in: Proc. STAK'94, Ilmenau, 1994, S. 135-159

Unterstützung der Ablaufsteuerung in Entwurfsumgebungen durch Versionierung und Konfigurierung

N. Ritter, B. Mitschang, T. Härder, U. Nink

Fachbereich Informatik
Universität Kaiserslautern
67653 Kaiserslautern
e-mail: {ritter / mitsch / haerder / nink }@informatik.uni-kl.de

Zusammenfassung

Eine wesentliche Aufgabe von Entwurfsumgebungen besteht in der Integration von einzelnen, eigenständigen Entwurfswerkzeugen sowohl bzgl. einer gemeinsamen Datenhaltung als auch einer gemeinsamen Ablaufsteuerung. Dies erfordert zum einen die integrierte Verwaltung aller Entwurfsdaten sowie die Bereitstellung der für den werkzeugspezifischen Entwurf relevanten Daten. Versionierung und Konfigurierung stellen hierbei die zentralen Konzepte zur umfassenden Beschreibung der Entwurfsdaten dar. Daneben ist auch die Einbindung der einzelnen Werkzeuganwendungen in den Gesamtablauf zu bewerkstelligen. Hierzu wird eine geeignete Entwurfsablaufsteuerung benötigt. Ihre Aufgaben bestehen vor allem in der Unterstützung einer kontrollierten Kooperation zwischen zusammenarbeitenden Entwerfern, in der Koordination ggf. vorgeplanter Folgen von Werkzeuganwendungen sowie in der Sicherung einer korrekten Interaktion der Werkzeuge mit der Entwurfsdatenverwaltung.

In diesem Artikel beschreiben wir die zentralen Charakteristika der drei grundlegenden Konzepte: Versionierung, Konfigurierung und Ablaufsteuerung. Weiterhin diskutieren wir das Zusammenwirken dieser Konzepte im Rahmen einer Entwurfsumgebung. Dabei kommt deutlich zum Vorschein, daß die Datenbeschreibungsaspekte auf der einen Seite und die Ablaufaspekte auf der anderen Seite sowie deren Zusammenspiel die Eigenschaften einer konkreten Entwurfsumgebung, wie z.B. die Aspekte des parallelen Entwurfs oder die Fehlerbehandlung, entscheidend mitbestimmen.

1. Einleitung

Der Einsatz *integrierter rechnergestützter Entwurfsumgebungen* (engl. "integrated computer-aided design environments") stellt einen wesentlichen Schritt zur Beherrschung der Komplexität des Entwurfs (etwa VLSI-Entwurf oder CAD in den Bereichen Maschinenbau, Architektur und Bau- und Raumwesen) dar. Entwurfsumgebungen bieten eine auf einen konkreten Anwendungsbereich zugeschnittene Infrastruktur für die Anwendung entsprechender CAD-Werkzeuge und unterstützen alle Phasen des Entwurfsprozesses in lückenloser und kontinuierlicher Weise. In diesem Sinne besteht die wesentliche Aufgabe von Entwurfsumgebungen in der Integration von einzelnen, eigenständigen Werkzeugen sowohl bzgl. einer gemeinsamen Datenhaltung als auch einer gemeinsamen Ablaufsteuerung. Dies erfordert im einzelnen:

(1) die integrierte Verwaltung der entwurfsrelevanten Daten sowie die Bereitstellung der werkzeugrelevanten Daten für den werkzeugspezifischen Entwurf,

(2) die Integration der einzelnen Werkzeuganwendungen in einen zielgerichteten Gesamtablauf.

Damit ergibt sich als direkte Konsequenz die Forderung nach entsprechenden Systemkomponenten, die die Infrastruktur zur Erstellung solch angepaßter und integrierter Umgebungen ermöglichen. Es ist dabei die Aufgabe von sog. *CAD-Frameworks* [17], die dazu erforderlichen Basisdienste in einer möglichst allgemeinen und damit wiederverwendbaren Betriebsumgebung bereitzustellen. Im folgenden betrachten wir nur die Basisdienste zur langfristigen Verwaltung aller entwurfs- und werkzeugrelevanten Daten sowie zur Kontrolle der Abläufe innerhalb eines Entwurfsprozesses. Andere, nicht weniger wichtige Basisdienste unterstützen z.B. die Werkzeugintegration oder etwa die Entwicklung spezifischer Benutzerschnittstellen.

Im Gegensatz zu allgemeinen Ablaufsteuerungen [8], [3] konzentrieren wir uns hier auf die spezielle Problematik des werkzeugbasierten Entwurfs und betrachten dementsprechend sog. *Entwurfsablaufsteuerungen*. Damit ergibt sich folgende Konretisierung bzw. Spezialisierung hinsichtlich Daten- und Kontrollfluß:

- Anstatt beliebiger Aktivitäten bzw. Aktoren handelt es sich nun um konkrete CAD-Werkzeuge, die in vorgegebenen Planungsbereichen eingesetzt werden.
- Die Anwendungen dieser Werkzeuge stellen einzelne Entwurfsschritte dar, die im Rahmen der Bearbeitung eines Entwurfs-(Teil-)Auftrages durchzuführen sind.
- Zur Koordination der verschiedenen Aktivitäten werden mit spezieller Semantik belegte Beziehungen berücksichtigt, wie z.B. Kooperationsbeziehungen, Auftragsbeziehungen, Ablaufreihenfolgebeziehungen.
- Datenfluß basiert auf einem Datengranulat, das sich an den unterschiedlichen Verarbeitungskontexten der verschiedenen Aktivitäten orientiert und im wesentlichen Entwurfsdatenzustände beschreibt.

Diese Spezialisierungen kommen explizit zum Ausdruck in unserem CONCORD-Modell [19] zur Entwurfsablaufsteuerung sowie in unserem Objekt-Versions-Datenmodell OVM [12] zur Datenhaltung. Offensichtlich sind Entwurfsablaufsteuerung und Entwurfsdatenverwaltung zwei wesentliche Komponenten, die eng zusammenarbeiten und aufeinander abgestimmt sein müssen. Zielgerichtete Abläufe in komplexen Entwurfsbereichen beruhen darauf, gegebenenfalls auf frühere Entwurfsdatenzustände (Versionen) zurückgreifen zu können, um dann alternative Wege zur Erreichung des gesetzten Entwurfsziels einzuschlagen. Weiterhin erfordert die notwendige Zerlegung komplexer Entwurfsaufgaben geeignete Möglichkeiten zur Synthese der einzelnen Teilergebnisse zu Gesamtergebnissen (Konfigurationen). Kooperation, die ebenfalls zur Bewältigung der Komplexität des Entwurfs unterstützt werden muß, besteht im wesentlichen aus dem Austausch von (vorläufigen) Entwurfsdaten zwischen den zusammenarbeitenden Ablaufeinheiten. Auch hier ist eine explizite Versionierung bzw. Konfigurierung der Entwurfsdaten sehr hilfreich.

Ziel der vorliegenden Arbeit ist es nun, die Zusammenhänge und Abhängigkeiten zwischen den beiden Diensten Entwurfsablaufsteuerung und Entwurfsdatenverwaltung zu analysieren und darauf aufbauend die Rolle von Versionierung und Konfigurierung für die Ablaufsteuerung zu diskutieren. Nachdem ein kleines überschaubares Anwendungsszenario in Kapitel 2 beschrieben ist, werden zuerst das CONCORD-Modell (Kapitel 3) und danach das OVM-Modell (Kapitel 4) vorgestellt. Daran schließt sich in den Kapiteln 5 und 6 eine detaillierte Diskussion der Zusammenhänge und des Zusammenspiels dieser beiden Modelle und der Auswirkungen auf die Datenhaltungskomponente an. Die wichtigen Ergebnisse dieser Analyse werden dann abschließend in Kapitel 7 zusammengefaßt.

2. Anwendungsbereich “Mechanische Konstruktion”

Zur Illustration des in den folgenden Kapiteln diskutierten Zusammenspiels von Entwurfsdatenverwaltung und Entwurfsablaufsteuerung werden wir in diesem Kapitel ein Anwendungsbeispiel aus dem Bereich der mechanischen Konstruktion skizzieren. Um jedoch Begriffsirritationen zu vermeiden, möchten wir vorab betonen, daß wir die in der Maschinenbau-Literatur mit unterschiedlicher Bedeutung belegten Begriffe *Konstruktion* und *Entwurf* [21] im Rahmen dieses Artikels synonym verwenden werden.

Den beiden Hauptaspekten dieser Arbeit folgend, werden wir zunächst eine den Entwurfsablauf bestimmende Entwurfsmethodik skizzieren und danach in diesem Anwendungsbereich vorherrschende Informationstrukturen besprechen.

2.1 Entwurfsmethodik

In [15] findet sich eine Gegenüberstellung der sechs bekanntesten Entwurfsmethodiken für die mechanische Konstruktion. Wir wollen exemplarisch die von Pahl/Beitz vorgeschlagene Methodik heranziehen [16]. Diese unterteilt den Gesamtprozeß in die Phasen ‘Spezifikation’ (der Aufgabenstellung), ‘Konzipieren’ (der Funktionsstrukturen), ‘Gestaltung’ (der technischen Strukturen) und ‘Ausarbeitung’ (der Fertigungsunterlagen). Aus Platzgründen können wir nicht auf alle Phasen näher eingehen, wir wollen daher die Phase der ***Gestaltung*** exemplarisch betrachten. Hier wird das technische Produkt (in der Regel eine Maschine oder Anlage) soweit in seiner Struktur gestaltet, daß anschließend die Fertigungsvorbereitung beginnen kann. Man tut dies durch die evtl. iterierte Ausführung der beiden folgenden Entwurfsaktivitäten, die man sich als Anwendung entsprechender Entwurfswerkzeuge vorstellen kann:

- *Grobgestaltung*
  Aus der Aufgabenbeschreibung werden die gestaltsbestimmenden Anforderungen abgeleitet und zur Erstellung einer ersten Baustruktur herangezogen. Man folgt dabei einer Top-down-Vorgehensweise, so daß ausgehend von dem gesamten Produkt Baugruppen definiert werden, die selbst wieder aus Baugruppen bzw. auf der untersten Ebene aus Einzelteilen zusammengesetzt sind. In **Bild 1** ist die Baugruppenstruktur eines Getriebes vereinfacht dargestellt. Das gesamte Getriebe setzt sich im Beispiel aus zwei Getriebestufen

zusammen, die über eine Welle als Verbindungselement verknüpft sind. Im Baugruppenentwurf wird eine Art “Schnittstelle” für eine Baugruppe spezifiziert, indem etwa geometrische Abmaße oder Anschlußpunkte zu anderen Baugruppen festgelegt werden. Wie dann die interne Realisierung der Baugruppe aussieht (z.B. ob eine Getriebestufe als Stirnrad- oder Planetenradgetriebe realisiert wird), ist davon zunächst unabhängig. Es können

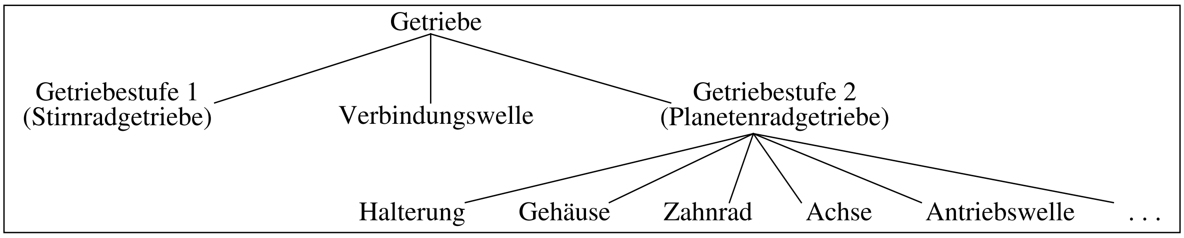

Bild 1: Beispiel einer Baugruppenstruktur für ein Getriebe

mehrere Varianten* für die Subbaugruppen entworfen werden, aus denen dann jeweils eine ausgewählt und damit die übergeordnete Baugruppe “konfiguriert” wird.

