**Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH**

Document
D-90-04

STEP

Überblick über eine zukünftige
Schnittstelle zum
Produktdatenaustausch

Ansgar Bernardi, Christoph Klauck,
Ralf Legleitner

September 1990

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

Das Deutsche Forschungszentrum für Künstliche Intelligenz (DFKI) mit Standorten in Kaiserslautern und Saarbrücken ist eine gemeinnützige GmbH, die 1988 von den Gesellschaftern ADV/Orga, AEG, IBM, Insiders, Fraunhofer Gesellschaft, GMD, Krupp-Atlas, Mannesmann-Kienzle, Nixdorf, Philips und Siemens gegründet wurde. Die am DFKI durchgeführten Projekte werden vom Bundesministerium für Forschung und Technologie, von den Gesellschaftern oder von externen Auftraggebern finanziert.

Am DFKI wird anwendungsorientierte Grundlagenforschung auf dem Gebiet der Künstlichen Intelligenz (KI) und angrenzender anderer Bereiche der Informatik durchgeführt. Ziel ist die Erstellung von "Intelligenten Fachsystemen", in denen ausgesuchte Probleme eines Anwendungsbereiches mit Hilfe von KI Methoden gelöst werden. Derzeitig werden am DFKI die folgenden Forschungsschwerpunkte verfolgt:

- Intelligente Ingenieur-Systeme
- Intelligente Benutzerschnittstellen
- Intelligente Kommunikations-Netzwerke
- Intelligente kooperative Systeme

Das DFKI strebt danach, seine wissenschaftlichen Ergebnisse der Forschungsgemeinschaft zugänglich zu machen. Aus diesem Grunde wurde ein Netzwerk von Kontakten zu in- und ausländischen Forschungseinrichtungen im akademischen und industriellen Bereich aufgebaut. Darüber hinaus veranstaltet das DFKI Wissenstransfer-Workshops, auf denen Gesellschafter und interessierte Dritte über den Forschungsstand informiert werden.

Seit seiner Gründung stellt das DFKI eine attraktive Arbeitsumgebung für namhafte Forscherinnen und Forscher aus dem In- und Ausland dar. Das Ziel ist, nach Beendigung der Aufbauphase ca. 100 Wissenschaftler zu beschäftigen.

Prof. Dr. Gerhard Barth
Technisch-wissenschaftlicher Geschäftsführer

STEP: Überblick über eine zukünftige Schnittstelle zum Produktdatenaustausch

Ansgar Bernardi, Christoph Klauck, Ralf Legleitner

DFKI-D-90-04

STEP

Überblick über eine zukünftige Schnittstelle zum Produktdatenaustausch

Ansgar Bernardi
Christoph Klauck
Ralf Legleitner

E-Mail:
bernardi@mail.dfki.uni-kl.de
klauck@mail.dfki.uni-kl.de
legleitner@mail.dfki.uni-kl.de

Inhalt

Zusammenfassung

STEP (*St*andard for the *E*xchange of *P*roduct Model Data) ist ein von der ISO entwickeltes Standardformat zur Abbildung produktdefinierender Daten (ISO TC 184/SC 4, NAM 96.4) im Gesamtkomplex der CIM-Techniken (*C*omputer *I*ntegrated *M*anufacturing), der 1993 weltweiter Standard werden soll. In diesem Bericht wird ein Überblick über den derzeitigen Entwicklungsstand von STEP gegeben. Dabei werden die bereits weitgehend stabilen Teile detailliert beschrieben.

1 Einleitung

Im Bereich der rechnerintegrierten Produktion (CIM) stellen CAD-Systeme (*C*omputer *A*ided *D*esign) Informationen über technische Produkte bereit, die von anderen Rechnersystemen übernommen und weiterverarbeitet werden können (CAP, CAM, ...). Die Notwendigkeit zur datentechnischen Verknüpfung der hauptsächlich als Insellösungen beteiligten Systeme ist unumstritten. Die unter dem Stichwort CIM-Schnittstellen zusammengefaßte Problematik soll durch standardisierte Schnittstellen gelöst werden. Einen Schwerpunkt bildet dabei die Entwicklung von STEP.

STEP spezifiziert ein Standardformat zur Abbildung produktdefinierender Daten. Alle während des Lebenszyklus eines Industrieproduktes auftretenden Daten sollen abgebildet werden (Vollständigkeit), um verschiedene CA-Systeme verbinden zu können (Kompatibilität) und als Formatdefinition zur Datenarchivierung zu dienen (Archivierungsfähigkeit). Desweiteren soll STEP

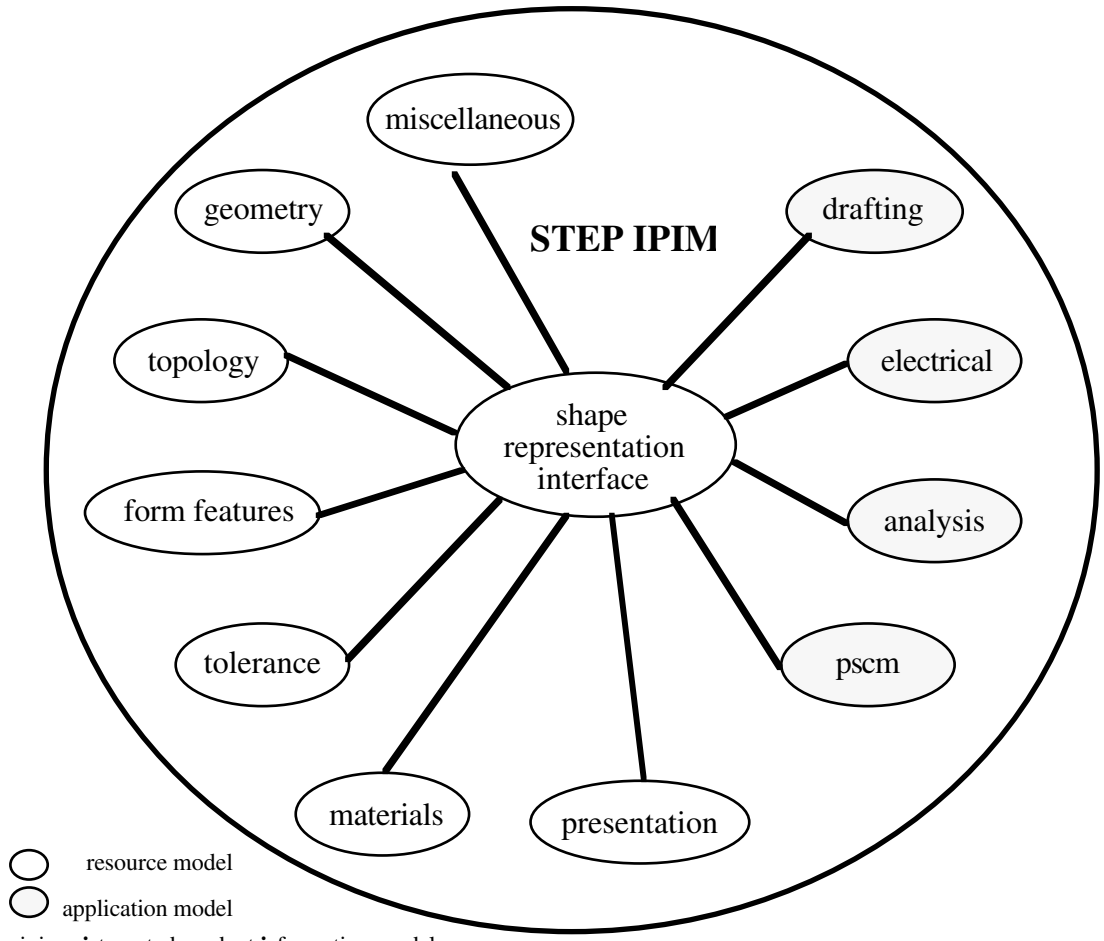
- erweiterbar sein,
- Pre- und Postprozessoren in Effizienz unterstützen,
- ein Minimum an Informationseinheiten (Entities) bieten (Redundanzfrei),
- unabhängig von der Rechnerumgebung sein,
- von der logischen Datenstruktur und dem physikalischen Speicherformat entkoppelt sein und
- fähig zur Bildung von anwendungsspezifischen Leistungsstufen sein.

Dies bedeutet, daß neben der Geometrie auch technologische Eigenschaften, die funktions- oder baugruppenbezogen sind, wie Materialeigenschaften, Oberflächengüte und Toleranzen, übertragen werden müssen. Um diesen Ansprüchen genügen zu können, definiert STEP Datenformate zur Beschreibung sehr unterschiedlicher Aspekte eines

Produkts. Die einzelnen Sichtweisen werden in den sogenannten *Partialmodellen* dargestellt. Deren Kombination ergibt dann das integrierte Produktmodell IPIM (*I*ntegrated *P*roduct *I*nformation *M*odel), in dem ein Produkt vollständig repräsentiert werden kann.

Die Konzeption des STEP-IPIM als Datenformat-Definition entspricht dem klassischen Schnittstellen-Ansatz: Unterschiedliche Systeme nutzen ein festgelegtes Datenformat zur Kommunikation, die Interpretation der erhaltenen Daten obliegt dem jeweiligen System. Die Semantik der im STEP-IPIM syntaktisch repräsentierten Daten kann selbst nicht in STEP übertragen werden; sie ist Gegenstand der in den STEP-Dokumentationen festgelegten Übereinkünfte.

Die folgende Abbildung soll das Zusammenspiel der einzelnen Partialmodelle zum IPIM veranschaulichen:



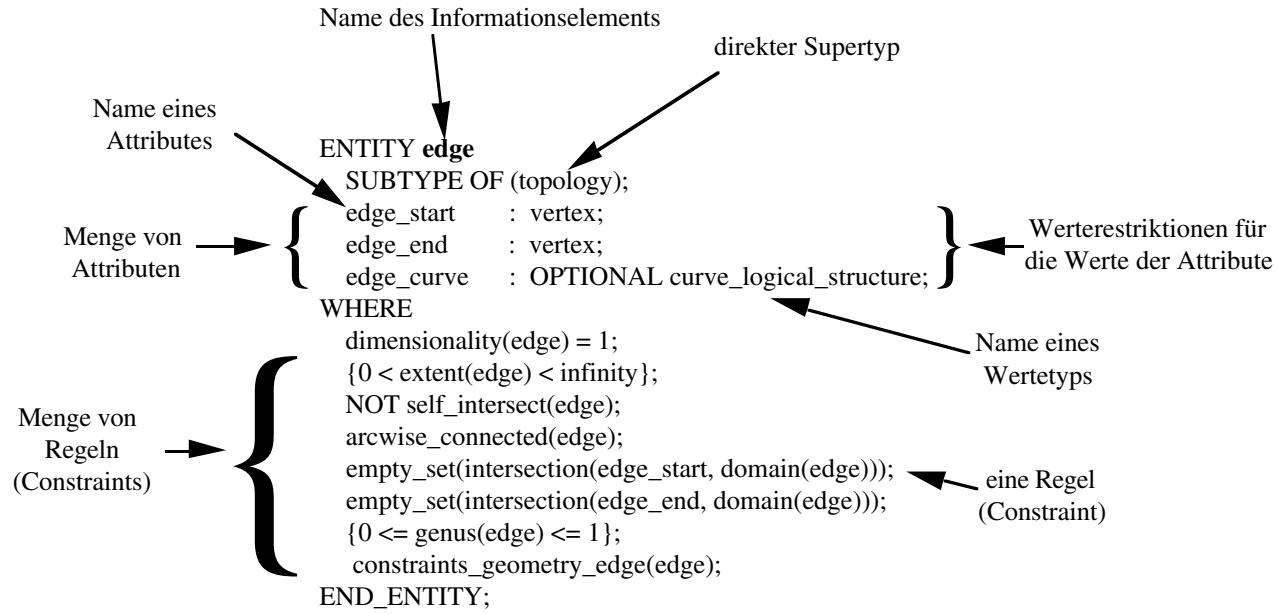
ipim : **i**ntegrated **p**roduct **i**nformation **m**odel
pscm : **p**roduct **s**tructure **c**onfiguration **m**anagement
step : **st**andard for the **e**xchange of **p**roduct model data

Das Shape Representation Interface bildet das Herzstück des IPIM und ist eine allgemeine Definition der Produktgestalt (Shape), an die spezielle Informationen, wie etwa Materialinformationen, geknüpft werden können. Die notwendigen Daten sind in Informationseinheiten, den sogenannten Entities, zerlegt, nach logischen Kriterien klassifiziert und entsprechenden Partialmodellen zugeordnet. Diese Partialmodelle gliedern sich in *Resource Models,* die der allgemeinen Produktdefinition dienen (anwendungsneutrale Informationen), und in *Application Models*, die anwendungsspezifische Informationen beinhalten. Die Partialmodelle sind somit die Informationsstellen, während das Shape Representation Interface die organisatorische Funktion einer systeminternen Schnittstelle übernimmt. Die einzelnen Partialmodelle des Resource Models werden in den folgenden Abschnitten detaillierter beschrieben; die Partialmodelle der Application Models sind zur Zeit noch nicht genügend dokumentiert.

Zur Dokumentation der Partialmodelle wird die für STEP erstellte formale Beschreibungssprache EXPRESS verwendet. Hierbei handelt es sich um eine hybride Sprache, die sowohl zur Informationsmodellierung als auch zur Abbildung von Datenstrukturen geeignet ist. Die in EXPRESS beschriebenen Informationen sind unabhängig von der späteren Implementierung. Durch die Verwendung dieser Beschreibungssprache wird die syntaktisch und semantisch eindeutige Repräsentation der Partialmodelle ermöglicht. Dies schließt Interpretationsfehler bei der Kommunikation zwischen Entwicklern und Anwendern aus und ermöglicht den Einsatz von Softwaretools für die Informationsmodellierung. Die Beschreibungssprache EXPRESS stellt in STEP zusammen mit den in ihr definierten Informationseinheiten die logische Ebene (Logical Layer) dar.

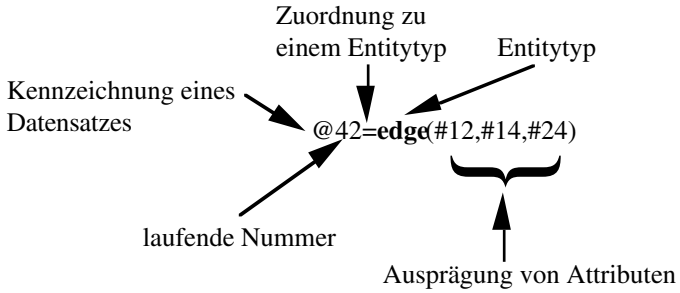
Die Informationseinheiten (Entities) der einzelnen Partialmodelle werden durch Einordnung in eine Hierarchie von Super-/Sub-Entities, die Angabe von Attribut-Wert-Paaren und durch in Form von Funktionsaufrufen dargestellten Constraints auf diesen Werten definiert.

Dazu sei folgendes Beispiel einer Definition aus dem Topologiemodell gegeben:



Die Typhierarchien der einzelnen Partialmodelle finden sich im Anhang wieder. Die Definitionen der Entities der Partialmodelle möge der interessierte Leser in der entsprechenden Literatur nachschlagen.

Zu der logischen Ebene wird in STEP eine Abbildungsvorschrift auf die physikalische Ebene definiert. Mit anderen Worten: Es ist festgelegt, wie eine in EXPRESS formalisierte Beschreibung auf eine reale (physikalische) Datei abgebildet wird.
Die gerade als Beispiel erwähnte Informationseinheit würde in der physikalischen Ebene von STEP (Physical File) nachfolgende Datenstruktur erzeugen:



Es wird deutlich, daß diese Datenstruktur keine semantischen Informationen mehr enthält. Benutzer, die mittels STEP Daten austauschen, müssen daher alle über dieselben Definitionen der Informationseinheiten verfügen. Das heißt insbesondere, daß bei Verwendung benutzerdefinierter Informationstypen (deren Definition in STEP grundsätzlich zulässig ist) Sender und Empfänger diese Informationstypen vor der Datenübertragung austauschen müssen.

2 Die Basis-Partialmodelle

Die Basis-Partialmodelle (Resource Models) dienen als Grundlage für die Anwendungs-Partialmodelle (Application Models) und stellen eine Datenstruktur zur Verfügung, in der sich alle produktdefinierenden Daten vollständig beschreiben lassen. Sie beinhalten somit nur anwendungsneutrale Informationen, wie etwa geometrische oder topologische Informationen und Informationen über Oberflächengüten.

Die Basis-Partialmodelle setzen sich aus dem Geometriemodell (geometry), dem Topologiemodell (topology), dem Modell der Form Feature (form feature), dem Toleranzenmodell (tolerance), dem Modell der Materialeigenschaften (materials), dem Präsentationsmodell (presentation) und dem Modell der allgemeinen Verwaltungsinformationen (miscellaneous) zusammen. In den folgenden Abschnitten werden die Partialmodelle entsprechend der aktuellen Dokumentation über die Schnittstellenspezifikation von STEP erläutert.

2.1 Geometriemodell

Das Geometriemodell von STEP enthält ausschließlich die geometrischen Grundelemente Punkt, Linie und Fläche; Körper und andere komplexere Gebilde werden durch das Topologiemodell von STEP definiert. Die Trennung ist dabei nicht ganz klar : Die *Curve_on_surface* (Oberflächenkurve) stellt bereits eine logische Beziehung zwischen Flächenelementen und Kurvenelementen her. Freiformflächen und Kurven lassen sich analytisch und approximativ beschreiben.

Die Beschreibung von 2D-Elementen ist durch eine optionale Definition der dritten Dimension realisiert. Die zweidimensionale Beschreibung ist somit ein Spezialfall der dreidimensionalen Beschreibung.

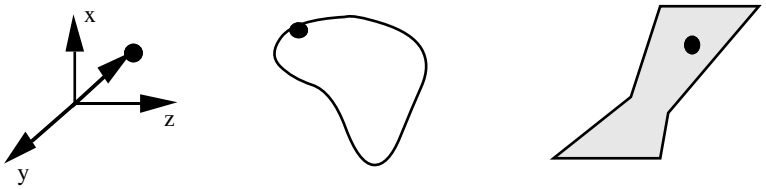
Die beschriebene Geometrie ist ausschließlich eine Geometrie von parametrisierten Kurven und Flächen, das heißt, daß Punkt und Richtungsvektor die elementarsten Elemente sind, aus denen sich alle anderen Elemente definieren lassen. Die definierten Elemente sind Punkt (point), Vektor (vector), Achsensystem (axis_placement), Kurve (curve), Oberfläche (surface), Transformation (transformation) und Koordinatensystem (coordinate_system).

Die Orientierung der geometrischen Elemente wird durch das Kreuzprodukt des Normalenvektors N des Elementes und der z-Achse des lokalen Koordinatensystems

bestimmt. Dadurch läßt sich immer eindeutig bestimmen, in welcher Richtung sich das Material befindet.

2.1.1 Point

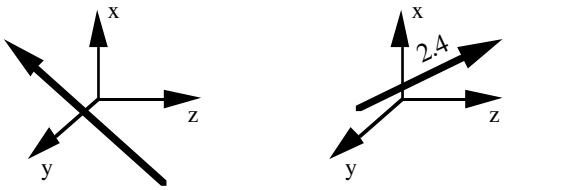
Ein *Point* (Punkt) ist ein Ort in irgendeinem Koordinatensystem. Es wird zwischen einem *Cartesian_point*, der durch Angabe der kartesischen Koordinaten definiert ist, einem *Point_on_curve*, der durch Angabe einer Kurve und eines Parameters für seinen Ort auf der Kurve definiert ist, und einem *Point_on_surface*, der durch Angabe einer Oberfläche und zweier Parameter für seinen Ort auf der Oberfläche definiert ist, unterschieden.



cartesian_point point_on_curve point_on_surface

2.1.2 Vector

Ein *Vector* (Vektor) ist ein 3-/2-Tupel. Es wird unterschieden zwischen dem Richtungsvektor *Direction* und dem Richtungsvektor mit Größe *Vector_with_magnitude*. Für die drei Achsen *x*, *y* und *z* werden vordefinierte Vektoren bereitgestellt (*default_x_axis*, *default_y_axis*, *default_z_axis*).

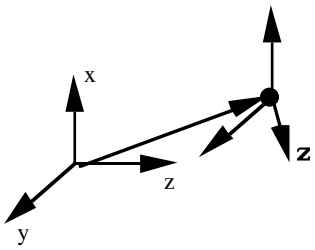


direction vector_with_magnitude

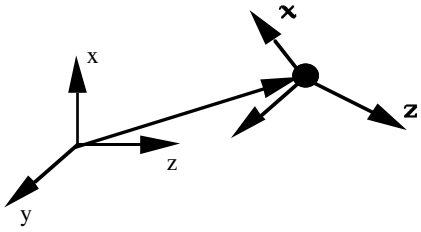
2.1.3 Axis_Placement

Ein *Axis_placement* (Achsensystem) definiert eine lokale Umgebung für die Definition eines geometrischen Objektes. Es lokalisiert das Objekt und gibt ihm eine Orientierung. Dabei wird zwischen einem *Axis1_placement*, das durch Angabe seines Ursprunges und der Richtung der *z*-Achse definiert wird, und einem *Axis2_placement*, das durch Angabe seines Ursprunges, der Richtung der *z*-Achse und der lokalen *x*-Achse

(Orientierung) definiert wird, unterschieden. Letzteres benötigt man vor allem, um nicht achsensymmetrische Objekte im Raum zu beschreiben.



axis1_placement



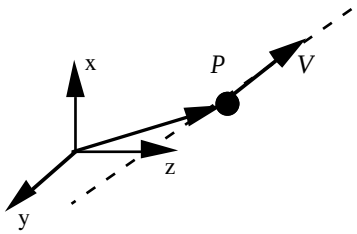
axis2_placement

2.1.4 Curve

Eine *Curve* (Kurve) ist ein Pfad eines Punktes, der sich durch ein Koordinatensystem bewegt. Dieses lokale Koordinatensystem wird bei der Definition mit angegeben. Es werden 5 Arten von Kurven unterschieden : *line*, *conic*, *bounded_curve*, *curve_on_surface* und *offset_curve*.

