die für eine Dose nötigen Parameter sind die X-, Y- und Z-Werte zum Bezugspunkt (dafür ist eine lokale Koordinatenkonvertierung des Steinkoordinatensystems in BauXML notwendig), Radius und Tiefe der Dose sowie der Durchmesser und die zulässige Schnitttiefe von dem benutzten Fräser.

Der Fräser geht zuerst zum Mittelpunkt, der durch den X-Wert und den Y-Wert festgelegt wurde, und fährt senkrecht in den Stein. Bei Dosen, deren Tiefe größer als die zulässige Schnitttiefe ist, wird der Mäander in der Ebene von einer Zustellschleife in die Tiefe überlagert. Während der Bearbeitung wird geprüft, ob die vorgegebene Schrittweite der Schnitttiefe größer als die gewünschte Tiefe der Dose ist. Falls ja, geht der Fräser einfach einmal auf die gewünschte Tiefe und läuft die **Ebenenschleife** (siehe Bild 5.27) auf dieser Tiefe ab. Falls nein, dann geht der Fräser durch die **Tiefenschleife** (siehe Bild 5.28). In jeder Tiefenschleife läuft der Fräser wieder durch die Ebenenschleife.

Schrittweite in Y-Richtung: D

Schleifenzahl einer Ebenenschleife: $m = \frac{R}{D}$

Schleifenzahl der Tiefenschleife: $n = \frac{T}{S}$

In der Ebenenschleife fährt der Fräser vom Dosen-Mittelpunkt aus entlang der Y-Achse um $\frac{D}{2}$ und fräst anschließend einen Kreis mit Radius D. Als Hilfspunkte werden hier Hilf1 und Hilf2 (siehe Bild 5.26) benutzt. Dann fährt er weiter entlang der Y-Achse um $(R - D)$ mm und fräst anschließend mit einem Kreis mit Radius R die Außenkontur der Dose. Als Hilfspunkte werden hier Hilf3 und Hilf4 (siehe Bild 5.26) benutzt. Danach läuft das Programm Schritt für Schritt durch die Ebenenschleife bis zum Mittelpunkt.

In der Robotersteuerung werden die Fertigungsprogramme (Makros) aller Fertigungsfeatures offline programmiert. In dem technologiespezifischen Organisationsprogramm CELL.SRC werden sie aufgerufen und ausgeführt. Aus Platzgründen werden die Programme nicht hier gezeigt sondern nur anhand des aufgeführten Beispiels einer Dose prinzipiell erläutert.

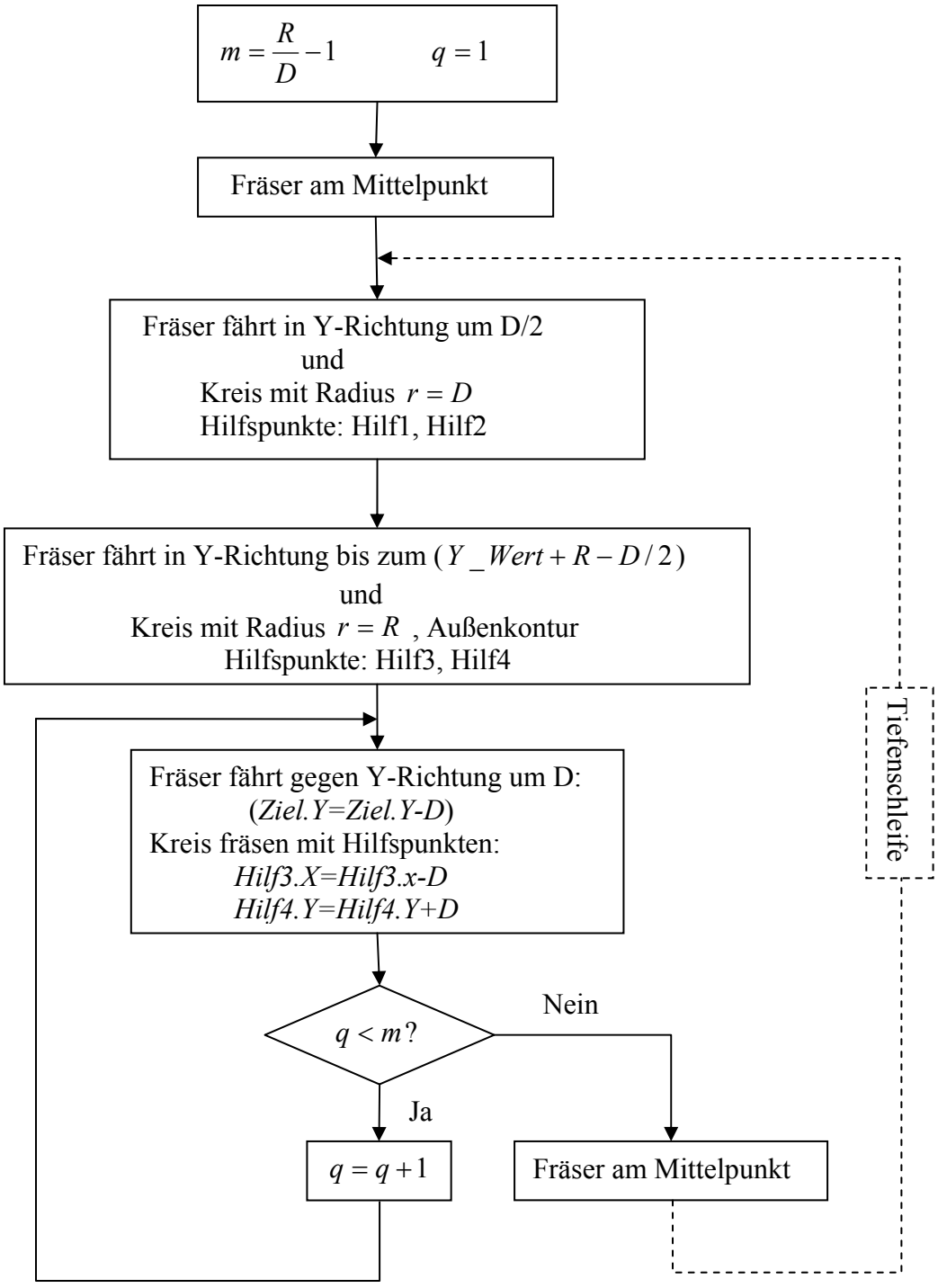
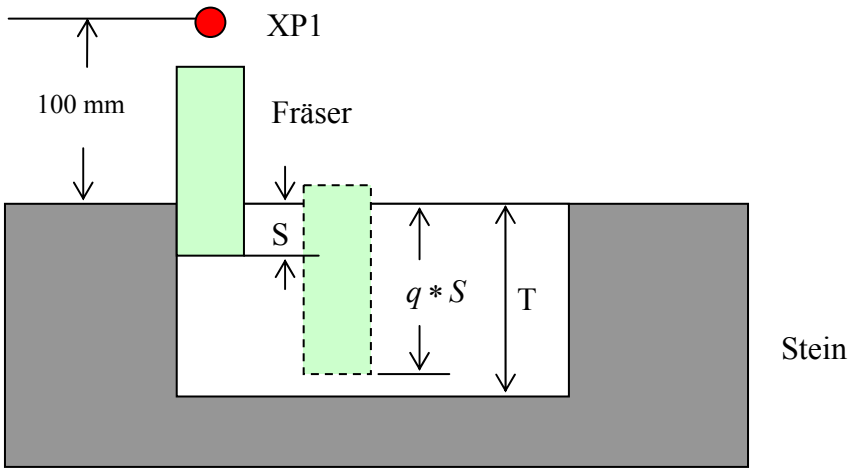


Bild 5.27: Ebenenschleife bei der Dosen-Programmierung



POS.Z: *aktuelle Z-Koordinate des Fräsers*
XP1.Z: *Z-Koordinate von XP1 (100mm über dem Stein)*

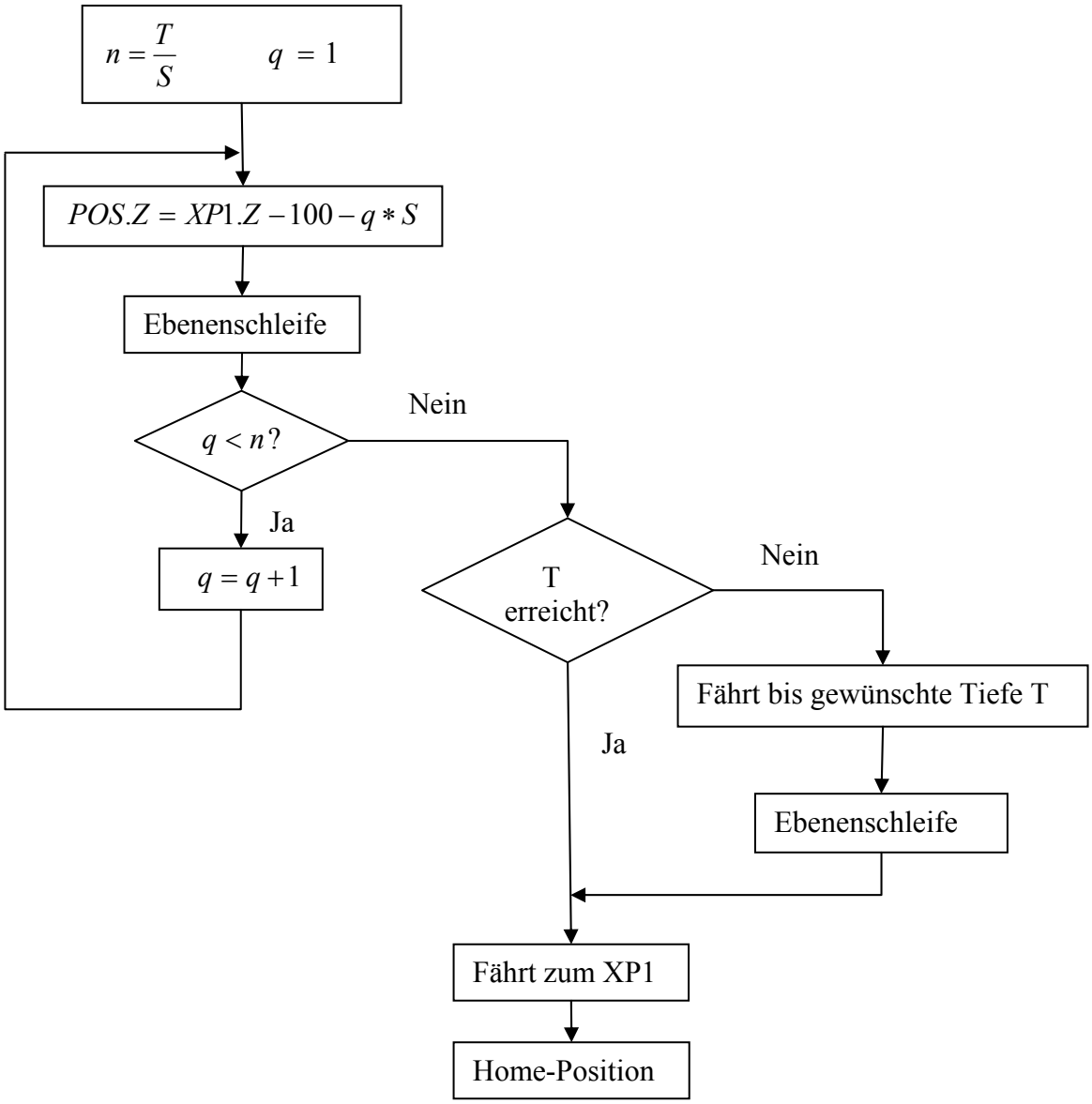


Bild 5.28: Algorithmus der Tiefenschleife

5.5 Softwaretechnische Umsetzung des Prototyps der Zellensteuerung

5.5.1 Projektübersicht

Als übergeordnete Zellensteuerung wird ein Programm „WallProducer“ in der Arbeit neu entwickelt. Im Programm wird jedes Mal eine BauXML-Datei eingelesen, die zuvor im Programm „IfcWallModificator“ generiert und gespeichert wurde. Mit einer 3D-Darstellung wird das gesamte BauXML-Modell anschaulich erkennbar. „WallProducer“ ermöglicht interaktive Aktionen, um das BauXML-Modell nachträglich zu manipulieren. Im „WallProducer“ können z.B. weitere Features wie beispielsweise Dübellöcher im Modell hinzugefügt werden.

Darüber hinaus ist das Programm „WallProducer“ auch ein OPC-Client, der mit den beiden zuvor konfigurierten zwei OPC-Servern verbunden wird. Über die OPC-Schnittstelle arbeitet das Programm „WallProducer“ als Prozessleit- und Überwachungssystem in der Vorfertigungswerkstatt. Es steuert die gesamte Fertigungsprozedur und überwacht die Zustände der Peripherie der Roboterzelle und des gesamten Fördersystems. Die Steine im BauXML-Modell werden sequenziell vom Programm bearbeitet. Dabei liest das Programm die Geometriedaten jedes Steins und schickt die Daten über Ethernet via OPC an die Robotersteuerung. Dort wird je nach Typ des Features das entsprechende Bearbeitungsprogramm aufgerufen.

Das Programm „WallProducer“ wurde auf Basis der MFC (Microsoft Foundation Classes) unter der Entwicklungsumgebung Microsoft Visual Studio 6 implementiert. Zur 3D-Visualisierung und Darstellung kommt die OpenGL-Technik zum Einsatz. Zum Einlesen und zur Verarbeitung der BauXML-Daten werden die gleichen Klassen wie im Programm „IfcWallModificator“ benutzt. Zur Benutzung der OPC-Technologie wurde das OPC-Client Entwicklungstoolkit der Firma WinTech [WINT] benutzt. Das OPC-Client Entwicklungstoolkit liefert ein einfaches API (Application Programming Interface) für die Integration einer kundenspezifischen Anwendung um Daten von jedem OPC-Server zu verwenden. Dabei werden alle Details von DCOM und von OPC durch die DLL (Dynamic Link Library) erledigt, die den Programmierer einfach auf die Daten vom OPC-Server zugreifen lässt, ohne die zugrunde liegenden Schnittstellen selbst anzufassen zu müssen. Unterstützt sind Standard OPC 1.0 und OPC 2.0. Zusammen mit der DLL WTclient.dll werden eine Bibliothekdatei WTclient.lib und zwei dafür notwendige Head-Dateien bereitgestellt. In dem Programm „WallProducer“ wurden sie in das Projekt eingeführt und mit dem Projekt verbunden.

Zur vereinfachten Datenhaltung und Benutzung wurden drei C++-Klassen für Server, Group und Item selbst implementiert. In der Klasse *cOpcServer* werden ein oder mehrere OPC-

Server verbunden bzw. getrennt. Der Zustand und der Name des OPC-Servers können immer nachgefragt werden. In der Klasse *cOpcGroup* werden bestimmte ausgewählte OPC-Items zusammengefasst. Die Klasse *cOpcItem* stellt alle relevanten OPC-Funktionen zum OPC-Item zur Verfügung, z.B. Lesen (Wert, Name, Zugriffsrecht, Qualität und Zeitstempel) und Schreiben des Wertes. Die Klassen werden anhand des UML-Diagramms (Bild 5.45) deutlich.

Bild 5.29: UML-Diagramm der Klassen von OPC

Die Steuerungen der Laufbahn und der Schiebebühne werden in zwei C++-Klassen realisiert: *CLaufbahn* und *CSchiebebahn*. Dabei kann das gesamte Fördersystem ferngesteuert werden, ohne auf die Tasten auf dem Handbediengerät zuzugreifen. Zusammen mit den beiden Klassen werden die Überwachungsfunktionen in der Klasse *CBahnDlg* realisiert (Bild 5.30).