- *Feingestaltung der Einzelteile*
  Es werden direkt oder mit Hilfe von Auswahlalgorithmen weitere technische Elemente (i. allg. aus Wiederholteil-, Normteil- oder Kaufteilkatalogen) ausgewählt und damit der Einzelteilentwurf detailliert.

Hier wird bereits deutlich, daß die zu entwerfenden mechanischen Objekte durch sehr komplexe Daten beschrieben sind. Diese *primären Entwurfsdaten* sind in einem anwendungsneutralen Produktmodell zu erfassen [20]. Dabei sind die verschiedenartigen, wesentlichen Eigenschaften in unterschiedlichen Partialmodellen [1] reflektiert (z.B. das geometrische, technische oder technologische Partialmodell). In diesem Papier wollen wir aus Gründen des einfacheren Verständnisses jedoch lediglich einen Ausschnitt des technischen Partialmodells betrachten. Dieser wird im folgenden Abschnitt erläutert.

2.2 Objektstrukturen

Die technischen Objektstrukturen sind in **Bild 2** am Beispiel des Einzelteils ‘Welle’ dargestellt (die durch Pfeile dargestellten Generalisierungsbeziehungen sind auf den Entwurf einer Welle zugeschnitten). Die wichtigsten Entitytypen sind grau hinterlegt und werden im folgenden kurz besprochen.

Baugruppen sind Teilgruppen einer Maschine oder Anlage, die eine bestimmte Funktion erfüllen. Der Aufbau wird durch die Funktion der Maschine oder Anlage bestimmt. Eine Baugruppe kann aus Baugruppen oder Einzelteilen bestehen. *Einzelteile* sind Teile einer Baugruppe und werden entweder aus einer Menge vorgefertigter Teile ausgewählt (Normteile, Kaufteile) oder für einen speziellen Zweck, das Erfüllen einer technischen Funktion, angefertigt (Zeichnungsteil). Ein Einzelteil (Zeichnungsteil) setzt sich aus Hauptelementen zusammen. Ein *Hauptele-*

*) Der Begriff der ‘Variante’ wird in dem vorliegenden Papier nicht vertieft. Es sei an dieser Stelle lediglich gesagt, daß zwei Baugruppen oder Einzelteile zueinander variant sind, falls sie beide herangezogen werden könnten, einen bestimmten strukturellen Bestandteil einer übergeordneten Baugruppe zu bilden. Damit ist für die Variantenbildung die Zeit nicht konstitutiv, was sie von der später in diesem Papier angesprochenen Versionsbildung abgrenzt [22].

ment (z.B. Absatz) bezeichnet ein (z.B. rotationssymmetrisches oder prismatisches) Formelement zur Beschreibung eines Einzelteils mit den dazugehörigen Eigenschaften (z.B. technologische oder geometrische Angaben). Jedem Hauptelement können ein Funktionselement und mehrere Nebenelemente zugeordnet werden, deren Auswahl aus einem Normteilkatalog erfolgen kann. Wie der Name schon sagt, repräsentiert ein *Funktionselement* (z.B. Lager, Paßfeder) die technische Funktion (z.B. Übertragung eines Drehmoments) eines Einzelteils. *Nebenelemente* (z.B. Fase, Freistich) spezifizieren das zugehörige Hauptelement näher. So wird beispielsweise eine Fase an einem Absatz angeordnet, damit in der späteren Fertigung etwa ein Zahnrad auf den Absatz geschoben werden kann.

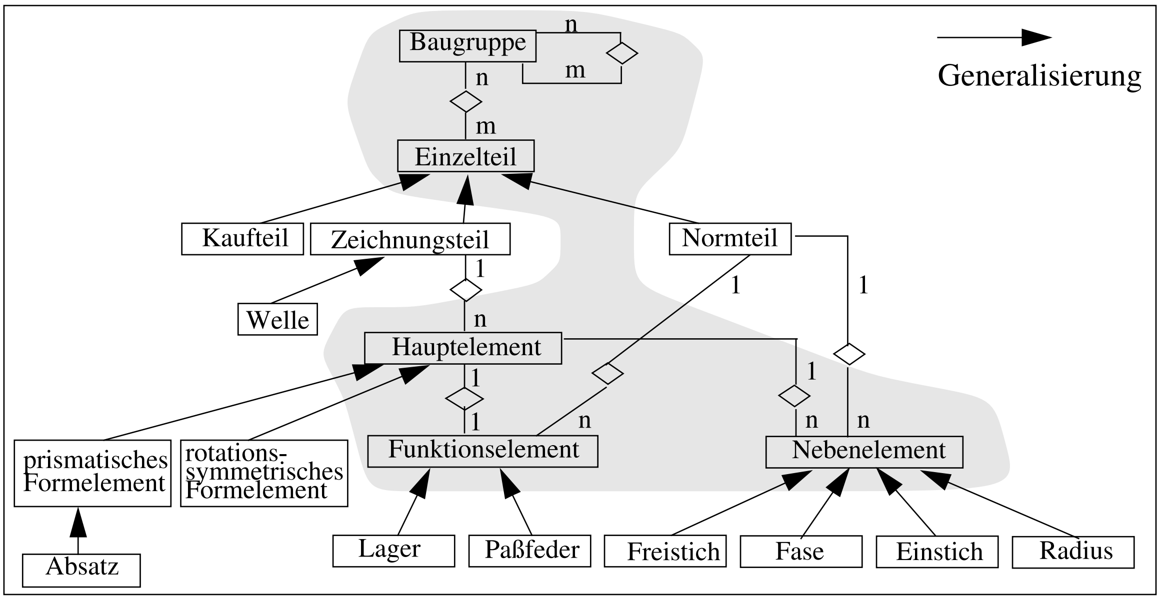

Bild 2: Ausschnitt des technischen Partialmodells

3. Der CONCORD-Ansatz zur Ablaufmodellierung

Das Ziel dieses Kapitels ist das Aufzeigen eines Modellierungsansatzes zur systemseitigen Erfassung der typischen Ablaufstrukturen in einem Entwurfsprozeß, beispielsweise dem der mechanischen Konstruktion (Kapitel 2). Ein Ablaufmodell kann mittels des CONCORD-Ansatzes (CONtrolling COopeRation in Design environments) erstellt werden. Dieser ist in [19] eingehend beschrieben. Im folgenden geben wir einen Überblick.

Der CONCORD-Ansatz erhebt den Anspruch, die wesentlichen Anforderungen von Entwurfsanwendungen zu befriedigen und damit die Komplexität des Entwurfsvorgangs beherrschbar und die Abläufe soweit wie möglich kontrollierbar zu machen, ohne die Kreativität des Entwerfers einzuschränken. Die Charakteristika des Modells reflektieren in direkter Weise die wichtigsten Anwendungsanforderungen:

- Die *hierarchische Verfeinerung und Dekomposition* der komplexen Entwurfsaufgabe in für einzelne Entwerfer handhabbare Teileinheiten (auch *Teilaufträge* genannt) wird unterstützt. Diese Zerlegung orientiert sich an den komplexen Strukturen der Entwurfsobjekte. In entgegengesetzter Richtung wirken die Primitive zur *Komposition (Synthese)* von Teilentwürfen zu vollständigeren Entwurfsergebnissen.

- *Teamorientierung* wird unterstützt durch *Kooperations*primitive, so daß Entwerfer, die unterschiedliche Teilaufträge bearbeiten, ihre Arbeit aufeinander abstimmen können.
- Jeder Teilauftrag ist spezifiziert durch ein lokales Entwurfsziel, und die *zielorientierte Vorgehensweise* in der Bearbeitung des Teilauftrags wird systemseitig unterstützt.
- Zielorientierung drückt sich aus in einer *schrittweisen Verbesserung* von Zwischenergebnissen, die bestimmten methodischen Ablaufmustern folgt.
- Ein Schritt zur Verbesserung eines Zwischenergebnisses wird i.a. durch Anwendung eines Entwurfswerkzeuges getan (z.B. Einzelteilgestaltung in der mechanischen Konstruktion, siehe Kapitel 2). *Werkzeuganwendungen* werden zu atomaren Ablaufeinheiten gekapselt, die die Erzeugung *konsistenter* Daten im Sinne der Datenhaltung (Schemakonsistenz) garantieren.

Eine natürliche Abbildung von Entwurfsabläufen kann durch die Unterscheidung der drei in **Bild 3** illustrierten Abstraktionsebenen erreicht werden, die im folgenden vorgestellt werden.