2.1.4.1 Line

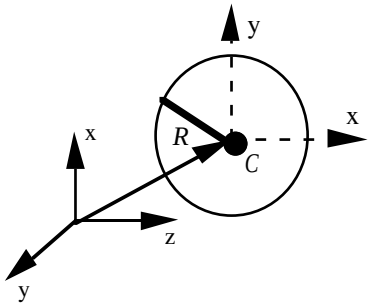
Eine *Line* (Linie) ist eine unbegrenzte Kurve, die durch einen Punkt auf ihr und ihren Richtungsvektor definiert wird. Die Kurve wird durch folgende Gleichung parametrisiert :
$\lambda(u) = P + u\langle V\rangle$, wobei P der Punkt ist und V der Richtungsvektor.



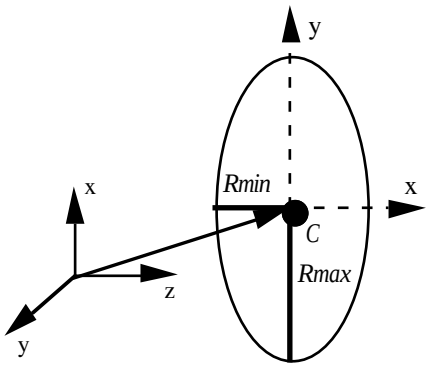
2.1.4.2 Conic

Ein *Conic* (Kegelschnitt) wird in Termen seiner eigenen Parameter beschrieben und nicht in Abhängigkeit mit anderen Geometrien. Es werden 4 Arten unterschieden : *circle*, *ellipse*, *hyperbola* und *parabola*.

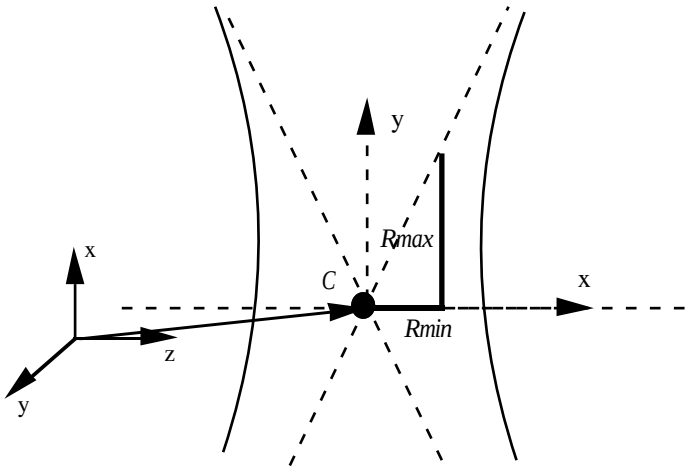
Circle : Ein Kreis wird durch seinen Radius, seinen Ursprung und seine Orientierung definiert. Er ist durch folgende Gleichung parametrisiert : $\lambda(u) = C + R(\cos u\mathrm{x} + \sin u\mathrm{y})$, wobei C der Ursprung ist, R der Radius ist und x und y die Orientierung angeben.



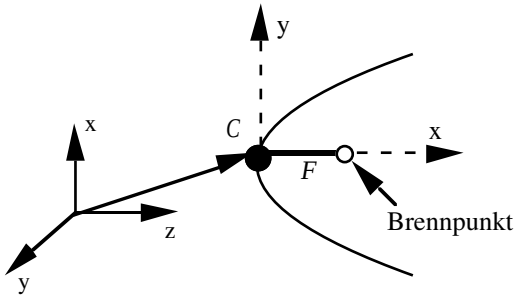
Ellipse : Eine Ellipse wird durch die Längen ihres maximalen und minimalen Radius, ihren Ursprung und ihre Orientierung definiert. Sie ist wie folgt parametrisiert :$\lambda(u) = C + R_{max}\cos u\mathrm{x} + R_{min}\sin u\mathrm{y}$, wobei C der Ursprung ist, $R_{\min}$ und $R_{\max}$ die beiden Radien sind und x und y die Orientierung angeben.



Hyperbola : Eine Hyperbel wird durch die Längen ihres maximalen und minimalen Radius, ihren Ursprung und ihre Orientierung definiert. Sie ist wie folgt parametrisiert :$\lambda(u) = C + R_{max}\cosh u\mathrm{x} + R_{min}\sinh u\mathrm{y}$, wobei C der Ursprung ist, $R_{\min}$ und $R_{\max}$ die beiden Radien sind und x und y die Orientierung angeben.



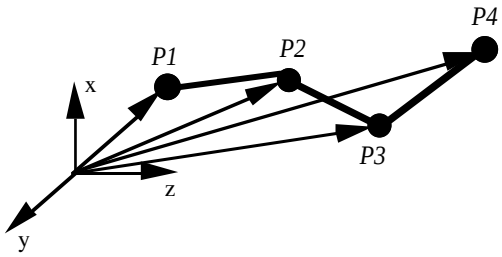
Parabola : Eine Parabel wird durch ihre Brennweite, ihren Ursprung und ihre Orientierung definiert. Die Brennweite bezeichnet den Abstand zwischen dem Ursprung und dem Brennpunkt. Die Parabel ist wie folgt parametrisiert :
$\lambda(u) = C + F(u^2 x + 2uy)$, wobei C der Ursprung ist, F der Abstand zwischen C und dem Brennpunkt ist und x und y die Orientierung angeben.



2.1.4.3 Bounded_curve

Eine *Bounded_curve* (begrenzte Kurve) ist eine endlich lange Kurve. Es werden 4 Arten unterschieden : *polyline*, *b-spline_curve*, *trimmed_curve* und *composite_curve*.

Polyline : Ein Polygonzug wird durch eine endliche Menge von Punkten definiert. Er wird wie folgt parametrisiert :
$\lambda(u) = P_i(i - u) + P_{i+1}(u + 1 - i)$, wobei die P_i die den Polygonzug definierenden Punkte sind.

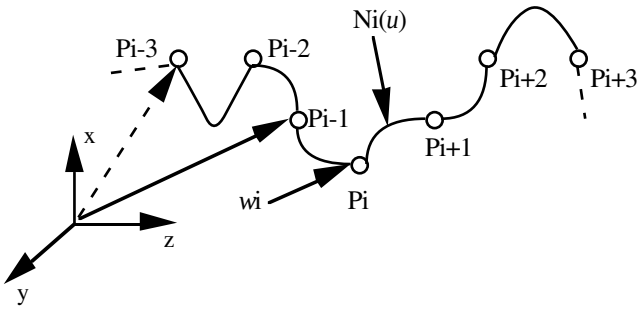


B-spline_curve : Eine B-Spline Kurve wird durch eine Menge von Kontrollpunkten

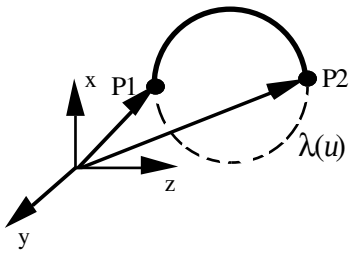
und den Grad der B-Splinefunktion definiert. Sie ist wie folgt parametrisiert :

$$\lambda(u) = \frac{\sum_{i=0}^{k} w_i P_i N_i(u)}{\sum_{i=0}^{k} w_i N_i(u)},$$

wobei k+1 die Anzahl der Kontrollpunkte ist, P_i Kontrollpunkte sind, N_i die normalisierten B-Splinefunktionen vom Grad d sind und w_i die Gewichte (Koeffizienten) der Kontrollpunkte sind.

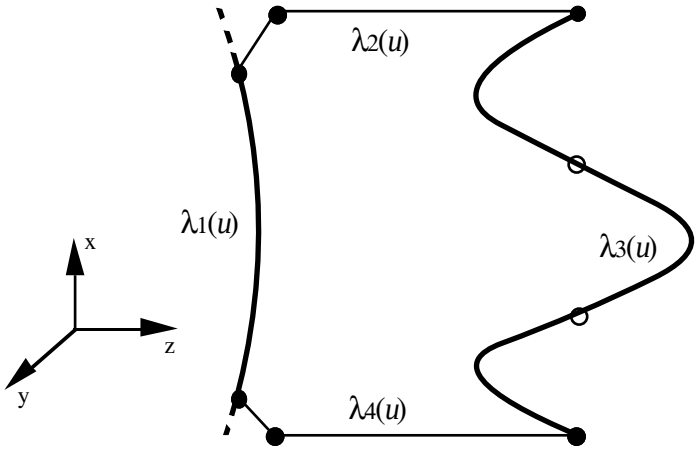


Trimmed_curve : Eine beschnittene Kurve wird durch eine begrenzte Kurve, zwei Schnittpunkte und einen Flag definiert. Der Flag gibt an, ob die Orientierung der beschnittenen Kurve mit der Orientierung der begrenzten Kurve übereinstimmt.



Composite_curve : Eine zusammengesetzte Kurve wird durch eine Menge von begrenzten Kurven definiert, die paarweise miteinander verbunden

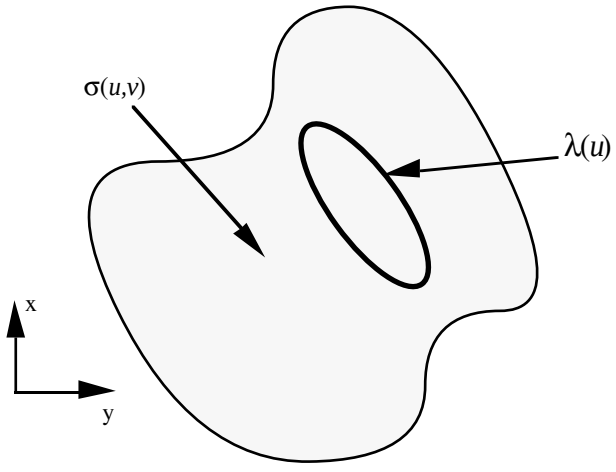
sind. Jedem Kurvensegment wird ein Flag zugeordnet, der angibt, ob die Orientierung des Segmentes mit der Orientierung der zusammengesetzten Kurve übereinstimmt.



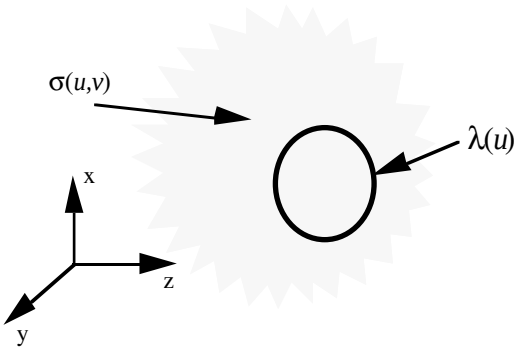
2.1.4.4 Curve_on_surface

Eine *Curve_on_surface* (Oberflächenkurve) ist eine Kurve, die auf einer parametrisierten Oberfläche verläuft. Es werden 4 Arten unterschieden : *pcurve*, *surface_curve*, *intersection_curve* und *composite_curve_on_surface*.

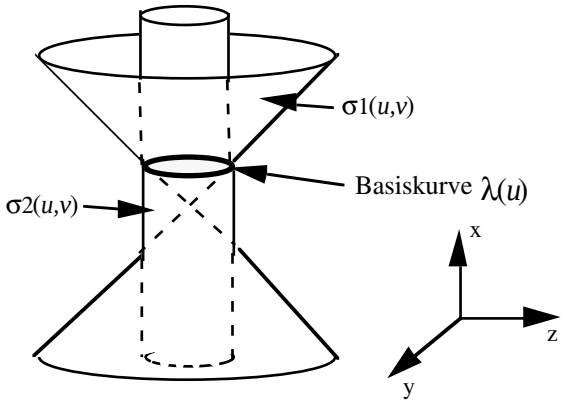
Pcurve : Eine Flächenkurve ist eine Kurve, die im zweidimensionalen parametrisierten Bereich einer Oberfläche definiert ist. Sie wird durch Angabe der Basisoberfläche und der Basiskurve definiert.



Surface_curve : Eine Oberflächenkurve ist eine Kurve, die im dreidimensionalen Bereich oder auch im zweidimensionalen parametrisierten Bereich einer Oberfläche definiert ist.

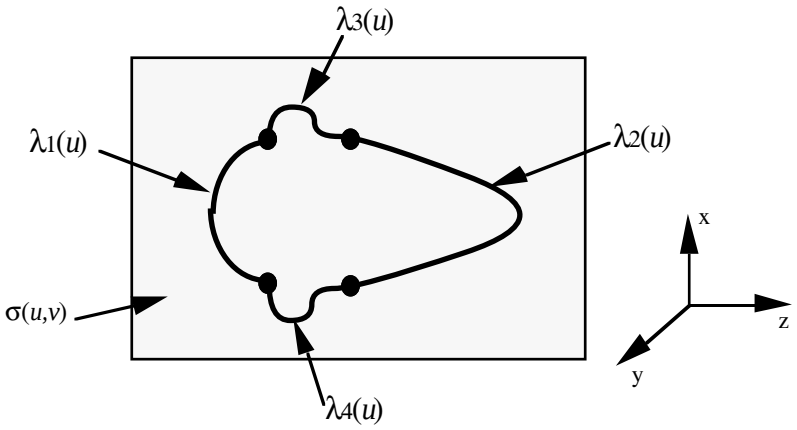


Intersection_curve : Eine Schnittkurve wird durch den Schnitt zweier Oberflächen definiert. Die Schnittkurve kann einmal als Kurve (curve) oder als Flächenkurve (Pcurve) auf einer der beiden sich schneidenden Oberflächen definiert werden.



Composite_curve_on_surface : Eine zusammengesetzte Oberflächenkurve ist als eine Menge von Oberflächenkurven definiert, die paarweise miteinander verbunden sind. Jedem Segment dieser Kurve wird ein Flag zugeordnet, der

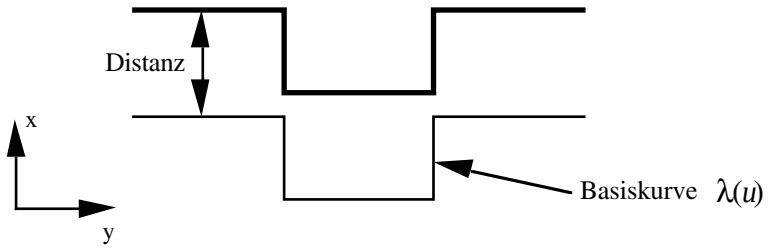
angibt, ob die Orientierung des Segmentes mit der Orientierung der zusammengestzten Oberflächenkurve übereinstimmt.



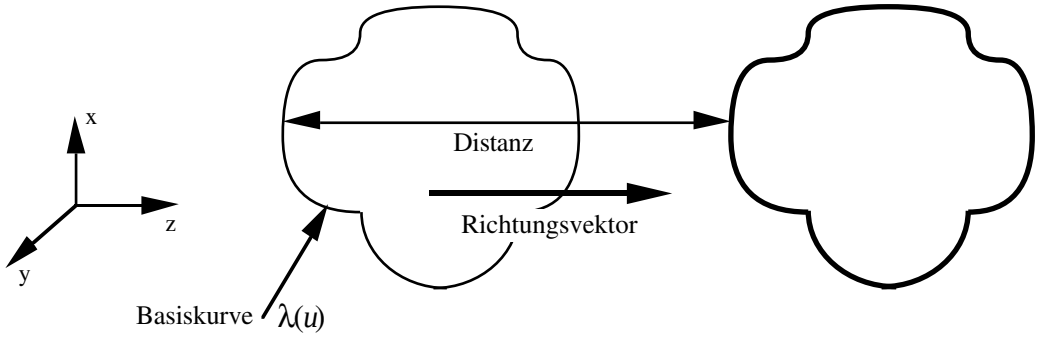
2.1.4.5 Offset_curve

Eine *Offset_curve* (Offsetkurve) ist eine Kurve, die einen konstanten Abstand zu einer (Basis-) Kurve hat. Die Basiskurve muß an jedem Punkt eine wohldefinierte Tangente besitzen.

D2_offset_curve : Eine zweidimensionale Offsetkurve wird durch Angabe einer (zweidimensionalen) Basiskurve und einer Distanz definiert. Sie verläuft in Richtung der Normalen der Fläche der Basiskurve mit dem Abstand der Distanz.



D3_offset_curve : Eine dreidimensionale Offsetkurve wird durch Angabe einer (dreidimensionalen) Basiskurve, einer Distanz und einem Richtungsvektor definiert. Sie verläuft im Abstand der Distanz in Richtung des Richtungsvektors entlang der Basiskurve.



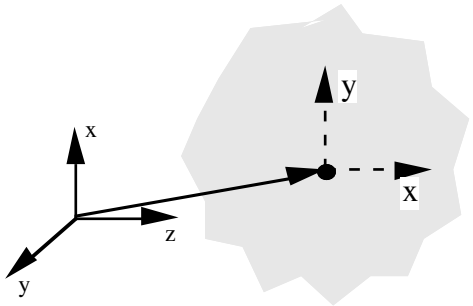
2.1.5 Surface

Eine *Surface* (Oberfläche) ist ein Pfad einer Kurve, der sich kontinuierlich durch ein Koordinatensystem bewegt. Dieses lokale Koordinatensystem wird bei der Definition mit angegeben. Es werden 4 Arten von Oberflächen unterschieden : *elementary_surface*, *swept_surface*, *bounded_surface* und *offset_surface*.

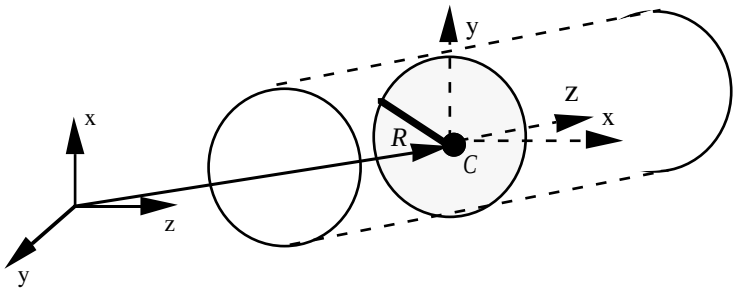
2.1.5.1 Elementary_surface

Die *Elementary_surface* (elementaren Oberflächen) sind unendlich große Flächen.

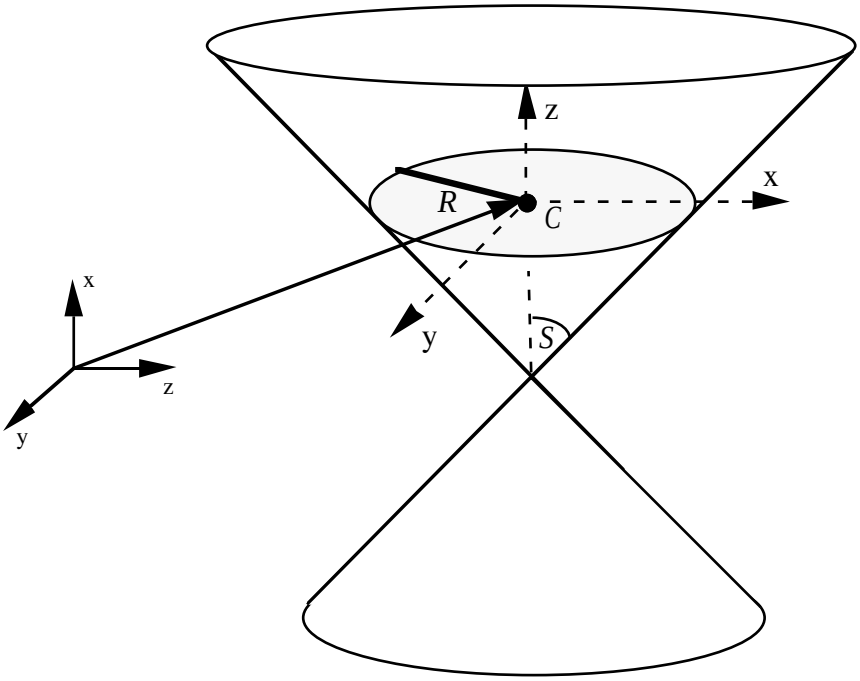
Plane : Eine Planfläche wird durch einen Punkt in dieser Fläche und ein Achsensystem definiert. Sie ist wie folgt parametrisiert : $\sigma(u,v) = C+xu+yv$, wobei C der Punkt in der Fläche ist und x und y die Orientierung angeben.



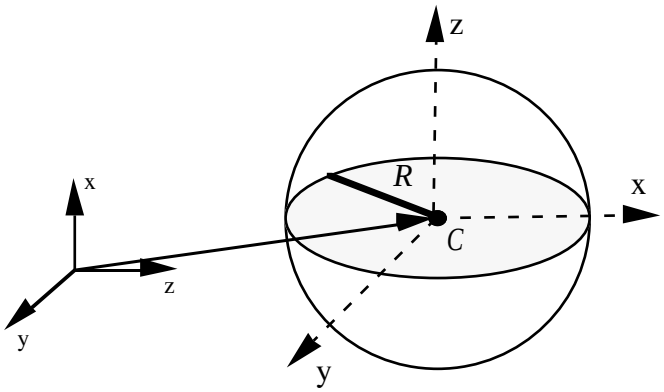
Cylindrical_surface : Eine Zylinderoberfläche wird durch ihren Radius, ihren Ursprung und ein Achsensystem definiert. Sie ist wie folgt parametrisiert : $\sigma(u,v) = C + R(\cos u x + \sin u y) + vz$, wobei C der Ursprung ist, R der Radius ist und x, y und z die Orientierung angeben.



Conical_surface : Eine Kegeloberfläche wird durch ihren Ursprung, ihren Halbwinkel, ihren Radius am Ursprung und ein Achsensystem definiert. Sie ist wie folgt parametrisiert :
$\sigma(u,v) = C + (R + v\tan S)(\cos u x + \sin u y) + vz$, wobei C der Ursprung ist, R der Radius ist, S der Halbwinkel ist und x, y und z die Orientierung angeben.