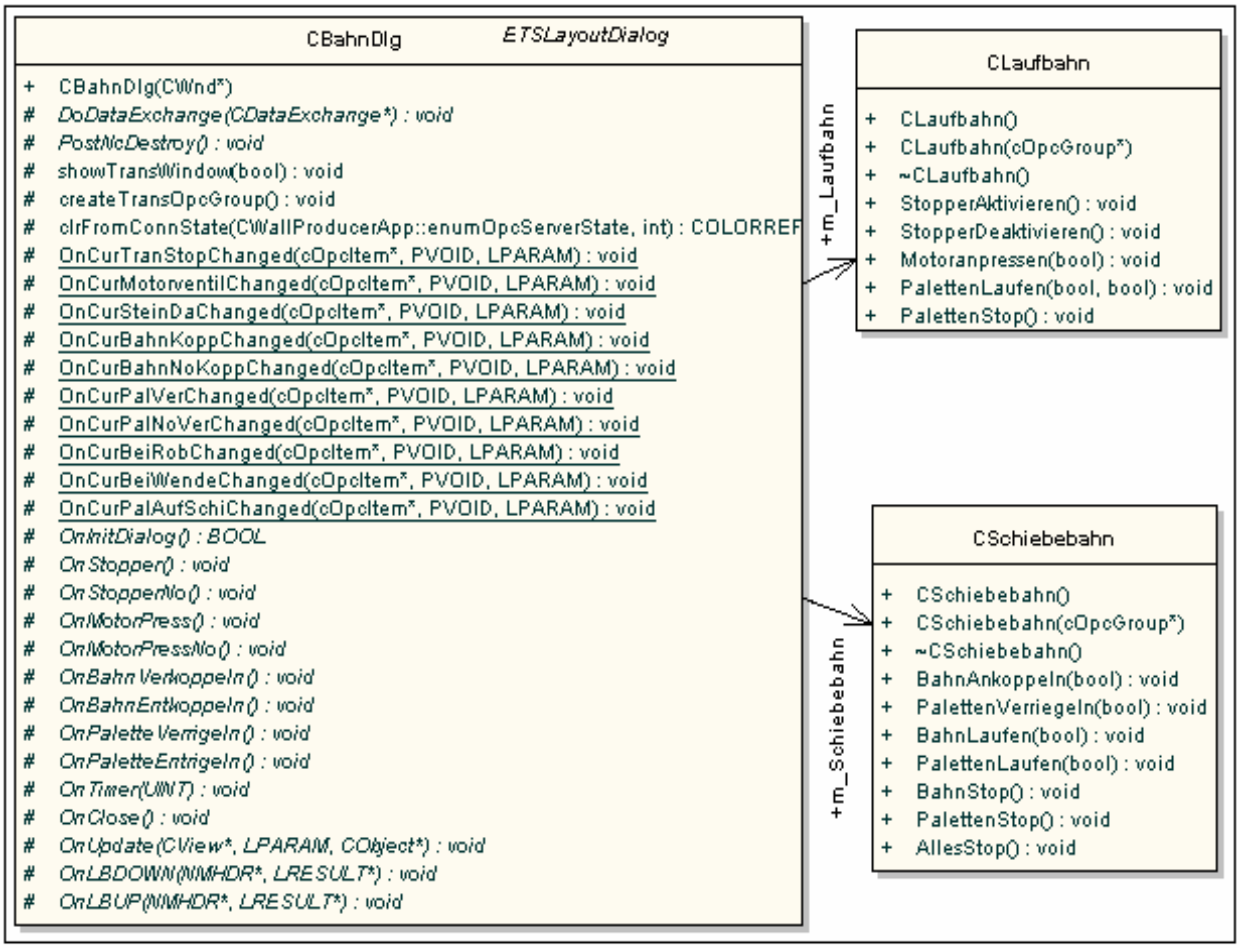


Bild 5.30: UML-Diagramm der Klassen des Fördersystems

Analog ist es auch so beim Roboter sowie seiner Peripherie. Die Steuerungen der Säge und der Frässpindel, z.B. Aufnehmen/Absetzen, Ein-/Ausschalten, werden durch die Klassen *CSaege* und *CSpindel* realisiert. In der Klasse *CRobotDlg* werden die gesamte Steuerung und Überwachung zusammengefasst (Bild 5.31).

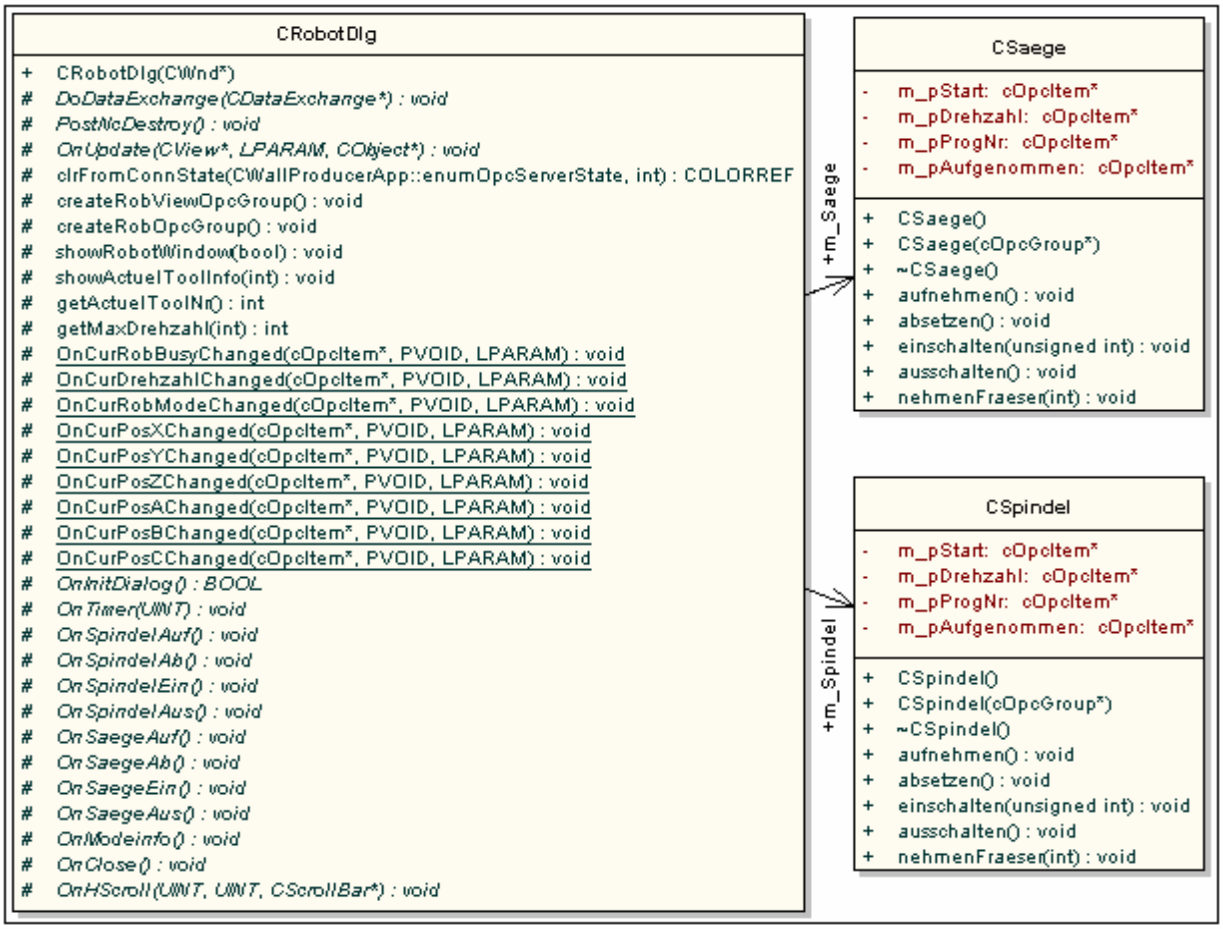


Bild 5.31: UML-Diagramm der Klassen des Roboters

5.5.2 Zellensteuerungstool – WallProducer

Die Benutzeroberfläche des Programms „WallProducer“ gliedert sich zu Beginn in drei Bereiche (siehe Bild 5.32). Im Hauptfenster ist eine 3D-Darstellung des gesamten BauXML-Modells dargestellt. Auf dem linken Teil befindet sich die Darstellung der Projektstruktur des eingelesenen BauXML-Modells. Ausgehend vom Gebäude bis hin zum einzelnen Feature können die darin enthaltenen Elemente angezeigt werden. Durch Klicken der linken Maustaste an der Position der Wand oder des Steins wird diese Wand (Zusammenfassung aller ihrer Steine) oder der Stein in der 3D-Darstellung rot gekennzeichnet. Die administrativen Informationen können optional durch Aktivierung des Menüpunkts „Ansicht→Auftragsdetails“ im Fenster gezeigt werden. Alle anderen Attribute des Objekts wie die OID, der Name oder das Placement können durch Klick der rechten Mausetaste gefragt werden. Unten in der Benutzeroberfläche befindet sich ein Meldungsfenster, in dem alle wichtigen Aktionen protokolliert werden.

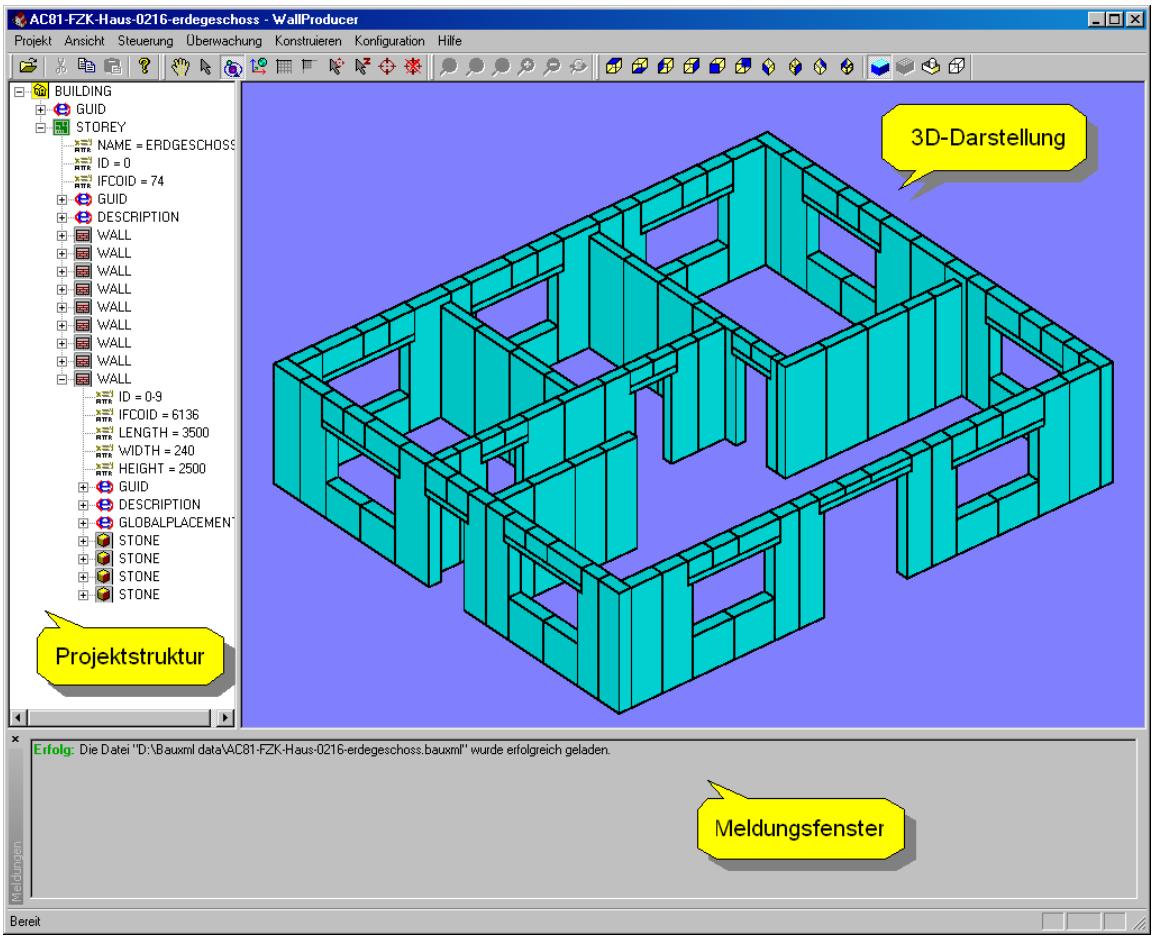


Bild 5.32: Benutzeroberfläche der Applikation WallProducer

Das Kontextmenü in der Baumstruktur (linkes Fenster) variiert je nach Objekttyp. Durch Klick auf „Fertigen“ im Kontextmenü kann entschieden werden, ob die Steine des gesamten Gebäudes oder nur ein Stockwerk, eine Wand oder ein einzelner Stein zu fertigen sind. Ein neues Feature kann zu einem Stein hinzugefügt werden, indem „Neues Feature hinzufügen…“ im Kontextmenu des Steinelements geklickt wird. Dann kommt das Dialogfenster, in dem ein bestimmtes Feature ausgewählt werden kann (Bild 5.33).

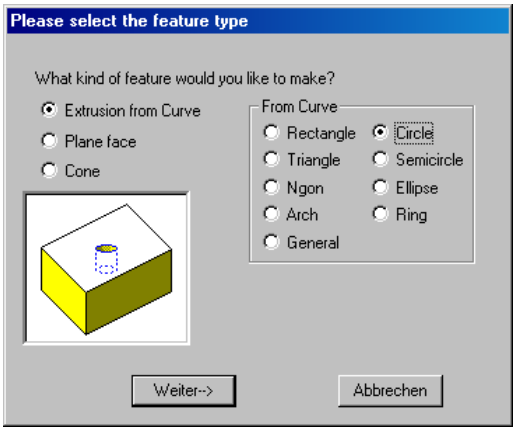


Bild 5.33: Dialogfenster zur Definition eines neuen Features

Durch den Klick „Weiter→“ wird im Eingabedialog (Bild 5.34)gelandet, wo die Geometrie und die Lage des neuen Features zum Stein definiert werden. Die originale Steingeometrie ist oben in einer 3D-Darstellung gezeigt. Die interaktive Darstellung ist sehr anschaulich.

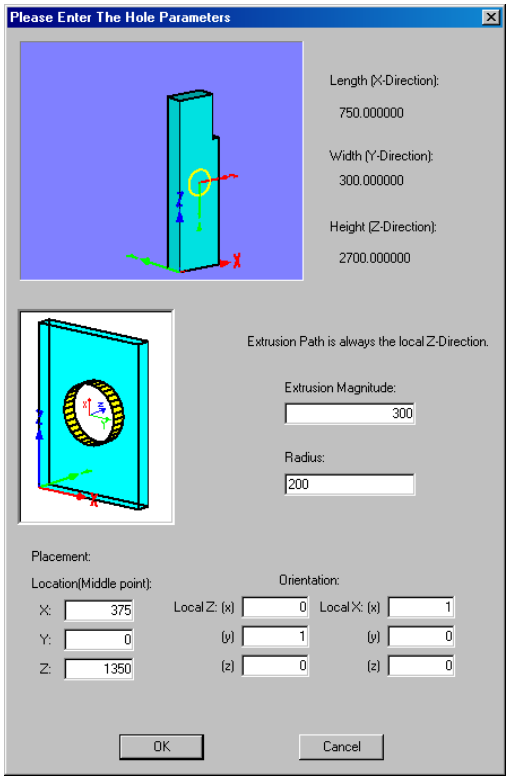


Bild 5.34: Dialogfenster zur Definition des neuen Features

Erfolgt der Klick auf „Ok“, wird das neu definierte Feature in das BauXML-Modell hinzugefügt und ebenfalls im Stein drei dimensional dargestellt. Im Meldungsfenster wird eine Erfolgsnachricht erscheinen.

Unter dem Menu „Konfiguration“ können die OPC-Server, die IP-Adressen für die Überwachungskamera und die SPS-Controller angegeben werden (Bild 5.35).

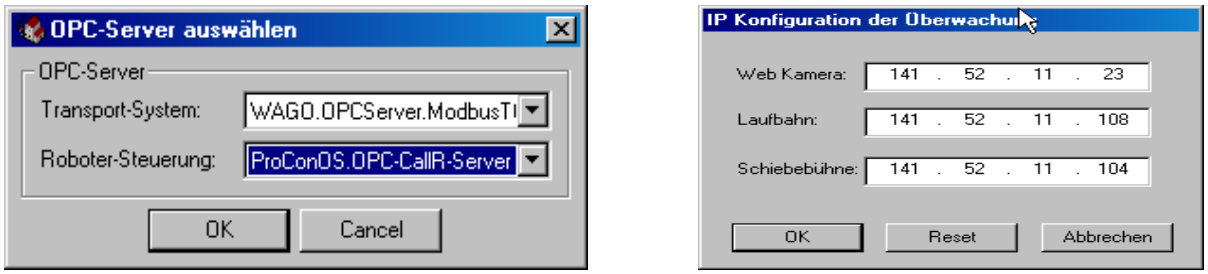


Bild 5.35: Konfigurationsfenster

Nach der Verbindung mit dem OPC-Server können alle OPC-Items der verbundenen OPC-Server in einer Liste gezeigt werden (siehe Bild 5.36). Alle typischen OPC-Eigenschaften des Items wie der Wert, der Name, die Qualität, der Datentyp, die Zugriffsrechte und ein Zeitstempel werden hier angezeigt.

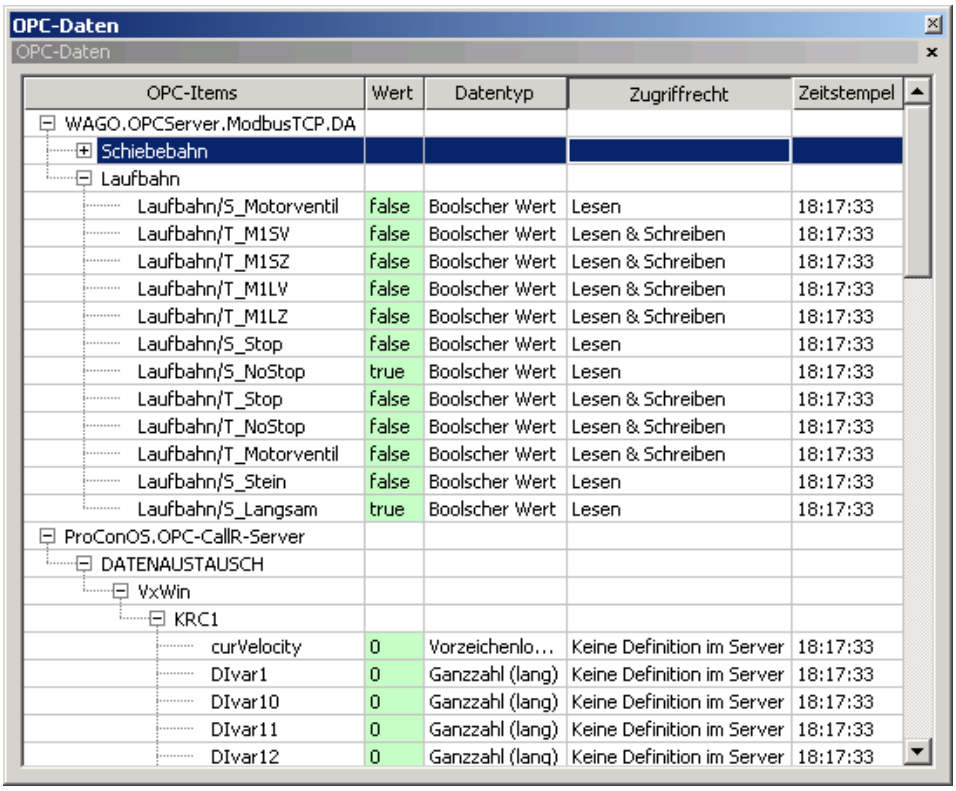


Bild 5.36: OPC-Items Browser

Mit dem HMI (Human Machine Interface) (Bild 5.37) können der Roboter und das Transportsystem gesteuert und die Zustände überwacht werden. Die Tasten im Fenster dienen nur zur handlichen Fernsteuerung. Die Zustände werden farbig verdeutlicht. Die Farbe blau bedeutet ein gesetztes Signal, weiß ein ungesetztes Signal. Die Farbe grün bedeutet Betriebsbereitschaft und rot bedeutet immer eine Gefahr.