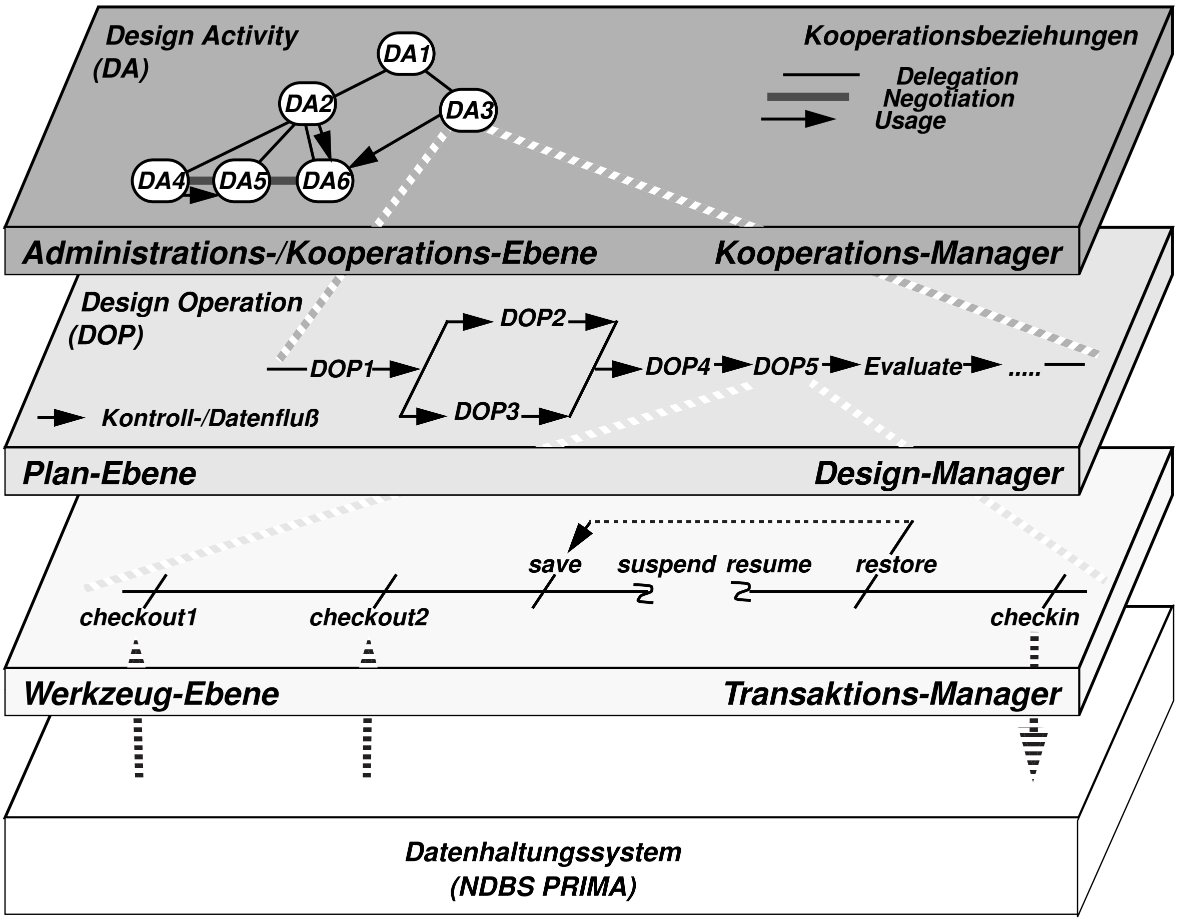

Bild 3: Abstraktionsebenen des CONCORD-Modells

3.1 Administrations-/Kooperations Ebene

Die Konzepte der obersten Ebene (*AC-Ebene*, engl. *administration/cooperation level*) sollen den kreativen und den administrativen Teil der Entwurfsarbeit unterstützen. Das zentrale Konzept ist die ***Design Activity (DA)***. Eine DA ist eine Ablaufeinheit, die alle notwendigen Schritte der Bearbeitung eines (Teil-)Auftrages umfaßt. Während des Entwurfsprozesses kann in dynamischer Weise eine Hierarchie von DAs aufgebaut werden, die eine Hierarchie von (nebenläufig aktiven) Teil-Aufträgen darstellt; idealerweise kann jeder DA jeweils ein Entwerfer zugeordnet werden (concurrent engineering). Der Entwurfsauftrag einer jeden DA ist in Form einer expliziten Spezifikation erfaßt; ferner werden verschiedenartige Kooperationsbeziehungen explizit modelliert. Eine Auftragsbeziehung (Beziehungstyp ***Delegation***) entsteht implizit bei der Auslagerung eines Teil-Auftrages durch Erzeugung einer untergeordneten DA (auch *sub-DA* genannt, während die übergeordnete DA als *super-DA* bezeichnet wird; die oberste DA heißt Top-Level-DA). Delegationsbeziehungen spannen die DA-Hierarchie auf. Kooperationsbeziehungen vom Typ ***Usage*** dienen dem kontrollierten Austausch von (vorläufigen) Entwurfsdaten. Der dritte Beziehungstyp (***Negotiation***) ermöglicht die Abstimmung der Entwurfsspezifikationen (Schnittstellen, technische und physikalische Eigenschaften, etc.) kooperierender DAs.

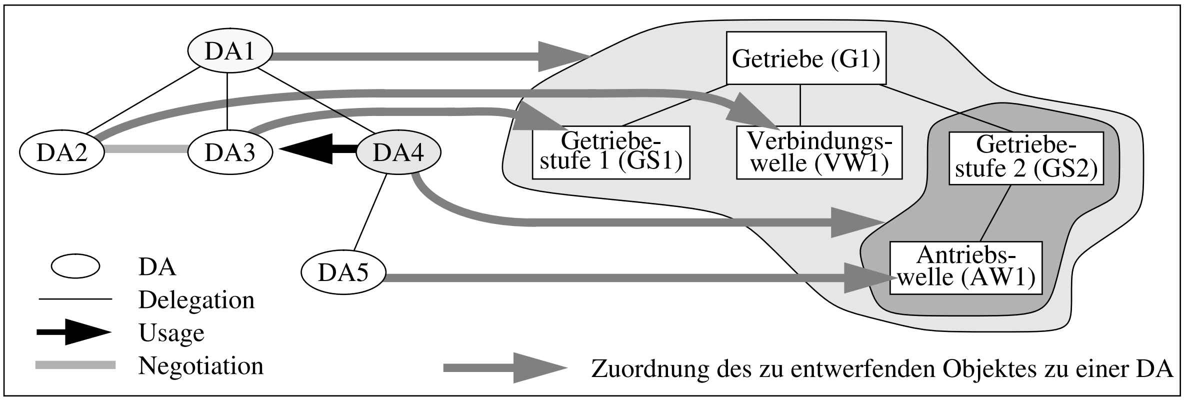

Bild 4: Beispiel einer DA-Hierarchie

Ein **Beispiel** eines Entwurfsprozesses ist in **Bild 4** illustriert. Wir gehen von dem Planungsauftrag, ein Getriebe zu entwerfen, aus. Dieser komplexe Entwurfsauftrag sei spezifiziert durch die Antriebs- und Abtriebsdrehzahl sowie die geometrischen Abmessungen des Getriebes. Der der Top-Level-DA zugeordnete Entwerfer (Chefkonstrukteur) ermittelt zunächst die Baugruppenstruktur des Getriebes G1 und delegiert daraufhin den Entwurf der untergeordneten Baugruppen (GS1, VW1, GS2). Dazu spezifiziert er die Entwurfsziele, die wiederum Antriebs- und Abtriebsdrehzahlen für GS1 und GS2 und die Festlegung der Krafteinteilungsstelle für VW1 sowie die Anschlußpunkte und Abmessungen der mechanischen Objekte beinhalten. Bild 4 illustriert weiter, daß der verantwortliche Entwerfer von DA4 die Einzelteildetaillierung der Antriebswelle an einen weiteren Entwerfer (Spezialisten) delegiert. Stellen wir uns vor, daß die Aufgabenspezifikationen in DA2, DA3 und DA4 Freiheitsgrade bzgl. der Anschlußpunkte enthalten. Daher treten DA2 und DA3 in eine Verhandlung (*Negotiation*) zum genauen Festlegen der gemeinsamen Anschlußpunkte. Eine andere wesentliche Form der Kooperation ist das Lesen eines Vorentwurfes von VW1 durch DA4 (*Usage*). Beispielsweise könnten auf diesem Weg die bereits im Vorentwurf festgelegten Anschlußpunkte zu GS2 von DA4 übernommen werden.

3.2 Plan-Ebene

Die Konzepte der zweiten Abstraktionsebene (*DC-Ebene*, engl. *design control level*) beschreiben die innere Struktur einer DA. Hier steht die Anordnung bestimmter atomarer Entwurfsschritte zwecks Erreichung des Entwurfszieles im Vordergrund (*Workflow*). Die Planebene in Bild 3 zeigt einen Ausführungsplan (***script***) für DA3. Ein solches Skript besteht aus der Spezifikation von Kontroll- und Datenfluß zwischen Werkzeuganwendungen. Die Ablaufeinheit zur Ausführung eines Werkzeuges ist die ***design operation (DOP)***. Es ist sehr wichtig, auf der DC-Ebene geeignete Mechanismen zur Spezifikation des *Workflow's* (*Skripte, Event-Condition-Action-Rules, Constraints*) bereitzustellen, um den Entwerfer einerseits bei der Kontrolle von Routineabläufen zu entlasten, ihn andererseits jedoch beim Treffen der notwendigen, kreativen Entwurfsentscheidungen nicht einzuschränken, sondern zu unterstützen.

Ein sehr einfaches Beispiel für die Anordnung von DOPs zur Einzelteilgestaltung der Antriebswelle AW1 (siehe Bild 4) ist die Nacheinanderausführung eines CAD-Modellierungs-Werkzeuges und eines Finite-Elemente-Programs zur Prüfung der Belastbarkeit der Welle.

3.3 Werkzeug-Ebene

Auf dieser Ebene (*TE-Ebene*, engl. *tool execution level*) betrachten wir einzelne DOPs. Es wurde bereits deutlich, daß wir mit der Einführung des Begriffes DOP von der einzelnen Werkzeuganwendung abstrahieren. Aus der Sicht der Datenhaltungskomponente (Datenbanksystem) ist ein DOP eine klassische Transaktion mit den bekannten ACID-Eigenschaften (Atomizität, Konsistenz, Isolation, Dauerhaftigkeit) [6]. Angesichts der möglichen langen Dauer von Werkzeugläufen halten wir jedoch eine interne Strukturierung mittels *Save/Restore-* und *Suspend/Resume*-Primitiven [7] für angebracht (die Anwendbarkeit dieser Strukturierungskonzepte ist natürlich vom Integrationsgrad des jeweiligen Werkzeugs [17] abhängig). Ein DOP verarbeitet Entwurfsobjekt-Versionen (eine genaue Definition dieses Begriffes wird im folgenden Kapitel gegeben, es genügt an dieser Stelle die intuitive Vorstellung) in drei Schritten. Im ersten Schritt werden Eingabe-Versionen aus der zentralen DB in einen anwendungsnahen Objekt-Puffer ausgelesen (*Checkout*-Operation). Der zweite Schritt besteht in der Verarbeitung der eingelesenen Daten. Der dritte Schritt umfaßt die Propagierung der Änderungsinformation in die DB (*Checkin*-Operation).

3.4 Entwurfsdatenkontexte auf den verschiedenen Abstraktionsebenen

Wir haben nun in den Abschnitten 3.1, 3.2 und 3.3 die Abstraktionsebenen des CONCORD-Ansatzes und die zugehörigen Ablaufeinheiten kennengelernt. Diese haben verschiedene

Sichten auf die Entwurfsdaten. Diese Sichten sollen erläutert werden, bevor im nächsten Kapitel ein Ansatz zur Modellierung und Verarbeitung von Entwurfsdaten angesprochen wird.