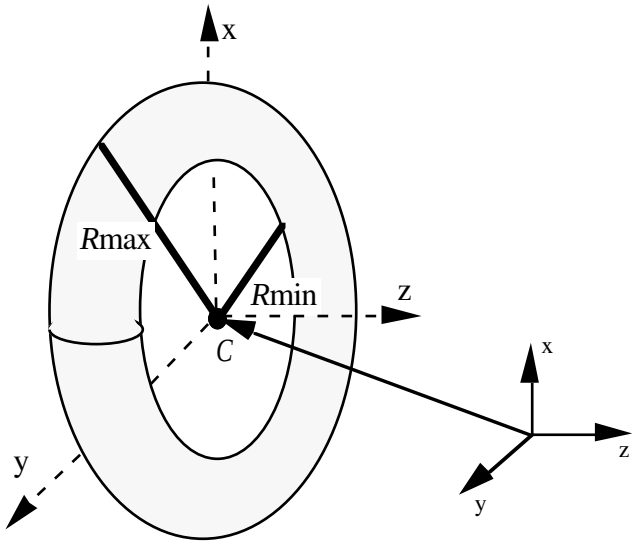


Spherical_surface : Eine Kugeloberfläche wird durch ihren Radius, ihren Ursprung und ein Achsensystem definiert. Sie ist wie folgt parametrisiert :
$\sigma(u,v) = C + R\cos v(\cos u x + \sin u y) + R\sin v z$, wobei R der Radius ist, C der Ursprung ist und x, y und z die Orientierung angeben.



Toroidal_surface : Eine Toroidoberfläche wird durch ihren maximalen und minimalen

Radius, ihren Ursprung und ein Achsensystem definiert. Sie ist wie folgt parametrisiert :
$\sigma(u,v) = C + (R_{max} + R_{min}\cos u)(\cos v x + \sin v y) + R_{min}\sin u z$,
wobei R_{min} und R_{max} die beiden Radien sind, C der Ursprung ist und x, y und z die Orientierung angeben.

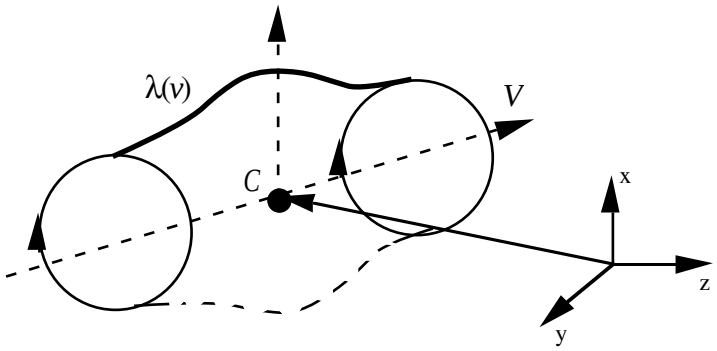


2.1.5.2 Swept_surface

Eine *Swept_surface* (gezogene Oberfläche) ist eine Oberfläche, die dadurch konstruiert wird, daß eine Kurve entlang einer anderen Kurve gezogen wird. Es werden 2 Arten unterschieden : *surface_of_revolution* und *surface_of_linear_extrusion*.

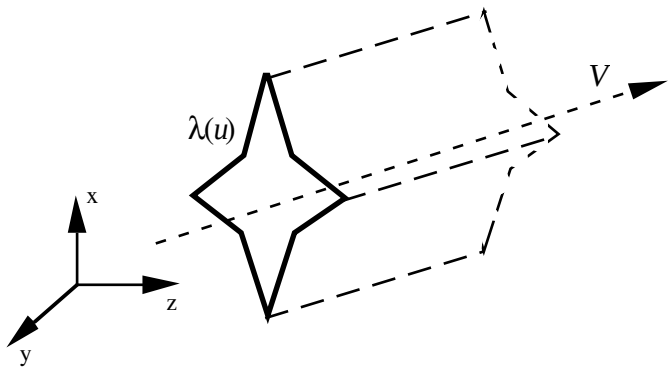
Surface_of_revolution : Eine Rotationsfläche ist eine Oberfläche, die durch eine (ganze) Rotation einer Kurve um eine Achse beschrieben wird. Sie ist definiert durch Angabe der Kurve und einem

Achsensystem. Sie ist wie folgt parametrisiert :
$\sigma(u,v) = C + (\lambda(v) - C)\cos u + ((\lambda(v) - C)V)V(1 - \cos u) + V(\lambda(v) - C)\sin u$, wobei C der Ursprung ist, V die Achse ist und $\lambda(v)$ die Parametrisierung der Kurve ist.



Surface_of_linear_extrusion : Eine Verschiebungsfläche ist ein generalisierter Zylinder, der dadurch gebildet wird, daß eine Kurve

entlang einer Achse gezogen wird. Sie wird durch die Kurve und eine Achse beschrieben. Ihre Parametrisierung ist wie folgt : $\sigma(u,v) = \lambda(u) + vV$, wobei $\lambda(u)$ die parametrisierte Kurve ist und V die Achse ist.

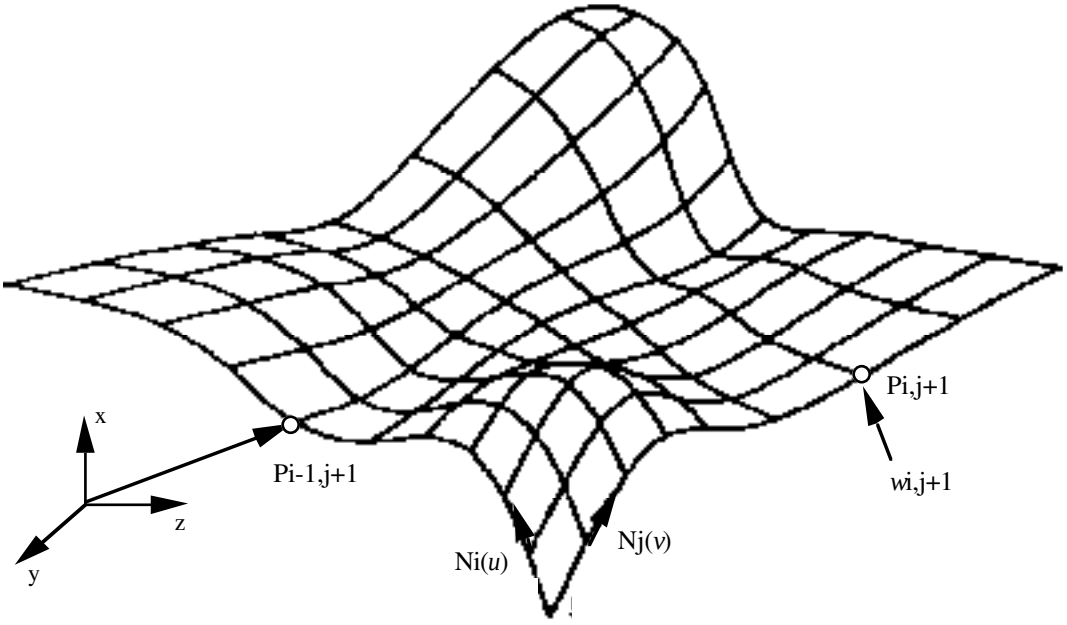


2.1.5.3 Bounded_surface

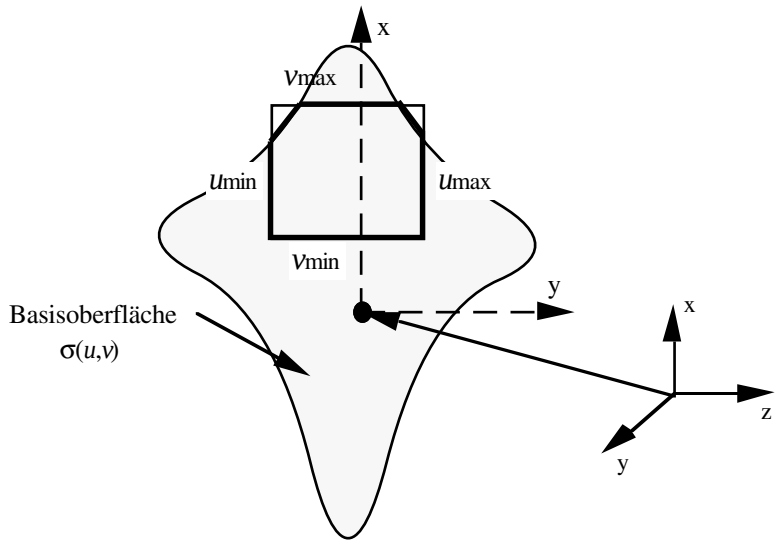
Eine *Bounded_surface* (begrenzte Oberfläche) ist eine endlich große Fläche. Es werden 4 Arten unterschieden: *b-spline_surface*, *rectangular_trimmed_surface*, *curve_bounded_surface* und *rectangular_composite_surface*.

B-spline_surface : Eine B-Spline Oberfläche wird durch eine Menge von Kontrollpunkten und den Grad der B-Splinefunktion in den beiden Flächenrichtungen definiert. Sie ist wie folgt parametrisiert :

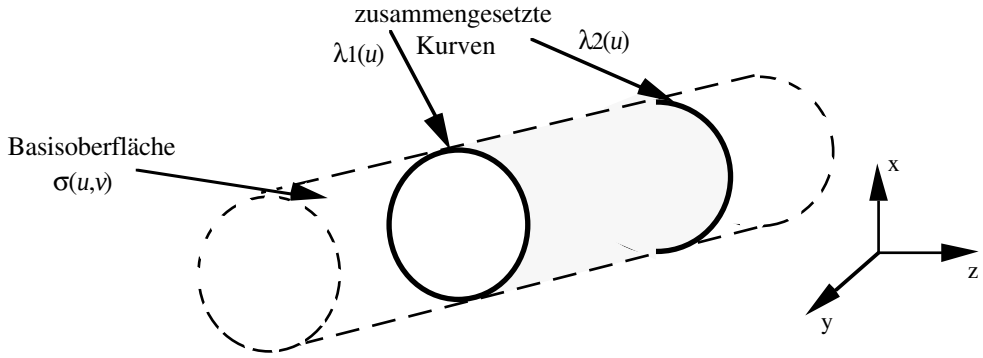
$$\sigma(u,v) = \frac{\sum_{i=0}^{k_1}\sum_{j=0}^{k_2} w_{ij}P_{ij}N_i(u)N_j(v)}{\sum_{i=0}^{k_1}\sum_{j=0}^{k_2} w_{ij}N_i(u)N_j(v)}$$, wobei $(k_1+1)*(k_2+1)$ die Anzahl der Kontrollpunkte ist, P_{ij} Kontrollpunkte sind, N_i, N_j die normalisierten B-Splinefunktionen vom Grad d_u bzw d_v sind und w_{ij} die Gewichte (Koeffizienten) der Kontrollpunkte sind.



Rectangular_trimmed_surface : Eine (rechteckige) beschnittene Oberfläche ist eine einfache begrenzte Oberfläche, der in Richtung ihrer Orientierung Grenzwerte gegeben sind. Sie wird durch Angabe dieser Grenzwerte (v_{min}, v_{max}, u_{min}, u_{max}) und der Basisoberfläche definiert.



Curve_bounded_surface : Eine durch Kurven begrenzte Oberfläche ist eine parametrisierte Oberfläche, die durch eine oder mehrere zusammengesetzte Kurven begrenzt wird. Sie ist durch die Angabe der Basisoberfläche und der zusammengesetzten Kurven definiert.



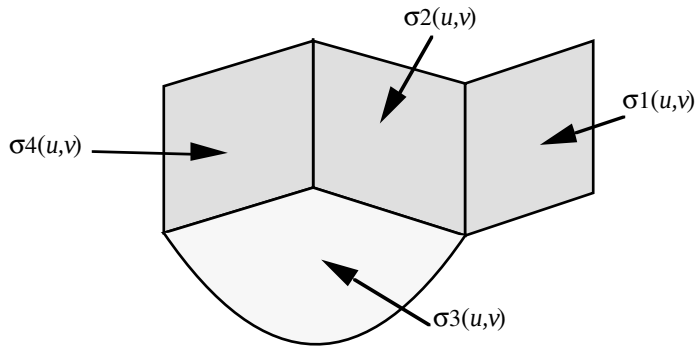
Rectangular_composite_surface : Eine rechteckige zusammengesetzte Oberfläche ist eine zusammengesetzte Oberfläche, die eine einfache

rechteckige Topologie hat. Sie ist definiert durch n und m begrenzte Oberflächenteile in Richtung ihrer Orientierung. Als begrenzte Oberflächen können durch die topologische Restriktion nur B-spline Oberflächen und (rechteckig) beschnittene Oberflächen zur Definition herangezogen werden.

Jedem Oberflächenteil wird ein Flag zugeordnet, der

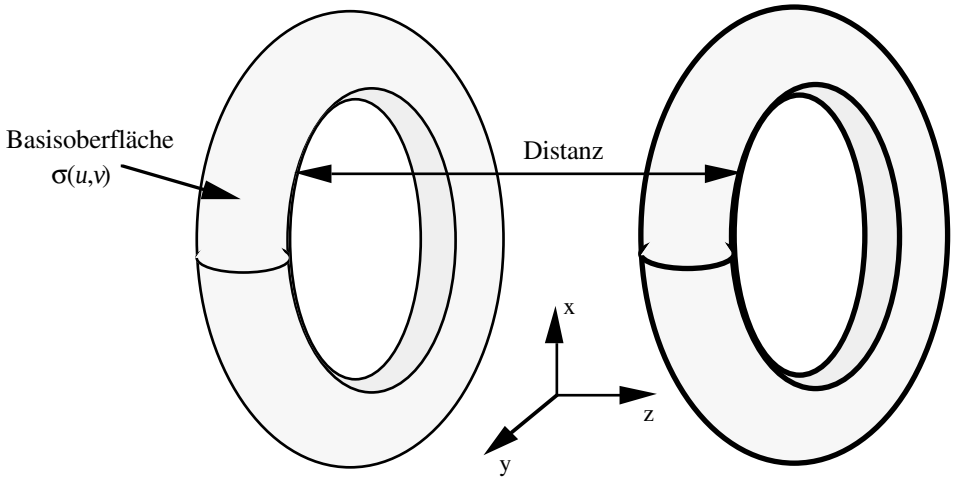
Auskunft darüber gibt, ob sein Minimum oder Maximum in Richtung seiner Orientierung mit dem Maximum oder Minimum der angrenzenden Flächen

verbunden wird. Für jede Orientierungsrichtung existiert ein Flag.



2.1.5.4 Offset_surface

Eine *Offset_surface* (Offsetoberfläche) ist eine Oberfläche, die einen konstanten Abstand zu einer (Basis-) Oberfläche hat. Sie wird durch Angabe der Basisoberfläche und der Distanz definiert. Das Kreuzprodukt der Orientierung von der Basisoberfläche bestimmt die Richtung, in der die Offsetoberfläche liegt.



2.1.6 Transformation

Eine *Transformation* (Transformation) definiert eine allgemeine geometrische Transformation, wie etwa eine Rotation, eine Spiegelung oder Verschiebung. Die Transformation für einen Punkt mit Positionsvektor P ist zum Beispiel definiert durch $P \rightarrow A + STP$, wobei A der lokale Ursprung ist, S der Vergrößerungswert ist und T die orthogonale Matrix darstellt, die durch drei normierte Vektoren aufgespannt wird. Falls $|T| = -1$ gilt, ist eine Spiegelung in der Transformation enthalten.

2.1.7 Coordinate_system

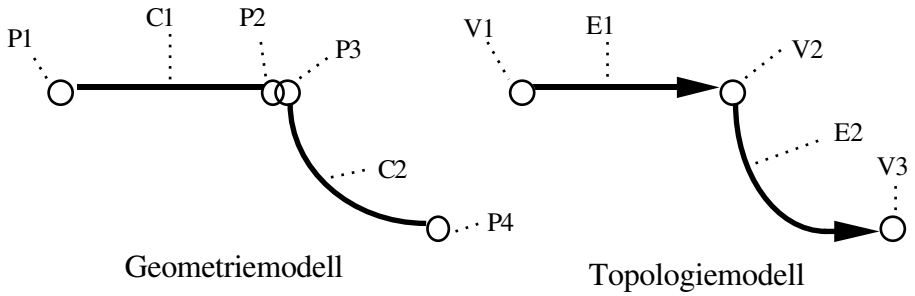
Ein *Coordinate_system* (Koordinatensystem) wird durch eine Transformation definiert, die sich auf ein anderes Koordinatensystem bezieht. Ursprung ist das globale Koordinatensystem.

2.2 Topologiemodell

In dem Topologiemodell von STEP werden Beziehungen zwischen den geometrischen Objekten repräsentiert. Dadurch wird ermöglicht, daß zum Beispiel geometrische Flächen als Grenzflächen eines Körpers definiert werden können oder daß Nachbarschaftsbeziehungen von geometrischen Flächen hergestellt werden können.

Definiert werden in dem Modell unter anderem Knoten (Vertex), Kanten (Edge) und Oberflächen (face), die als *Schnittstellen* zu den im Geometriemodell definierten Elementen Punkt (point), Kurve (curve) und Oberfläche (surface) verwendet werden können. Darüber hinaus sind Elemente für zusammengesetzte Kanten (connected_vertex_set), Pfade von Kanten (path), Schleifen von Kanten (loops), Körperteile (shell), zusammengesetzte Flächen (connected_edge_set), Bereiche (region) und Teilflächen (subfaces) definiert.

Die Elemente des Topologiemodells repräsentieren die *logischen* Beziehungen zwischen den geometrischen Elementen; konkrete Maßangaben kommen daher nicht direkt vor. Damit kann beispielsweise der aus der NC-Technik bekannte Fehler vermieden werden, daß durch Rechenungenauigkeiten des Prozessors ein geschlossenes NC-Programmiersystem einen übertragenen Konturzug als nicht geschlossen erkennt.



Verschiedene topologische Objekte verwenden einen Flag zur Repräsentation der Orientierung :

- TRUE : Die Orientierung des referenzierten Elementes ist gleich der Orientierung des referenzierenden Elementes.
- FALSE : Die Orientierung des referenzierten Elementes ist entgegengesetzt der Orientierung des referenzierenden Elementes.

Es wird kein direkter Bezug auf die lokale Orientierung der zugehörigen geometrischen Elemente genommen.

In der Topologie kann eine Kette solcher Flags auftreten. Zur Handhabung dieser Erscheinung ist auf der Menge der Flags Φ := { True, False } eine kommutative und

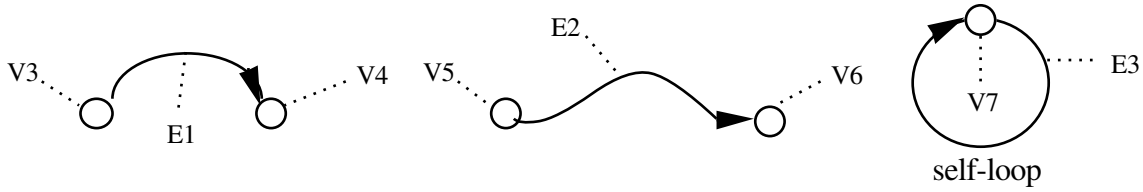
assoziative Funktion *not exclusive or* : $\otimes : \Phi \rightarrow \Phi$ definiert, mit $T \otimes T = T$, $T \otimes F = F$ und $F \otimes F = T$, so daß $(\otimes,\Phi)$ eine kommutative Gruppe mit Idempotenz bildet.

2.2.1 Vertex

Ein *Vertex* (Knoten) ist ein topologisches Objekt, das mit einem Punkt (point) im Geometriemodell assoziiert werden kann.

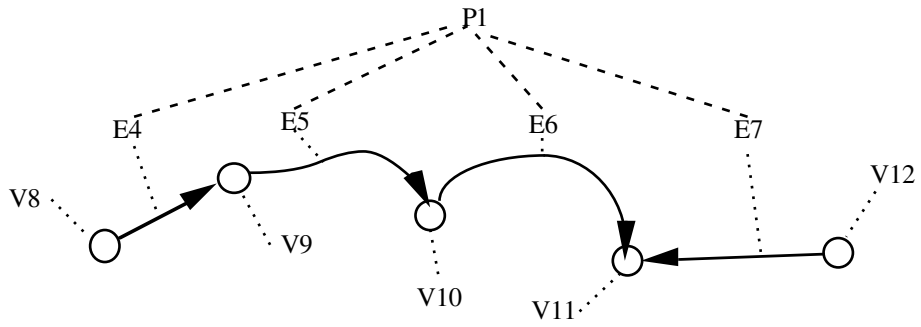
2.2.2 Edge

Eine *Edge* (Kante) stellt eine logische Beziehung zwischen zwei Knoten dar; sie stimmt mit einer Linie überein, die zwischen den beiden Knoten gemalt ist. Die Knoten (Begrenzungen) einer Kante gehören nicht zu ihrer Domain. Ihr kann eine Kurve im Geometriemodell zugeordnet sein, wobei ein Flag angibt, ob die Orientierung der assoziierten Kurve mit der Orientierung der Kante übereinstimmt. Die Orientierung einer Kante wird durch die Verbindung der Knoten gegeben : Sie geht vom ersten Knoten zum zweiten. Sind die beiden Knoten gleich, so wird die Kante als eine *self-loop* bezeichnet.



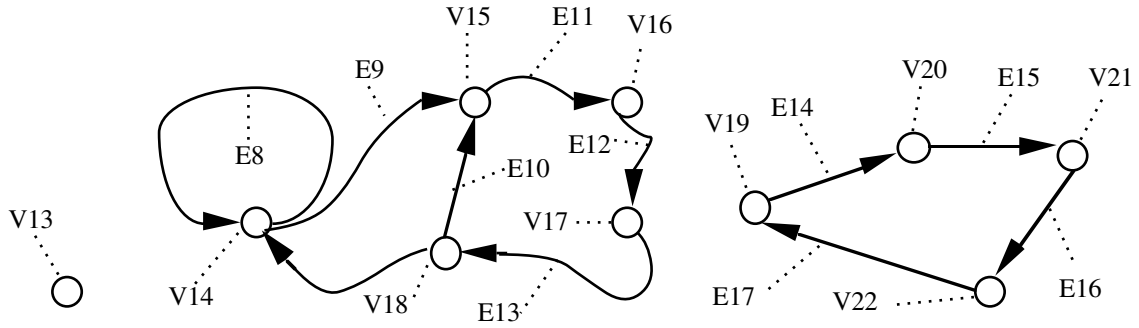
2.2.3 Path

Ein *Path* wird durch die Verbindung von Kanten und Knoten so gebildet, daß keine Kante und kein Knoten zweimal verwendet wird. Die Begrenzungen eines Pfades gehören nicht zu seiner Domain. Die Orientierung geht vom ersten Knoten zum letzten. Jeder dem Path angehörigen Kante wird ein Flag zugeordnet, der Auskunft darüber gibt, ob die Orientierung von Kante und Path übereinstimmen.