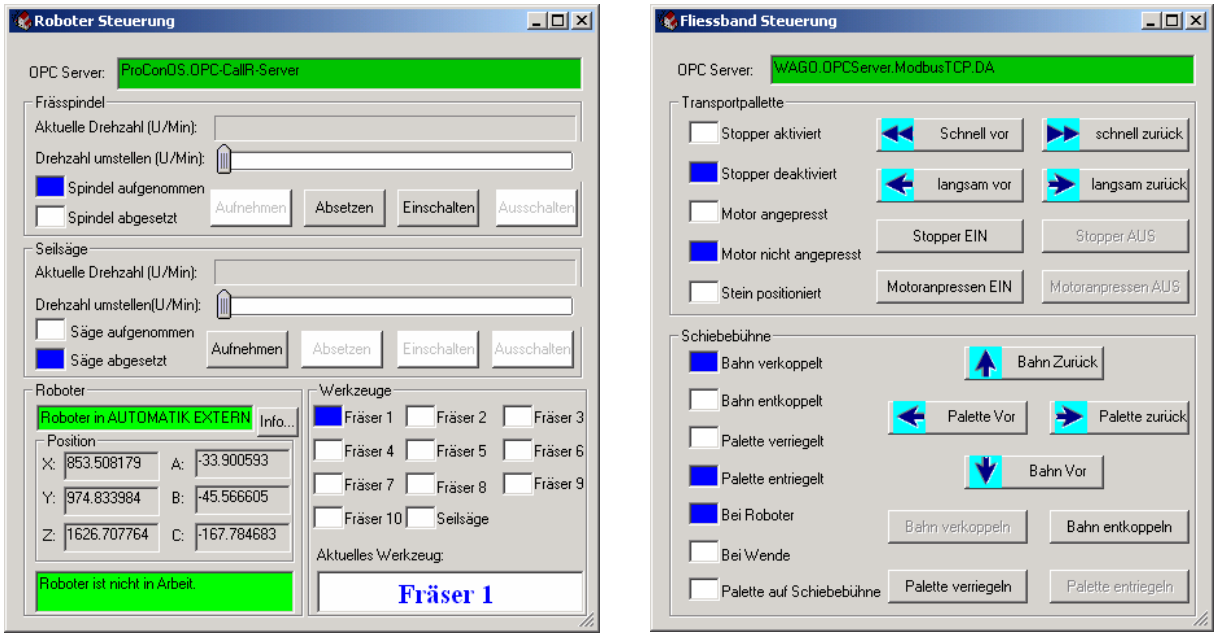


Bild 5.37: Steuerungs- und Überwachungsfenster

5.5.3 Automatische Fertigungsprozedur

In der automatischen Fertigungsprozedur werden die Steine nach der Reihenfolge im BauXML-Modell sequenziell bearbeitet. Ein Stein wird auf der Palette liegend platziert und dort gespannt. Das obere Eck des Steins, das am „Stopper“ liegt, wird als der Ursprung des lokalen Steinkoordinatensystems im BauXML-Modell betrachtet. Die obere Steinseite ist dann die Vorderseite in stehendem Zustand. Durch die Y-Koordinate des Ursprungs vom Feature-Placement innerhalb des Steinkoordinatensystems wird erkannt, ob ein Feature auf der Rückseite vorhanden ist. Die Features in einem Stein werden prinzipiell nach der Vorkommensreihenfolge sequenziell bearbeitet. Um einen ständigen wiederholenden Wechsel zwischen der Säge und der Frässpindel zu vermeiden und um die Fertigungszeit zu minimieren, werden die Arbeiten auf zwei Arten aufgeteilt, nämlich die Sägearbeit und die Fräsarbeit. Ob eine Sägearbeit oder eine Fräsarbeit vorzunehmen ist, entscheidet der Typ des zu bearbeitenden Features. Die Features mit einem „offenen“ Profil wie z.B. einem runden U-Profil, einem rechteckgen U-Profil, einem V-Profil oder Ebene werden durch eine Sägearbeit zuerst bearbeitet. Alle anderen „geschlossenen“ Features werden danach gefräst. Wenn der vom Roboter aufgenommene Fräser für die nächste Arbeitsphase geeignet ist, ist kein Fräserwechsel vorzunehmen. Die Fertigungsparameter wie die Drehzahl, die Vorschubgeschwindigkeit und die Schnitttiefe werden nach der Fräsernummer entschieden und per OPC an die Bearbeitungsmakros übertragen [LIS1].

Nach der Fertigung eines Features wird sein Fertigungszustand geändert. Ist ein Feature fertig bearbeitet, wird die Kennzeichnung des Fertigungszustands des Features auf „true“ gesetzt. Damit wird eine mögliche wiederholte Fertigung des gleichen Features nach Schichtwechsel vermieden.

Nach der Fertigung wird die Palette mit dem Stein auf die Schiebebühne transportiert und die Schiebebühne fährt die Palette weiter zum gegenüber liegenden Schienenstrang. Dafür wird eine Reihe von Befehlen sequentiell durchgeführt, so als ob das ganze Fließband mit den beiden Handbediengeräten sequenziell bedient worden wäre.

5.5.4 Beispiel eines realisierten Hauses

In der vorgestellten Arbeit wurde ein Weg geschaffen, die gesamte CAD/CAM-Kette vom IFC-Modell bis zu einer robotergestützten Fertigung zu automatisieren [LIS2][LIS3]. Um das hier entwickelten Verfahren mit einem „realen“ Haus zu evaluieren, wurde ein kleines „Kinderhaus“ im CAAD-Programm ArchiCAD modelliert (Bild 5.38 links). Das Modell hat vier Wände, zwei Fenster, eine Tür und sechs kleine Wandöffnungen für die Dachpfetten. Gemäß

der vorhandenen Steinmaße wurde es mit den Parametern (maximale Steinlänge 500mm, minimale Steinlänge 200mm, Fenstersturzhöhe 120mm und Auflagerlänge 120mm) segmentiert und anschließend ein BauXML-Modell generiert (Bild 5.38 rechts).

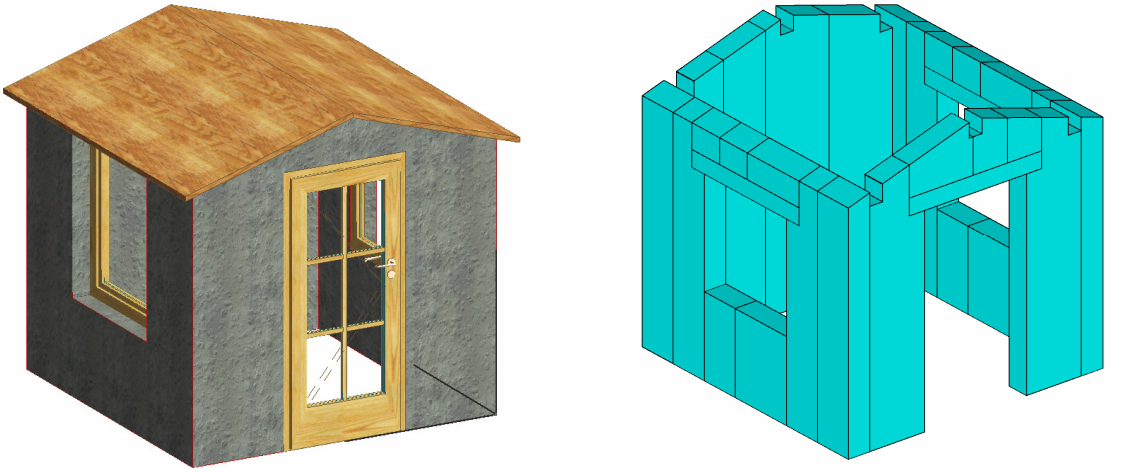


Bild 5.38: CAD- und BauXML-Modell des „Kinderhauses“

In der prototypischen Vorfertigungsfabrik wurden alle Steine des Hauses durch den KUKA-Roboter mit Säge und Frässpindel (Bild 5.39) innerhalb weniger als zwei Stunden gefertigt.

Bild 5.39: Roboter mit einer Seilsäge (links) und einer Frässpindel (rechts) in der Arbeit

Nach der Fertigung wurden alle Steine zusammengebaut und auf dem Kindergartengelände des Forschungszentrums Karlsruhe fertig aufgestellt (Bild 5.40).

Bild 5.40: Fertig aufgebautes „Kinderhaus“

6 Zusammenfassung

Im Vergleich zu vielen anderen Wirtschaftszweigen wird der Einsatz der Informations-, der Kommunikations- und der Automatisierungstechnik im Bauwesen leider nicht ausgenutzt. Auf der Baustelle wird oft noch mittelalterlich traditionelle handwerkliche Arbeit verrichtet, welche die hohen Baukosten verursacht. Zur Verbesserung der Situation wurde eine Methodik entwickelt, die die Montage eines Hauses aus raumhohen Steinen oder Betonfertigelementen mit integrierter Haustechnik vorsieht, die in einer computergestützten Vorfertigung individuell hergestellt werden.

Das Ziel der vorliegenden Arbeit bestand darin, einen Weg von der individuellen Hausplanung bis zur Fertigung in Losgröße Eins zu finden und diesen gesamten Prozess mit Hilfe von Informationstechnologien zu automatisieren. Ausgehend von einem digitalen Gebäude-Produktmodell wurde ein zweistufiges Verfahren entwickelt. Im ersten Schritt werden alle Wände in geschosshohe Steine mit allen Fertigungsfeatures aufgeteilt. Im zweiten Schritt wird jeder Stein automatisiert vorgefertigt. Aufgrund seiner Verbreitung und Interoperabilität wurde als Schnittstelle zu den Hausentwürfen der Architekten auf das IFC-Produktdatenmodell gesetzt. Aus dem IFC-Hausmodell werden alle Wände extrahiert, bearbeitet und als ein BauXML-Modell abgelegt. Ausgehend vom BauXML-Modell werden die Steine des Hauses in der Vorfertigung sequentiell produziert.

Für die Entwicklung des neuen Verfahrens wird im Kapitel 2 das IFC-Produktdatenmodell aus Sicht der Bedürfnisse dieser Arbeit analysiert und untersucht. Im Kapitel 3 wird hinsichtlich der Fertigung ein neues Produktdatenformat BauXML entworfen, das sich an Fertigungsfeatures orientiert und auf XML-Technologie basiert. Die Bearbeitungsobjekte werden im BauXML-Format geometrisch und topologisch beschrieben. Es dient als Datenquelle für die Vorfertigung.

In Kapitel 4 wird das IFC-Hausmodell geometrisch erarbeitet. Dabei werden zuerst die Wandverbindungen fertigungs- und montagegerecht manipuliert. Die Wände des Hauses werden hinsichtlich statischer und fertigungstechnischer Aspekte in einzelne Steine aufgeteilt. Gemäß dem BauXML-Schema wird ein BauXML-Modell generiert, in dem sämtliche Steine inklusive aller Fertigungsfeatures beschrieben werden. Zur Überprüfung der Regenerierbarkeit des Architektenentwurfes aus den Steinen wird das automatisch generierte BauXML-Modell dreidimensional dargestellt.

In Kapitel 5 wird eine Vorfertigungsfabrik für Wandelemente aus Porenbetonsteinen konzipiert und die Realisierung des Prototyps einer automatisierten Roboterzelle zur Wandvorfertigung vorgestellt. In die Roboterzelle werden die Peripherie und die Robotersteuerung integriert, um zahlreiche Funktionen zu erledigen. Realisiert sind eine OPC-basierte Zellensteuerung und eine SPS-basierte dezentrale Steuerung des Fließbands.

Die wichtigsten Ergebnisse der Arbeit sind:

1. Konzeption eines Verfahrens zur Leistungs- und Produktivitätssteigerung im Bauwesen durch Verlagerung der handwerklichen Arbeit von der Baustelle in die Vorfertigungsfabrik. Dort werden raumhohe Mauerwerksteine aller Wände mit integrierter Haustechnik maschinell produziert und dann auf die Baustelle transportiert, um dort zusammenmontiert zu werden. Dies ermöglicht eine schnelle, kostensparende und rationalisierte Bauweise.
2. Auswahl des IFC-Produktdatenmodells als ein passendes Datenformat für das digitale Hausmodell. Dank der Interoperabilität der IFC ermöglicht das Format gezielte Zugriffe und eine parametrisierte Darstellung der die Vorfertigung interessierenden Bauobjekte.
3. Entwurf und Spezifikation eines Fertigungsfeature-orientierten Datenmodells, nämlich des BauXML-Modells. Auswahl und Implementierung des XML-Schemas als formale Beschreibungssprache für das hierarchische und strukturierte BauXML-Modell. Damit ermöglicht es den Benutzern, eigene Datentypen auf Basis zahlreicher Datentypen im XML-Schema aufzubauen und die Datenstruktur zu erweitern. Darüber hinaus ist die grammatikalische Richtigkeit der eingehenden BauXML-Datei für die Fertigung durch eine Validierung gegen das BauXML-Schema gesichert.
4. Erarbeitung und Manipulation des IFC-Hausmodells. Darunter sind die Korrektur und die Manipulation der Wandverschnitte, die Aufteilung der Wand in einzelne raumhohe Mauerwerksteine und am Ende die Generierung des entsprechenden BauXML-Modells zu verstehen.
5. Implementierung und Erweiterung modifizierter Computeralgorithmen zur Behandlung computergrafischer Probleme und 3D-Darstellung graphischer Daten. Diese Arbeiten wurden in dem entwickelten Programm „IfcWallModificator“ zusammengefasst.

6. Konzeption und Realisierung einer automatisierten Roboterzelle zur Wandvorfertigung. Auswahl und Integration der Sensoren und Aktoren für die Peripherie und Einbindung in die KUKA-Robotersteuerung durch das DeviceNet bzw. eine serielle Schnittstelle. Bereitstellung der Fertigungsprogramme aller Fertigungsfeatures und der Hilfsfunktionen wie dem Werkzeugwechsel und der Bedienung der Peripherie in der Robotersteuerung.
7. SPS-Programmierung des Fließbands und der Datenübertragung mit der Robotersteuerung. Auswahl und Nutzung der OPC-Technologie als offene Kommunikationsschnittstelle zur Visualisierung, Optimierung und Kontrolle von Prozessdaten. Entwickelt wurden eine OPC-basierte zentrale Zellensteuerung und eine SPS-basierte dezentrale Steuerung auf der Feldebene. Dadurch entsteht ein flaches, dezentrales und flexibles Steuerungskonzept.
8. Entwicklung eines übergeordneten Steuerungsprogramms „WallProducer“ für die OPC-basierte zentrale Zellensteuerung. Implementierung der Bedienung und Visualisierung des Roboters und des Fließbands im Programm. „WallProducer“ übernimmt die HMI-Funktionen zur Bedienung und Beobachtung der Produktionsprozesse. Es automatisiert kompakt und komfortabel die Fertigungszelle und nutzt die PC-basierte Bedienoberfläche als „Production Screen“ für die Roboterzelle.
9. Verifikation der Regenerierbarkeit des Entwurfs aus dem BauXML-Modell anhand mehrerer konkreter Hausmodelle und experimentelle Evaluierung der erarbeiteten Verfahren bei Steinen mit unterschiedlichen Features.
10. Experimentelle Erprobung des entwickelten Verfahrens anhand eines realisierten Kindhauses.

Im Vergleich zu anderen Vorfertigungsverfahren bietet das in dieser Arbeit entwickelte Verfahren neben der Arbeitserleichterung eine erhebliche Kundenorientierung und Kostenersparnis bei erhöhter Fertigungspräzision. Für eine Umsetzung als Produkt können der Betonhersteller oder die Fertigteilanbieter das IFC-Format und den „IfcWallModificator“ verwenden, um Wände automatisch in einzelne raumhohe Steine aufzuteilen. Die in der Arbeit entwickelte Roboterzelle und die Steuerungen können in die Praxis umgesetzt werden. Die kundenspezifischen Anlagen können aber auch in dem entwickelten Steuerungskonzept integriert und angepasst werden. Ein Versuch zur Überführung des in dieser Arbeit entwickelten Knowhows in die Industrie ist ein momentan laufender Prozess.