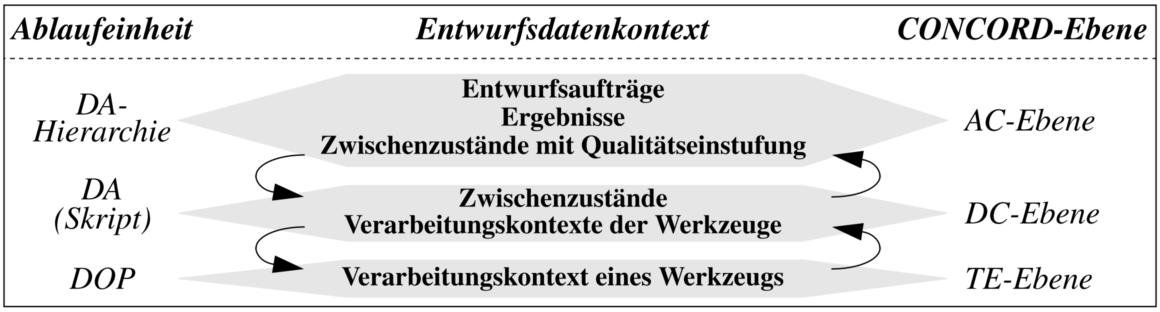

Bild 5: Entwurfsdatenkontexte

Bild 5 zeigt die verschiedenen, den CONCORD-Abstraktionsebenen zuzuordnenden Datensichten. Hinter dieser Zuordnung steht die Frage, welche Daten für die spezifischen Ablaufeinheiten einer Ebene relevant sind und aus welchen Eigenschaften heraus sich diese Relevanz begründet. In diesem Sinne sind auf der AC-Ebene die zwischen DAs fließenden Entwurfsdaten (Außenwirkung der DA), auf der DC-Ebene die innerhalb einer DA, d.h. zwischen DOPs, fließenden Daten (Innenwirkung einer DA) und auf der TE-Ebene der Verarbeitungskontext [5] einer DOP zu betrachten.

Voraussetzung für erfolgreiches Zusammenarbeiten ist es, daß einer DA die Entwurfsaufträge (zu entwerfende Objekte und spezifiziertes Entwurfsziel) aller weiteren DAs der Hierarchie (*AC-Ebene*) bekannt sind. Der kooperative Austausch von Daten zwischen DAs ist sinnvollerweise auf Ergebnisdaten (entlang von *Delegation*-Beziehungen) und solche Entwurfsdatenzustände einzuschränken, die bzgl. ihrer Qualität (Zielerreichungsgrad) eingestuft werden können (*Usage*-Beziehungen). Letzteres erlaubt einer DA, die vorläufige Daten einer anderen DA liest, einzuschätzen, wie stabil diese Daten bereits sind.

Innerhalb einer DA (*DC-Ebene*) sind nicht *nur* solche Zwischenzustände von Bedeutung, die auch sinnvoll von kooperierenden DAs genutzt werden könnten. Auch hier sind zwar Qualitätseinstufungen als Grundlage für das Treffen kreativer Entwurfsentscheidungen wichtig, jedoch geht es darüber hinaus auch darum, die gesamte Entwicklungsgeschichte im Auge zu behalten. Dies beinhaltet auch solche Daten, die keinen definierten Zwischenzustand des von der DA zu entwerfenden Objektes repräsentieren, sondern lediglich zwischen (im Skript) aufeinanderfolgenden DOPs weitergegeben werden. Die Komponente zur Skript-Abarbeitung benötigt in erster Linie die Informationen, welche Werkzeuge zu welchem Zeitpunkt mit welchen Daten aufgerufen werden müssen bzw. bereits aufgerufen wurden. Letzteres ist insbesondere im Fehlerfall oder beim Zurückkehren zu einem früheren Zustand wichtig.

Auf der niedrigsten Abstraktionsstufe (*TE-Ebene*) ist dann nur noch der spezielle Verarbeitungskontext einer DOP relevant, der die Menge der von dem entsprechenden Werkzeug zu bearbeitenden Daten umfaßt.

Wir wollen im folgenden Kapitel einen Modellierungsansatz für Entwurfsdaten untersuchen, der die Repräsentation und die Handhabung der verschiedenen Datenkontexte unterstützt.

4. Das Objekt- und Versionsdatenmodell OVM

In [11], [12] wird ein Objekt- und Versionsdatenmodell (OVM) zur Entwurfsdatenverwaltung beschrieben. Um dessen angemessene Unterstützung für die Belange des CONCORD-Ablaufmodell zu zeigen, genügt eine Beschreibung der zentralen OVM-Eigenschaften in einem OVM-Minimalmodell. **Bild 6** illustriert die Konzepte dieses Minimalmodells.

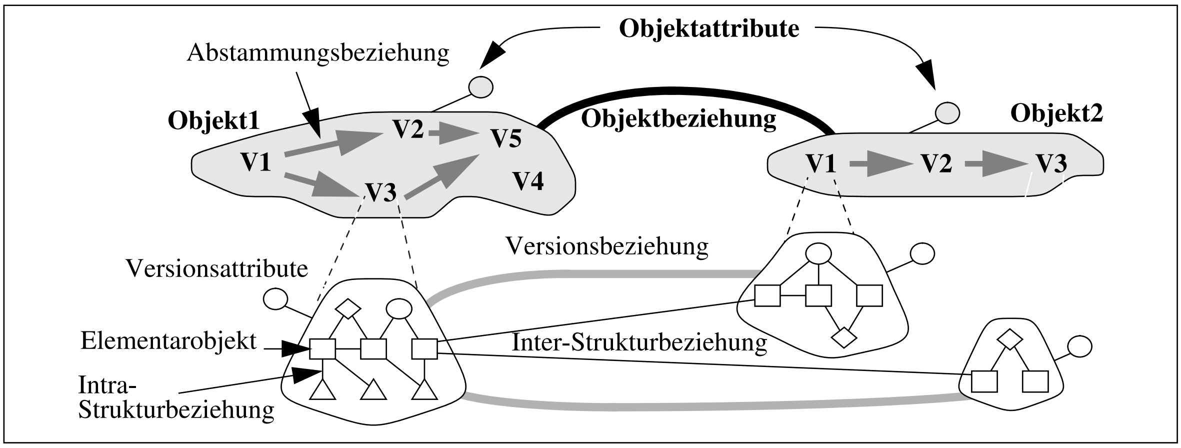

Bild 6: OVM-Konzepte

4.1 Komplexe Objekte und explizite Versionen komplexer Objekte

Komplexe Objekte sind eindeutig benennbare Ausprägungen von komplexen Objekttypen und stellen strukturierte Ansammlungen von elementaren Daten dar. Ein komplexes Objekt ist in seiner Gesamtheit durch eine Reihe von sog. *Objektattributen* beschrieben. Ein Objekt faßt sogenannte *Elementarobjekte* zusammen, die den Tupeln des Relationenmodells entsprechen und in Bild 6 durch kleine Kreise, Quadrate und Rauten symbolisiert sind. Sie sind ebenfalls durch Attribute, die sog. *Elementarobjektattribute*, beschrieben (nicht in Bild 6 dargestellt). Über das Relationenmodell hinausgehend können Elementarobjekte durch sog. (typisierte) *Strukturbeziehungen* verbunden sein. Offensichtlich besteht eine der Hauptaufgaben des Entwurfs darin, 'Inhalte' von Objekten, d.h. Netze von Elementarobjekten (aus der Sicht der Datenhaltung), zu erzeugen. Der Entwurf ist i.a. ein iterativer Prozeß, in dem üblicherweise mehrere ähnliche Elementarobjektnetze entstehen, die wir im folgenden als ***Versionen*** bezeichnen. Versionen sind somit u*nterschiedliche Zustände der im Objekt zusammengefaßten Elementarobjektnetze**. Analog den Objekten können die in Versionen zusammengefaßten Daten in ihrer Gesamtheit durch sog. *Versionsattribute* charakterisiert werden. Erzeuger und Teststatus sind typische Beispiele für Versionsattribute. Die Abhängigkeiten zwischen (Ausgangs-)Versionen und den daraus entwickelten (neuen) Versionen, die die Fortentwicklung der beteiligten Objekte beschreiben, werden in Form sog. *Abstammungsgraphen* repräsentiert. Der Abstammungsgraph ist gerichtet und kann sich als Liste, Baum oder allgemein als azyklischer Graph entwickeln.

*) Demzufolge werden hier explizite, dem Benutzer sichtbare Versionen modelliert, wobei das Komplexobjekt (und *nicht* Elementarobjekt oder Attribut) das Granulat der Versionierung bildet.

4.2 Beziehungen zwischen Objekten und zwischen Versionen

Objektbeziehungen repräsentieren zwischen Objekten bestehende Relationen [23]. Offensichtlich gelten diese Beziehungen auch für die Versionen der entsprechenden Objekte. Das heißt, ***Versionsbeziehungen*** treten als Verfeinerungen der Objektbeziehungen auf. Sie stellen Beziehungen zwischen Versionen dar, deren Objekte durch eine Objektbeziehung verbunden sind. Dies bedeutet, daß Versionsbeziehungen von der Existenz entsprechender Objektbeziehungen abhängig sind bzw. durch diese erst legitimiert werden.

Zusätzlich zu den beschriebenen *expliziten* Verknüpfungen zwischen Versionen können weitere *implizite* Beziehungen vorhanden sein. Sie ergeben sich durch *Datenüberlappungen* als Folge von überlappenden Objekttypdefinitionen. Die Datenüberlappungen können dabei über Elementarobjekte selbst oder über sogenannte *Inter-Strukturbeziehungen* entstehen, die im Gegensatz zu *Intra-Strukturbeziehungen* Objektversionsgrenzen überschreiten. Beziehen wir uns auf Bild 6 und vergegenwärtigen wir uns, daß jede Objektversion das Objekt vollständig beschreibt; dies bedeutet, daß Version 3 von Objekt 1 entweder nur mit Version 1 oder nur mit Version 3 von Objekt 2 in Beziehung stehen darf. Aus diesem Grund setzt man Konfigurationen ein, d.h., *Inter-Strukturbeziehungen dürfen nur innerhalb einer Konfiguration interpretiert werden*. In unserem Fall müßten also zwei Konfigurationen (Objekt 1, Version 3 mit Objekt 2 Version 1 und Objekt 1, Version 3 mit Objekt 2, Version 3) gebildet werden. Offensichtlich beeinflußt die Objektüberlappung wiederum den *Konfigurierungsvorgang* [14], [2], [9].

4.3 Konfigurationen

Wie bereits angesprochen, ist es das Ziel jeder *Konfigurierung*, in einer Menge von (versionierten) Objekten eine Versionsauswahl so zu treffen, daß die ausgewählten Versionen eine konsistente Einheit (Aggregat, Baugruppe), ***Konfiguration*** genannt, bilden. Daher ist eine Konfiguration definiert als eine Ausprägung eines Konfigurationstyps, die aus jeweils einer Ausprägung eines jeden dem Konfigurationstyp zugeordneten Objekttyps genau eine Version auswählt und diese Versionen zu einer (aufgrund von Versionsbeziehungen) kohärenten und konsistenten Einheit zusammenfügt. Wir bezeichnen die einem Konfigurationstyp zugeordneten Objekttypen auch als seine *Komponententypen* und die impliziten Beziehungstypen zwischen Konfigurationstyp und Komponententypen als *Komponentenbeziehungstypen*.

Der eingeführte Konfigurationsbegriff erlaubt die Definition von Konfigurationstypen mit überlappenden Mengen von Komponententypen. Für den Fall, daß alle Komponententypen eines Konfigurationstyps KT1 in der Menge der Komponententypen eines weiteren Konfigurationstyps KT2 enthalten sind, können Ausprägungen von KT1 vollständig in Konfigurationen vom Typ KT2 eingehen. KT1-Konfigurationen können daher als konsistente ***Teilkonfigurationen*** von KT2-Konfigurationen betrachtet werden.