2.2.4 Loop

Eine *Loop* (Schleife) wird ebenfalls wie der Path durch eine Verbindung von Kanten und Knoten gebildet. Die Begrenzungen einer Schleife gehören zu ihrer Domain. Üblicherweise verwendet man eine Schleife, um eine Fläche, die auf einer Oberfläche (surface) liegt, zu begrenzen. Man unterscheidet zwischen einer *Vertex_loop*, die nur aus einem Knoten besteht, einer *Edge_loop*, die aus einer Menge von Kanten und Knoten besteht, und einer *Poly_loop*, die aus einer Menge von Punkten im Geometriemodell besteht. Die Orientierung der Schleife wird analog zur Orientierung des Path gebildet : Sie geht vom ersten Knoten zum letzten.

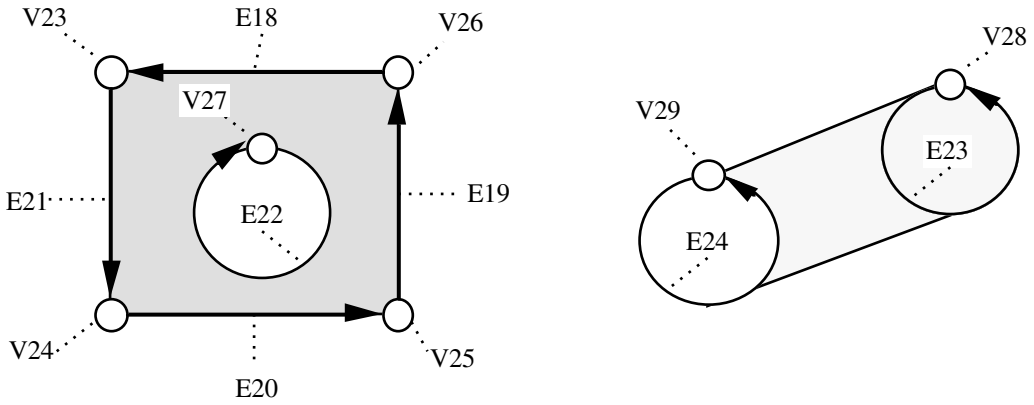


Vertex_loop Edge_loop Poly_loop

Die Orientierung der Edge_loop verläuft vom ersten Knoten zum letzten. Die Kanten der Edge_loop haben Flags, die ihre Orientierung mit der Orientierung ihrer Edge_loop in Verbindung setzt. Die Orientierung der Poly_loop entspricht der Orientierung der sie bildenden Liniensegmente. Im Gegensatz zur Edge_loop werden bei der Poly_loop die Kanten implizit durch die Knoten gebildet : Sie sind Geraden von einem Knoten zum nächsten.

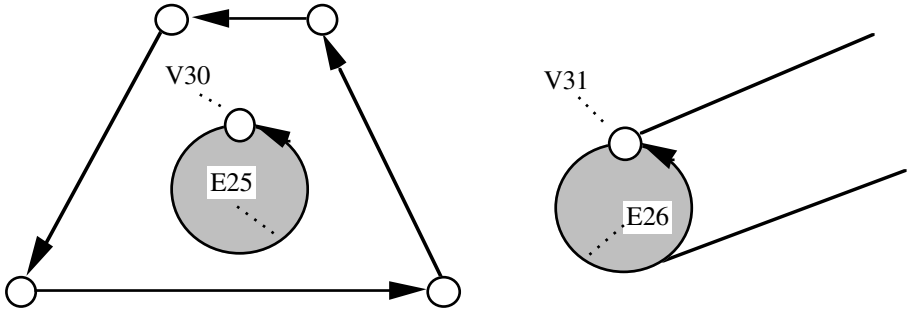
2.2.5 Face

Eine *Face* (Fläche) wird mit einer begrenzten Oberfläche im Geometriemodell assoziiert, die Löcher (Flächen mit entgegengesetzter Orientierung) enthalten kann. Sie wird durch Angabe ihrer Begrenzungen, die Schleifen sind, repräsentiert. Die Begrenzungen einer Fläche gehören nicht zu ihrer Domain. Die Orientierung der Fläche wird in Abhängigkeit ihres Normalenvektors n gebildet. Das Kreuzprodukt von n mit den Tangenten t der die Fläche begrenzenden Schleifen ($n \times t$) zeigt in das Innere der Fläche. Jede Schleife erhält einen Flag, der ihre Orientierung mit der Orientierung der Fläche in Beziehung setzt.



2.2.6 Subface

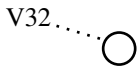
Eine *Subface* (Teilfläche) ist ein Ausschnitt einer Fläche oder einer Teilfläche. Es wird nur bei der Teilfläche vermerkt, zu welcher Fläche sie gehört. Die Teilfläche wird genau wie die Fläche repräsentiert. Flächen lassen sich nicht aus Teilflächen zusammensetzen.



2.2.7 Shell

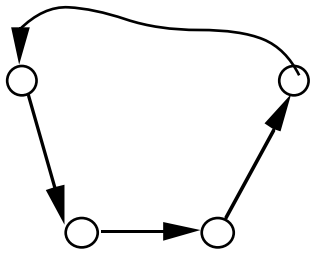
Eine *Shell* (Körperteil) wird in Abhängigkeit ihrer Dimension mit Punkten, Kurven oder Oberflächen im Geometriemodell assoziiert. Sie wird vor allem zur Begrenzung von Bereichen (Region), sprich Körpern, verwendet, wie auch Kanten zur Begrenzung von Flächen dienen. Die Begrenzungen eines Körperteil sind selbst Teil des Körperteils. Man unterscheidet zwischen einer *Vertex_shell*, die nur aus einer Vertex_loop besteht, einer *Wire_shell*, die aus einer Edge_loop besteht, einer *Open_shell*, die aus Flächen besteht, und einer *Closed_shell*, die ebenfalls aus Flächen besteht.

Vertex_shell : Sie repräsentiert ein Teil, das nur aus einem Knoten besteht.



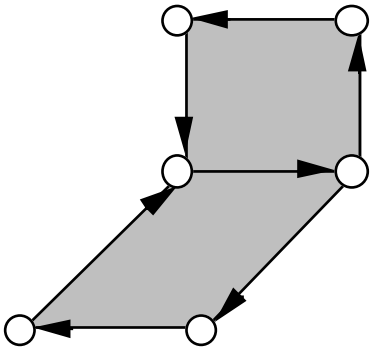
Vertex_shell

Wire_shell : Sie besteht aus einer Menge von Edge_loop, deren Vereinigung einen verbundenen Graph definiert, der aus Knoten und Kanten besteht.



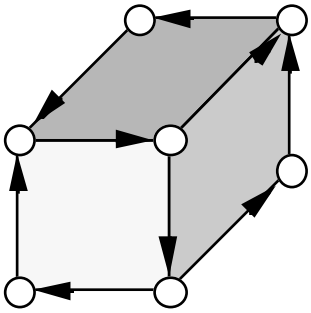
Wire_shell

Open_shell : Sie besteht aus einer Menge von Flächen. Jede einzelne Fläche erhält einen Flag, der ihren Normalenvektor mit dem des Körperteils in Beziehung setzt. Sie kann mit einer offenen Oberfläche im Geometriemodell assoziiert werden und entspricht somit einer *Closed_shell* mit Löchern. Im Gegensatz zu allen anderen Teilen, beinhaltet die Domain der Open_shell nicht ihre Begrenzungen; Begrenzungen sind die Kanten, die nur einmal von den Flächen verwendet werden.



Open_shell

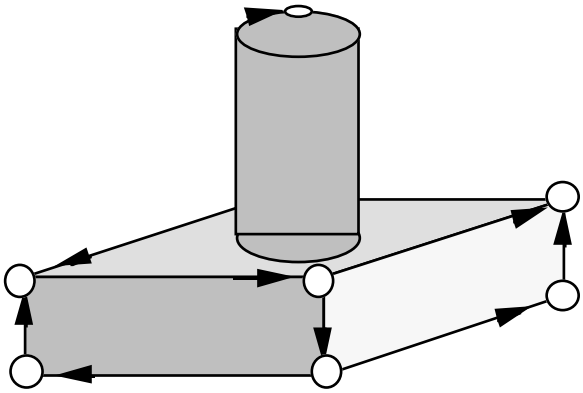
Closed_shell : Sie besteht aus einer Menge von Flächen, wobei jede einzelne Fläche einen Flag erhält, der ihren Normalenvektor mit dem des Körperteils in Beziehung setzt. Sie kann mit einer geschlossenen Oberfläche im Geometriemodell assoziiert werden. Da jede Kante genau zweimal verwendet wird, besitzt sie keine Begrenzungen.



Closed_shell

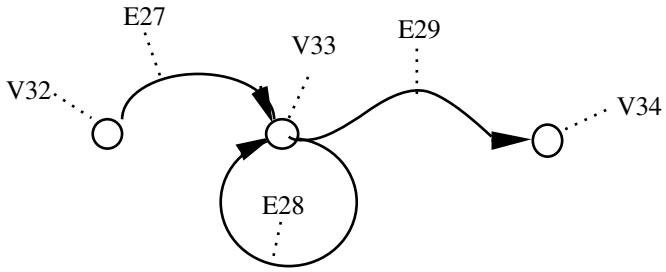
2.2.8 Region

Ein *Region* (Bereich) ist ein dreidimensionales Objekt, das durch Körperteile beschränkt wird. Durch diese Begrenzungen wird ein Bereich repräsentiert. Die Begrenzungen sind nicht Teil des Bereiches. Jedes Körperteil erhält einen Flag, der seine Normalenvektoren mit denen des Bereiches in Beziehung setzt.



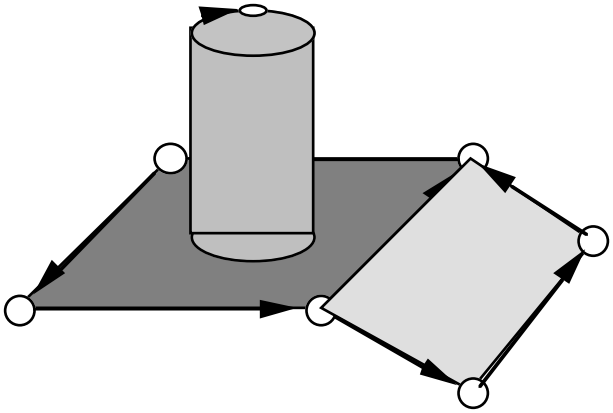
2.2.9 Connected Edge Set

Ein *Connected_Edge_Set* (verbundene Menge von Kanten) ist eine Menge von Kanten, die über ihre Knoten verbunden sind.



2.2.10 Connected Face Set

Ein *Connected_Face_Set* (verbundene Menge von Flächen) ist eine Menge von Flächen, die über ihre Kanten verbunden sind.



2.3 Form Feature Information Modell (FFIM)

Das Form Feature Information Modell von STEP soll die Repräsentation der Gestalt eines Produkts oberhalb der Geometrieebene erlauben. Da die Makrogeometrie im Geometriemodell dargestellt wird, besteht eine gewisse Konkurrenz zwischen diesen beiden Modellen. Es läßt sich aber eine Aussage über die Einsatzbereiche der verschiedenen Modelle treffen. Während das Geometriemodell die Mächtigkeit der CSG und B-Rep. Modelle der CAD-Systeme besitzt und man damit auch sehr komplexe Gestalten modellieren kann, soll über das FFIM die Möglichkeit bestehen, einfache sich oft wiederholende Geometrien schnell zu modellieren. Es soll hier ein kurzer Überblick über die Zusammensetzung des Feature Modells gegeben werden.

2.3.1 Definition Form Feature nach STEP

Ein Form Feature (FF) ist ein Aspekt einer Gestalt, der mit einem zuvor beschriebenen Muster übereinstimmt und der von allgemeiner Bedeutung ist.

Ein FF ist ein stereotyp wiederkehrender Bereich einer Gestalt und wird normalerweise mit Bohrungen, Taschen oder Nuten etc. in Verbindung gebracht. Ein FF beschreibt jedoch nicht ein komplettes Teil. Dementsprechend sollen auch keine Standardteile dargestellt werden. In einem FF werden nur Formeigenschaften betrachtet, die jedoch mit technologischen Attributen verknüpft werden können. FF enthalten sowohl eine Beschreibung der zu modifizierenden Teilgestalt, als auch die Lokalisierung des Features, auf der bereits vordefinierten Gestalt des Produktes. Dabei kann ein FF auch auf die Lage eines anderen FF bezogen werden. Im FFIM werden dem Benutzer nur wenige, dafür aber in der Industrie sehr wichtige Features zur Verfügung gestellt. Das FFIM V 5.0 erlaubt es dem Benutzer noch nicht, eigene FF zu definieren. Es ist ebenfalls noch nicht erlaubt, eigene Relationen zwischen einzelnen Features zu definieren. Es müssen immer die vom FFIM bereitgestellten Relationen compounding, tolerances, bounding/adjacency, repetition usw. benützt werden.

2.3.2 Features allgemein

Entsprechend des Gestaltbezuges der Features lassen sich zwei Arten von Features benennen:

Volumenfeatures sind Incremente oder Decremente von Volumen oder des sie umgebenden Volumenelementes. Beispiele sind Taschen, Löcher oder Knöpfe.

Ein **Übergang** (Transition) separiert oder vermischt zwei Oberflächen einer Gestalt. Beispiele sind Leisten (Fillets) und Auskehlungen (Chamfers).

Weiterhin sind sog. Form Feature-Elemente Teile von FF, die eine feste Position innerhalb eines FF haben, so z.B. die Öffnung einer Mulde.

Man kann generell zwischen drei verschiedenen Arten von Features unterscheiden:

1. Application Features (AF):
AF sind mit einer Anwendung eng verknüpft und enthalten sowohl FF-Daten, als auch Anwendungsdaten.

2. Form Features (FF):
FF haben keinen direkten Anwendungsbezug und enthalten nur Gestaltsinformationen

3. Form Feature Representation (FF rep.):
FF rep. beschreiben die Gestalt, Größe, Lokalisation und Orientierung von FF.

Im FFIM werden nur FF und FF rep. behandelt, es sind aber Verbindungen zwischen AF und FF in der Sprache der Anwendungsebene möglich, z.B. mit: "hat als Gestalt". Verbindungen zwischen FF und FF rep. sind ebenfalls möglich über das Konstrukt "wird repräsentiert mit". Das FFIM erlaubt verschiedene Wege ein bestimmtes Feature zu beschreiben. Dabei ist die Feature-Repräsentation in weiten Grenzen unabhängig von dem von einem CAD-System benützten Geometriemodell (CSG, B-Rep. Feature-Based).

Entsprechend den obigen Punkten 2. Form Feature und 3. Form Feature Representation ist die Beschreibung der FF analog zur Shape-Darstellung (siehe Shape-Modell). Auf der abstrakten Ebene wird das FF durch das Konstrukt "FORM FEATURE" dargestellt, auf der Repräsentationsebene wird das FF durch das Konstrukt "FORM FEATURE REPRESENTATION" dargestellt. Beide Ebenen werden analog zum Shape-Modell über das Konstrukt "wird repräsentiert mit" verbunden, sind aber prinzipiell eigenständig und abgeschlossen.

2.3.3 Repräsentationskonzepte

FF rep. sind typischerweise sehr eng mit den geometrischen Modellen verknüpft. Es gibt zwei Typen der Repräsentation:

1. Explicit FF:

   Dies sind Beschreibungen von Features durch die Angabe einer Liste von Elementen des Geometrie- oder Topologiemodells. Z.B. eine Tasche braucht 5 bestimmende Flächen (Faces).

2. Implicit FF:

   Dies ist eine Beschreibung eines Features durch seinen Namen und durch Parameter, die das Feature genau beschreiben.Z.B. eine Bohrung ist definiert durch Radius und Anfangs- und Endpunkt.

Da im FFIM bis auf eine Ausnahme nur Implicit FF dargestellt sind, wird im weiteren nur auf diese eingegangen.

Implicit FF rep.:

Die Beschreibung eines Implicit FF durch Parameter und oder Geometrieinformationen muß so genau sein, daß die maschinelle Berechnung der Gestalt und Lokalisierung des Features möglich ist. Dies bedeutet somit auch, daß mit der Definition eines Features eine Vorschrift angegeben werden muß, nach der das Feature evaluiert werden kann. Durch diese Beschreibung soll auch die maschinelle Erkennung von Implicit FF ermöglicht werden. Gleichzeitig soll die Erweiterung des Geometriemodells um die Geometrie-Entities des beschriebenen FF's automatisch geschehen können. Es wird deutlich, daß die Informationen des Featuremodells entweder redundant zum Geometriemodell sind, oder aber, daß die Beschreibung des Produktes im Geometrie- und Topologiemodell nicht vollständig ist, solange die Implicit FF nicht alle evaluiert wurden. Dieser Fall tritt immer dann auf, wenn das Produktmodell zumindest teilweise über FF erstellt wurde. In diesem Fall muß das vollständige Geometrie- bzw. Topologiemodell durch einen Prozessorlauf erstellt werden, wodurch im Produktmodell die oben angesprochene Redundanz geschaffen wird. Andererseits müssen nicht alle implicit FF evaluiert werden, um sie gebrauchen zu können.

Da in der Industrie viele Maße mit Toleranzen versehen werden müssen, ist es notwendig, daß man den Parametern einer Featurebeschreibung sowohl Dimensionen als auch Toleranzen zuordnen kann. Dies geschieht dadurch, daß man die Parameter eines Features über Relationen mit Entities aus dem Toleranzenmodell verknüpft. Weiterhin ist es möglich, über ein Constraint System die Konsistenz der dem Feature zur Verfügung gestellten Parameter (auch dimensional) zu überprüfen. Implicit FF rep. enthalten sowohl eine Beschreibung der zu modifizierenden Gestalt als auch eine Beschreibung der

Lokalisation und Orientierung des FF, wie oben schon beschrieben wurde. Das FFIM stellt drei verschiedene Methoden zur Lokalisierung von FF zu Verfügung.

Lokalisierungsmethoden :

1. Lokales Koordinatensystem (Axis Placement)
2. Referenz zu bestehender Gestalt
3. Bezug zu Geometriemodell Entities

Im folgenden sollen einige implicit FF rep. des Step-Modells vorgestellt werden:

1. Volumetrische Repräsentationen, bei denen Features über ein addiertes oder subtrahiertes Volumen beschrieben werden.

Sweep representations spezifizieren das addierte oder subtrahierte Volumen, das entsteht, wenn z.B. eine Fläche durch den Raum bewegt wird. Das Volumen kann begrenzt oder unbegrenzt sein.

Profile sweeps enthalten den Verschiebungsweg und die verschobene Form. Lineare und zirkuläre Wege werden unterstützt.

Axisymmetric sweeps enthalten die Rotation einer Kurve um die Mittellinie.

2. CSG FF rep. enthalten eine Darstellung der Boolschen Operationen, durch die ein CSG-Baum erzeugt wurde.

3. Delta volume representation besteht aus einem Additions- oder Subtraktionsoperator und aus einem Solid-Modell des addierten oder subtrahierten Volumens. Diese Repräsentation unterscheidet sich von den vorherigen Darstellungen dadurch, daß es sich um eine eigenständige Verknüpfung handelt, die zusammen mit jedem geometrischen Modell verwendet werden kann.

4. Edge blend representations beschreiben Abrundungen und sanfte Übergänge von Oberflächenschnitten.

Corner blend rep. beschreiben Abrundungen von Ecken.

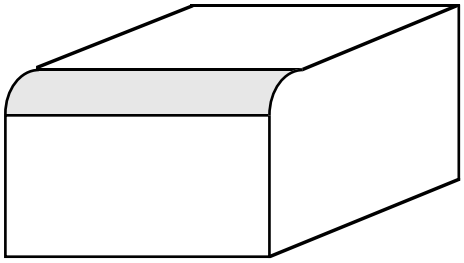
Replicate rep. beschreibt ein Feature, indem eine bestehende Beschreibung eines Features verwendet wird, plus eine Verschiebung oder Rotation des bestehenden Features, wobei alle Inhalte des "Urfeatures" erhalten bleiben .

Pattern rep. ist eine strenge Form der Replikation, bei der geometrische Muster kopiert werden.

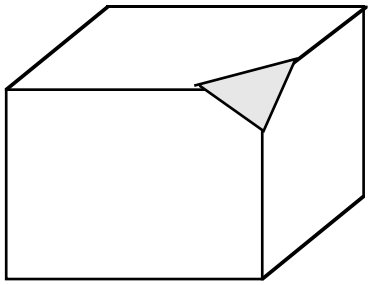
Pattern member rep. spezifiziert ein best. Feature innerhalb eines Musters durch eine Mustererkennung.

Im folgenden werden einige der FF des FFIM dargestellt. Es handelt sich hierbei um eine mehr oder weniger zufällige Auswahl von Repräsentationen, die vollständige Beschreibung der Features kann im STEP Form Feature Info. Model, Version 5 (Working) vom 18.4.90 eingesehen werden.