Um eine möglichst praxisnahe Segmentierung durchzuführen, sind die Randbedingungen zur Segmentierung noch weiter zu untersuchen. Als Ergebnis der Generierung des BauXML-Modells ist es auch möglich ein 2D Steinlayout aller Wände bereitzustellen. In IFC sind die Wände ohne fest vorgegebene Reihenfolge definiert. Als Folge werden die Steine im BauXML-Modell nach der Vorkommensreihenfolge der Wände in IFC nacheinander „platziert“ und produziert. Aus Sicht der Arbeit auf der Baustelle soll aber die Reihenfolge der Steine frei auswählbar sein. Die Fertigungsreihenfolge der Steine in der Vorfertigung muss also entsprechend festgelegt werden können. Eine Erweiterung des Programms „WallProducer“ kann zudem dazu dienen, die Ausführungsreihenfolge der Steine vor der Fertigung anzugeben. Als nächstes Entwicklungspotenzial sollten auch die Reihenfolge des Fertigungsauftrags und der Logistik untersucht und optimiert werden.

7 Literatur

[ALTO] ALTOVA GMBH: *http://www.altova.com/xmlspy*

[ANDL] R. ANDERL; D. TRIPPNER (HRSG.): *STEP - Standard for the Exchange of Product Model Data: Eine Einführung in die Entwicklung, Implementierung und industrielle Nutzung der Normenreihe ISO 10303 (STEP)*. B.G. Teubner Verlag, Stuttgart, ISBN 3-519-06377-8, 2000.

[BERG] M. DE BERG; M. VAN KREVELD; M. OVERMARS; O. SCHWARZKOPF: *Computational Geometry – Algorithms and Applications*. Second, revised Edition, Springer Verlag. ISBN 3-540-65620-0, 2000.

[BENN] J. BENNER, K. LEINEMANN, A. LUDWG: *Übertragung von Geometrie und Semantik aus IFC-Gebäudemodellen in 3D-Stadtmodelle*. CORP 2004 & Geomultimedia, Seite 573, ISBN 3-901673, 2004.

[BERN] R. BERNHARDT; W. BERNHARDT: *CAD-CAM Anwendungsbeispiele aus der Praxis*. Offenbach, VDE-Verlag, ISDN 3-8007-1323-3, 1984.

[BITT] D. BITTRICH: *Industry Foundation Classes: Ein Überblick. Lehrstuhl Informatik im Bauwesen*. Bauhaus-Universität Weimar, März 1998.

[BRET] G. BRETTHAUER; S. DIETZE; K.-H. HÄFELE; J. ISELE; J. JÄKEL: *Nachhaltiges Planen, Bauen und Wohnen im Informationszeitalter*. Wissenschaftliche Berichte FZKA 6626, Forschungszentrum Karlsruhe, 2001.

[DANF] FIRMA DANFOSS: *Empfehlung für Schnittstellenkonverter*. Online-Dokument: http://www.danfoss.de/desca/elektrischeantriebe/Files/Konverteranschluß.pdf

[DEMB] K. DEMBOWSKI: *Computerschnittstelle und Bussysteme*. Markt- und Technik Verlag., 1993, ISBN 3-87791-440-3

[DIAI] IAI-GERMANY: *http://www.iai-ev.de*

[ECCO] PDTEC GMBH: *http://www.pdtec.de*

[EHLT] D. EHLTING: *Vorfertigung komplexer Ausbau-Bausysteme für offene Bausweisen*. Dissertation, Universität Dortmund, 2001.

[ESSE] D. BOX; A. SKONNARD; J. LAM: *Essential XML-XML für die Softwareentwicklung*. Addison-Wesley Verlag, ISBN 3-8273-1769-X, 2001.

[EVER] W. EVERSHEIM; M. PHORNPRAPHA; M. WESTEKEMPER: Migrationskonzept für DV-Systeme der NC-Verfahrenskette - Einbindung feature-basierter Planungsme-

thoden in technologieorientierte NC-Programmiersysteme. „Werkstattstechnik (wt)-online“, 03-2001, Seite 117.

[GANZ] W. GANZ: *Schnell und effektiv planen*. Mikado 3 (1996) 38-42.

[GEIG] A. GEIGER: *Manipulation der Wandverschnitte im IFC-Produktdatenmodell*. Diplomarbeit, Forschungszentrum Karlsruhe, 2002.

[GRA1] H. GRABOWSKI: *35 Jahre Plattenbau in der DDR: Vom Versuchsbau zur materiall-technischen Basis des Wohnungsbaus*. In: Bauzeitung 43 (1989), Nr. 5. Seite 202-205.

[GRA2] H. GRABOWSKI: *Rechnerunterstütztes Konstruieren und Erstellen von Fertigungsunterlagen*. Vorlesungsskript, Universität Karlsruhe, 2001.

[HAAS] W. HAAS: *Datenaustausch und Datenintegration - STEP und IAI als Beiträge zur Standarisierung*.

[HAHN] A. STEINLE, V. HAHN: *Bauen mit Betonfertigteilen im Hochbau*. Ernst & Sohn, Berlin, 1998.

[HANP] J. HAN; M. PRATT; W.C. REGLI: *Manufacturing Feature Recognition from Solid Models: a statusreport*. IEEE Transaction on Robotics and Automation, ISBN 1042-296X, Vol. 16, Issue. 6, December 2000.

[HERB] G. HERBERT: *The Dream of the Factory-Made House*. Cambridge/Mass, Seite 15, 1986.

[HULI] H. LI: *Bestimmung der Fräsparameter für die Zerspanung von Porenbeton*. Studienarbeit, Forschungszentrum Karlsruhe, 2003.

[IAI1] PRESSEMITTEILUNG: *Der IFC Standard der IAI jetzt erstmals in kommerziellen Produkten*. München, 3. Nov. 2000.

[IFC1] INTERNATIONAL ALLIANCE FOR INTEROPERABILITY (IAI): *IFC 2x2 Specifications*.

[IFC2] INTERNATIONAL ALLIANCE FOR INTEROPERABILITY (IAI): *IFC 2x Extension Modelling Guide*.

[IFC3] INTERNATIONAL ALLIANCE FOR INTEROPERABILITY (IAI): *IFC 2x2 Model Implementation Guide*.

[ISEL] J. ISELE: *Präsentation der Fertigungsfabrik*. Interne Dokumentation, Forschungszentrum Karlsruhe, Mai 2000.

[ISO1] ISO 10303-224: *Mechanical product definition for process planning using machine features*. ISO TC184/SC4/WG3 N405, 1995.

[JUNG] K. JUNGHANNS: *Das Haus für alle: zur Geschichte der Vorfertigung in Deutschland*. Ernst & Sohn Verlag, Berlin, ISBN 3-433-01274-1, 1994.

[KOTU] B. KOTULLA: B.-P. URLAU-CLEVER: *Industrielles Bauen*. 2. Auflage, ISBN 3-8169-1145-5, Expert Verlag, 1994.

[KUKA] KUKA ROBOTER GMBH: *http://www.kuka.de/*

[LANG] B. von LANGENBECK; A. VON COLER; F. WERNER: *Die transportable Lazarettbaracke*. Zweite vermehrte Auflage, Seite 29, Berlin, 1890.

[LENG] E. LENGYEL: *„Mathematics for 3D Game Programming and Computer Graphics“*. Charles River Media, Inc., ISBN 1-58450-037-9, 2002.

[LIS1] S. LI: *Automatisierte Einzelfertigung-Übertragung der Produktbeschreibung aus einer XML-Datei via OPC in ein Roboterprogramm*. Diplomarbeit, Forschungszentrum Karlsruhe, 2002.

[LIS2] S. LI; J. ISELE; K.-H. HÄFELE; A. GEIGER: *CAD/CAM Integrated Building Prefabracation based on a Product Data Model*. Proceeding of the 11th International Conference on Computing in Civil and Building Engineering (ICCCBE-XI), ISBN 2-921145-57-X, Montreal, Canada, 2006.

[LIS3] S. LI; J. ISELE; G. BRETTHAUER: *Application of IFC Product Data Model in Computer Integrated Building Prefabrication.* Proceeding of the ASCE International Conference on Computing in Civil Engineering 2007, Pittsburgh, USA, 2007.

[LUST] T. FETTERS: *The Lustron Home – The History of a Postwar Prefabricated Housing Experiment*. McFarland & Company, ISBN 0786411333, 2002.

[MASS] W. MASSBERG; J. XU: *CAD/CAP-Integration: Einsatz Feature-basierter Modelle: Berücksichtigung des Concurrent Engineering*. VDI-Z ISSN 0042-1766, 1995, Vol. 137, Seite 41-44. Springer-VDI

[MENT] J. MENTZEL: *Framework zur Bearbeitung des IFC Produktdatenmodells am Beispiel der Wandsegmentierung*. Diplomarbeit, Forschungszentrum Karlsruhe, 2003.

[MUNZ] H. MUNZ: *Echtzeit trotz PC - LP-VxWIN*. Firmenschrift der LP Elektronik GmbH, Weingarten, März 1997.

[NEUB] F. NEUBERG: Ein Softwarekonzept zur Internet-basierten Simulation des Ressourcenbedarfs von Bauwerken. Dissertation, 2003, Technische Universität München.

[NOVA] http://www.fertighaus.biz/fertighaeuser/fertighaus/Hanse_Haus_129.htm

[OPCF] OPC FOUNDATION: *OPC- Data Access Custom Interface Standards*. Version 2.0, October 14, 1998.

[PROS] PROSTEP IVIP VEREIN: *http://www.prostep.org/de*

[RANK] E. RANK: *Vorlesungsskript "Bauinformatik"*, TU München.

[RIOL] E. SCHLITZ-RIOL: *Baukonstruktive Innovation für den Geschoßwohnungsbau unter nachhaltigen Kriterien*. Dissertation, 1998, Bauhaus-Universität Weimar.

[ROMH] R. ROMHEIN: *Computergestützte Konstruktion, Fertigung und Arbeitssteuerung von Ausbaubauteilen*. Dissertation, Universität Dortmund, ISBN 3-89825-482-8, 2002.

[SCHN] G. SCHNEIDER: *„Steuerungstechnik für Industrieroboter“*, KUKA Roboter GmbH, Augsburg. 1998.

[SCNE] P. J. SCHNEIDER; D. H. EBERLY: *„Geometric Tools for Computer Graphics“*. Morgan Kaufmann Publishers, ISBN 1-55860-594-0, 2003.

[SOFT] SOFTING AG: *OPC- ein Standard für die Praxis*, *www.softing.de*

[STVO] *http://bundesrecht.juris.de/bundesrecht/stvo/__22.html*

[TÖNS] H. K. TÖNSHOFF; P.-O. WOELK: „Feature-based Data Model for Integration of Design and Process Planning”. Proceedings of ProSTEP Science Days 2000 “SMART Engineering”, Seite 157-168, Sindelfingen, September 13-14 2000.

[VDI9] VDI-Richtlinie 2218 „Feature-Technologie“. Entwurf 1999-11, Beuth Verlag, Berlin 1999.

[VLT5] DANFOSS GMBH: Projektierungshandbuch VLT Serie 5000.

[W3C] W3C: *http://www.w3c.org/XML*

[WACH] K. WACHSMANN: *Holzhausbau – Technik und Gestaltung*. Neuausgabe des Originals von 1930: Birkhäuser, Basel, 1995.

[WAGO] WAGO Kontakttechnik GmbH & Co. KG: *http://www.wago.com/*

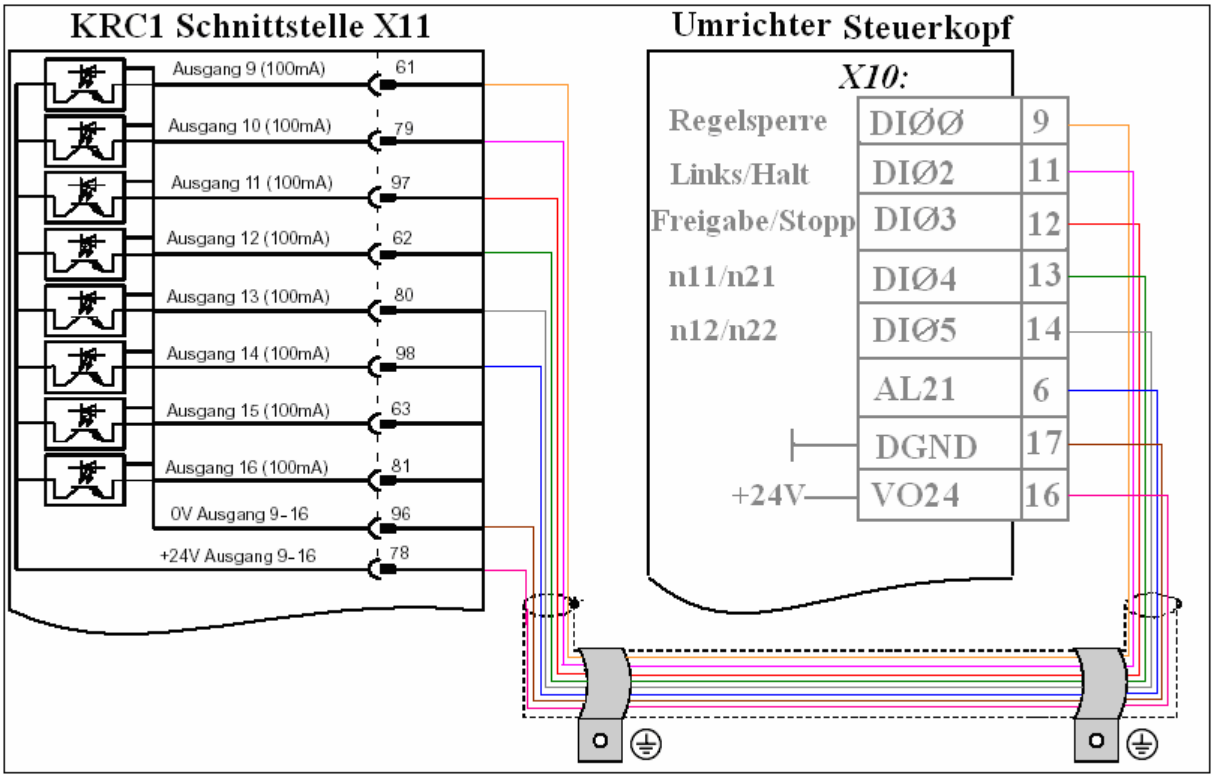
[WEBE] C. WEBER; F.-L. KRAUSE: *Neue VDI-Richtlinie über Feature-Technologie*. Zeitschrift „Konstruktion“ 4-2000, Seite 31.

[WINT] WINTECH SOFTWARE DESIGN: *http://www.win-tech.com/*

[WÖRN] H. WÖRN: *Steuerungstechnik für Roboter*. Vorlesungsfolien, Institut für Prozessrechentechnik, Automation und Robotik (IPR), Universität Karlsruhe.