Die bisher besprochenen Konfigurationen sollen als ***explizite Konfigurationen*** bezeichnet werden, da sie Ausprägungen eines explizit definierten Konfigurationstyps des zugrundliegenden OVM-Schemas sind. Der wesentliche Aspekt der Konfigurierung ist jedoch die Bildung von Einheiten, die eine bestimmte Klasse von Konsistenzbedingungen (z. B. alle versionsübergreifenden Bedingungen, siehe folgender Abschnitt) erfüllen; eine Konfigurierung wird verboten, falls eine nicht konsistente Einheit entstehen würde. Ausgehend von diesem Gedanken der Konsistenzeinheit werden wir eine Menge von Versionen, auf der die relevanten Konsistenzbedingungen ebenfalls erfüllt sind, die aber trotzdem keine Konfiguration bildet, als ***implizite Konfiguration*** betrachten, für die kein Konfigurationstyp existieren muß.

4.4 Integrität der in OVM beschreibbaren Datengranulate

Den verschiedenen, mittels der eingeführten OVM-Konzepte beschreibbaren Datengranulaten, können bestimmte logische Konsistenzgrade zugeordnet werden. Zur Erläuterung soll folgende Klassifikation von Integritätsbedingungen hinsichtlich ihrer Reichweite dienen:

(1) Bedingungen, die auf *einer Version* eines Objektes ausgewertet werden können:
Wesentliche Vertreter sind hier die elementarobjektinternen Bedingungen (z.B. Wertebereichsbedingungen zu Attributen) und elementarobjektübergreifende Bedingungen (z.B. Kardinalitätsrestriktionen von Intra-Strukturbeziehungen).

(2) Bedingungen, die auf einer *impliziten Teilkonfiguration* ausgewertet werden können:
Hier sind alle versionsübergreifenden Bedingungen (z.B. Kardinalitätsrestriktionen von Versionsbeziehungen oder Bedingungen auf mehreren durch Versionsbeziehungen verbundenen Versionen) zu betrachten.

(3) Bedingungen, die auf *einer expliziten Konfiguration* auszuwerten sind:
Diese Klasse setzt sich zusammen aus den Bedingungen, die sich aus dem allgemeinen Konfigurationsbegriff ergeben (z.B. die Forderung, daß von einem Objekt höchstens eine Version in eine bestimmte Konfiguration eingehen kann), Bedingungen, die durch die Definition der Komponentenbeziehungstypen gegeben sind (z.B. Kardinalitätsrestriktionen) sowie aus allgemeinen Integritätsbedingungen, die vom Benutzer einem Konfigurationstyp zugeordnet sind.

Die der Stufe 2 und 3 zugeordneten Bedingungen können nur dann erfüllt sein, wenn auch die der darunterliegenden Stufe(n) erfüllt sind. Die gegebene Klassifikation detailliert unser Verständnis der Schemakonsistenz einzelner Versionen (Stufe 1), impliziter Teilkonfigurationen (Stufe 2) und vollständiger expliziter Konfigurationen (Stufe 3);

4.5 Logisches Repräsentationsmodell für Objekte und Versionen

Um im folgenden Kapitel einige Ablaufaspekte anschaulicher diskutieren zu können, soll an dieser Stelle auch das logische Repräsentationsmodell der eingeführten Objekt- und Versionsstrukturen erläutert werden. Die in **Bild 7** gezeigte Darstellung soll verdeutlichen, daß zu jedem Objekt ein Objektrepräsentant angelegt wird. Dieser trägt die Objektattribute sowie beziehungsbeschreibende Attribute (Objektbeziehungen). Weiterhin wird zu jeder Version ein Versionsrepräsentant gespeichert, der mit dem zugehörigen Objektrepräsentanten verbunden ist. Er enthält die Versionsattribute sowie Attribute für Abstammungsbeziehungen und für Beziehungen zu Versionen anderer Objekte (Versionsbeziehungen). Die eigentlichen versionsspezifischen Daten werden von den Elementarobjekten getragen, die mit dem jeweiligen Versionsrepräsentanten verbunden sind und untereinander in strukturellen Beziehungen stehen. Durch die oben angesprochenen Komponentenbeziehungen werden die ausgewählten Versionen als Komponenten mit ihrem Konfigurationsobjekt verbunden.

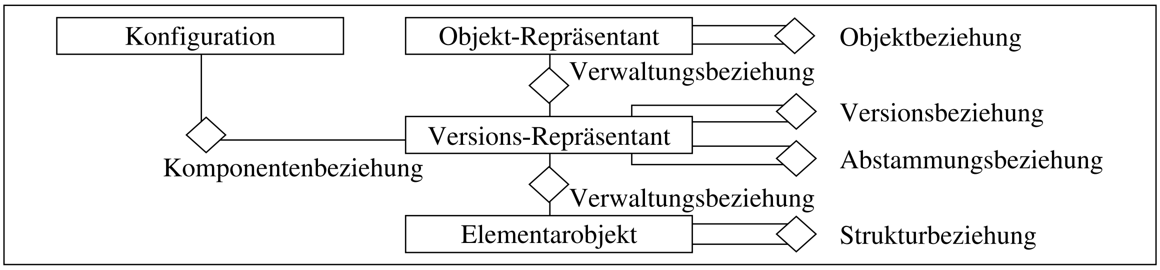

Bild 7: Repräsentationsmodell

4.6 Beispiel

Bild 8 zeigt einen Ausschnitt aus der Definition eines OVM-Schemas, das das in Kapitel 2 eingeführte technische Partialmodell für mechanische Objekte abbildet. Die Sprachelemente der Datendefinitionssprache sind weitestgehend selbsterklärend. Die Definition der Elementarobjekttypen und der Strukturbeziehungstypen ist lediglich angedeutet. Der komplexe Objekttyp *Teil* (Bild 8b), dessen Elementarobjekttypen in der AS-Klausel angegeben werden, dient sowohl der Beschreibung von Einzelteilen als auch von Baugruppen. Daher verbinden Ausprägungen des Objektbeziehungstyps *Aggregation* (Bild 8c) Objekte des Typs *Teil* miteinander, wobei die Rollen *Oberteil* und *Unterteil* zu unterscheiden sind. Dieser Beziehungstyp dient der Beschreibung einer Stücklistenstruktur [23]. Der Konfigurationstyp *Konf* (Bild 8d) besitzt als Komponententyp ebenfalls nur den Objekttyp *Teil*, jedoch ebenfalls in den zwei angesprochenen Rollen. Dabei darf nur jeweils ein *Teil* in der Rolle des *Oberteils* auftreten (Komponentenbeziehungstyp *strukturiert* vom Typ 1:1), während mehrere *Unterteile* (Komponentenbeziehungstyp *ordnet_zu* vom Typ 1:n) zugeordnet werden können.

```
(a)   DEFINE ELEMENTARY_OBJECT_TYPE Hauptelement
          ATTRIBUTES ... ;
      DEFINE STRUCTURAL_RELATIONSHIP_TYPE
          Hauptelement.to_nebenelement (0,*), Nebenelement.to_hauptelement (0,1);

(b)   DEFINE OBJECT_TYPE Teil AS Hauptelement, Funktionselement, ... VERSIONED
          OBJECT_ATTRIBUTES      ...
          VERSION_ATTRIBUTES     ...
          DERIVATION IS DAG;

(c)   DEFINE OBJECT_LINK Aggregation BETWEEN Oberteil(Teil), Unterteil(Teil)
          CARDINALITY      FROM Oberteil TO Unterteil IS (0,*) FOR VERSIONS (0,*),
                           FROM Unterteil TO Oberteil IS (0,*) FOR VERSIONS (0,*);

(d)   DEFINE CONFIGURATION_TYPE Konf FOR Oberteil(Teil), Unterteil(Teil)
          ATTRIBUTES ...
          strukturiert:    Oberteil (1,1),
          ordnet_zu:       Unterteil (1, *);
```

Bild 8: Modellierung technischer Objekte (Beispiel)

5. Die Rolle von Versionierung und Konfigurierung in CONCORD

In den beiden vorangegangen Kapiteln haben wir mit dem CONCORD-Ansatz ein Ablaufmodell für komplexe, kooperative Entwurfsvorgänge erläutert und die Grundbegriffe des OVM-Modells zur Erfassung der Entwurfsdaten eingeführt. Darauf aufbauend ist es das Ziel dieses Kapitels zu untersuchen, wie die Entwurfsdatenkontexte der CONCORD-Ablaufeinheiten mit den Mitteln des OVM beschrieben und verwaltet werden können. Umgekehrt soll auch hinterfragt werden, inwieweit Versionierungs- und Konfigurierungsprimitive notwendig sind für eine natürliche Abbildung von kooperativen Entwurfsprozessen und welchen Anforderungen diese zu genügen haben, speziell im Kontext von CONCORD.

Zur Beantwortung dieser Fragen werden die CONCORD-Abstraktionsebenen absteigend betrachtet und die jeweilige Bedeutung von Versionierung und Konfigurierung zur Beschreibung der Entwurfsdatenkontexte (siehe Bild 5) diskutiert. Dabei werden die Datengranulate mit spezifischen Bezeichnern belegt.

5.1 AC-Ebene

Die Administrations-/Kooperations-Ebene stellt Primitive zur Verfügung, die Entwurfsarbeit im Sinne von “concurrent engineering” [18] ermöglichen. Daher interessieren wir uns in diesem Abschnitt in erster Linie für solche Entwurfsdaten, die zwischen den Ablaufeinheiten dieser Ebene, d.h. den DAs einer Hierarchie, fließen. Um diese zu beschreiben, müssen zunächst die Daten, die eine DA im Laufe ihrer Entwurfsarbeit erzeugt, betrachtet werden.

5.1.1 Entwurfsobjekte und Entwurfsobjektzustände

Wir werden in diesem Abschnitt deutlich machen, daß Entwurfsobjekte nicht allein durch die im OVM beschreibbaren Komplexobjekte modellierbar sind, sondern daß hier die zusätzliche, durch den Konfigurationsbegriff eingebrachte Flexibilität zur Beschreibung der Entwurfsdatenkontexte herangezogen werden muß.

Eine DA wird mittels eines Beschreibungsvektors, der die Gestalt <*DOT, SPEC, Designer, DC*> hat, initialisiert. Dabei ordnet der Parameter *Designer* den verantwortlichen Entwerfer zu, und *DC* enthält Ablaufsteuerungsinformationen, die in Abschnitt 5.2 detailliert werden. Im folgenden sollen die ersten beiden Parameter im Mittelpunkt der Diskussion stehen.