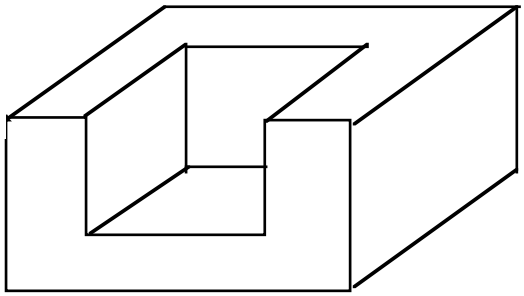
Transition Area



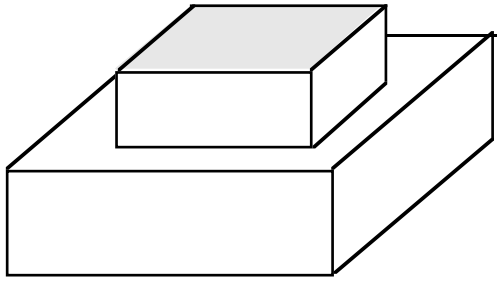
Trans eliminated Corner



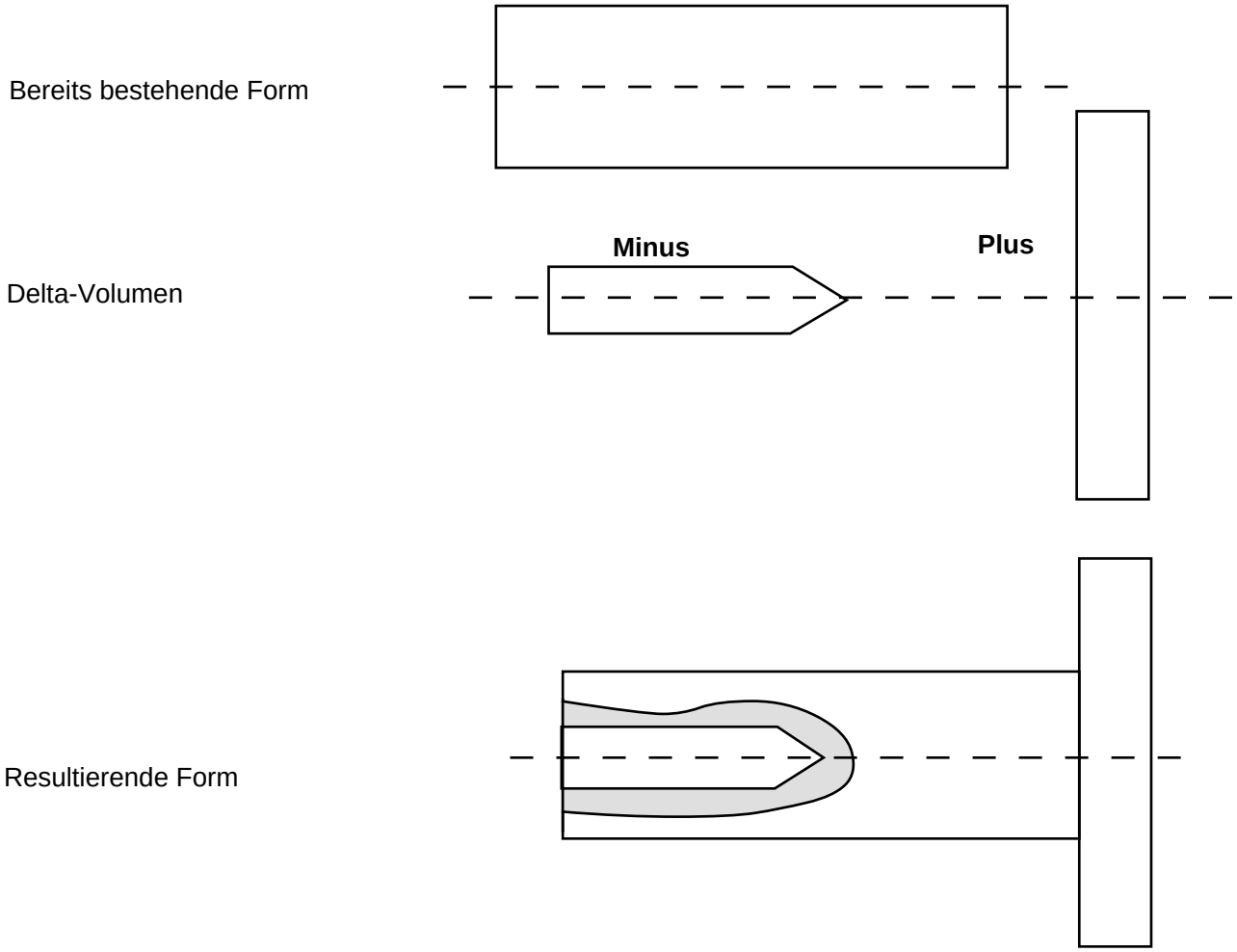
Depression



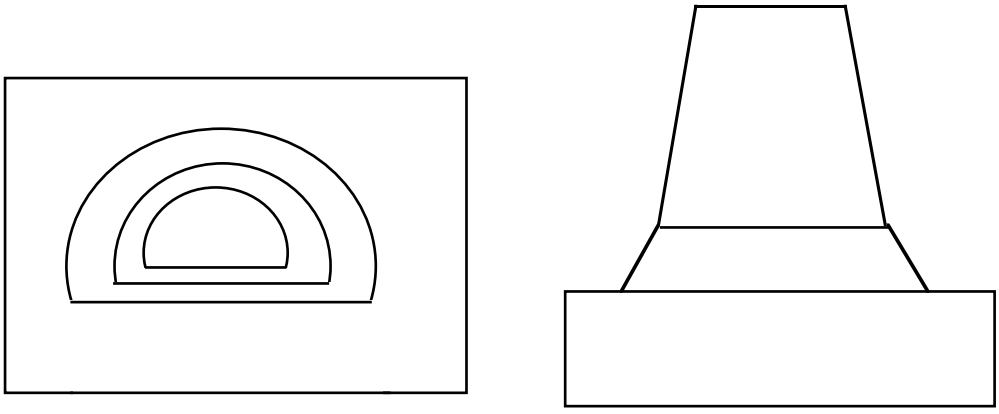
Protrusion



Delta Volume FF rep.



Contoured linear sweep

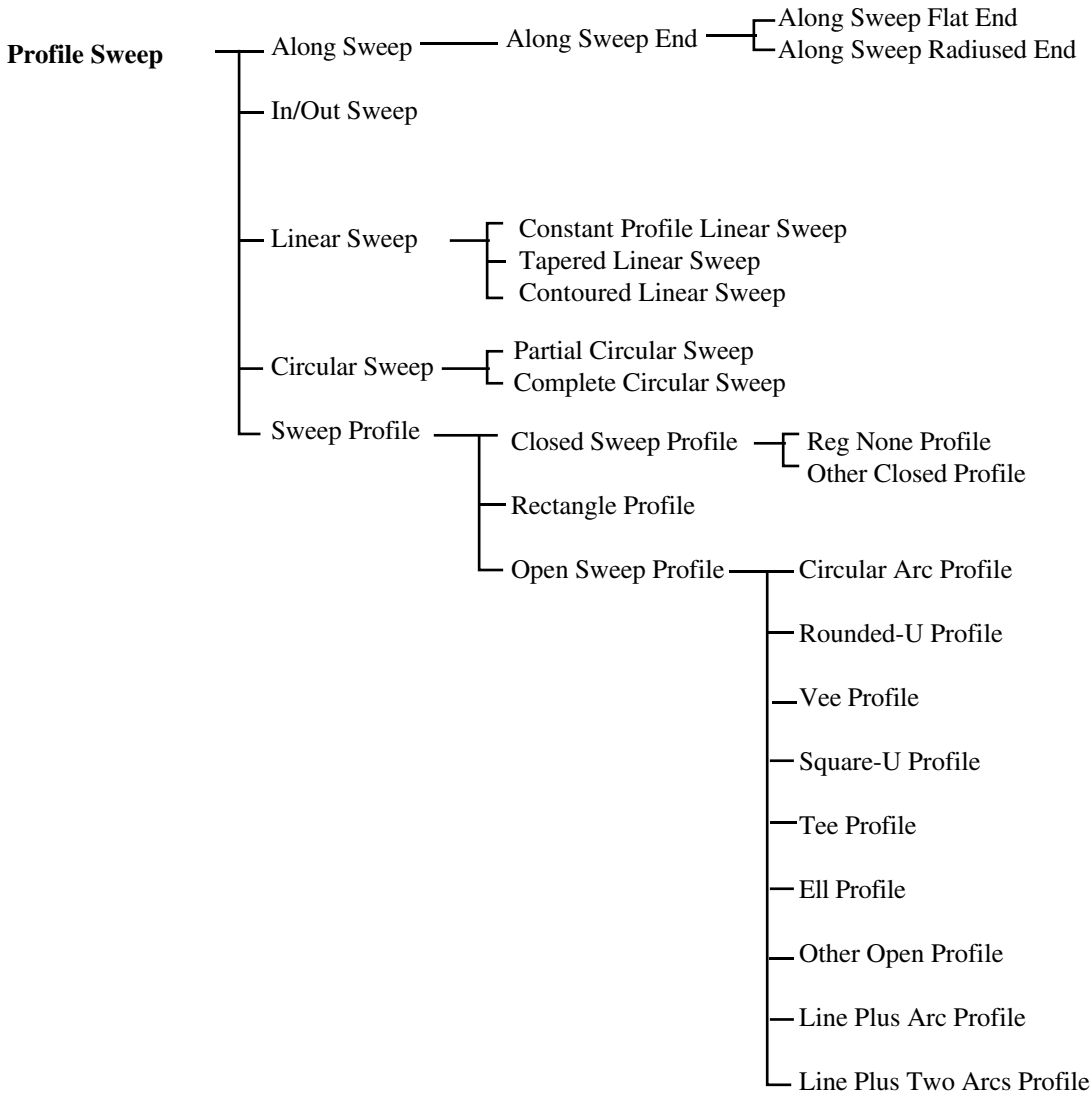


Contoured linear sweep ist ein Subtype von **Linear Sweep**, dessen Profilgröße sich nichtlinear über die Länge der Verschiebung ändert. Bei der Skalierungsfunktion handelt es sich somit um eine positive 2-dimensionale Kurve, deren Parameter entlang der Schieberichtung geändert wird.

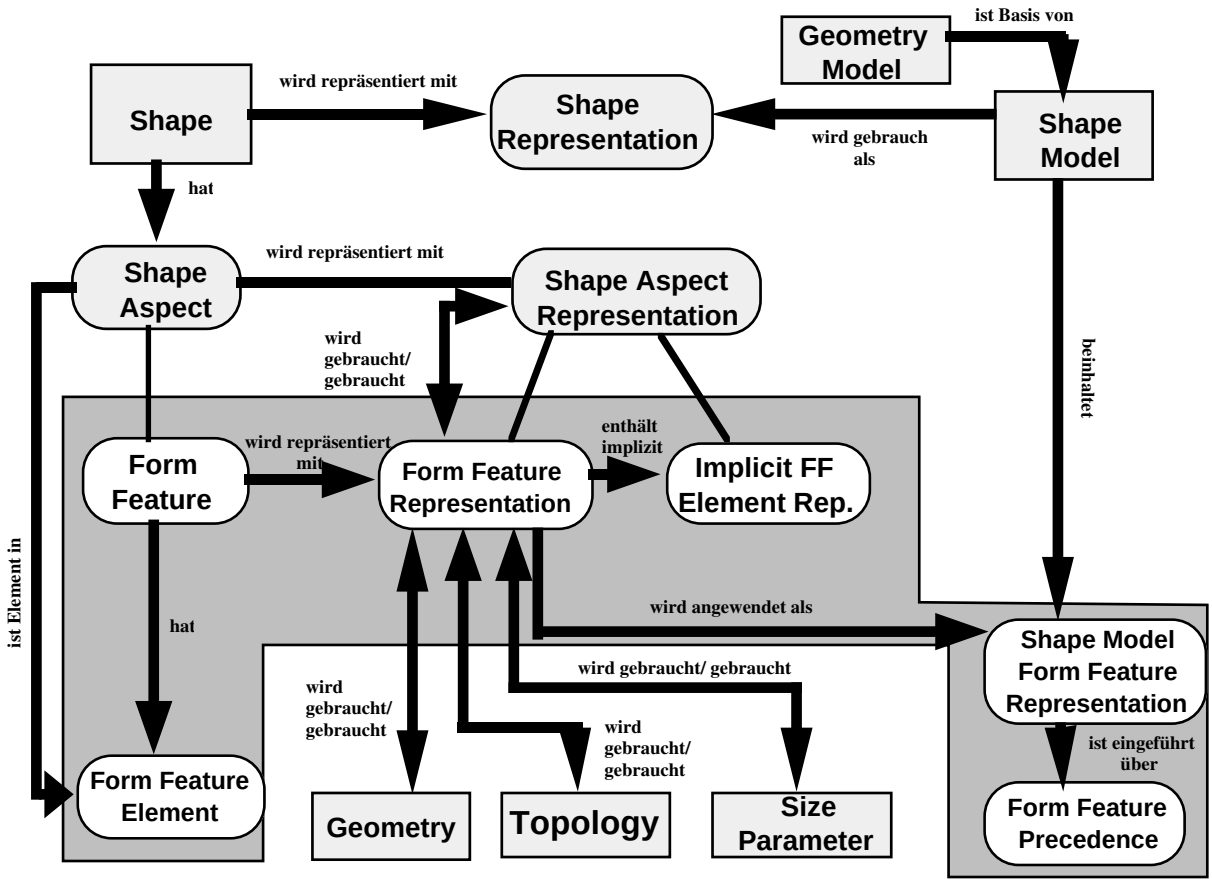
Linear Sweep ist ein Subtype von **Profile Sweep**, dessen Beschreibungsmerkmal eine gerade Linie ist.

Profile Sweep ist eine Repräsentation eines Volumenfeatures als Verschiebung eines ebenen Profils.

Das folgende Bild spiegelt die Struktur eines Teils des FFIM wieder, wobei der gewählte Ausschnitt den oben behandelten Teil **Profile Sweep** wiedergibt.

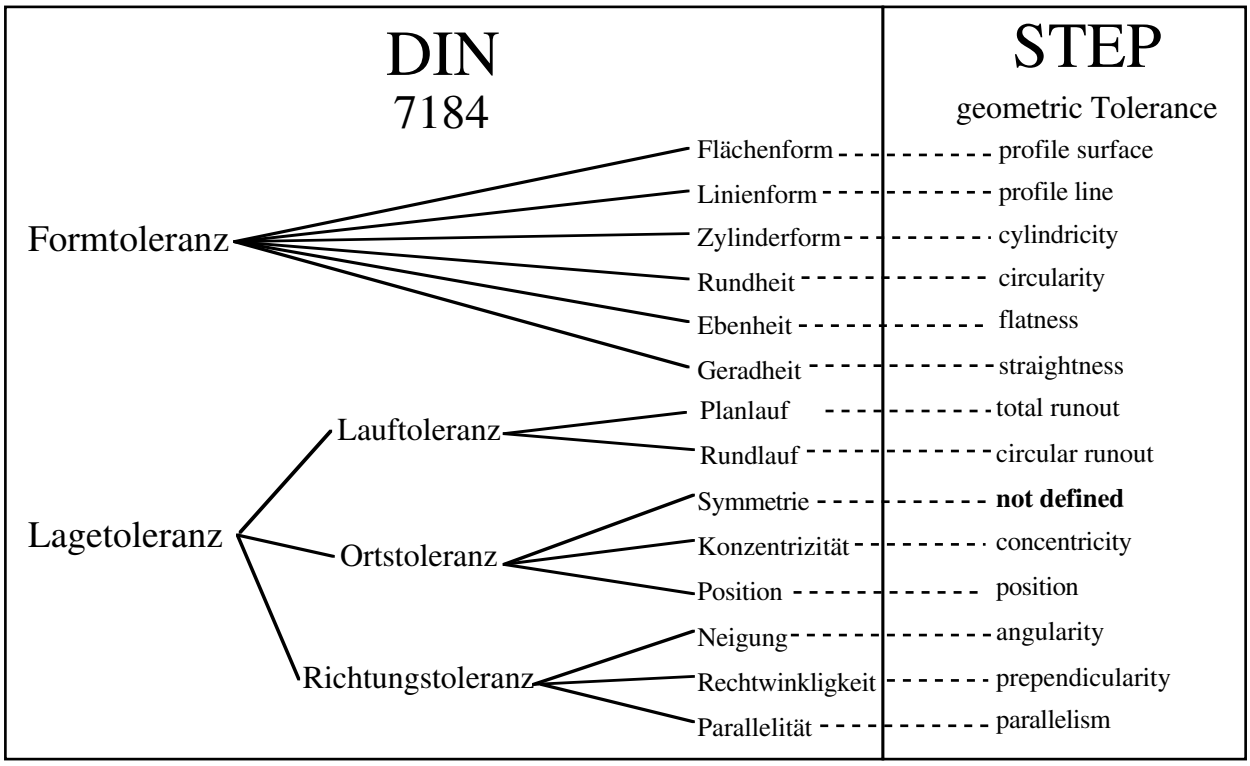


Das folgende Bild gibt einen Überblick über die Einordnung des FFIM in das IPIM von STEP.



2.4 Toleranzenmodell

Im Toleranzmodell von STEP werden die systematischen oder auch zufallsbedingten Abweichungen vom Sollmaß eines technischen Erzeugnisses durch Vorgabe von Grenzmaßen, innerhalb derer die volle Funktionsfähigkeit des Erzeugnisses gewährleistet wird, vorgegeben. Da dieses Modell noch nicht stabil ist und auch noch nicht alle wichtigen Toleranzen berücksichtigt sind, wird ein Überblick über den derzeitigen Stand der Definitionen gegeben. Anschließend wird zur Illustration der Darstellung von Toleranzen in STEP ein Beispiel einer Entity-Definition gegeben



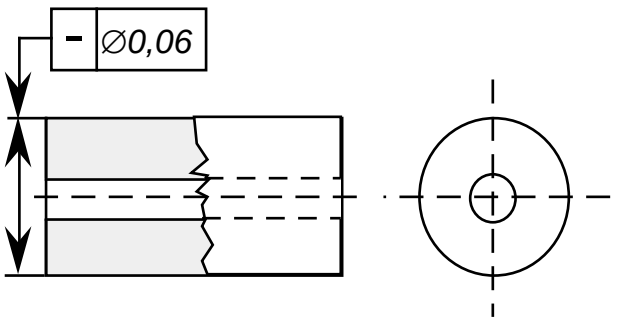
DIN 7182	STEP
Nennmaßtoleranzen	Tolerance range

DIN 7168 T1 Freimaßtoleranzen	STEP **not defined**

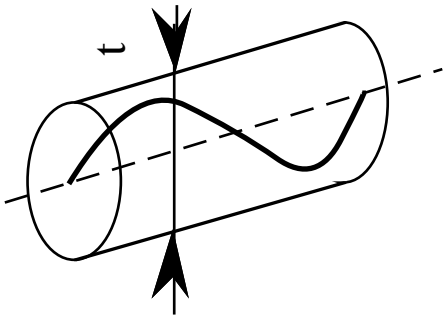
DIN 7168 T2 Allgemeintoleranzen für Lage und Form	STEP **not defined**

DIN 7150 ISO-Toleranzen	STEP **not defined**

Als Beispiel einer Toleranz-Definition in STEP wird nun die Geradheit der Formtoleranz nach DIN 7184 (ISO/R 1101) betrachtet. Nachfolgend ist eine Zeichnungseintragung und die Toleranzzone dargestellt : Die tolerierte Achse des (äußeren) Zylinders muß innerhalb eines Zylinders vom Durchmesser 0,06 mm liegen.

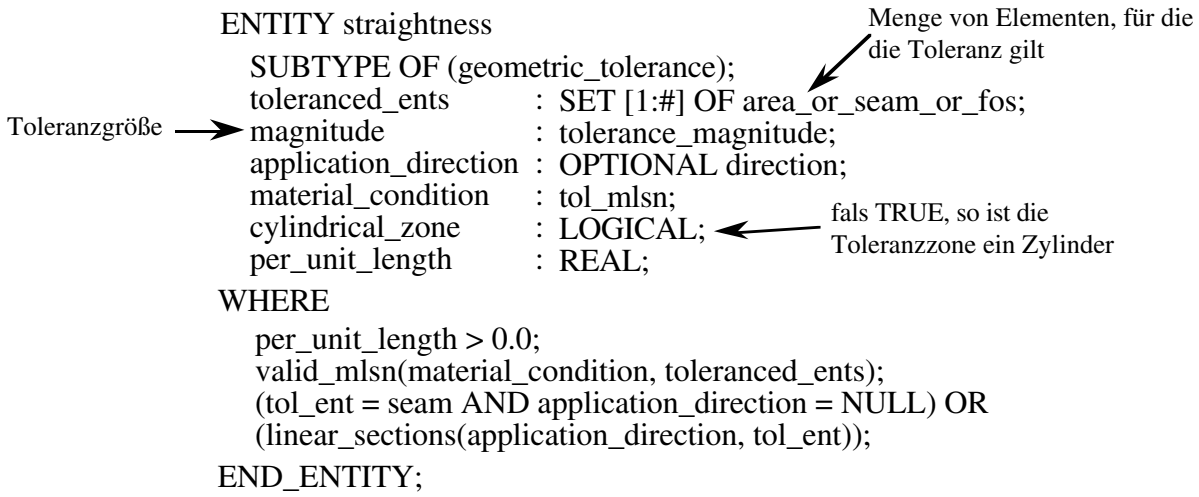


Zeichnungseintragung



Toleranzzone

In STEP kann diese Toleranz durch das Entity *straightness* vom Typ *geometric_tolerance* repräsentiert werden. Es definiert die erlaubte Maßtoleranz für Geraden und Geraden von Oberflächenfeatures. Die Toleranzzone für Geraden sind Zylinder, Parallelepipede und parallele Flächen. Für die Oberflächenfeatures spezifiziert es die Distanz von zwei parallelen Linien, zwischen denen die Gerade der Oberfläche mit einer spezifizierten Richtung liegen muß.




```
ENTITY straightness
   SUBTYPE OF (geometric_tolerance);
   toleranced_ents        : SET [1:#] OF area_or_seam_or_fos;
   magnitude              : tolerance_magnitude;
   application_direction  : OPTIONAL direction;
   material_condition     : tol_mlsn;
   cylindrical_zone       : LOGICAL;
   per_unit_length        : REAL;
WHERE
    per_unit_length > 0.0;
    valid_mlsn(material_condition, toleranced_ents);
    (tol_ent = seam AND application_direction = NULL) OR
    (linear_sections(application_direction, tol_ent));
END_ENTITY;
```

2.5 Materialmodell

Das Materialmodell soll alle werkstoffbeschreibenden Informationen eines Produktes repräsentieren. Die in diesem Partialmodell bisher definierten Materialkennwerte eines Werkstoffes sind die Materialelastizitätsmatrix sowie materialspezifische Koeffizienten, wie etwa Ausdehnungs-, Wärmeübertragungs- und Wärmeleitfähigkeitskoeffizienten. Aus der Sicht der NC-Technik fehlen noch die wichtigsten Werkstoffschlüssel, wie etwa für Härte oder Werkstoffzusammensetzung. Ein Überblick über die bisherige Struktur der Entities findet sich im Anhang.