8 Anhang

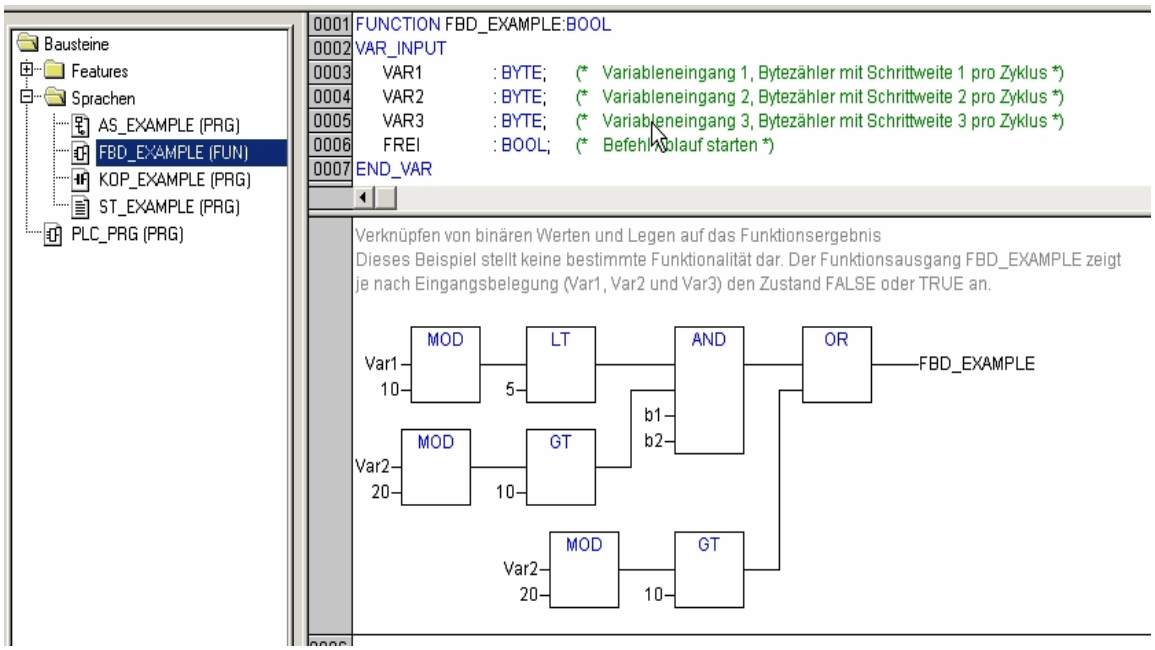
8.1 Verkabelung zwischen der KRC1 und dem SEW-Antriebumrichter



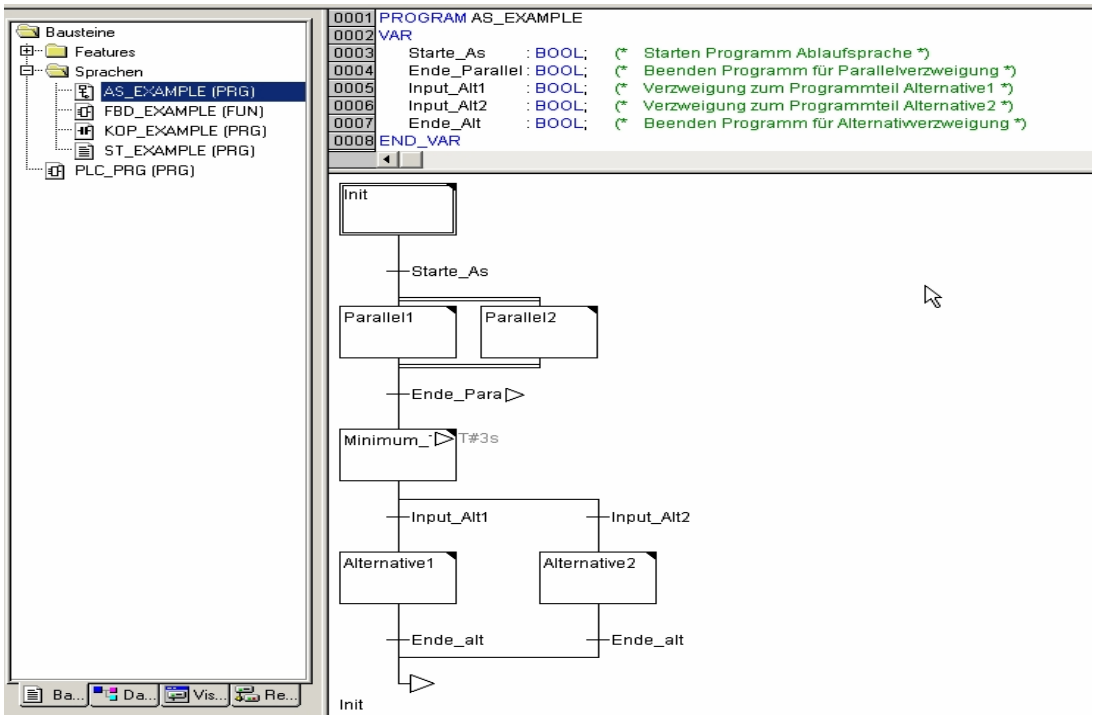
8.2 SPS-Programmiersprache Norm IEC 61131-3 [WAGO]

Die Norm IEC 61131-3 von 1993 ist ein internationaler Standard für moderne Systeme mit SPS-Funktionalität. Die deutsche Übersetzung ist in der europäischen Norm EN 61131-3 enthalten und hat den Status einer Deutschen Norm. Aufbauend auf einem strukturierten Softwaremodell definiert sie eine Reihe leistungsfähiger Programmiersprachen, die für unterschiedliche Automatisierungsaufgaben eingesetzt werden können. Insgesamt werden fünf Programmiersprachen definiert, und zwar Funktionsplan (FUP), Ablaufsprache (AS), Kontaktplan (KOP), Strukturierter Text (ST), und Anweisungsliste (AWL).

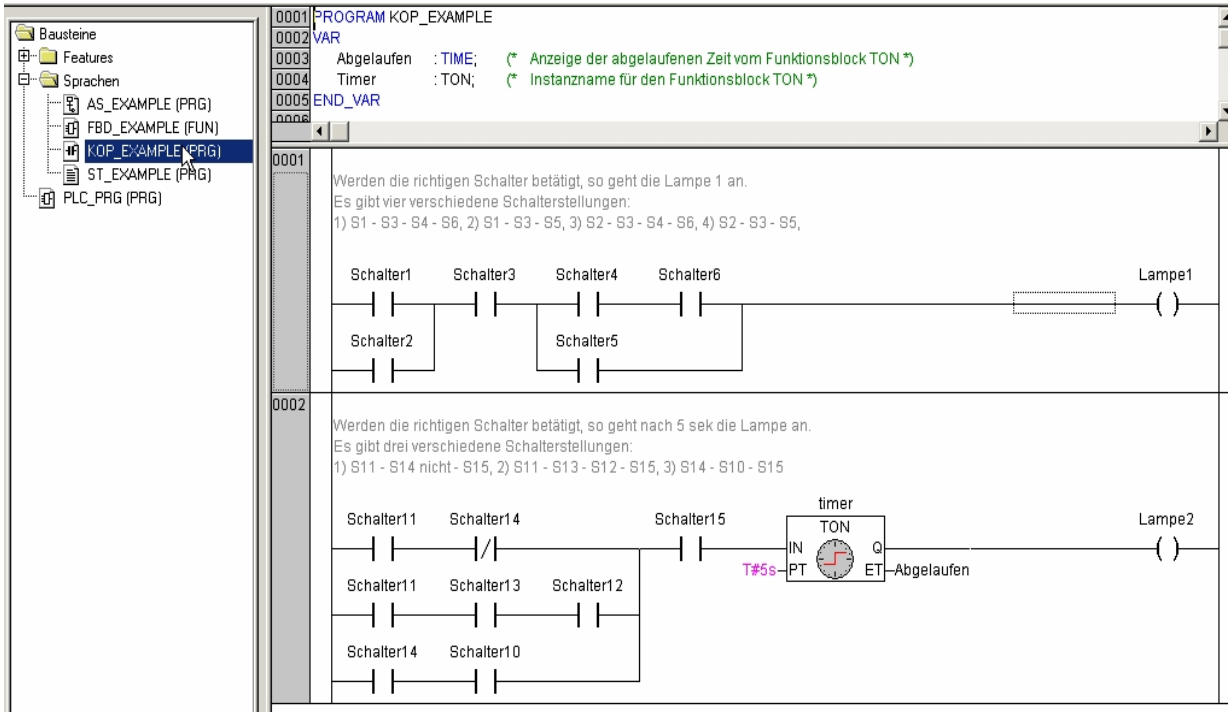
Funktionsplan (FUP): Der Funktionsplan (FUP) ist eine grafisch orientierte Programmiersprache. Er arbeitet mit einer Liste von Netzwerken, wobei jedes Netzwerk eine Struktur enthält, die jeweils einen logischen bzw. arithmetischen Ausdruck, den Aufruf eines Funktionsblocks, einen Sprung oder eine Return-Anweisung darstellt.



Ablaufssprache (AS): Ablaufssprache (AS) ist eine grafisch orientierte Sprache, die es ermöglicht, zeitliche Abfolgen verschiedener Aktionen innerhalb eines Programms zu beschreiben.



Kontaktplan (KOP): Kontaktplan (KOP) ist ebenfalls eine grafisch orientierte Programmiersprache, die zur Darstellung von Verknüpfungen das Prinzip einer elektrischen Schaltung mit Kontakten und Spulen in Netzwerk verwendet.



Strukturierter Text (ST): Strukturierter Text (ST) besteht aus einer Reihe von Anweisungen, die wie in Hochsprache (z.B. Pascal) bedingt („IF..THEN ..ELSE “) oder in Schleifen („WHILE..DO“) ausgeführt werden können.

```
PROGRAM ST_EXAMPLE
VAR
    xVal          : INT       := 0;     (*  X-Position des Greifers *)
    yVal          : INT       := 0;     (*  Y-Position des Greifers *)
    xValBall      : INT       := 0;     (*  X-Position des Balles *)
    yValBall      : INT       := 0;     (*  Y-Position des Balles *)
    updirection   : BOOL      :=TRUE;   (*  Bewegung Heben *)
    downdirection : BOOL      :=FALSE;  (*  Bewegung Senken *)
    leftdirection : BOOL      :=FALSE;  (*  Bewegung nach links *)
    rightdirection : BOOL     :=FALSE;  (*  Bewegung nach rechts *)
    withball      : BOOL      :=TRUE;   (*  Bewegung mit Ball (TRUE... mit Ball / FALSE...ohne Ball *)
    run           : BOOL      :=FALSE;  (*  Bewegung gestartet ( TRUE...Greifer läuft /
```

```
run_string:='Start';                    (*  Defaultanzeige für Start und Stop Taste: Start
                                            - bedeutet: Anlage ist aus *)

IF NOT run THEN
    RETURN;                             (*  Ist run = FALSE, wird das Programm an dieser Stelle
                                            abgebrochen *)
END_IF;                                 (*  Ansonsten wird das Programm weiter bearbeitet. *)
run_string:='Stop';                     (*  Anzeige für Start und Stop Taste: Stop
                                            - bedeutet: Anlage läuft *)

IF updirection THEN                     (*  Starten mit Heben, wenn updirection = TRUE *)
    IF yVal = -100  THEN                (*  Ist die Y-Position = -100 (Greifer oben), dann wird *)
        updirection := FALSE;           (*  das Heben gestoppt. *)
        IF xVal = 0 THEN                (*  Ist die X-Position = 0 (Greifer links), dann wird nach
                                            dem Heben *)
            rightdirection := TRUE;     (*  nach recht gefahren. *)
        ELSE                            (*  Ist die X-Position <> 0 (Greifer rechts), dann wird nach
                                            dem Heben *)
            leftdirection := TRUE;      (*  nach links gefahren. *)
        END_IF;
    ELSE
```

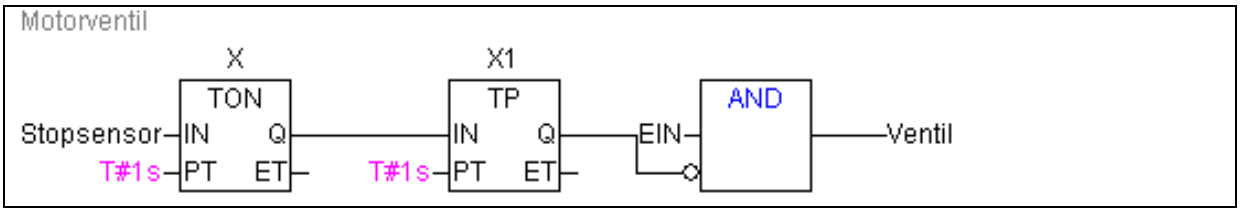
Anweisungsliste (AWL): Die Anweisungsliste ist eine weit verbreitete Assembler-nahe Textsprache für SPS.

```
WARTEN (FB-AWL)
0001 FUNCTION_BLOCK WARTEN
0001     LD    ZAB.Q
0002     JMPC  marke
0003
0004     CAL   ZAB(IN:=FALSE)
0005     LD    ZEIT
0006     ST    ZAB.PT
0007     CAL   ZAB(IN:=TRUE)
0008     JMP   ende
0009
0010 marke:
0011     CAL   ZAB
0012 ende:
0013     LDN   ZAB.Q
0014     ST    OK
0015     RET
```

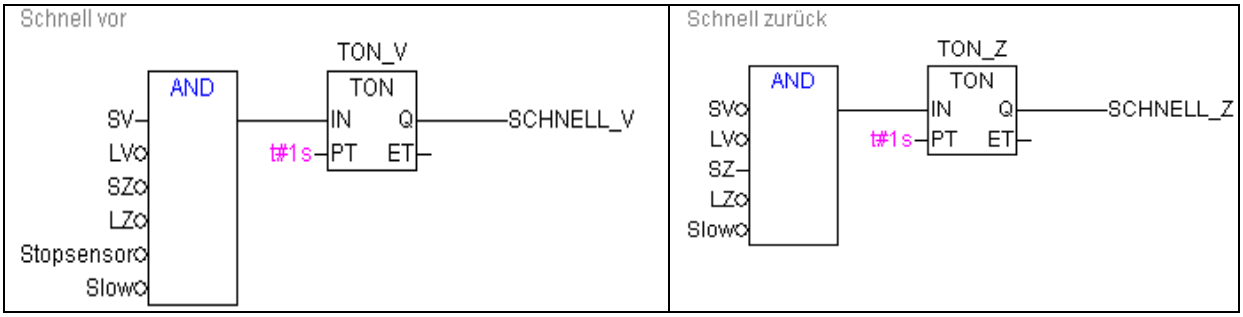
8.3 SPS-Steuerungsprogramme der Laufbahn

Funktionen zur Aktivierung/Deaktivierung des Stoppers

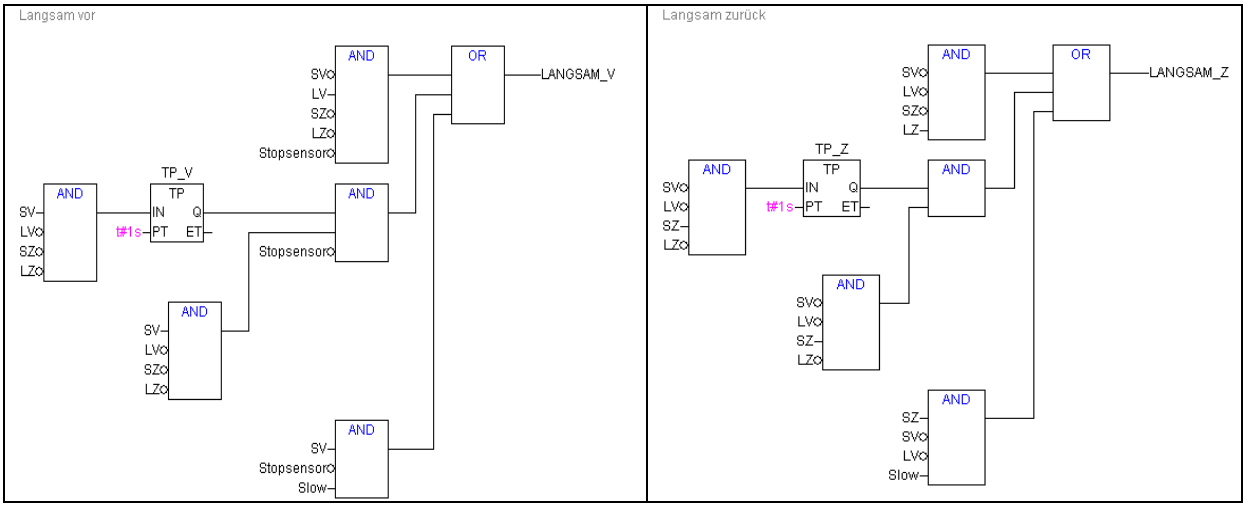
Funktion zum Anpressen des Motors:



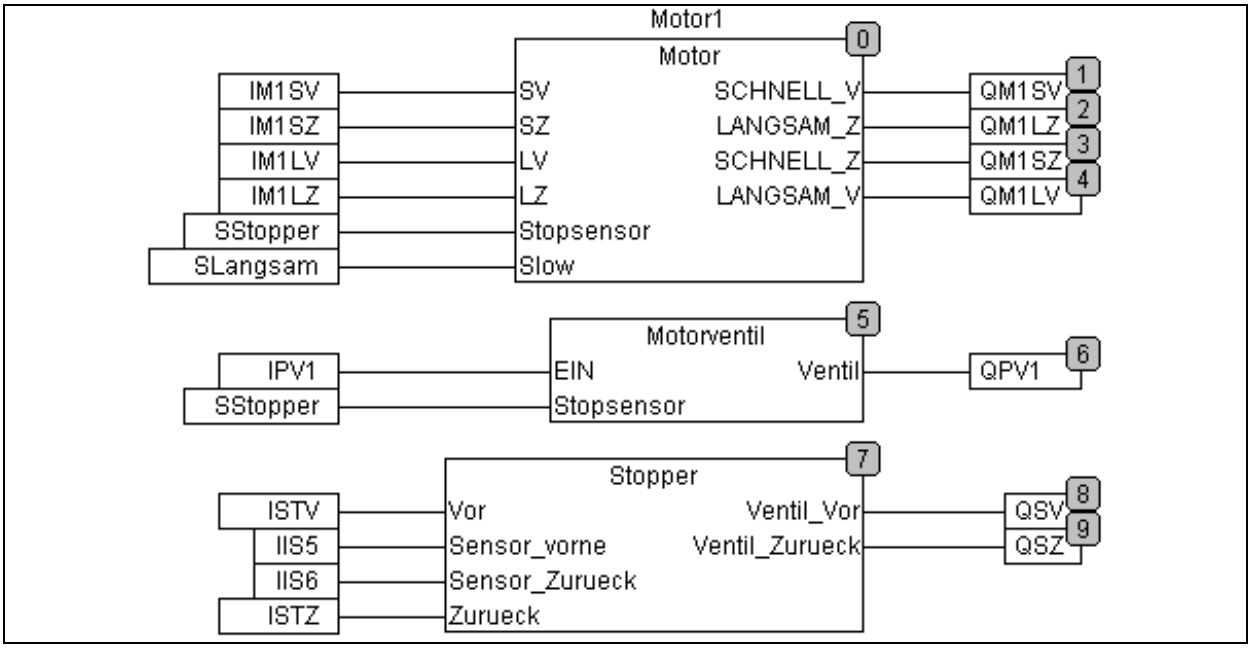
Funktionen, um den Motor schnell zu drehen:



Funktionen, um den Motor langsam zu drehen:

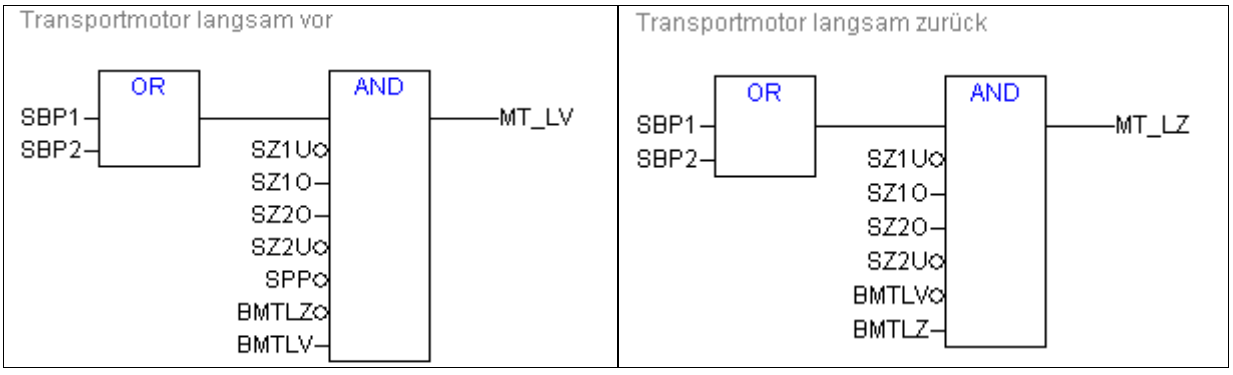


Gesamte Steuerung PLC-PRG der Laufbahn:

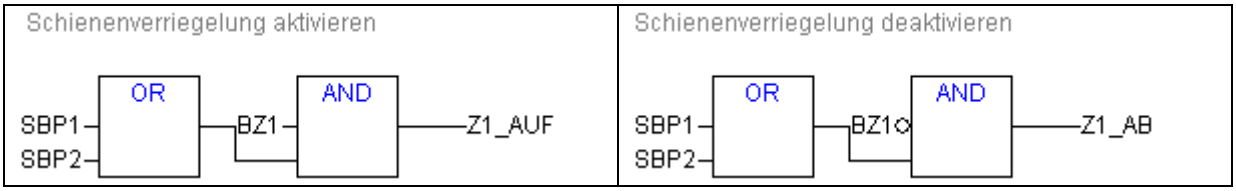


8.4 SPS-Steuerungsprogramme der Schiebebühne

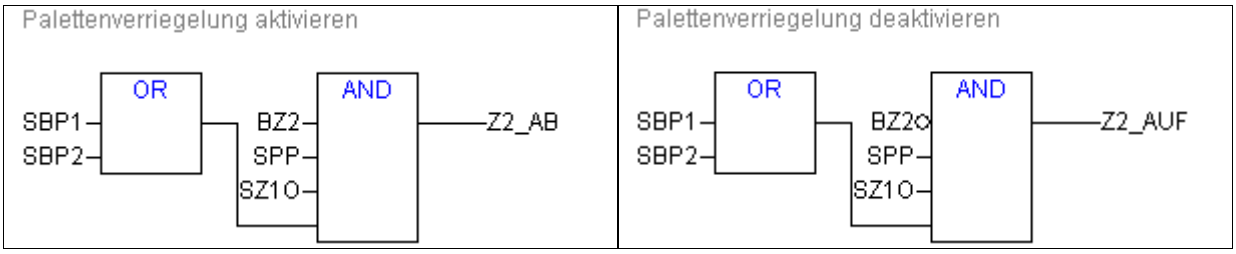
Funktionen zum Antrieb der Schiebebühne:



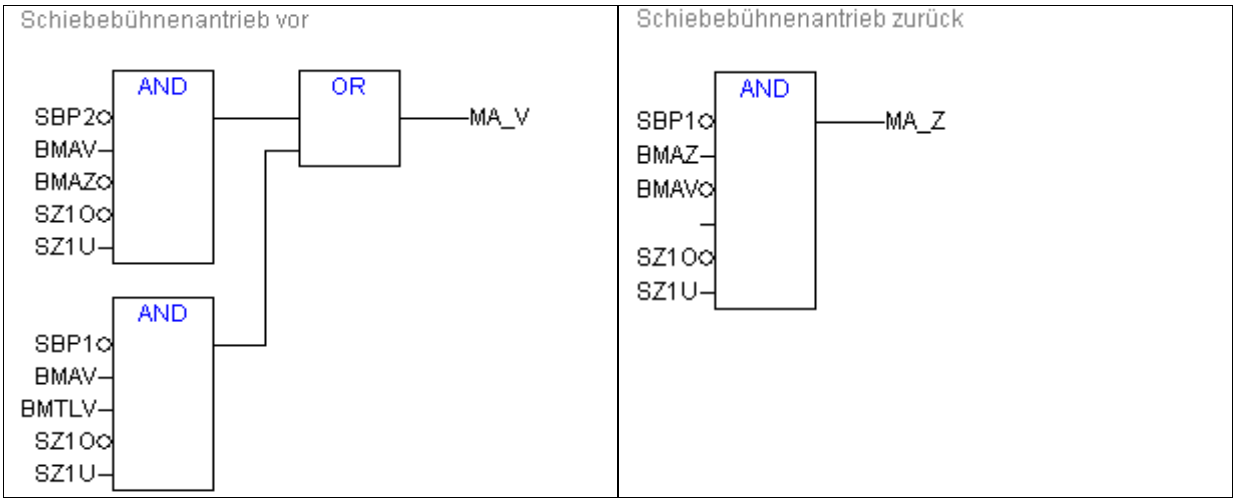
Funktionen zur Kopplung der Schienen:



Funktionen zur Verriegelung der Palette:

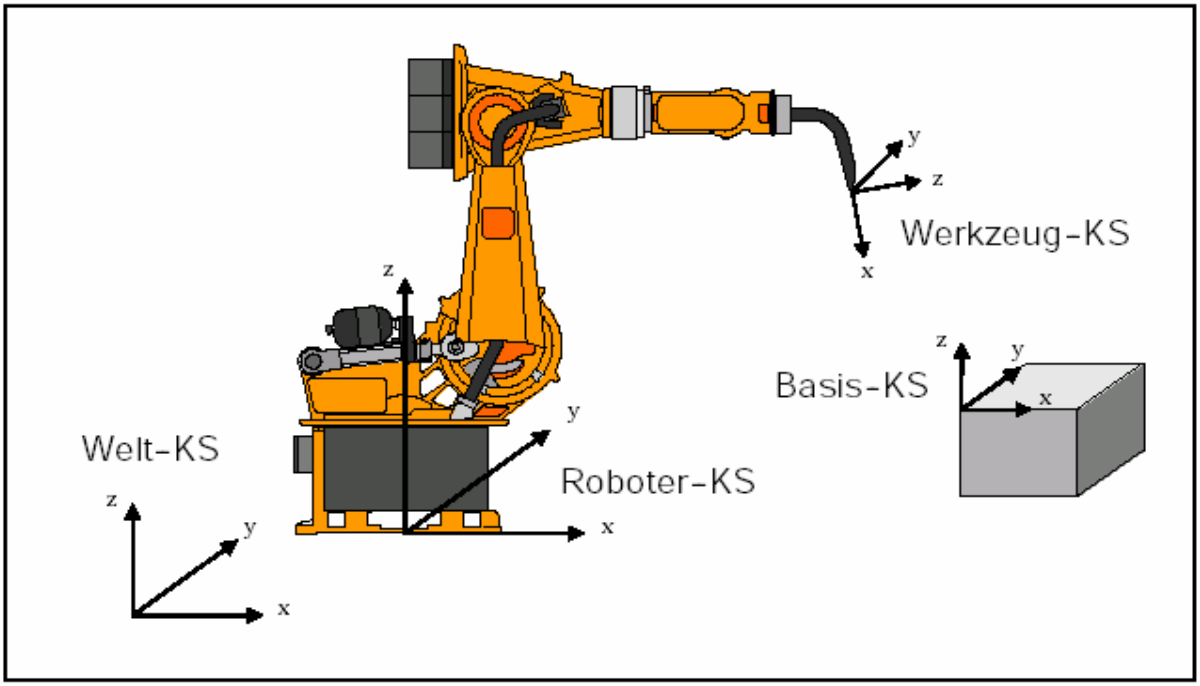


Funktionen für den Antrieb der Schiebebühne:



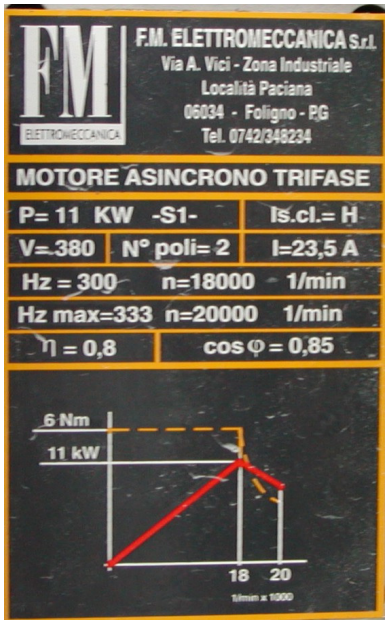
Funktionen um Anpressen des Transportmotors:

8.5 Kartesische Koordinatensysteme KRC1 [KUKA]



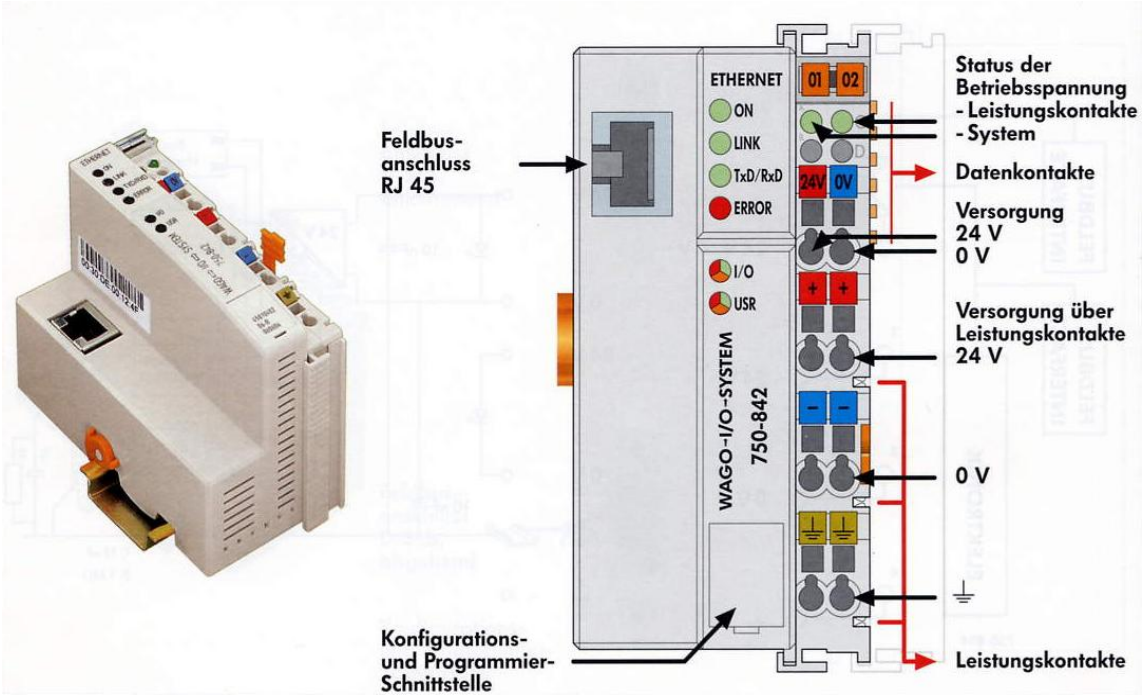
8.6 Technische Daten

Frässpindel



Programmierbarer ETHERNET-Controller: 750-842

Der programmierbare Feldbus-Controller für ETHERNET kombiniert den WAGO-Feldbuskoppler für ETHERNET mit der Funktionalität einer SPS.

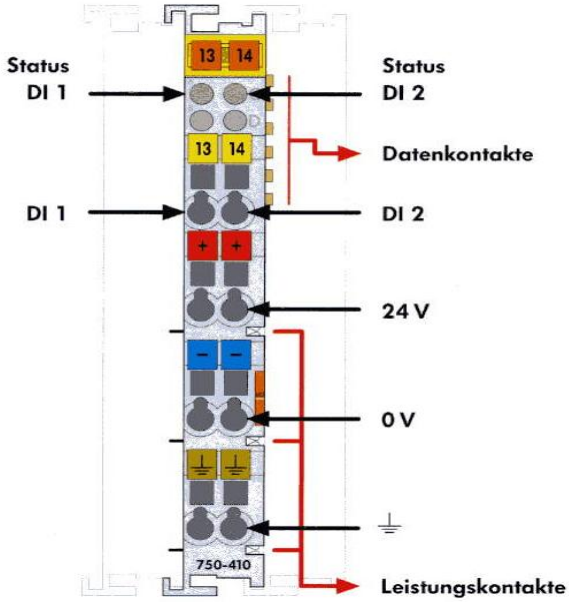


Wichtige Technische Daten:

Anzahl Busklemmen	64
Übertragungsmedium	Twisted Pair S-UTP 100 Ω Cat 5
Busanschluss	RJ 45
Max. Bussegmentlänge	100m zwischen Hub und 750-842, Max. Netzwerk-länge durch ETHERNET-Spezifikation limitiert.
Übertragungsrate	10 Mbit/s
Protokolle	MODBUS/TCP, http, BootP, MODBUS/UDP
Programmierung	WAGO-I/O-PRO 32
IEC 61131-3	AWL, KOP, FUP, ST, AS
Programmspeicher	128 kByte
Datenspeicher	64 kByte
Zykluszeit	< 3 ms für 1000 Bitanweisungen / 256 dig. E/A's
Spannungsversorgung	DC 24 V (-15% ... +20%)
Eingangsstrom $_{max.}$	500 mA bei 24 V
Interne Stromaufnahme	200 mA bei 5 V
Potenzialtrennung	500 V System / Versorgung
Spannung über Leistungskontakte	DC 24 V (-15% ... +20%)
Strom über Leistungskontakte $_{max.}$	DC 10 A
Betriebstemperatur	0 °C … +55 °C

2-Kanal Digital Eingangsklemme DC 24 V: 750-401

Die digitalen Eingangsklemmen erfassen Steuersignale aus dem Feldbereich z.B. über Sensoren.

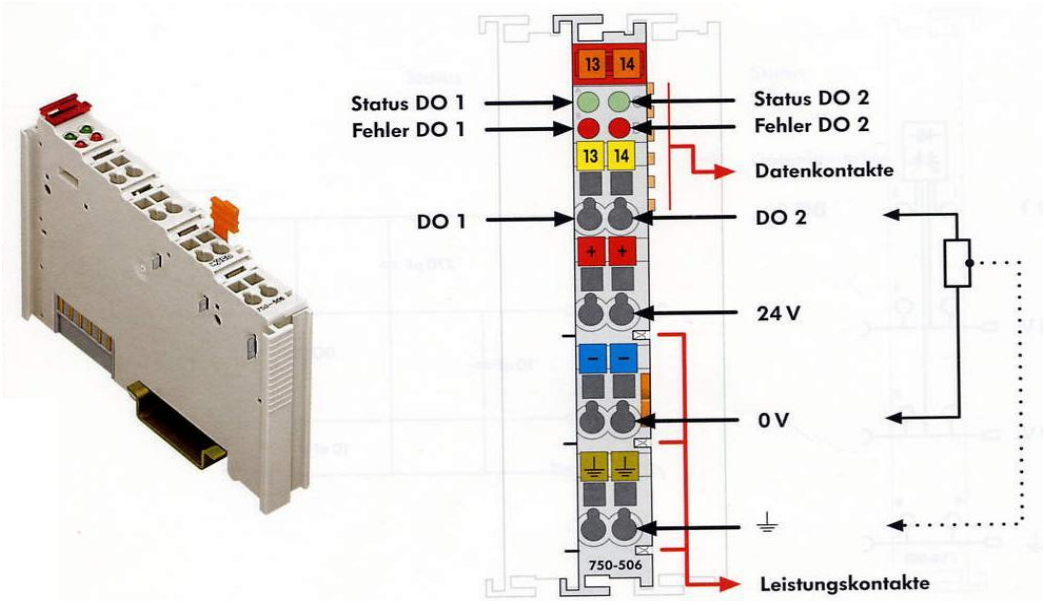


Wichtige Technische Daten:

Anzahl der Ausgänge	2
Stromaufnahme $_{max.}$ (intern)	2,5 mA
Spannung über Leistungskontakte	DC 24 V (-15% ... +20%)
Signalspannung (0)	DC -3V bis +5V
Signalspannung (1)	DC 11V bis 30V
Eingangsstrom	8 mA
Potenzialtrennung	500 V System / Versorgung
Datenbreite intern	2 Bit
Betriebstemperatur	0 °C … +55 °C

2-Kanal Digital Ausgangsklemme DC 24 V: 750-506

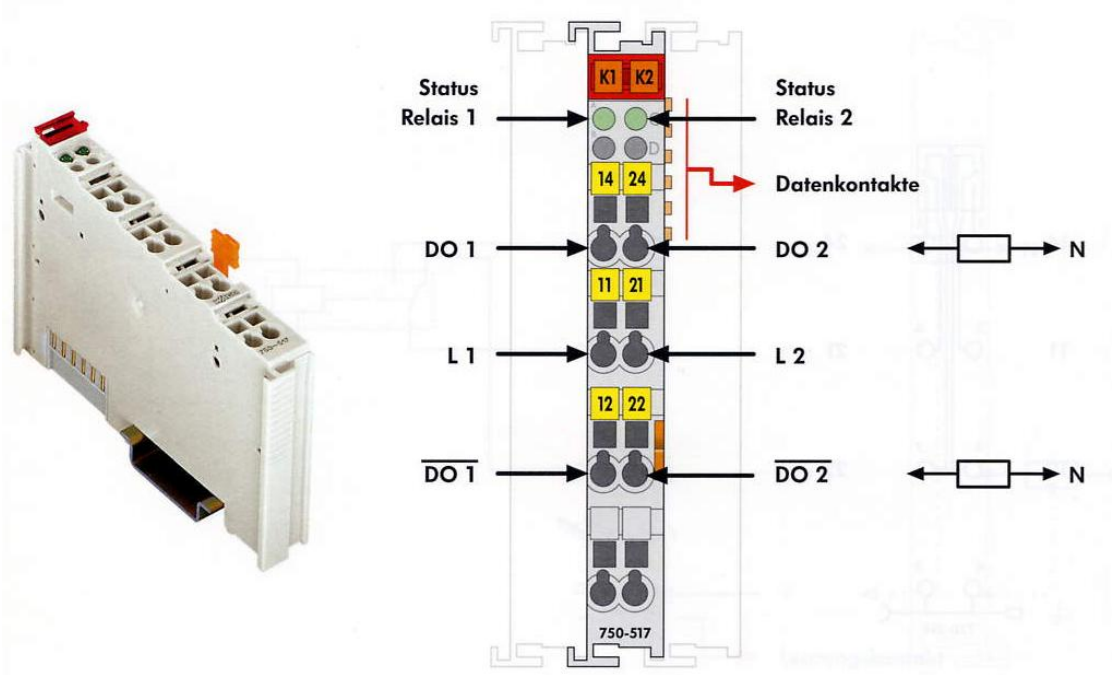
Über die digitalen Ausgangsklemmen werden Steuersignale aus dem Automatisierungsgerät an die angeschlossenen Aktoren weitergegeben.