DOT steht für ***Entwurfsobjekttyp*** (engl. design object type) und beinhaltet die Typinformation der für eine DA relevanten primären Entwurfsdaten. Bei ihrer Terminierung wird von einer DA eine Ausprägung des DOT erwartet, die eine Lösung des Entwurfsauftrages der DA darstellt. Der DOT kann entweder *ein* Objekt-Typ oder *ein* Konfigurations-Typ *zusammen* mit seinen Komponententypen sein. Im ersten Fall sprechen wir von einem *einfachen DOT*, im zweiten von einem *strukturierten*. **Bild 9**(a) und (b) illustrieren diese möglichen DOT-Strukturen. In unserem laufenden Beispiel (Getriebeentwurf) haben die DAs DA1, ..., DA4 aufgrund der rekursiven Stücklistenstrukturen identische DOTs; es handelt sich hier um den in Bild 9(c) gezeigten strukturierten DOT. DA5 unseres Beispiels hat einen einfachen DOT (nicht dargestellt), gegeben durch den Objekttyp *Teil*. Wir werden uns im folgenden nur noch auf den strukturierten DOT beziehen, wenn wir unser Beispiel ansprechen.

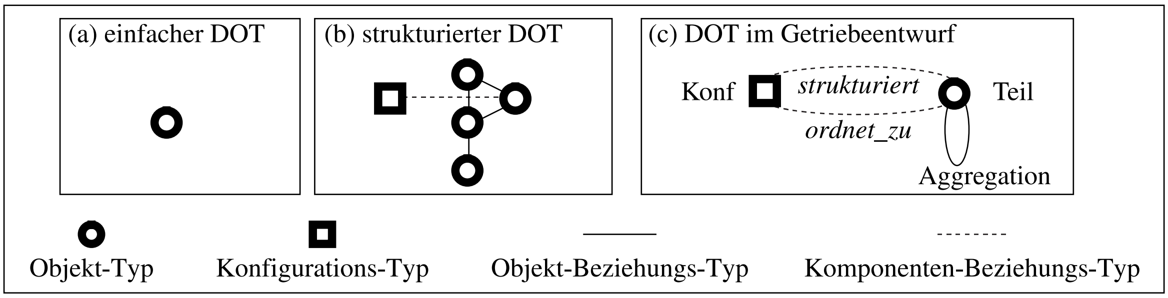

Bild 9: Illustration von Entwurfsobjekttypen

Für die Ausprägungsebene hat die Struktur des DOT folgende Bedeutung. Im Falle eines einfachen DOT können als Ausprägungen beliebig viele Objektausprägungen des Objekttyps, der den DOT darstellt, mit jeweils beliebig vielen Versionen angelegt werden. Anderenfalls (strukturierter DOT) können als Ausprägungen beliebig viele Konfigurationen von dem im DOT enthaltenen Konfigurationstyp entstehen. Betrachten wir unser Beispiel, so können von einer DA mit dem in Bild 9(c) gezeigten DOT (theoretisch) beliebig viele Objekte vom Typ *Teil* angelegt werden. Für die Konfigurierung gilt jedoch folgendes. Jede Konfiguration muß genau eine Version eines Objektes, das in der Rolle des *Oberteils* auftritt, und mindestens eine Objektversion in der Rolle *Unterteil* beinhalten (siehe Kardinalitätsrestriktionen zu den Komponentenbeziehungstypen der Konfiguration *Konf* in Bild 8d). Es kommen jedoch nur solche *Unterteil*-Versionen in Frage, die mit der gewählten *Oberteil*-Version in Aggregationsbeziehung stehen. Dies ergibt sich aus dem allgemeinen, mit Konfigurationen assoziierten Integritätsbegriff [14].

Die Gesamtheit dieser Ausprägungsinformation einer DA bezeichnen wir auch als ihren ***lokalen Scope***. Dieser ist ohne weitere Authorisierung nicht für andere DAs sichtbar. Der *globale Scope* (oder kurz *Scope*) einer DA umfaßt alle Daten, die sie zur Berechnung heranziehen kann. Neben den Daten des lokalen *Scope*'s sind darin auch solche Daten enthalten, die allgemein zugänglich sind oder die der DA mittels Kooperationsmechanismen sichtbar gemacht wurden. Diese werden in einem der folgenden Abschnitte genauer betrachtet.

Der lokale *Scope* enthält unterschiedliche Zustände der in der DA-Bearbeitung befindlichen Entwurfsobjekte. Wie angesprochen, können diese Bearbeitungszustände gegeben sein durch Versionen (einfacher DOT) oder Konfigurationen (strukturierter DOT). Um von dieser Unterscheidung zu abstrahieren, nennen wir diese Bearbeitungszustände allgemein ***Entwurfsobjektzustände***, abgekürzt ***DOS*** (engl. design object state).

Es wurde bereits mehrfach betont, daß alle Aktivitäten innerhalb einer DA darauf abzielen, einen bestimmten Entwurfsauftrag oder Teilauftrag zu erfüllen. Dieser Auftrag wird beschrieben durch eine explizite ***Entwurfszielspezifikation***, die sich hinter dem Parameter *SPEC* im Beschreibungsvektor der DA verbirgt. Die Spezifikation besteht aus einer Reihe von geforderten Eigenschaften, den sogenannten ***Features*** [10]. Im einfachsten Fall entspricht ein Feature einer Attribut-Wertebereichs-Bedingung; es kann sich beispielsweise aber auch um die Forderung handeln, daß ein bestimmtes Test-Werkzeug oder ein Simulations-Werkzeug erfolgreich passiert werden muß. Im Hinblick auf die Zielspezifikation können die DOSs einer DA verschiedenen Qualitätsstufen zugeordnet werden [19]; für den Kontext dieses Papiers genügt jedoch die folgende Unterscheidung: Wir sprechen von einem *vorläufigen DOS*, solange noch nicht alle Features erfüllt sind und von *einem Ergebnis-DOS*, sobald die vollständige Zielspezifikation erreicht ist.

5.1.2 Delegation von Teilaufträgen

Eine DA kann einen Teilauftrag (ihres eigenen Entwurfsauftrages) durch Erzeugung einer Sub-DA auslagern. Die Erfüllung des Teilauftrags ist Voraussetzung für die erfolgreiche Bearbeitung des eigenen Auftrages. Diese Delegation orientiert sich in aller Regel an der DOT-Struktur der Super-DA. Es sind verschiedene Muster für die Aufspaltung von Entwurfsaufgaben in Teilaufgaben denkbar (disjunkte, überlappende oder identische DOTs in den Sub-DAs). Ein Beispiel für den Fall überlappender DOTs ist in **Bild 10** illustriert.

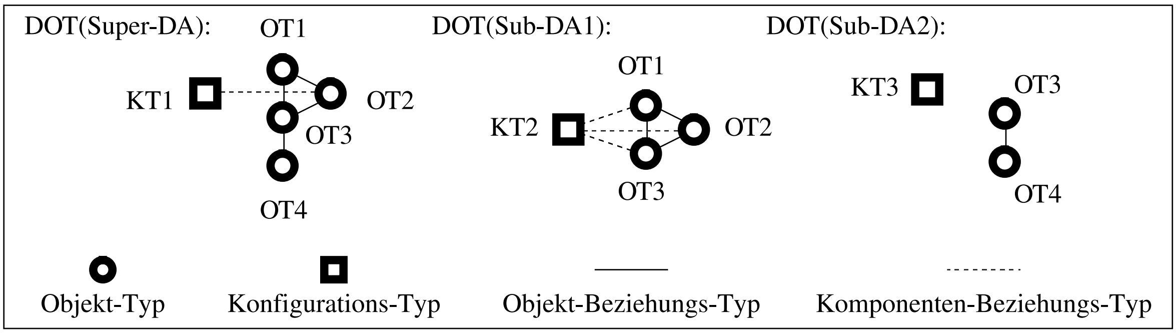

Bild 10: Delegation entlang der DOT-Struktur (hier: Erzeugung überlappender DOTs)

Unterschiedliche Motivationen sind vorstellbar für die Erzeugung von Sub-DAs mit überlappenden oder sogar identischen DOTs. Einerseits kann die Überlappung aufgrund rekursiver Objektstrukturen notwendig sein. Dies ist in unserem laufenden Beispiel der Fall, in dem Stücklisten durch rekursive Oberteil-Unterteil-Beziehungen modelliert sind. Eine ganz andere Motivation ist die, daß die Super-DA das beste unter den von den (ggf. parallel aktiven) Sub-DAs gelieferten Ergebnissen auswählen möchte.

Eine *Initialisierung des lokalen Scope's* einer Sub-DA ist bei deren Erzeugung durch die Super-DA möglich, so daß erstere die bereitgestellten Daten als Basis für ihre Berechnungen heranziehen muß. Dazu kann zu den Objekttypen im DOT der Sub-DA jeweils ein *initiales Objekt* (nur Objektrepräsentant mit Objektattributen) oder eine *initiale Version* (Objektausprägung mit einer ersten Version) zugeordnet werden. Die umfangreichste Initialisierung ist das Anlegen einer *initialen Konfiguration*.

In unserem Beispiel ist eine Initialisierung der *Scopes* für DA2, DA3 und DA4 durch initiale Konfigurationen sinnvoll, da der Chefkonstrukteur (DA1) vor der Auftragszerlegung das Grobgestalten durchgeführt und damit bereits die gesamte Baustruktur des Getriebes bis zu den Einzelteilen vorgeplant hat.

Wie in der obigen Diskussion bereits angesprochen wurde, sind im Normalfall die lokalen *Scopes* der DAs einer Hierarchie disjunkt. Durch Initialisierung können zwar mehrere direkte Sub-DAs einer DA Versionen desselben Objektes oder sogar dieselbe Version als Ausgangsbasis erhalten, jedoch werden zunächst alle abgeleiteten Versionen lokal zur erzeugenden DA gehalten. Es kann jedoch bei überlappenden DOTs sinnvoll sein, daß sich die lokalen *Scopes* überlappen. Falls dies von der Super-DA spezifiziert wird, sind für eine Sub-DA auch die von anderen Sub-DAs abgeleiteten Versionen der Objekte im Überlappungsbereich sichtbar, und jede beteiligte Sub-DA darf zu allen Versionen im Überlappungsbereich Nachfolger ableiten. Wir sprechen in diesem Fall von einem gemeinsamen ***Object-Pool*** (OP) der betroffenen DAs. Diese Ableitungsvorgänge auf gemeinsamen Daten beeinträchtigen die Konsistenz der Daten und Korrektheit der Abläufe nicht, da bestehende Versionen nicht geändert werden können, sondern lediglich Nachfolger-Versionen abgeleitet werden dürfen. Es kommt jedoch folgende sehr wichtige Integritätsbedingung hinzu: Die Ergebnis-Konfigurationen aller am *Object-Pool* beteiligten Sub-DAs müssen aus den gemeinsamen Objekten *dieselbe* Version beinhalten, um das spätere Integrieren der von den Sub-DAs gelieferten Ergebnisse im Rahmen ihrer Super-DA zu ermöglichen. In der folgenden Diskussion zur Synthese der von den Sub-DAs gelieferten Ergebnisse werden wir hierzu ein Beispiel betrachten. Es werden hier Mechanismen benötigt, die das Herbeiführen einer Einigung der betroffenen Sub-DAs auf *eine* Version ermöglichen. Wir werden diese im Abschnitt zur Kooperation kurz ansprechen.