2.6 Präsentationsmodell

Das Präsentationsmodell definiert Vorschriften zur Visualisierung eines mit STEP übertragenen Produktmodells. Es können Ansichten generiert werden, Schrifttypen festgelegt und vergleichbare Manipulationen zur Sichtbarmachung der Modellinformation durchgeführt werden, um eine verbindliche graphische Darstellung des Produktmodells für den Empfänger zu generieren. Bei dieser Datenübertragung wird das gesamte rechnerinterne Modell und eine Abbildungsvorschrift übertragen.

2.7 Verwaltungsinformationenmodell

Das Verwaltungsinformationenmodell repräsentiert allgemeine Informationen über die organisatorischen Daten, wie etwa der Name des Bearbeiters und der zugehörigen Firma, das Erstellungsdatum, benutzte Maßsysteme und ähnliches.

3 Nominal Shape Schema

Im Nominal Shape Schema wird die tatsächliche Gestalt des Werkstücks repräsentiert. Dazu werden die Entities aus Geometry und Topology verwendet.

Die im Nominal Shape Schema definierten Begriffe beschäftigen sich mit den grundsätzlichen Verfahren, die Gestalt eines Körpers zu beschreiben. Sie dienen damit als Schnittstelle zwischen den Resource Models, mit deren Hilfe die konkreten Beschreibungen erstellt werden, und den Application Models, die auf die Beschreibung des Körpers zugreifen.

Das Nominal Shape Schema definiert das

```
ENTITY shape_model
        SUPERTYPE OF (solid_model XOR
                surface_model XOR
                wireframe_model XOR
                geometric_set);
END_ENTITY;
```

Die Untertypen dieses Entitys beschreiben die grundsätzlich verschiedenen Darstellungsweisen, die vom Shape Schema unterstützt werden.

3.1 Solid Model

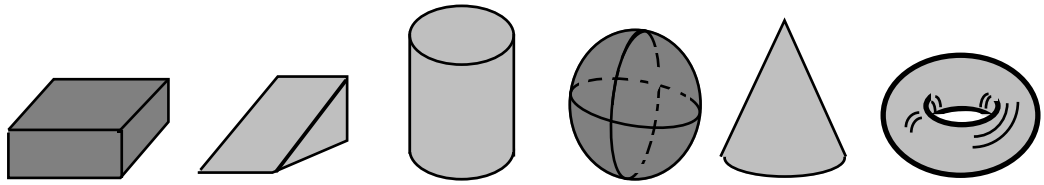
Die erste Möglichkeit, die Gestalt eines Werkstücks darzustellen, trägt die Bezeichnung solid_model (Festkörper- oder Raummodell). Ein solches Raummodell ist eine vollständige Repräsentation der Gestalt eines Werkstücks, so daß für jeden Punkt im Raum angegeben werden kann, ob er innerhalb des Werkstücks, außerhalb oder auf der Begrenzung des Werkstücks liegt.
Die Repräsentation des Raummodells kann auf verschiedene Weise erfolgen:

```
ENTITY solid_model
        SUPERTYPE OF (boolean_expression XOR
                csg_primitive XOR
                csg_solid XOR
                facetted_brep XOR
                half_space XOR
                manifold_solid_brep XOR
                solid_instance XOR
                swept_area_solid)
        SUBTYPE OF (shape_model);
END_ENTITY;
```

Die ersten drei Unterpunkte dieses Entitys gehören dabei zum CSG-Modell von STEP.

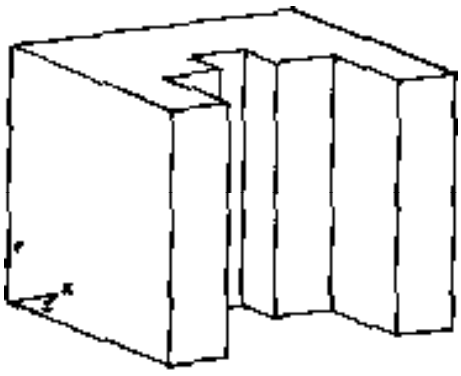
3.1.1 Das CSG Modell

Eine Möglichkeit, Körper in STEP darzustellen, wird durch das CSG-Modell (constructive solid geometry) realisiert. CSG ist eine weitverbreitete Repräsentationsmethode für Körper.
Das CSG-Modell benutzt zwei Kategorien von Entities, geometrische und strukturelle. Die geometrischen Entities sind primitive Körper, die als Basisprimitive für die CSG-Darstellung dienen. Sie umfassen Quader, Keil, Zylinder, Kugel, Torus und "swept surface".

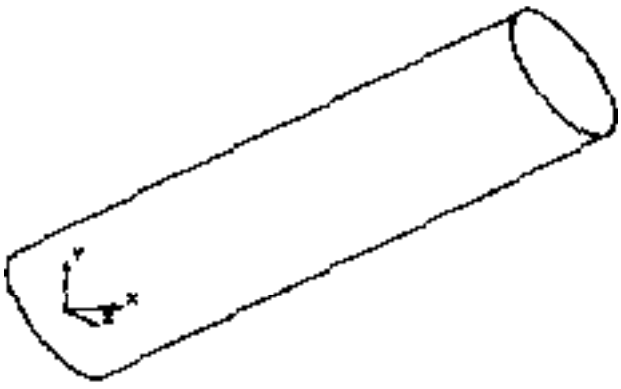


CSG-Basisprimitive

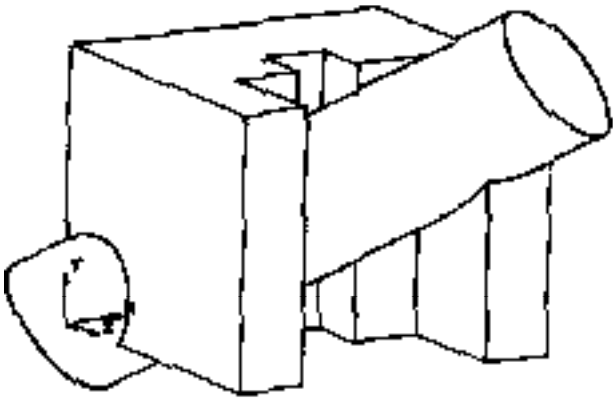
Die strukturellen Entities sind csg_solid und boolean_expression. Die boolean_expression umfassen normalisierte Vereinigung, normalisierten Durchschnitt und normalisierte Mengendifferenz. Sie dienen der Verknüpfung von Körpern, die dabei als Punktmengen aufgefaßt werden.



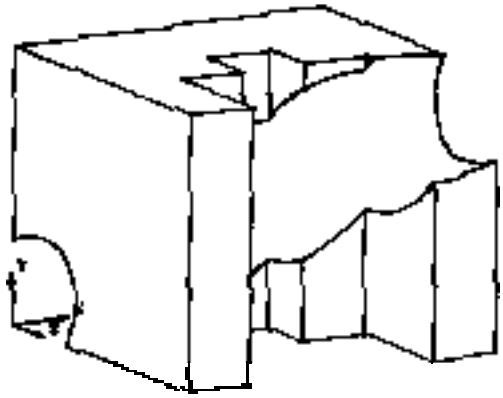
Basiskörper A



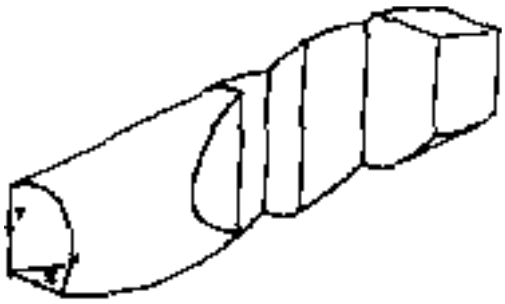
Additivkörper B



Addition A∪B

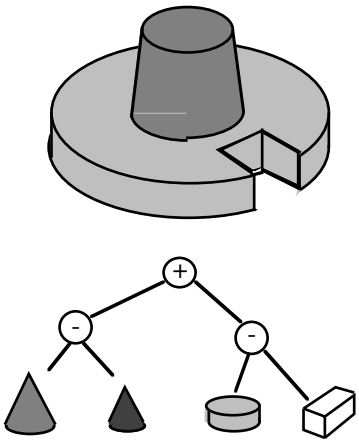


Subtraktion A \ B



Durschnitt A∩B

Das csg_solid verweist dann auf einen aus den boolean_expression und den Basisprimitiven gebildeten azyklischen Graphen. Dieser Graph unterscheidet sich von einem Baum nur dadurch, daß Äste weiter unten wieder in einem Knoten zusammenfließen können. Die Blattknoten des baumähnlichen Graphen werden durch Basisprimitive gebildet, die anderen Knoten sind boolean_expression. Wurzelknoten solcher Graphen beschreiben dann den gesamten repräsentierten Körper, und zwar durch die im Graph dargestellte Kombination der Basisprimitive mit Hilfe der boole'schen Operationen.

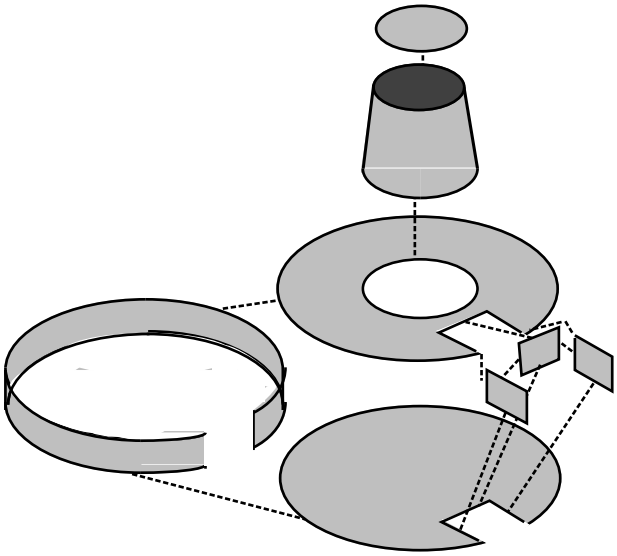


CSG-Darstellung eines Werkstücks

Mit anderen Worten: Im CSG-Modell wird die Gestalt eines Körpers dadurch beschrieben, daß angegeben wird, wie er aus den Basiskörpern zusammengesetzt werden kann.

3.1.2 Das BREP Modell

Die zweite wichtige Möglichkeit, in STEP einen Körper als SOLID zu beschreiben, heißt Boundary Representation (BREP). Der Körper wird beschrieben, indem seine Oberflächen angegeben werden. Dazu wird ein Graph aus Ecken und Kanten erzeugt, die die Oberfläche des Körpers in Faces unterteilen. Diese Beschreibung verwendet die Sprachelemente des Topologiemodells, die einzelnen Faces sind mit Elementen des Geometriemodells beschrieben.



Der gleiche Gegenstand in BREP-Darstellung

Eine Einschränkung des BREP-Modells stellt das facetted brep dar. Eine Reihe existierender Systeme erlaubt nur die Darstellung von Körpern, die durch ebene Flächen begrenzt werden; Rundungen werden dann angenähert. Um den Datenaustausch zwischen solchen Systemen zu ermöglichen, wurde das facetted brep model vorgesehen.

3.2 Unvollständige Darstellungen

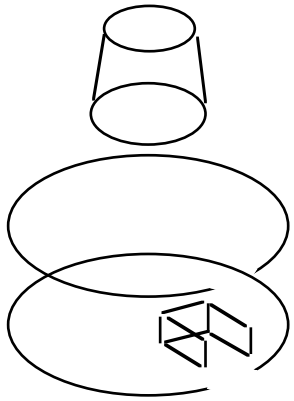
Wenn die Darstellungen des Solid Models angewendet werden, kann für einen beliebigen Punkt gesagt werden, ob er innerhalb oder außerhalb des dargestellten Gegenstandes liegt. Die folgenden Darstellungen beschreiben einen Gegenstand nicht mehr vollständig. Für viele Anwendungen ist solch eine Darstellung jedoch adäquat und ausreichend.

3.2 1 Surface Model

In manchen Fällen wird ein Gegenstand durch Oberflächen beschrieben, die nicht notwendigerweise die vollständige Boundary des Gegenstandes bilden. Für solche Fälle steht das Surface Model zur Verfügung.

3.2.2 Wireframe Model

Eine Wireframe-Representation (Drahtmodell-Repräsentation) eines Körpers beschreibt diesen, indem nur die Kanten beschrieben werden. Die Kanten entstehen als Schnitte der verschiedenen Oberflächen des Körpers.



Wireframe ("Drahtmodell") - Darstellung

3.2.3 Geometric Set

Die Repräsentationsform des Geometric Set dient der Übertragung von Modellen, wenn keine topologischen Informationen vorhanden sind. Die Beschreibung reduziert sich auf eine Menge von Punkten, Kurven und Flächen.

4 Die Anwendungs-Partialmodelle

In den Anwendungs-Partialmodellen (Application Models) werden die Produktinformationen der Basis-Partialmodelle um anwendungsspezifische Definitionen, wie etwa aus den Bereichen Elektronik oder Mechanik, erweitert. Sie beinhalten somit Informationen, die von der Art des Produktes beziehungsweise von der jeweiligen Produktionsstufe abhängen. Ihre Spezifikation ist im allgemeinen noch nicht soweit aus-gearbeitet wie die der Basis-Partialmodelle.

Die Anwendungs-Partialmodelle setzen sich zur Zeit aus dem Draft-Modell (drafting), dem Modell zur Verwaltung von Produktstrukturen (product structure configuration management), dem AEC-Modell (architecture, engineering, construction), dem Elektronikmodell (electrical), dem Modell der FEM[1]-Analyse (analysis) und dem Datentransfermodell (data transfer application) zusammen. Diese werden im folgenden kurz erläutert.

Drafting

Das Draft-Modell dient zur Übertragung von relevanten Zeichnungsinformationen und ist derzeit noch als unstabil zu bezeichnen. Die Produktinformation wird gängigen Normen entsprechend nur symbolisch repräsentiert. Spezifiziert wurden Datentypen und Datenstrukturen, die das Maßbild einer technischen Zeichnung, die Darstellung von technischen Objekten in verschiedenen Ansichten und Schnitten sowie das Zeichnungslayout repräsentieren.

PSCM

Das PSCM-Modell definiert die kennzeichnenden Informationen , die zur Abbildung komplexer Produktstrukturen notwendig sind. Dazu gehört zum Beispiel die Baugruppenhierarchie und auch die Aufnahme von nicht detailliert beschriebenen Zukaufteilen in die Produktstruktur. Das Modell dient somit zur rechnerinternen Produktversionsverwaltung. Aus der Datenstruktur des Modells lassen sich die Stücklisten und die Baugruppenstruktur ableiten.

[1] **F**inite **E**lemente **M**ethode

AEC

Das AEC-Modell bezieht sich auf Anwendungen des Schiffbaus und des Bauwesens. Es wird unterteilt in die Modelle zur Übertragung strukturierter Daten im Schiffbau (Ship Model) und zur Erfüllung bauspezifischer Anforderungen (Core Model).

Electrical

Das Elektronikmodell spezifiziert Modelle zur Beschreibung der Funktionsstruktur von elektronischen Bauteilen. Dabei wird in drei Betrachtungsebenen unterteilt : die Ebene der funktionalen Hierarchie (Topologie), die Ebene der Charakteristik und des Verhalten (Layered Electrical Product) und die Ebene der physikalischen Beschreibung von Leiterplatten (Printed Wiring Board).

Analysis

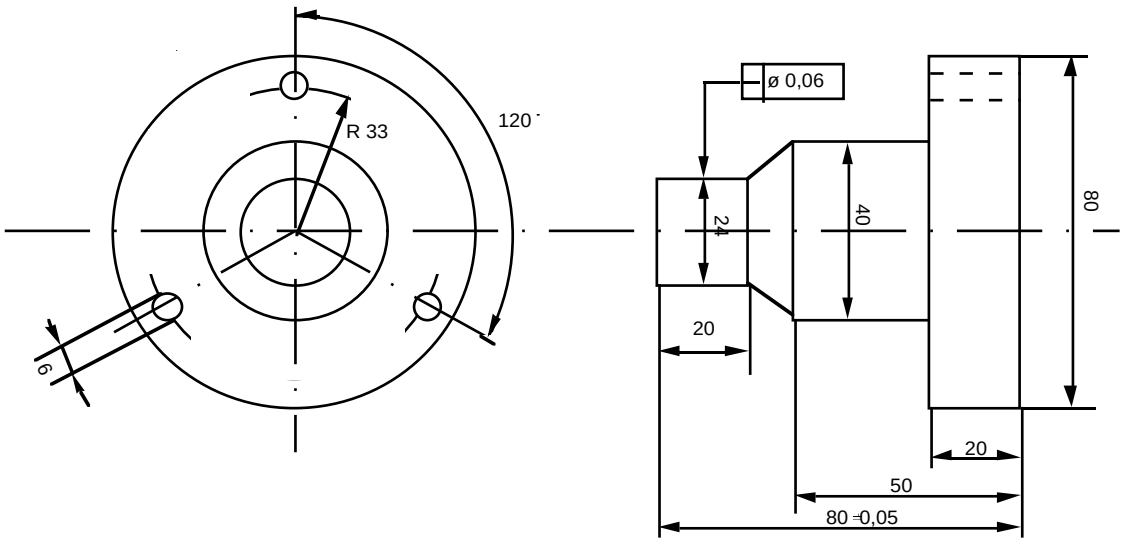
Das Analyse-Modell dient der Analyse eines Produktes.In dem Modell ist bisher nur die FEM-Analyse realisiert worden

Data Transfer

Das Datentransfermodell erlaubt den Zugriff auf externe Informationen, die nicht in dem STEP-Übertragungsfile vorhanden sind. Damit kann zum Beispiel eine Aufforderung zur interaktiven Dateneingabe übermittelt werden, aber auch auf (externe) Tabellen zugegriffen werden.

5 Beispielrepräsentation eines Produktes

Als Beispiel eines STEP-Datenformats wird folgendes Drehteil mit drei Bohrungen in den Modellen für Geometrie und Topologie repräsentiert. Die Form wird durch nachfolgende Skizze angegeben :



Das Beispiel-Drehteil

Die anhand dieses Beispiels erarbeitete Repräsentationen folgt auf den nächsten Seiten.