Wichtige Technische Daten:

Anzahl der Ausgänge	2
Stromaufnahme (intern)	15 mA
Spannung über Leistungskontakte	DC 24 V (-15% ... +20%)
Lastart	Ohmsch, induktiv, Lampenlast
Ausgangsstrom	0,5 A
Kurzschlussbegrenzung	1,5 A
Diagnose	Leerlauf, Kurzschluss, Überlast
Stromaufnahme (Feldseite)	15 mA + Last
Datenbreite intern	4 Bit In, 4 Bit Out
Betriebstemperatur	0 °C … +55 °C

2-Kanal Relaisausgangsklemme AC 230 V: 750-517

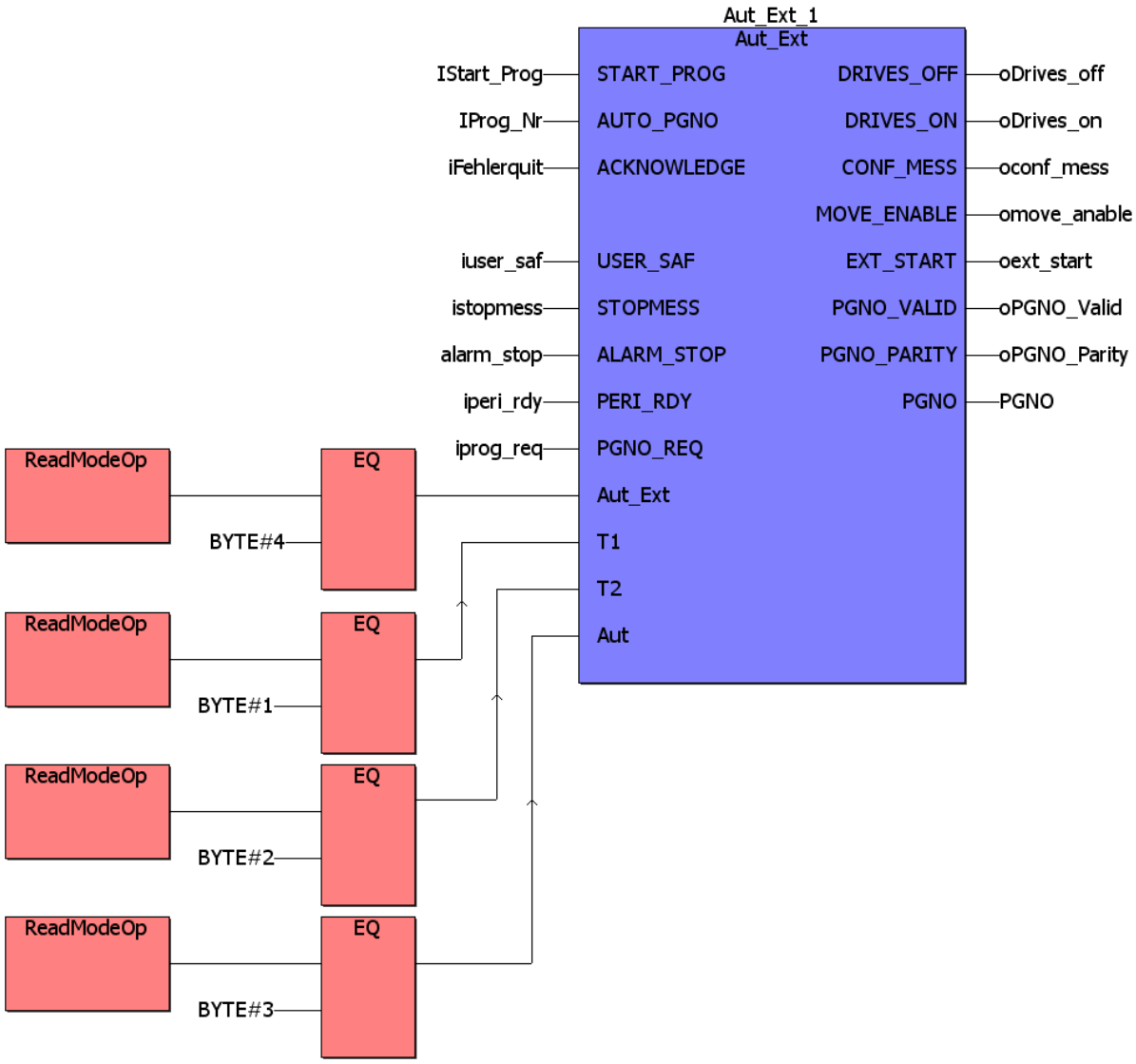


Wichtige Technische Daten:

Anzahl der Ausgänge	2 Wechsler
Stromaufnahme $_{max.}$ (intern)	105 mA
Schaltspannung $_{max.}$	AC 150 V/DC 300V
Schaltstrom $_{max.}$	AC 1A, DC 1A bei DC 40V, DC 0,15 A bei DC 300V
Schaltstrom $_{min.}$	AC/DC 10 mA bei 5V
Schaltfrequenz $_{max.}$	6/min
Datenbreite intern	2 Bit
Betriebstemperatur	0 °C … +55 °C

8.7 Quellcode der KUKA Soft-SPS Programmierung

Quellcode für „Automatik Extern“:



Lokale Variablendeklaration für „Automatik Extern“:

```
VAR
Aut_Ext_1      :      Aut_Ext;
iprog_req      :      BOOL;
iperi_rdy      :      BOOL;
alarm_stop     :      BOOL;
istopmess      :      BOOL;
iuser_saf      :      BOOL;
IProg_NR       :      INT;
IStart_Prog    :      BOOL;
oDrives_off    :      BOOL;
oDrives_on     :      BOOL;
oconf_mess     :      BOOL;
omove_anable:         BOOL;
```

```
oext_start      :       BOOL;
oPGNO_Valid:    BOOL;
oPGNO_Parity:   BOOL;
PGNO            :       BYTE;
iFehlerquit     :       BOOL;
END_VAR
```

Codeblatt für „Datenaustausch“

```
(* real Variable *)
WriteSenReal_1 (Index := BYTE #1,Value := x_wert);
WriteSenReal_2 (Index :=BYTE #2,Value := y_wert);
WriteSenReal_3 (Index :=BYTE #3,Value := Rvar3);
WriteSenReal_4 (Index :=BYTE #4,Value := Rvar4);
WriteSenReal_5 (Index :=BYTE #5, Value := Rvar5);
WriteSenReal_6 (Index :=BYTE #6, Value := Rvar6);
WriteSenReal_7 (Index := BYTE #7, Value := Rvar7);
WriteSenReal_8 (Index := BYTE #8, Value := Rvar8);
WriteSenReal_9 (Index := BYTE #9, Value := Rvar9);
WriteSenReal_10 (Index := BYTE #10, Value := Rvar10);
WriteSenReal_11 (Index := BYTE #11, Value := Rvar11);
WriteSenReal_12 (Index := BYTE #12, Value := Rvar12);
WriteSenReal_13 (Index := BYTE #13, Value := Rvar13);
WriteSenReal_14 (Index := BYTE #14, Value := Rvar14);
WriteSenReal_15 (Index := BYTE #15, Value := Rvar15);
WriteSenReal_16 (Index := BYTE #16, Value := Rvar16);
WriteSenReal_17 (Index := BYTE #17, Value := Rvar17);
WriteSenReal_18 (Index := BYTE #18, Value := Rvar18);
WriteSenReal_19 (Index := BYTE #19, Value := Rvar19);
WriteSenReal_20 (Index := BYTE #20, Value := Rvar20);

(* Integer Variable *)
WriteSenInt_1 (Index := BYTE #1, Value := DIvar1);
WriteSenInt_2 (Index := BYTE #2, Value := DIvar2);
WriteSenInt_3 (Index := BYTE #3, Value := DIvar3);
WriteSenInt_4 (Index := BYTE #4, Value := DIvar4);
WriteSenInt_5 (Index := BYTE #5, Value := DIvar5);
WriteSenInt_6 (Index := BYTE #6, Value := DIvar6);
WriteSenInt_7 (Index := BYTE #7, Value := DIvar7);
WriteSenInt_8 (Index := BYTE #8, Value := DIvar8);
WriteSenInt_9 (Index := BYTE #9, Value := DIvar9);
WriteSenInt_10 (Index := BYTE #10, Value := DIvar10);
WriteSenInt_11 (Index := BYTE #11, Value := DIvar11);
WriteSenInt_12 (Index := BYTE #12, Value := DIvar12);
WriteSenInt_13 (Index := BYTE #13, Value := DIvar13);
WriteSenInt_14 (Index := BYTE #14, Value := DIvar14);
WriteSenInt_15 (Index := BYTE #15, Value := DIvar15);
WriteSenInt_16 (Index := BYTE #16, Value := DIvar16);
WriteSenInt_17 (Index := BYTE #17, Value := DIvar17);
WriteSenInt_18 (Index := BYTE #18, Value := DIvar18);
WriteSenInt_19 (Index := BYTE #19, Value := DIvar19);
WriteSenInt_20 (Index := BYTE #20, Value := DIvar20);

(* Write/read the KRC1 variable $OV_PRO representing the reduction of current velocity in percent *)
curVelocity:=ReadOvPro();
WriteOvPro_1(OvPro:=VelocityToBe);

(* read the KRC1 variable $MODE_OP containing the current operating mode. *)
Rob_Mode:=ReadModeOp();

(* Read $PPOS_ACT representing the current position actual value of the robot trajectory *)
```

```
ReadPosAct_1();
Pos_Valid:=ReadPosAct_1.bValid;
Pos_X:=ReadPosAct_1.X;
Pos_Y:=ReadPosAct_1.Y;
Pos_Z:=ReadPosAct_1.Z;
Pos_A:=ReadPosAct_1.A;
Pos_B:=ReadPosAct_1.B;
Pos_C:=ReadPosAct_1.C;
```

Globale Variablen:

```
VAR_GLOBAL
cycle_count                 :  INT;

 (*Variable for Automatik Extern*)
oDrives_off     AT %QX20.0  :  BOOL;      (* $OUT[481] *)
oDrives_on      AT %QX20.1  :  BOOL;      (* $OUT[482] *)
oconf_mess      AT %QX20.2  :  BOOL;      (* $OUT[483] *)
omove_anable    AT %QX20.3  :  BOOL;      (* $OUT[484] *)
oext_start      AT %QX20.4  :  BOOL;      (* $OUT[485] *)
oPGNO_Valid     AT %QX20.5  :  BOOL;      (* $OUT[486] *)
oPGNO_Parity    AT %QX20.6  :  BOOL;      (* $OUT[487] *)
PGNO            AT %QB21    :  BYTE;      (*$OUT[489]~$OUT[496]*)
IProg_NR                    :  INT    {CSV};
IStart_Prog                 :  BOOL   {CSV};
iFehlerquit                 :  BOOL   {CSV};
iuser_saf       AT %IX20.0  :  BOOL;      (* $IN[481] *)
istopmess       AT %IX20.1  :  BOOL;      (* $IN[482] *)
alarm_stop      AT %IX20.2  :  BOOL;      (* $IN[483] *)
iperi_rdy       AT %IX20.3  :  BOOL;      (* $IN[484] *)
iprog_req       AT %IX20.4  :  BOOL;      (* $IN[485] *)

(* Variable to be writen *)
x_wert   : REAL  {CSV};
y_wert   : REAL  {CSV};
Rvar3    : REAL  {CSV};
Rvar4    : REAL  {CSV};
Rvar5    : REAL  {CSV};
Rvar6    : REAL  {CSV};
Rvar7    : REAL  {CSV};
Rvar8    : REAL  {CSV};
Rvar9    : REAL  {CSV};
Rvar10   : REAL  {CSV};
Rvar11   : REAL  {CSV};
Rvar12   : REAL  {CSV};
Rvar13   : REAL  {CSV};
Rvar14   : REAL  {CSV};
Rvar15   : REAL  {CSV};
Rvar16   : REAL  {CSV};
Rvar17   : REAL  {CSV};
Rvar18   : REAL  {CSV};
Rvar19   : REAL  {CSV};
Rvar20   : REAL  {CSV};
DIvar1   : DINT  {CSV};
DIvar2   : DINT  {CSV};
DIvar3   : DINT  {CSV};
DIvar4   : DINT  {CSV};
DIvar5   : DINT  {CSV};
DIvar6   : DINT  {CSV};
DIvar7   : DINT  {CSV};
DIvar8   : DINT  {CSV};
```

```
DIvar9   : DINT  {CSV};
DIvar10 : DINT  {CSV};
DIvar11 : DINT  {CSV};
DIvar12 : DINT  {CSV};
DIvar13 : DINT  {CSV};
DIvar14 : DINT  {CSV};
DIvar15 : DINT  {CSV};
DIvar16 : DINT  {CSV};
DIvar17 : DINT  {CSV};
DIvar18 : DINT  {CSV};
DIvar19 : DINT  {CSV};
DIvar20 : DINT  {CSV};

curVelocity      : BYTE  {CSV};
VelocityToBe     : BYTE  {CSV};

(* Robot operating mode *)
Rob_Mode         : BYTE  {CSV};

(* actual position value of the trajectory *)
Pos_Valid        : BOOL  {CSV};
Pos_X            : REAL  {CSV};
Pos_Y            : REAL  {CSV};
Pos_Z            : REAL  {CSV};
Pos_A            : REAL  {CSV};
Pos_B            : REAL  {CSV};
Pos_C            : REAL  {CSV};

(*Boolsche Variablen für den OPC-Server*)
(* Signal ob ein Werkzeug auf dem Roboter ist.*)
ToolOn  AT %IX0.1  : BOOL   {CSV}; (* $IN[322] *)

(*Signale für die 10 Fräsern ob sie in Position sind. *)
Tool1    AT %IX1.0  : BOOL   {CSV}; (* $IN[329] *)
Tool2    AT %IX1.1  : BOOL   {CSV}; (* $IN[330] *)
Tool3    AT %IX1.2  : BOOL   {CSV}; (* $IN[331] *)
Tool4    AT %IX1.3  : BOOL   {CSV}; (* $IN[332] *)
Tool5    AT %IX1.4  : BOOL   {CSV}; (* $IN[333] *)
Tool6    AT %IX1.5  : BOOL   {CSV}; (* $IN[334] *)
Tool7    AT %IX1.6  : BOOL   {CSV}; (* $IN[335] *)
Tool8    AT %IX1.7  : BOOL   {CSV}; (* $IN[336] *)
Tool9    AT %IX2.0  : BOOL   {CSV}; (* $IN[337] *)
Tool10   AT %IX2.1  : BOOL   {CSV}; (* $IN[338] *)

(* Signale, ob der Spindel order die Säge ist auf der Ablage *)
SpindelFrei      AT %IX2.4  : BOOL   {CSV}; (* $IN[341] *)
SaegeFrei        AT %IX2.7  : BOOL   {CSV}; (* $IN[344] *)
Rob_Busy         AT %IX10.0 : BOOL   {CSV}; (* $IN[401] *)
Vert_steckdose   AT %QX10.0 : BOOL   {CSV}; (* $OUT[401]*)
END_VAR
```