Die *Synthese* der von den Sub-DAs gelieferten Ergebnis-DOSs (evtl. mit den eigenen Ergebnissen der Super-DA) geschieht durch einen Konfigurierungsvorgang. Wir betrachten hier nur den allgemeineren Fall, in dem der Ergebnis-DOS einer Sub-DA als eine Konfiguration gegeben ist. Aus der Sicht der Super-DA ist dieses Ergebnis eine *Teilkonfiguration* (siehe Kapitel 4). Daher muß sie diese Teilkonfiguration in einen eigenen Konfigurierungsvorgang einbeziehen, um einen eigenen DOS zu erzeugen, der die Ergebnisse der Sub-DAs verwertet. **Bild 11** illustriert den allgemeinen Fall (beachte: Bild 11 zeigt im Gegensatz zu den Bildern 9 und 10 Ausprägungsinformation).

In dem in Bild 11 illustrierten Fall liegt Objekt 3 (O3) im *Object-Pool* der Sub-DAs 1 und 2, so daß die Ergebniskonfigurationen dieselbe Version beinhalten (Version V2 von Objekt O3, im Bild durch den Zusatz OP gekennzeichnet) und somit auf der Ebene der Super-DA eine Synthese im Sinne des Zusammensetzens ‘kompatibler’ DOS möglich ist.

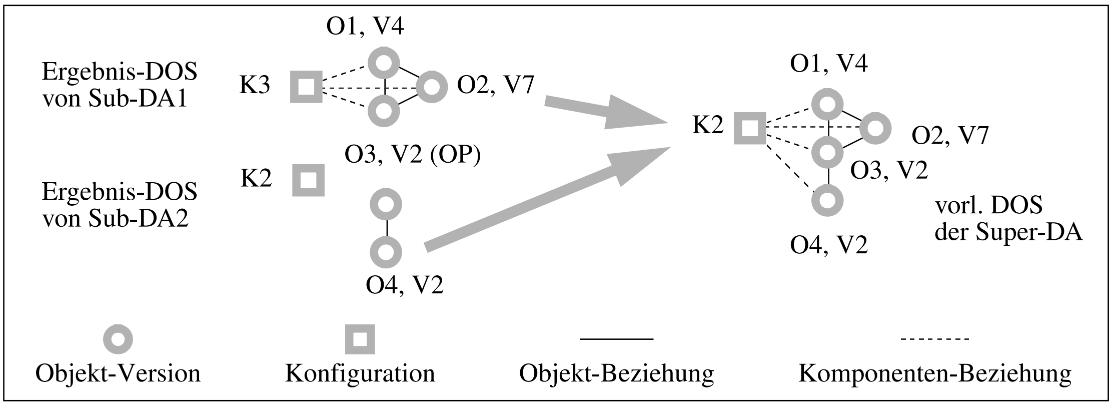

Bild 11: Synthese der Sub-DA-Ergebnisse durch Konfigurierung in der Super-DA

5.1.3 Kooperation

Neben der auftragsbezogenen Kommunikation entlang von Delegationsbeziehungen werden durch den CONCORD-Ansatz zwei weitere wesentliche Arten der Kooperation unterstützt. Dies geschieht durch explizite Modellierung von Verhandlungs- und Datenaustauschbeziehungen. Kooperation erfolgt durch die Ausführung von Kooperations-Operationen entlang solcher Beziehungen, der wiederum bestimmte Protokolle zugrundeliegen. Wir können diese Aspekte hier leider nicht vertiefen und verweisen auf die angegebene Literatur [19], [4], [10]. Hier wollen wir lediglich den jeweiligen Zweck der Kooperationsbeziehungen und den Gegenstand der Kooperation ansprechen.

Beziehungen vom Typ *Negotiation* dienen dem Verhandeln von Zielspezifikationen zwischen Sub-DAs. Eine DA kann einer anderen DA verfeinerte bzw. neue Features zur Aufnahme in deren Spezifikation vorschlagen, und die angesprochene DA kann diese Vorschläge annehmen oder verwerfen. Es wird also vorausgesetzt, daß einer DA (bzw. dem zugeordneten Entwerfer) die Zielsetzungen anderer DAs des gleichen Entwurfsprozesses bekannt sind.

Der Feature-Mechanismus und der angesprochene Verhandlungsprozeß ist auch einzusetzen, um eine Einigung der an einem *Object-Pool* beteiligten DAs bzgl. der in die jeweilige Ergebnis-DOS aufzunehmenden Versionen zu erreichen. Dazu kann eine DA eine Version mit einer Menge von Features markieren und damit ausdrücken, daß sie diese Version bzw. einen durch eine andere DA erzeugten Nachfolger dieser Version nutzen will, falls die zugeordneten Features Gültigkeit behalten. Auch über diese Art von Features kann verhandelt werden.

Beziehungen vom Typ *Usage* dienen dem kontrollierten Austausch vorläufiger Entwurfsdaten zwischen verschiedenen DAs. Auch hier wird vorausgesetzt, daß die Spezifikationen der einzelnen DAs allen beteiligten Entwerfern bekannt sind. Es wird dann eine Beziehung aufgebaut, sobald eine DA von einer anderen DA einen DOS anfordert, der eine bestimmte Entwurfsqualität aufweist. Diese Mindestqualität wird wiederum durch eine Feature-Menge spezifiziert. Falls nun ein DOS mit der angeforderten Qualität existiert und dieser durch die angesprochene DA (mittels einer dafür vorgesehenen Operation) zur Nutzung freigegeben wurde, kann die anfordernde DA diesen in die eigenen Berechnungen als Eingabe einbeziehen, wohlwissend, daß es sich um einen vorläufigen DOS handelt. Es ist die Aufgabe des Systems, die entstehende Abhängigkeit zu warten und bei Invalidierung (des bereitgestellten DOS) entsprechende Maßnahmen (z.B. Benachrichtigung der anfordernden DA) einzuleiten. Aufgrund der Relevanz der Entwurfsqualität für den Datenaustausch sind die Einheiten dieses Austausches vom Granulat DOS.

5.2 DC-Ebene

Es wurde bereits mehrfach angesprochen, daß die Aktionen zur Erfüllung des Entwurfsauftrages einer DA im wesentlichen als Anwendungen von Entwurfswerkzeugen gegeben sind. Die DC-Ebene betrachtet die kontrollierte Ausführung von Werkzeugen gemäß einer vorgegebenen Methodik und bietet Konzepte zur Spezifikation des *Workflows* [8] und zur Abarbeitung der die *Workflow*-Spezifikationen beinhaltenden Skripte. Damit sind die wichtigsten, mit den

Primitiven dieser Ebene verfolgten Ziele (Vermeidung von groben Fehlern in der Entwurfsvorgehensweise, Vorplanen und Durchführen der notwendigen Prüfungen auf den primären Entwurfsdaten, möglichst weitgehende Automatisierung von Routineabläufen) erreicht. In diesem Artikel soll der Schwerpunkt der Betrachtungen jedoch auf dem Datenfluß und nicht dem Kontrollfluß liegen. Daher betrachten wir im folgenden die Strukturen der Daten, die zwischen Werkzeugen weitergereicht werden.

5.2.1 Implizite (Teil-) Konfigurationen

Wie bereits eingeführt, abstrahieren wir von konkreten Entwurfswerkzeugen mit der Betrachtung der Ablaufeinheit DOP. Die Eingaben einer DOP sind i.allg. vorläufige Daten der DA, d.h. eine oder mehrere Versionen der Objekte im *Scope* der DA. Das Ergebnis ist dementsprechend eine Menge von neu abgeleiteten Versionen. Dabei ist folgendes zu beachten: das zugrundeliegende Verarbeitungskonzept für DOPs sieht vor, daß alle Änderungsoperationen, die innerhalb einer DOP an einer eingelesenen Version vorgenommen werden, zusammen zu genau einer Nachfolgeversion (der Eingabeversion) führen. Dies bedeutet, daß die Ablaufeinheit zur Erzeugung einer neuen Version eine DOP ist und damit der Ableitungsvorgang durch Transaktionseigenschaften geschützt ist. Aufgrund dieses Prinzips der Versionserzeugung kann die DOP mehrere Versionen, die jedoch zu jeweils verschiedenen Objekten gehören, einlesen und wiederum mehrere, durch Beziehungen verbundene* Nachfolgerversionen schreiben. Wir sprechen daher bei den Ergebnisdaten einer DOP von einer ***impliziten Teilkonfiguration***. Ähnlich wie bei den bisher besprochenen expliziten Konfigurationen wird hier eine Menge von durch Beziehungen verbundenen Versionen betrachtet, die unterschiedlichen Objekten im lokalen *Scope* der DA angehören. Es ist jedoch kein verbindendes Konfigurationsobjekt angelegt, da implizite Konfigurationen nur temporär zur Übergabe an nachfolgende DOPs benötigt werden und in aller Regel nicht direkt Entwurfsobjektzustände der DA repräsentieren.

Es wird deutlich, daß die Verarbeitungskontexte von Werkzeuganwendungen i. allg. nicht mit den DOSs der DA übereinstimmen. Vielmehr findet die explizite Konfigurierung (eines DOS) zu einem späteren Zeitpunkt als die Erzeugung der betroffenen Versionen statt. So wird im Beispiel durch die DOP ‘Einzelteildetaillierung’ (siehe Kap. 2.1) eine neue Version eines Einzelteiles entwickelt, das jedoch erst zu einem späteren Zeitpunkt durch einen Konfigurationsvorgang in die Baugruppe eingeht, die einen DOS der zugehörigen DA repräsentiert. Dies ist aus Flexibilitätsgründen notwendig, wie wohl leicht einsehbar ist. Explizite Konfigurierung gehört zu den Konzepten der AC-Ebene und wurde oben besprochen.

5.2.2 Aktionen auf DC-Ebene

Die vorangegangenen Diskussionen machen deutlich, daß während der Abarbeitung eines Skriptes sowohl die impliziten Teilkonfigurationen als auch die voll konfigurierten DOSs der

*) Die Ergebnisdaten einer DOP bilden i. allg. eine bedeutungsvolle Einheit. Wir gehen daher davon aus, daß, falls mehrere Versionen geschrieben werden, diese auch durch Versionsbeziehungen zu einer solchen Einheit zusammengebunden sind.

DA dem Datenkontext zugerechnet werden müssen. Oft beenden Konfigurierungsvorgänge einzelne Entwurfsphasen, während innerhalb einer Phase lediglich die Erzeugung und die Weitergabe einzelner Versionen bzw. impliziter Teilkonfigurationen zwischen Werkzeuganwendungen notwendig ist. Dabei steht das Ermöglichen einer flexiblen und fehlertoleranten Skript-Abarbeitung im Vordergrund. Dies betrifft beispielsweise den bereits genannten Fall, daß der Entwurf an einer früheren Stelle im Skript auf dem zugehörigen, früheren DOS wiederaufsetzen soll, um eine Folge von Entwurfsschritten mit den gleichen Werkzeugen (und Entwurfsdaten), aber variierten Kontrollparametern zu wiederholen. Ebenso ist es der Entwurfsarbeit zuträglich, eine Folge von Werkzeuganwendungen wiederholen zu können, jedoch den Eingabe-DOS zu variieren (z.B. unterschiedliche Konfigurationen auszuprobieren).