5.1 Repräsentation des Beispielteils im Geometriemodell

```
@1 = cylindrical_surface(
      radius: 12
      position: #11)
@2 = cylindrical_surface(
      radius: 20
      position: #11)
@3= cylindrical_surface(
      radius: 40
      position: #11)
@4 = cylindrical_surface(
      radius: 3
      position: #25)
@5 = cylindrical_surface(
      radius: 3
      position: #26)
@6 = cylindrical_surface(
      radius: 3
      position: #27)
@7 = conical_surface(
      semi_angle: 32
      radius: 12
      position: #12)
@11 = axis2_placement(
      location: #8
      axis: #10
      ref_direction: #9)
@12 = axis2_placement(
      location: #13
      axis: #10
      ref_direction: #9)
@25 = axis2_placement(
      location: #14
      axis: #81
      ref_direction: #9)
@26 = axis2_placement(
      location: #15
      axis: #81
      ref_direction: #9)
@27 = axis2_placement(
      location: #16
      axis: #81
      ref_direction: #9)
@8 = cartesian_point(
      x_coordinate: 0.0
      y_coordinate: 0.0
      z_coordinate: 0.0)
@9 = default_x_direction(
      x: 1.0
      y: 0.0
      z: 0.0)
```

```
@10 = default_z_direction(
      x: 0.0
      y: 0.0
      z: 1.0)
@13 = cartesian_point(
      x_coordinate: 0.0
      y_coordinate: 0.0
      z_coordinate: 20.0)
@14 = cartesian_point(
      x_coordinate: 33.0
      y_coordinate: 0.0
      z_coordinate: 0.0)
@15 = cartesian_point(
      x_coordinate: -19.8
      y_coordinate: 26.4
      z_coordinate: 0.0)
@16 = cartesian_point(
      x_coordinate: -19.8
      y_coordinate: -26.4
      z_coordinate: 0.0)
@17 = plane(
      position: #20)
@18 = plane(
      position: #21)
@19 = plane(
      position: #22)
@20 = axis2_placement(
      location: #8
      axis: #10
      ref_direction: #9)
@21 = axis2_placement(
      location: #23
      axis: #10
      ref_direction: #9)
@22 = axis2_placement(
      location: #24
      axis: #10
      ref_direction: #9)
@23 = cartesian_point(
      x_coordinate: 0.0
      y_coordinate: 0.0
      z_coordinate: 60.0)
@24 = cartesian_point(
      x_coordinate: 0.0
      y_coordinate: 0.0
      z_coordinate: 80.0)
@28 = intersection_curve(
      surface_s1: #17
      surface_s2: #1
      basis_curve: #40
      self_intersect: False)
@29 = intersection_curve(
      surface_s1: #1
      surface_s2: #7
      basis_curve: #41
      self_intersect: False)
@30 = intersection_curve(
      surface_s1: #7
      surface_s2: #2
      basis_curve: #42
      self_intersect: False)
@31 = intersection_curve(
      surface_s1: #2
      surface_s2: #18
      basis_curve: #43
      self_intersect: False)
@32 = intersection_curve(
      surface_s1: #18
      surface_s2: #3
      basis_curve: #44
      self_intersect: False)
@33 = intersection_curve(
      surface_s1: #3
      surface_s2: #19
      basis_curve: #45
      self_intersect: False)
@34 = intersection_curve(
      surface_s1: #18
      surface_s2: #4
      basis_curve: #46
      self_intersect: False)
@35 = intersection_curve(
      surface_s1: #18
      surface_s2: #5
      basis_curve: #47
      self_intersect: False)
@36 = intersection_curve(
      surface_s1: #18
      surface_s2: #6
      basis_curve: #48
      self_intersect: False)
@37 = intersection_curve(
      surface_s1: #4
      surface_s2: #19
      basis_curve: #49
      self_intersect: False)
@38 = intersection_curve(
      surface_s1: #5
      surface_s2: #19
      basis_curve: #50
      self_intersect: False)
@39 = intersection_curve(
      surface_s1: #6
      surface_s2: #19
      basis_curve: #51
      self_intersect: False)
@40 = circle(
      radius: 12
      position: #52)
@41 = circle(
      radius: 12
      position: #53)
@42 = circle(
      radius: 20
      position: #54)
@43 = circle(
      radius: 20
      position: #55)
@44 = circle(
      radius: 40
      position: #56)
@45 = circle(
      radius: 40
      position: #57)
@46 = circle(
      radius: 3
```

```
      position: #58)
@47 = circle(
      radius: 3
      position: #59)
@48 = circle(
      radius: 3
      position: #60)
@49 = circle(
      radius: 3
      position: #61)
@50 = circle(
      radius: 3
      position: #62)
@51 = circle(
      radius: 3
      position: #63)
@52 = axis2_placement(
      location: #8
      axis: #10
      ref_direction: #9)
@53 = axis2_placement(
      location: #13
      axis: #10
      ref_direction: #9)
@54 = axis2_placement(
      location: #64
      axis: #10
      ref_direction: #9)
@55 = axis2_placement(
      location: #23
      axis: #10
      ref_direction: #9)
@56 = axis2_placement(
      location: #23
      axis: #10
      ref_direction: #9)
@57 = axis2_placement(
      location: #24
      axis: #10
      ref_direction: #9)
@58 = axis2_placement(
      location: #65
      axis: #81
      ref_direction: #9)
@59 = axis2_placement(
      location: #66
      axis: #81
      ref_direction: #9)
@60 = axis2_placement(
      location: #67
      axis: #81
      ref_direction: #9)
@61 = axis2_placement(
      location: #68
      axis: #81
      ref_direction: #9)
@62 = axis2_placement(
      location: #69
      axis: #81
      ref_direction: #9)
@63 = axis2_placement(
      location: #70
      axis: #81
      ref_direction: #9)
@64 = cartesian_point(
      x_coordinate: 0.0
      y_coordinate: 0.0
      z_coordinate: 30.0)
@65 = cartesian_point(
      x_coordinate: 33.0
      y_coordinate: 0.0
      z_coordinate: 60.0)
@66 = cartesian_point(
      x_coordinate: -19.8
      y_coordinate: 26.4
      z_coordinate: 60.0
@67 = cartesian_point(
      x_coordinate: -19.8
      y_coordinate: -26.4
      z_coordinate: 60.0)
@68 = cartesian_point(
      x_coordinate: 33.0
      y_coordinate: 0.0
      z_coordinate: 80.0)
@69 = cartesian_point(
      x_coordinate: -19.8
      y_coordinate: 26.4
      z_coordinate: 80.0)
@70 = cartesian_point(
      x_coordinate: -19.8
      y_coordinate: -26.4
      z_coordinate: 80.0)
@71 = curve_bounded_surface(
      basis_surface: #17
      boundaries: (#28)
      outer_present: True)
@72 = curve_bounded_surface(
      basis_surface: #1
      boundaries: (#28,#29)
      outer_present: True)
@73 = curve_bounded_surface(
      basis_surface: #7
      boundaries: (#29,#30)
      outer_present: True)
@74 = curve_bounded_surface(
      basis_surface: #2
      boundaries: (#30,#31)
      outer_present: True)
@75 = curve_bounded_surface(
      basis_surface: #18
      boundaries:
      (#31,#32,#34,#35,#36)
      outer_present: True)
@76 = curve_bounded_surface(
      basis_surface: #3
      boundaries: (#32,#33)
      outer_present: True)
@77 = curve_bounded_surface(
      basis_surface: #19
      boundaries:
      (#33,#37,#38,#39)
      outer_present: True)
@78 = curve_bounded_surface(
      basis_surface: #4
      boundaries: (#34,#37)
      outer_present: True)
```

```
@79 = curve_bounded_surface(
      basis_surface: #5
      boundaries: (#35,#38)
      outer_present: True)
@80 = curve_bounded_surface(
      basis_surface: #6
      boundaries: (#36,#39)
      outer_present: True)
@81 = direction(
      x: 0.0
      y: 0.0
      z: -1.0)
```

5.2 Repräsentation des Beispielteils im Topologiemodell

```
@101 = edge(
      edge_start: #102
      edge_end: #102
      edge_curve: #103)
@102 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: 0.0
      y_coordinate: 0.0
      z_coordinate: 12.0))
@103 = curve_logical_structure(
      curve_element: #28
      flag: true)
@104 = edge(
      edge_start: #105
      edge_end: #105
      edge_curve: #106)
@105 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: 0.0
      y_coordinate: 0.0
      z_coordinate: 32.0))
@106 = curve_logical_structure(
      curve_element: #29
      flag: true)
@107 = edge(
      edge_start: #108
      edge_end: #108
      edge_curve: #109)
@108 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: 0.0
      y_coordinate: 0.0
      z_coordinate: 50.0))
@109 = curve_logical_structure(
      curve_element: #30
      flag: true)
@110 = edge(
      edge_start: #111
      edge_end: #111
      edge_curve: #112)
@111 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: 0.0
      y_coordinate: 0.0
      z_coordinate: 80.0))
@112 = curve_logical_structure(
      curve_element: #31
      flag: true)
@113 = edge(
      edge_start: #114
      edge_end: #114
      edge_curve: #115)
@114 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: 0.0
      y_coordinate: 0.0
      z_coordinate: 100.0))
@115 = curve_logical_structure(
      curve_element: #32
      flag: true)
@116 = edge(
      edge_start: #117
      edge_end: #117
      edge_curve: #118)
@117 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: 0.0
      y_coordinate: 0.0
      z_coordinate: 120.0))
@118 = curve_logical_structure(
      curve_element: #33
      flag: true)
@119 = edge(
      edge_start: #120
      edge_end: #120
      edge_curve: #121)
@120 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: 36.0
      y_coordinate: 0.0
      z_coordinate: 60.0))
@121 = curve_logical_structure(
      curve_element: #34
      flag: false)
@122 = edge(
      edge_start: #123
      edge_end: #123
      edge_curve: #124)
@123 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: -16.8
      y_coordinate: 26.4
      z_coordinate: 60.0))
@124 = curve_logical_structure(
      curve_element: #35
      flag: false)
@125 = edge(
      edge_start: #126
      edge_end: #126
      edge_curve: #127)
@126 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: -16.8
      y_coordinate: -26.4
```

```
      z_coordinate: 60.0))
@127 = curve_logical_structure(
      curve_element: #36
      flag: false)
@128 = edge(
      edge_start: #129
      edge_end: #129
      edge_curve: #130)
@129 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: 36.0
      y_coordinate: 0.0
      z_coordinate: 80.0))
@130 = curve_logical_structure(
      curve_element: #37
      flag: false)
@131 = edge(
      edge_start: #132
      edge_end: #132
      edge_curve: #133)
@132 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: -16.8
      y_coordinate: 26.4
      z_coordinate: 80.0))
@133 = curve_logical_structure(
      curve_element: #38
      flag: false)
@134 = edge(
      edge_start: #135
      edge_end: #135
      edge_curve: #136)
@135 = vertex(vertex_point:
      cartesian_point(
      x_coordinate: -16.8
      y_coordinate: -26.4
      z_coordinate: 80.0))
@136 = curve_logical_structure(
      curve_element: #39
      flag: false)
@137 = vertex(
      vertex_point: #8)
@138 = vertex(
      vertex_point: #13)
@139 = vertex(
      vertex_point: #14)
@140 = vertex(
      vertex_point: #15)
@141 = vertex(
      vertex_point: #16)
@142 = vertex(
      vertex_point: #23)
@143 = vertex(
      vertex_point: #24)
@144 = vertex(
      vertex_point: #64)
@145 = vertex(
      vertex_point: #65)
@146 = vertex(
      vertex_point: #66)
@147 = vertex(
      vertex_point: #67)
@148 = vertex(
      vertex_point: #68)
@149 = vertex(
      vertex_point: #69)
@150 = vertex(
      vertex_point: #70)
@151= surface_logical_structure(
      surface_element: #72
      flag: true)
@152= surface_logical_structure(
      surface_element: #73
      flag: true)
@153= surface_logical_structure(
      surface_element: #74
      flag: true)
@154= surface_logical_structure(
      surface_element: #76
      flag: true)
@155= surface_logical_structure(
      surface_element: #78
      flag: false)
@156= surface_logical_structure(
      surface_element: #79
      flag: false)
@157= surface_logical_structure(
      surface_element: #80
      flag: false)
@158= surface_logical_structure(
      surface_element: #71
      flag: true)
@159= surface_logical_structure(
      surface_element: #75
      flag: true)
@160= surface_logical_structure(
      surface_element: #77
      flag: true)
@161 = edge_logical_structure(
      edge_element: #101
      flag: true)
@162 = edge_logical_structure(
      edge_element: #104
      flag: true)
@163 = edge_logical_structure(
      edge_element: #107
      flag: true)
@164 = edge_logical_structure(
      edge_element: #110
      flag: true)
@165 = edge_logical_structure(
      edge_element: #113
      flag: true)
@166 = edge_logical_structure(
      edge_element: #116
      flag: true)
@167 = edge_logical_structure(
      edge_element: #119
      flag: false)
@168 = edge_logical_structure(
      edge_element: #122
      flag: false)
@169 = edge_logical_structure(
      edge_element: #125
      flag: false)
@170 = edge_logical_structure(
```

```
      edge_element: #128
      flag: false)
@171 = edge_logical_structure(
      edge_element: #131
      flag: false)
@172 = edge_logical_structure(
      edge_element: #134
      flag: false)
@173 = edge_loop(
      loop_edges: (#161))
@174 = edge_loop(
      loop_edges: (#162))
@175 = edge_loop(
      loop_edges: (#163))
@176 = edge_loop(
      loop_edges: (#164))
@177 = edge_loop(
      loop_edges: (#165))
@178 = edge_loop(
      loop_edges: (#166))
@179 = edge_loop(
      loop_edges: (#167))
@180 = edge_loop(
      loop_edges: (#168))
@181 = edge_loop(
      loop_edges: (#169))
@182 = edge_loop(
      loop_edges: (#170))
@183 = edge_loop(
      loop_edges: (#171))
@184 = edge_loop(
      loop_edges: (#172))
@185 = loop_logical_structure(
      loop_element: #173
      flag: true)
@186 = loop_logical_structure(
      loop_element: #174
      flag: true)
@187 = loop_logical_structure(
      loop_element: #175
      flag: true)
@188 = loop_logical_structure(
      loop_element: #176
      flag: true)
@189 = loop_logical_structure(
      loop_element: #177
      flag: true)
@190 = loop_logical_structure(
      loop_element: #178
      flag: true)
@191 = loop_logical_structure(
      loop_element: #179
      flag: true)
@192 = loop_logical_structure(
      loop_element: #180
      flag: true)
@193 = loop_logical_structure(
      loop_element: #181
      flag: true)
@194 = loop_logical_structure(
      loop_element: #182
      flag: true)
@195 = loop_logical_structure(
      loop_element: #183
      flag: true)
@196 = loop_logical_structure(
      loop_element: #184
      flag: true)
@197 = face(
      outer_bound: #
      bounds: (#185, #186)
      face_surface: #151)
@198 = face(
      outer_bound: #
      bounds: (#186, #187)
      face_surface: #152)
@199 = face(
      outer_bound: #
      bounds: (#187, #188)
      face_surface: #153)
@200 = face(
      outer_bound: #
      bounds: (#189, #190)
      face_surface: #154)
@201 = face(
      outer_bound: #
      bounds: (#191, #194)
      face_surface: #155)
@202 = face(
      outer_bound: #
      bounds: (#192, #195)
      face_surface: #156)
@203 = face(
      outer_bound: #
      bounds: (#193, #196)
      face_surface: #157)
@204 = face(
      outer_bound: #185
      bounds: (#185)
      face_surface: #158)
@205 = face(
      outer_bound: #189
      bounds: (#188, #189, #191,
                  #192, #193)
      face_surface: #159)
@206 = face(
      outer_bound: #190
      bounds: (#190, #194, #195,
                  #196)
      face_surface: #160)
@207 = face_logical_structure(
      face_element: #197
      flag: true)
@208 = face_logical_structure(
      face_element: #198
      flag: true)
@209 = face_logical_structure(
      face_element: #199
      flag: true)
@210 = face_logical_structure(
      face_element: #200
      flag: true)
@211 = face_logical_structure(
      face_element: #201
      flag: false)
@212 = face_logical_structure(
```

```
      face_element: #202
      flag: false)
@213 = face_logical_structure(
      face_element: #203
      flag: false)
@214 = face_logical_structure(
      face_element: #204
      flag: true)
@215 = face_logical_structure(
      face_element: #205
      flag: true)
@216 = face_logical_structure(
      face_element: #206
      flag: true)
@217 = open_shell(
      shell_boundary: (##207,
            #208, #209, #210,
            #211, #212, #213,
            #214, #215, #216))
@218 = open_shell(
      shell_boundary:   (#207,
#208,            #209,   #213,
#214,            #215, #216))
@219 = open_shell(
      shell_boundary: (#210))
@225 = open_shell(
      shell_boundary: (#211))
@226 = open_shell(
      shell_boundary: (#212))
@220 = shell_logical_structure(
      shell_element: #218
      flag: true)
@221 = shell_logical_structure(
      shell_element: #219
      flag: false)
@227 = shell_logical_structure(
      shell_element: #225
      flag: false)
@228 = shell_logical_structure(
      shell_element: #226
      flag: false)
@222 = shell_logical_structure(
      shell_element: #217
      flag: false)
@223 = region(
      outer_region_boundary: #
      region_boundaries:  (#220,
            #221, #227, #228))
@224 = region(
      outer_region_boundary: #
      region_boundaries: (#222))
```

6. Literatur

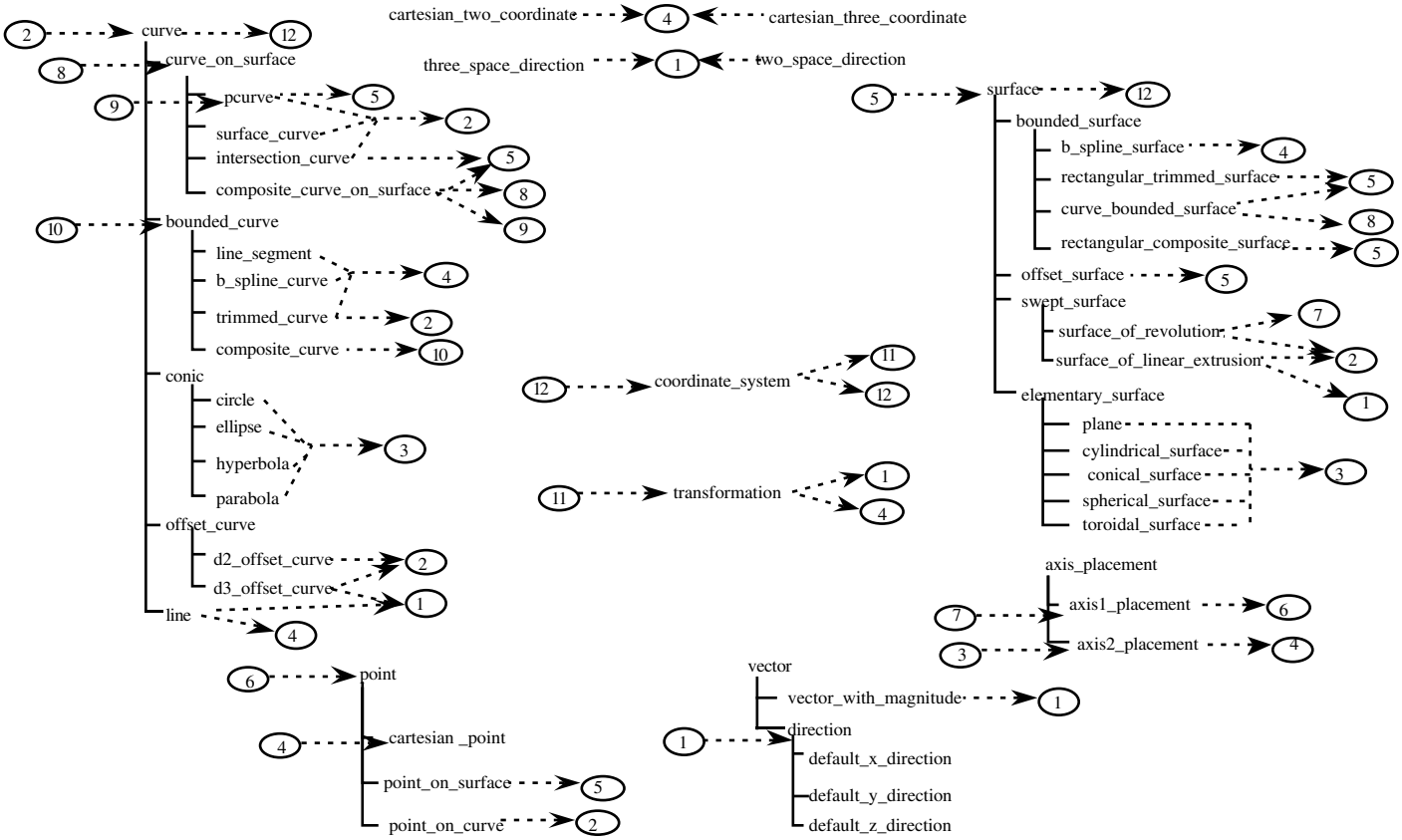
Anderl, R., Schilli, B.: *STEP - Eine Schnittstelle zum Austausch Integrierter Modelle.* In : CAD-Datenaustausch und Datenverwaltung - Schnittstellen in Architektur, Bauwesen und Maschinenbau. Berlin 1988

Anderl, R., Grabowski, H., Schilli, B. , Schmitt, M.: *STEP - Entwicklung einer Schnittstelle zum Produktdatenaustausch.* VDI-Z 131 (1989), Nr. 9 - Spetember 1989, Seiten 68-76

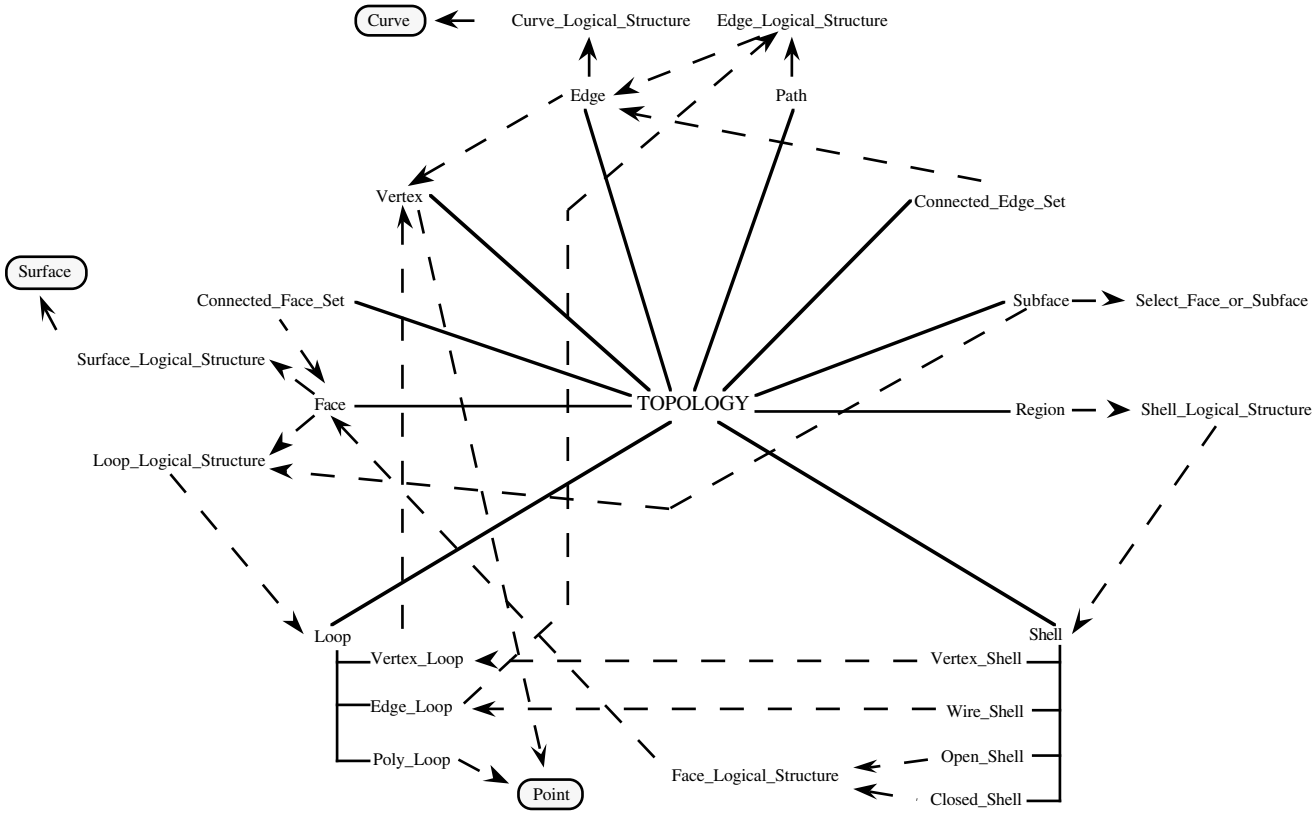
Holland, M. , Prengemann, U., Marczinski, G.., Mittmann, B.: *Anwendungsorientierte Analyse des zukünftigen Schnittstellen-Standards STEP.* ZwF 84 (1989) 8, Seiten 456-461.