I/O-Konfiguration:

```
PROGRAM o_iKRC1 : OUTPUT
(
  VAR_ADR := 0,                  (*Start adress ProConOS output byte 0*)
  END_VAR_ADR := 24,             (*End adress ProConOS output byte 25, then 200 output bits *)
  DEVICE := DRIVER,
  DRIVER_NAME := 'KRCIODRV',
  DATA_TYPE := BYTE,
  DRIVER_PAR1 := 40,             (*Start adress KRC output byte 40($OUT[321])*)
```

```
  DRIVER_PAR2 := 1              (*Type of I/O's 0 -> Peripheral I/O's*)
);

PROGRAM i_oKRC1 : INPUT
(
  VAR_ADR := 0,                 (*Start adress ProConOS input byte 0*)
  END_VAR_ADR := 24,            (*End adress ProConOS input byte, then 200
                                  input bits of ProConOs*)
  DEVICE := DRIVER,
  DRIVER_NAME := 'KRCIODRV',
  DATA_TYPE := BYTE,
  DRIVER_PAR1 := 40,            (*Start adress KRC input byte 40($IN[321])*)
  DRIVER_PAR2 := 1              (*Type of I/O's 0 -> Peripheral I/O's*)
);
```

8.8 Quellcode des BauXML-Schemas

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 rel. 3 sp2 (http://www.altova.com) by (FZK IAI) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:bau="urn:bauxml" targetNamespace="urn:bauxml" elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:element name="Adress" type="bau:AddressType"/>
<xs:complexType name="AddressType">
	<xs:all>
		<xs:element name="Name" type="xs:string"/>
		<xs:element name="Street" type="xs:string"/>
		<xs:element name="Postcode" type="xs:string"/>
		<xs:element name="city" type="xs:string"/>
		<xs:element name="Country" type="xs:string"/>
	</xs:all>
</xs:complexType>
<xs:simpleType name="AngleUnitType">
	<xs:restriction base="xs:string">
		<xs:enumeration value="deg"/>
		<xs:enumeration value="rad"/>
	</xs:restriction>
</xs:simpleType>
<xs:element name="BaseShape" type="bau:BaseShapeType"/>
<xs:complexType name="BaseShapeType">
	<xs:choice>
		<xs:element name="BlockBaseShape" type="bau:BlockBaseShapeType"/>
		<xs:element name="CylindricalBaseShape" type="bau:CylindricalBaseShapeType"/>
		<xs:element name="NgonBaseShape" type="bau:NgonBaseShapeType"/>
	</xs:choice>
</xs:complexType>
<xs:complexType name="BlockBaseShapeType">
	<xs:attribute name="Length" type="xs:double" use="required"/>
	<xs:attribute name="Width" type="xs:double" use="required"/>
	<xs:attribute name="Height" type="xs:double" use="required"/>
</xs:complexType>
<xs:complexType name="BuildingInfoType">
	<xs:all>
		<xs:element name="Address" type="bau:AddressType"/>
		<xs:element name="Description" type="xs:string" minOccurs="0"/>
	</xs:all>
</xs:complexType>
<xs:element name="Building" type="bau:BuildingType"/>
<xs:complexType name="BuildingType">
	<xs:sequence>
		<xs:element name="BuildingInfo" type="bau:BuildingInfoType" minOccurs="0"/>
```

```
                <xs:element name="Storey" type="bau:StoreyType" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="Guid" type="bau:GuidType" use="required"/>
</xs:complexType>
<xs:complexType name="CartesianPoint3DType">
        <xs:attribute name="X" type="xs:double" use="required"/>
        <xs:attribute name="Y" type="xs:double" use="required"/>
        <xs:attribute name="Z" type="xs:double" use="required"/>
</xs:complexType>
<xs:element name="CircleProfile" type="bau:CircleProfileType" substitutionGroup="bau:Profile"/>
<xs:complexType name="CircleProfileType">
        <xs:complexContent>
                <xs:extension base="bau:ProfileType">
                        <xs:sequence>
                                <xs:element name="Radius" type="xs:double"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
<xs:element name="CircleSectorProfile" type="bau:CircleSectorType" substitutionGroup="bau:Profile"/>
<xs:complexType name="CircleSectorType">
        <xs:complexContent>
                <xs:extension base="bau:CircleProfileType">
                        <xs:sequence>
                                <xs:element name="TiltAngle" type="xs:double"/>
                                <xs:element name="ProfileAngle" type="xs:double"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
<xs:element name="CircleSegmentProfile" type="bau:CircleSegmentType" substitutionGroup="bau:Profile"/>
<xs:complexType name="CircleSegmentType">
        <xs:complexContent>
                <xs:extension base="bau:CircleProfileType">
                        <xs:sequence>
                                <xs:element name="TiltAngle" type="xs:double"/>
                                <xs:element name="ProfileAngle" type="xs:double"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
<xs:element name="Cone" type="bau:ConeType"/>
<xs:complexType name="ConeType">
        <xs:sequence>
                <xs:element ref="bau:Placement"/>
                <xs:element name="Depth" type="xs:double"/>
                <xs:element name="BeginRadius" type="xs:double"/>
                <xs:element name="EndRadius" type="xs:double"/>
        </xs:sequence>
</xs:complexType>
<xs:complexType name="CylindricalBaseShapeType">
        <xs:attribute name="Diameter" type="xs:double" use="required"/>
        <xs:attribute name="Height" type="xs:double" use="required"/>
</xs:complexType>
<xs:complexType name="DirectionType">
        <xs:attribute name="ratioX" type="xs:double" use="required"/>
        <xs:attribute name="ratioY" type="xs:double" use="required"/>
        <xs:attribute name="ratioZ" type="xs:double" use="required"/>
</xs:complexType>
<xs:element name="EllipseProfile" type="bau:EllipseProfileType" substitutionGroup="bau:Profile"/>
<xs:complexType name="EllipseProfileType">
```

```
        <xs:complexContent>
            <xs:extension base="bau:ProfileType">
                <xs:sequence>
                    <xs:element name="SemiAxis1" type="xs:double"/>
                    <xs:element name="SemiAixs2" type="xs:double"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
</xs:complexType>
<xs:element name="ExtrusionSolid" type="bau:ExtrusionSolidType"/>
<xs:complexType name="ExtrusionSolidType">
        <xs:sequence>
            <xs:choice>
                <xs:element ref="bau:GeneralProfile"/>
                <xs:element ref="bau:RectangleProfile"/>
                <xs:element ref="bau:CircleProfile"/>
                <xs:element ref="bau:CircleSegmentProfile"/>
                <xs:element ref="bau:CircleSectorProfile"/>
                <xs:element ref="bau:EllipseProfile"/>
                <xs:element ref="bau:NgonProfile"/>
                <xs:element ref="bau:RingProfile"/>
                <xs:element ref="bau:RoundedUProfile"/>
                <xs:element ref="bau:SquareUProfile"/>
                <xs:element ref="bau:VeeProfile"/>
            </xs:choice>
            <xs:element name="ExtrusionPath" type="bau:VectorType"/>
        </xs:sequence>
</xs:complexType>
<xs:element name="Feature" type="bau:FeatureType"/>
<xs:complexType name="FeatureType">
        <xs:sequence>
            <xs:element name="Description" type="xs:string" minOccurs="0"/>
            <xs:choice>
                <xs:element name="ExtrusionSolid" type="bau:ExtrusionSolidType"/>
                <xs:element name="Cone" type="bau:ConeType"/>
                <xs:element name="Plane" type="bau:PlaneType"/>
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="IsFabricated" type="xs:boolean" use="required"/>
        <xs:attribute name="Guid" type="bau:GuidType" use="required"/>
</xs:complexType>
<xs:element name="GeneralProfile" type="bau:GeneralProfileType" substitutionGroup="bau:Profile"/>
<xs:complexType name="GeneralProfileType">
        <xs:complexContent>
            <xs:extension base="bau:ProfileType">
                <xs:sequence>
                    <xs:element name="Point" type="bau:CartesianPoint3DType"
                    minOccurs="3" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
</xs:complexType>
<xs:element name="Guid" type="bau:GuidType"/>
<xs:simpleType name="GuidType">
        <xs:restriction base="xs:normalizedString">
            <xs:minLength value="22"/>
            <xs:maxLength value="22"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="LengthUnitType">
        <xs:restriction base="xs:string">
```

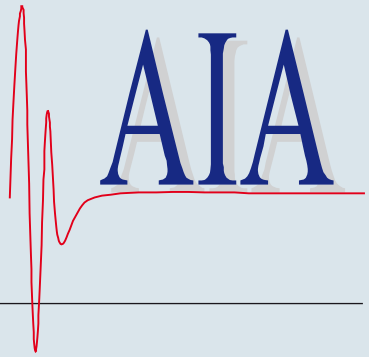
```
                <xs:enumeration value="mm"/>
                <xs:enumeration value="cm"/>
                <xs:enumeration value="m"/>
                <xs:enumeration value="in"/>
        </xs:restriction>
</xs:simpleType>
<xs:complexType name="NgonBaseShapeType">
        <xs:attribute name="IsCircumscribed" type="xs:boolean" use="required"/>
        <xs:attribute name="NumberOfSides" type="xs:positiveInteger" use="required"/>
        <xs:attribute name="Diameter" type="xs:double" use="required"/>
        <xs:attribute name="Height" type="xs:double" use="required"/>
</xs:complexType>
<xs:element name="NgonProfile" type="bau:NgonProfileType" substitutionGroup="bau:Profile"/>
<xs:complexType name="NgonProfileType">
        <xs:complexContent>
                <xs:extension base="bau:ProfileType">
                        <xs:sequence>
                                <xs:element name="IsCircumscribed" type="xs:boolean" default="true"/>
                                <xs:element name="Diameter" type="xs:double"/>
                                <xs:element name="NumberOfSides" type="xs:positiveInteger"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
<xs:element name="OrderInfo" type="bau:OrderInfoType"/>
<xs:complexType name="OrderInfoType">
        <xs:all>
                <xs:element name="OrderNumber" type="xs:string"/>
                <xs:element name="InitiatedDate" type="xs:string"/>
                <xs:element name="DeliveryDate" type="xs:string"/>
                <xs:element name="QuantityOrdered" type="xs:nonNegativeInteger"/>
                <xs:element name="OrderStatus" type="bau:OrderStatus"/>
                <xs:element name="Creator" type="xs:string"/>
        </xs:all>
</xs:complexType>
<xs:simpleType name="OrderStatus">
        <xs:restriction base="xs:string">
                <xs:enumeration value="Created not yet manufactured"/>
                <xs:enumeration value="In Manufacturing"/>
                <xs:enumeration value="Manufactured not yet transpported"/>
                <xs:enumeration value="Transported"/>
        </xs:restriction>
</xs:simpleType>
<xs:element name="Placement" type="bau:PlacementType"/>
<xs:complexType name="PlacementType">
        <xs:sequence>
                <xs:element name="Location" type="bau:CartesianPoint3DType"/>
                <xs:element name="Axis" type="bau:DirectionType"/>
                <xs:element name="RefAxis" type="bau:DirectionType"/>
        </xs:sequence>
</xs:complexType>
<xs:element name="Plane" type="bau:PlaneType"/>
<xs:complexType name="PlaneType">
        <xs:sequence>
                <xs:element name="Point" type="bau:CartesianPoint3DType"/>
                <xs:element name="Normal" type="bau:DirectionType"/>
                <xs:element name="RefDirection" type="bau:DirectionType"/>
                <xs:element name="Border" minOccurs="0">
                        <xs:complexType>
                                <xs:sequence>
                                        <xs:element name="Point" type="bau:CartesianPoint3DType"
```

```
                                        minOccurs="3" maxOccurs="unbounded"/>
                                </xs:sequence>
                        </xs:complexType>
                </xs:element>
        </xs:sequence>
</xs:complexType>
<xs:element name="Profile" type="bau:ProfileType"/>
<xs:complexType name="ProfileType">
        <xs:sequence>
                <xs:element ref="bau:Placement"/>
                <xs:element name="IsArea" type="xs:boolean" default="true"/>
        </xs:sequence>
</xs:complexType>
<xs:element name="Project" type="bau:ProjectType"/>
<xs:complexType name="ProjectType">
        <xs:all>
                <xs:element name="Guid" type="bau:GuidType"/>
                <xs:element name="Description" type="xs:string" minOccurs="0"/>
                <xs:element name="CustomerInfo" type="bau:AddressType" minOccurs="0"/>
                <xs:element name="OrderInfo" type="bau:OrderInfoType" minOccurs="0"/>
                <xs:element name="UnitInfo" type="bau:UnitInfoType"/>
                <xs:element name="Building" type="bau:BuildingType"/>
                <xs:element name="SegmentSettings" type="bau:SegmentSettingsType" minOccurs="0"/>
        </xs:all>
</xs:complexType>
<xs:element name="RectangleProfile" type="bau:RectangleProfileType" substitutionGroup="bau:Profile"/>
<xs:complexType name="RectangleProfileType">
        <xs:complexContent>
                <xs:extension base="bau:ProfileType">
                        <xs:sequence>
                                <xs:element name="Length" type="xs:double"/>
                                <xs:element name="Width" type="xs:double"/>
                        </xs:sequence>
                </xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:element name="RingProfile" type="bau:RingProfileType" substitutionGroup="bau:Profile"/>
<xs:complexType name="RingProfileType">
        <xs:complexContent>
                <xs:extension base="bau:ProfileType">
                        <xs:sequence>
                                <xs:element name="TiltAngle" type="xs:double"/>
                                <xs:element name="ProfileAngle" type="xs:double"/>
                                <xs:element name="LargeRadius" type="xs:double"/>
                                <xs:element name="SmallRadius" type="xs:double"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
<xs:element name="RoundedUProfile" type="bau:RoundedUProfileType" substitutionGroup="bau:Profile"/>
<xs:complexType name="RoundedUProfileType">
        <xs:complexContent>
                <xs:extension base="bau:ProfileType">
                        <xs:sequence>
                                <xs:element name="Width" type="xs:double"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
<xs:complexType name="SegmentSettingsType">
        <xs:all>
```

```
                <xs:element name="SegMin" type="xs:double"/>
                <xs:element name="SegMax" type="xs:double"/>
                <xs:element name="LintelHeight" type="xs:double" minOccurs="0"/>
                <xs:element name="BearingLength" type="xs:double" minOccurs="0"/>
        </xs:all>
</xs:complexType>
<xs:element name="SquareUProfile" type="bau:SquareUProfileType" substitutionGroup="bau:Profile"/>
<xs:complexType name="SquareUProfileType">
        <xs:complexContent>
                <xs:extension base="bau:ProfileType">
                        <xs:sequence>
                                <xs:element name="BaseWidth" type="xs:double"/>
                                <xs:element name="FirstAngle" type="xs:double"/>
                                <xs:element name="SecondAngle" type="xs:double"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
<xs:complexType name="StoneType">
        <xs:sequence>
                <xs:element name="Description" type="xs:string" minOccurs="0"/>
                <xs:element name="LocalPlacement" type="bau:PlacementType"/>
                <xs:element name="BaseShape" type="bau:BaseShapeType"/>
                <xs:element name="Feature" type="bau:FeatureType" minOccurs="0"
                maxOcurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="ID" type="xs:string" use="required"/>
        <xs:attribute name="Material" type="xs:string" use="required"/>
        <xs:attribute name="IsFabricated" type="xs:boolean" use="optional" default="false"/>
        <xs:attribute name="Guid" type="bau:GuidType" use="required"/>
</xs:complexType>
<xs:complexType name="StoreyType">
        <xs:sequence>
                <xs:element name="Description" type="xs:string" minOccurs="0"/>
                <xs:element name="Wall" type="bau:WallType" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="ID" type="xs:string" use="required"/>
        <xs:attribute name="IFCOID" type="xs:unsignedLong" use="optional"/>
        <xs:attribute name="Guid" type="bau:GuidType" use="required"/>
</xs:complexType>
<xs:complexType name="UnitInfoType">
        <xs:all>
                <xs:element name="LengthUnit" type="bau:LengthUnitType"/>
                <xs:element name="AngleUnit" type="bau:AngleUnitType"/>
        </xs:all>
</xs:complexType>
<xs:complexType name="VectorType">
        <xs:sequence>
                <xs:element name="Orientatition" type="bau:DirectionType"/>
                <xs:element name="Magnitude" type="xs:double"/>
        </xs:sequence>
</xs:complexType>
<xs:element name="VeeProfile" type="bau:VeeProfileType" substitutionGroup="bau:Profile"/>
<xs:complexType name="VeeProfileType">
        <xs:complexContent>
                <xs:extension base="bau:ProfileType">
                        <xs:sequence>
                                <xs:element name="TiltAngle" type="xs:double"/>
                                <xs:element name="ProfileAngle" type="xs:double"/>
                        </xs:sequence>
```

```
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
<xs:complexType name="WallType">
        <xs:sequence>
                <xs:element name="Description" type="xs:string" minOccurs="0"/>
                <xs:element name="GlobalPlacement" type="bau:PlacementType"/>
                <xs:element name="Stone" type="bau:StoneType" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="ID" type="xs:string" use="required"/>
        <xs:attribute name="IFCOID" type="xs:unsignedLong" use="optional"/>
        <xs:attribute name="Length" type="xs:double" use="required"/>
        <xs:attribute name="Width" type="xs:double" use="required"/>
        <xs:attribute name="Height" type="xs:double" use="required"/>
        <xs:attribute name="Guid" type="bau:GuidType" use="required"/>
</xs:complexType>
/xs:schema>
```

Institut für Angewandte Informatik /
Automatisierungstechnik
Universität Karlsruhe (TH)



ISSN: 1614-5267
ISBN: 978-3-86644-114-9

www.uvka.de