STEP Draft Version 1.0

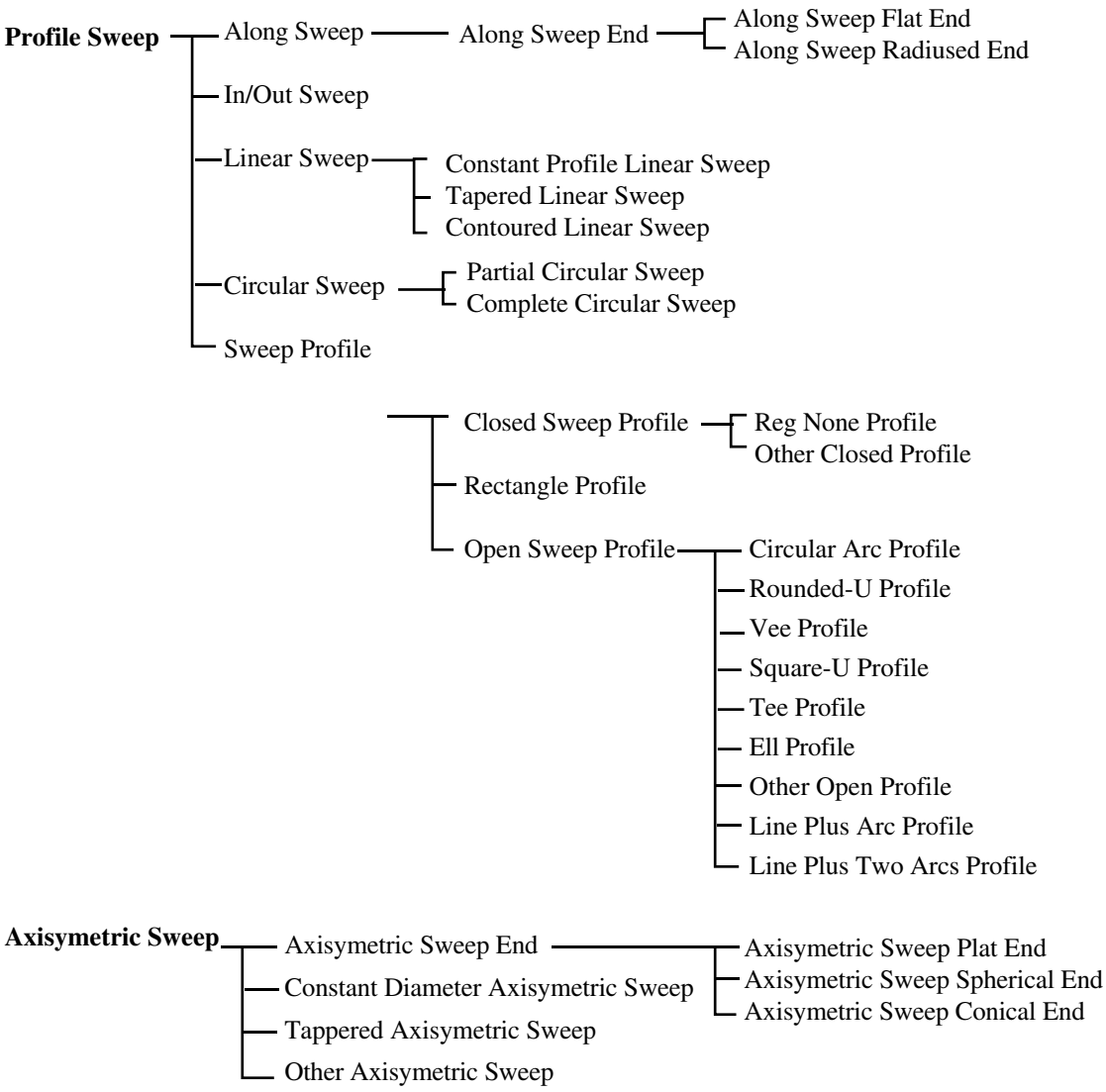
Anhang: Strukturierung der Elemente vom Typ Geometry



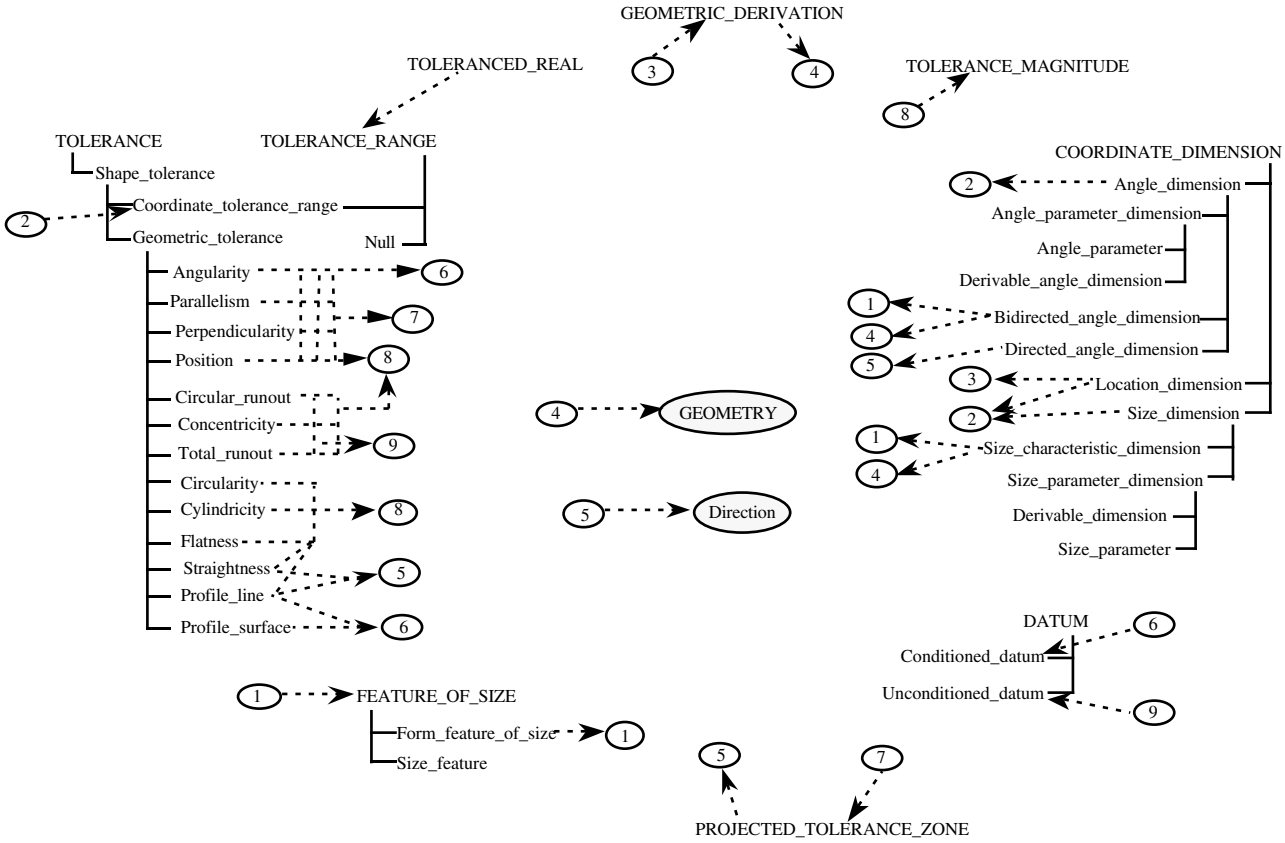
Strukturierung der Elemente vom Typ Topology



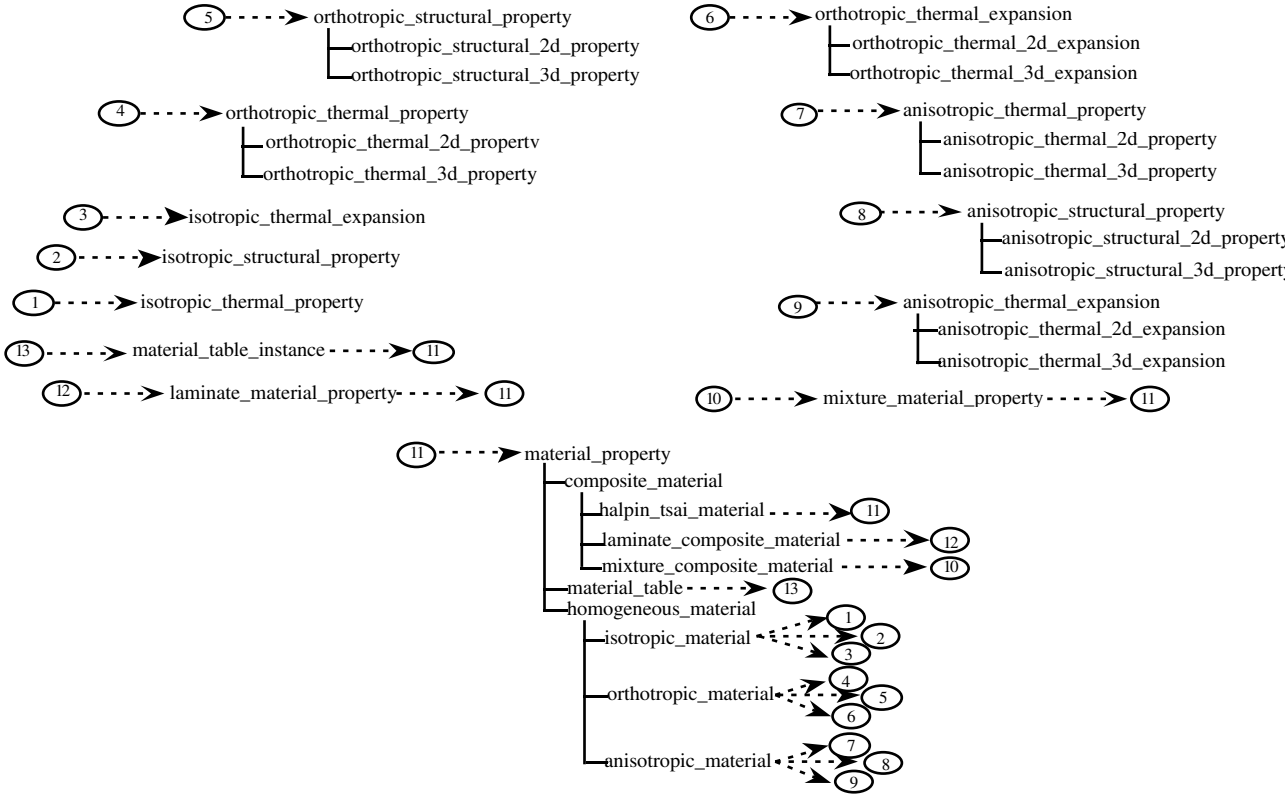
Beispielstrukturierung von Elementen vom Typ Form-Feature



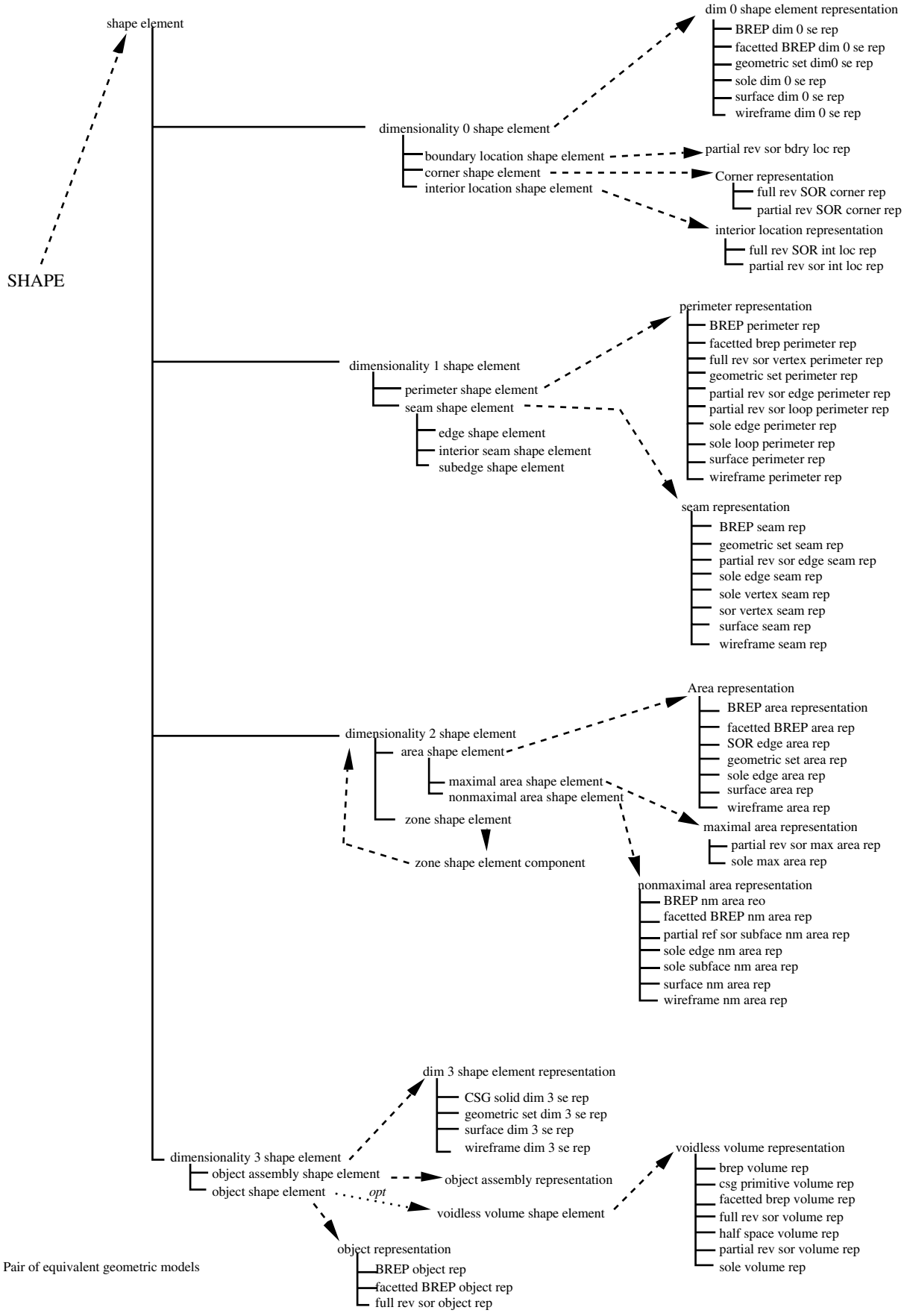
Strukturierung der Elemente vom Typ Tolerance



Strukturierung der Elemente vom Typ Material



Strukturierung der Elemente vom Typ Shape



Pair of equivalent geometric models

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
67608 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder per anonymem ftp von ftp.dfki.uni-kl.de (131.246.241.100) unter pub/Publications bezogen werden.

Die Berichte werden, wenn nicht anders gekenn-zeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or via anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) under pub/Publications.

The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-93-04
Christoph Klauck, Johannes Schwagereit:
GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-05
Franz Baader, Klaus Schulz: Combination Tech-niques and Decision Problems for Disunification
29 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: On Skolemization in Constrained Logics
40 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: Concept Logics with Function Symbols
36 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer: COLAB: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-09
Philipp Hanschke, Jörg Würtz:
Satisfiability of the Smallest Binary Program
8 pages

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf: Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-11
Bernhard Nebel, Hans-Juergen Buerckert:
Reasoning about Temporal Relations:
A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-12
Pierre Sablayrolles: A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-13
Franz Baader, Karl Schlechta:
A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-14
Joachim Niehren, Andreas Podelski,Ralf Treinen: Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster: PLUS - Plan-based User Support
Final Project Report
33 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz: Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-17
Rolf Backofen:
Regular Path Expressions in Feature Logic
37 pages

RR-93-18
Klaus Schild: Terminological Cycles and the Propositional m-Calculus
32 pages

RR-93-20
Franz Baader, Bernhard Hollunder:
Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-22
Manfred Meyer, Jörg Müller:
Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy:
Comparative Study of Connectionist Simulators
20 pages

RR-93-24
Rainer Hoch, Andreas Dengel:
Document Highlighting — Message Classification in Printed Business Letters
17 pages

RR-93-25
Klaus Fischer, Norbert Kuhn: A DAI Approach to Modeling the Transportation Domain
93 pages

RR-93-26
Jörg P. Müller, Markus Pischel: The Agent Architecture InteRRaP: Concept and Application
99 pages

RR-93-27
Hans-Ulrich Krieger:
Derivation Without Lexical Rules
33 pages

RR-93-28
Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker: Feature-Based Allomorphy
8 pages

RR-93-29
Armin Laux: Representing Belief in Multi-Agent Worlds viaTerminological Logics
35 pages

RR-93-30
Stephen P. Spackman, Elizabeth A. Hinkelman:
Corporate Agents
14 pages

RR-93-31
Elizabeth A. Hinkelman, Stephen P. Spackman:
Abductive Speech Act Recognition, Corporate Agents and the COSMA System
34 pages

RR-93-32
David R. Traum, Elizabeth A. Hinkelman:
Conversation Acts in Task-Oriented Spoken Dialogue
28 pages

RR-93-33
Bernhard Nebel, Jana Koehler:
Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-34
Wolfgang Wahlster:
Verbmobil Translation of Face-To-Face Dialogs
10 pages

RR-93-35
Harold Boley, François Bry, Ulrich Geske (Eds.): Neuere Entwicklungen der deklarativen KI-Programmierung — *Proceedings*
150 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

RR-93-36
Michael M. Richter, Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Gabriele Schmidt: Von IDA bis IMCOD: Expertensysteme im CIM-Umfeld
13 Seiten

RR-93-38
Stephan Baumann: Document Recognition of Printed Scores and Transformation into MIDI
24 pages

RR-93-40
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf:
Queries, Rules and Definitions as Epistemic Statements in Concept Languages
23 pages

RR-93-41
Winfried H. Graf: LAYLAB: A Constraint-Based Layout Manager for Multimedia Presentations
9 pages

RR-93-42
Hubert Comon, Ralf Treinen:
The First-Order Theory of Lexicographic Path Orderings is Undecidable
9 pages

RR-93-43
M. Bauer, G. Paul: Logic-based Plan Recognition for Intelligent Help Systems
15 pages

RR-93-44
Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, Martin Staudt: Subsumption between Queries to Object-Oriented Databases
36 pages

RR-93-45
Rainer Hoch: On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition
21 pages

RR-93-46
Philipp Hanschke: A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning
81 pages

RR-93-48
Franz Baader, Martin Buchheit, Bernhard Hollunder: Cardinality Restrictions on Concepts
20 pages

RR-94-01
Elisabeth André, Thomas Rist:
Multimedia Presentations:
The Support of Passive and Active Viewing
15 pages

RR-94-02
Elisabeth André, Thomas Rist:
Von Textgeneratoren zu Intellimedia-Präsentationssystemen
22 pages

RR-94-03
Gert Smolka:
A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 pages

RR-94-05
Franz Schmalhofer, J.Stuart Aitken, Lyle E. Bourne jr.:
Beyond the Knowledge Level: Descriptions of Rational Behavior for Sharing and Reuse
81 pages

RR-94-07
Harold Boley: Finite Domains and Exclusions as First-Class Citizens
25 pages

RR-94-08
Otto Kühn, Björn Höfling: Conserving Corporate Knowledge for Crankshaft Design
17 pages

RR-94-10
Knut Hinkelmann,Helge Hintze:
Computing Cost Estimates for Proof Strategies
22 pages

RR-94-11
Knut Hinkelmann: A Consequence Finding Approach for Feature Recognition in CAPP
18 pages

RR-94-12
Hubert Comon, Ralf Treinen:
Ordering Constraints on Trees
34 pages

DFKI Technical Memos

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh:
A Cognitiv Analysis of Event Structure
21 pages

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben:
The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

TM-93-01
Otto Kühn, Andreas Birk: Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta:
Conlfict Resolving Negotiation for COoperative Schedule Management
21 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer:
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

TM-93-04
Hans-Günther Hein: Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

TM-93-05
Michael Sintek: Indexing PROLOG Procedures into DAGs by Heuristic Classification
64 pages

DFKI Documents

D-93-03
Stephan Busemann, Karin Harbusch(Eds.):
DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

D-93-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1992
194 Seiten

D-93-05
Elisabeth André, Winfried Graf, Jochen Heinsohn, Bernhard Nebel, Hans-Jürgen Profitlich, Thomas Rist, Wolfgang Wahlster:
PPP: Personalized Plan-Based Presenter
70 pages

D-93-06
Jürgen Müller (Hrsg.):
Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz, Saarbrücken, 29. - 30. April 1993
235 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-07
Klaus-Peter Gores, Rainer Bleisinger:
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-08
Thomas Kieninger, Rainer Hoch:
Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

D-93-09
Hans-Ulrich Krieger, Ulrich Schäfer:
TDL ExtraLight User's Guide
35 pages

D-93-10
Elizabeth Hinkelman, Markus Vonerden,Christoph Jung: Natural Language Software Registry
(Second Edition)
174 pages

D-93-11
Knut Hinkelmann, Armin Laux (Eds.):
DFKI Workshop on Knowledge Representation Techniques — Proceedings
88 pages

D-93-12
Harold Boley, Klaus Elsbernd, Michael Herfert, Michael Sintek, Werner Stein:
RELFUN Guide: Programming with Relations and Functions Made Easy
86 pages

D-93-14
Manfred Meyer (Ed.): Constraint Processing – Proceedings of the International Workshop at CSAM'93, July 20-21, 1993
264 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-15
Robert Laux: Untersuchung maschineller Lernverfahren und heuristischer Methoden im Hinblick auf deren Kombination zur Unterstützung eines Chart-Parsers
86 Seiten

D-93-16
Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Gabriele Schmidt: Design & KI
74 Seiten

D-93-20
Bernhard Herbig:
Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus
97 Seiten

D-93-21
Dennis Drollinger:
Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken
53 Seiten

D-93-22
Andreas Abecker: Implementierung graphischer Benutzungsoberflächen mit Tcl/Tk und Common Lisp
44 Seiten

D-93-24
Brigitte Krenn, Martin Volk:
DiTo-Datenbank: Datendokumentation zu Funktionsverbgefügen und Relativsätzen
66 Seiten

D-93-25
Hans-Jürgen Bürckert, Werner Nutt (Eds.):
Modeling Epistemic Propositions
118 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-26
Frank Peters:
Unterstützung des Experten bei der Formalisierung von Textwissen
INFOCOM - Eine interaktive Formalisierungskomponente
58 Seiten

D-94-01
Josua Boon (Ed.):
DFKI-Publications: The First Four Years 1990 - 1993
75 pages