5.3 TE-Ebene

Für die Diskussionen im vorangegangenen Abschnitt war es bereits notwendig, die für eine DOP relevanten Daten (einzelne Versionen, implizite Teilkonfigurationen) zu identifizieren. Eine DOP verarbeitet diese Daten in drei Phasen (Checkout, Verarbeitung, Checkin). In der Checkout-Phase kann eine DOP eine (beliebige) Menge von Versionen aus dem globalen *Scope* der initiierenden DA lesen. Es kann lediglich vorausgesetzt werden, daß für jede einzelne dieser Versionen die Integritätsbedingungen der Stufe 1 (siehe Abschnitt 4.4) erfüllt sind. Aus Flexibilitätsgründen müssen auf der betrachteten Menge von Versionen eventuelle Inkonsistenzen bzgl. der zusätzlich in Stufe 2 erfaßten Bedingungen bei der DOP-Eingabe toleriert werden. Betrachten wir hierzu das Beispiel eines anwendungsspezifischen *Merge*-Vorgang [9] (d.h. der Verschmelzung zweier Versionen eines Objektes zu *einer* neuen Version desselben Objektes). Aufgrund der ACID-Eigenschaften von DOPs ist es sinnvoll, den *Merge*-Vorgang im Rahmen einer DOP durchzuführen. Da die Eingabeversionen jedoch zu demselben Objekt gehören, können sie nicht die Bedingungen der Stufe 2 erfüllen (vgl. Definition in Abschnitt 4.4). Es kann aber davon ausgegangen werden, daß der Anwendung diese Inkonsistenzen bekannt sind.

Für die von einer DOP erzeugten Daten können stärkere Garantien gegeben werden. Ein DOP erzeugt immer eine konsistente Einheit von Versionen (implizite Teilkonfiguration), so daß die im Checkin propagierte Menge von Ergebnisversionen einer DOP die Bedingungen der Stufe 2 immer erfüllt.

5.4 Zusammenfassung

Wie in den vorangegangenen Abschnitten erläutert, können die in Bild 5 gezeigten Entwurfsdatenkontexte verfeinert und mit Hilfe der Beschreibungsmittel des OVM dargestellt werden. **Bild 12** macht diese Verfeinerung deutlich und ordnet jeweils die entsprechenden Konsistenzgrade zu (vgl. Klassifikation in Abschnitt 4.4).

Die natürliche Vorgehensweise beim Entwurf verlangt sowohl die Erzeugung und Aufbewahrung verschiedener Entwurfsobjektzustände (DOS entspricht Konfiguration), als auch ihre Bewertung hinsichtlich der Entwurfszielspezifikation des zu bearbeitenden Entwurfs-

(Teil-) Auftrages (Qualitätseinstufung). Entwurfsobjektzustände sind im allgemeinen nicht identisch mit den Verarbeitungskontexten von Werkzeugen. Dies gilt insbesondere dann, wenn man eine flexible Einbindung von neuen Werkzeugen unterstützen will. Infolgedessen muß es den Werkzeugen erlaubt sein, auf Teilen der Entwurfsobjektzustände (Verarbeitungskontext eines Werkzeuges entspricht Menge von Versionen) zu arbeiten und auch Teile dieser Zustände (implizite Teilkonfigurationen) zu erzeugen, die durch nachfolgende Konfigurierung zu vollständigen Entwurfsobjektzuständen weiterentwickelt werden können.

Bild 12: Entwurfsdatenkontexte in CONCORD/OVM Notation (mit Konsistenzgraden)

6. Verwaltung der Entwurfsdatenkontexte in der OVM-Datenhaltungskomponente

Die Basis des CONCORD-Systemmodells ist die Datenhaltungskomponente PRIMA [13] (siehe Bild 3), die zum einen die Verwaltung von Objekten, Versionen und Konfigurationen erlaubt und damit die Bildung der verschiedenen angesprochenen Entwurfsdatenkontexte unterstützt. Zum anderen werden auch geeignete elementare Ablaufeinheiten (Transaktionen des DBS) für das Checkout und Checkin der Entwurfsdatenkontexte bereitgestellt. Für die beiden zentralen Aspekte dieser Arbeit, Entwurfsdatenhaltung und Entwurfsablaufsteuerung, sind insbesondere die Propagierung der in den CONCORD-Abläufen erzeugten Daten als neue Versionen und die Synchronisation der zugehörigen PRIMA-Transaktionen zu betrachten. Daher sollen diese beiden Punkte im folgenden erläutert und damit die Wechselwirkungen zwischen dem CONCORD-Ablaufsystem und PRIMA verdeutlicht werden.

6.1 Versionsfortschreibung

Jede Version in der von einer DOP erzeugten impliziten Teilkonfiguration wird in den Abstammungsgraphen des zugehörigen Objektes eingefügt; zusätzlich wird die Abstammungsbeziehung zur Vorgängerversion eingerichtet. Dabei sind die zwei folgenden Vorgehensweisen denkbar (vgl. Kapitel 4, logisches Repräsentationsmodell):

- *Redundante Versionsfortschreibung:*
  Es wird ein neuer Versionsrepräsentant angelegt und mit dem entsprechenden Objektrepräsentanten verbunden. Alle Elementarobjekte der Version werden neu angelegt und mit dem Versionsrepräsentanten verbunden. Dies bedeutet, daß die ausgehend von der Vorgängerversion nicht geänderten Elementarobjekte ebenfalls kopiert werden.

- *Inkrementelle Versionsfortschreibung:*
  Auch hier wird ein neuer Versionsrepräsentant angelegt, ebenso werden alle ausgehend von der Vorgängerversion hinzugefügten oder geänderten Elementarobjekte neu erzeugt. Nicht geänderte Elementarobjekte werden im Gegensatz zur redundanten Fortschreibung nicht kopiert, sondern ihre Zugehörigkeit wird durch Referenzen ausgedrückt (Elementarobjekt-Sharing zwischen Versionen desselben Objektes).

Natürlich ist die inkrementelle der redundanten Versionsfortschreibung schon angesichts der Speicherplatzersparnis vorzuziehen. Es sprechen jedoch auch anwendungsspezifischere Gründe für die Wahl dieser Technik.

In den komplex strukturierten Entwurfsdaten findet man häufig den Fall, daß *Objektüberlappungen* vorliegen. Beispielsweise ist es in dem Anwendungsbereich der mechanischen Konstruktion sinnvoll, in Normteilkatalogen erfaßte Nebenelemente nur einmal zu repräsentieren und ihre Zuordnung zu verschiedenen Hauptelementen und Einzelteilen durch entsprechende Referenzen zu modellieren. Solche Objektüberlappungen können ebenfalls durch das Referenzieren gemeinsamer Elementarobjekte erreicht werden. Angenommen ein bestimmtes Nebenelement ist mehreren Versionen unterschiedlicher Objekte zugeordnet und im Rahmen einer dieser Versionen wird seine Position in einem nachfolgenden Entwurfsschritt verändert. Erst dann wird das zugehörige Elementarobjekt für die entsprechende Nachfolgeversion neu angelegt, so daß die übrigen Objekte von dieser Änderung nicht betroffen sind, da sie weiterhin das “alte” Elementarobjekt referenzieren.

In diesem Sinne erlaubt das Elementarobjekt-Sharing zwischen Versionen bzw. Objekten eine Zuordnung der die wesentlichen Informationen tragenden Elementarobjekte zu verschiedenen für den Entwurf relevanten Datenkontexten (Versionen gleicher/verschiedener Objekte und Konfigurationen). Im folgenden Abschnitt werden wir sehen, daß das mit der inkrementellen Versionsfortschreibung verbundene Prinzip des Neuanlegens von Elementarobjekten bei Änderung auch zu einer wesentlichen Reduzierung des Synchronisationsaufwandes führt.

6.2 Synchronisationsaspekte

Der Zugriff auf die Entwurfsdatenbestände durch nebenläufige DOPs muß selbstverständlich in geeigneter Weise synchronisiert werden. Aufgrund der eingeführten Versionierung ist es jedoch im allgemeinen Fall zur Erhaltung der Konsistenz der Daten lediglich notwendig, die Datenstruktur des Ableitungsgraphen nur während des Checkin einer Version für nebenläufige Checkin/Checkout-Transaktionen zu sperren. Es handelt sich also hier nur um kurze Sperren, die nicht für die gesamte Dauer eines DOPs gehalten werden müssen. Aus Anwendungsgesichtspunkten kann jedoch die Möglichkeit des Haltens von Ableitungssperren für die gesamte DOP-Dauer hinzukommen. Diese Problematik wurde bereits in [19] diskutiert und soll hier nicht wiederholt werden.

Aufgrund des beschriebenen Elementarobjekt-Sharing zwischen Versionen desselben Objektes genügt während des Checkin einer neuen Version das Belegen des Repräsentanten der Vorgängerversion mit einer kurzen Schreibsperre*. Zur Realisierung einer Ableitungssperre reicht ebenfalls das Belegen des Versionsrepräsentanten mit einer Schreibsperre aus, die jedoch für die gesamte Dauer des DOPs gehalten werden muß, um die gleichzeitige Ableitung eines zweiten Nachfolgers zu verhindern. Damit wird einsichtig, daß durch die inkrementelle Versionsfortschreibung auf der Basis des eingeführten logischen Repräsentationsmodells eine minimale Anzahl von Sperren (eine Sperre pro Versionsrepräsentant) zu verwalten ist, so daß der Gesamtaufwand zur Synchronisation entscheidend vermindert werden kann.

7. Zusammenfassung und Ausblick

Zusammenfassend kann man sagen, daß eine Unterstützung der Ablaufsteuerungskomponente durch geeignete Versionierungs- und Konfigurationskonzepte entscheidend für die Effektivität einer integrierten rechnergestützten Entwurfsumgebung ist und damit einen wesentlichen Schritt zur Beherrschung der Komplexität des Entwurfs darstellt. Das Zusammenwirken dieser Konzepte im Rahmen einer Entwurfsumgebung ermöglicht zum einen eine natürliche Vorgehensweise beim Entwurf und zum anderen die unbedingt notwendige Flexibilität zum Einsatz von Werkzeugen im schrittweisen Entwurfsprozeß. Ferner haben wir gezeigt, daß sich für eine Entwurfsablaufsteuerung der Synchronisationsaufwand durch ein entsprechendes Objekt- und Versionsmodell entscheidend reduzieren läßt.

Die enge Verbindung zwischen Versionierung und Konfigurierung auf der einen Seite und der Entwurfsablaufsteuerung auf der anderen Seite zeigt sich insbesondere in der neuen Bedeutung der Konfigurierung als Mechanismus zur nachträglichen, semantischen Lösung von Konflikten zwischen nebenläufig aktiven Ablaufeinheiten des Entwurfsprozesses. Die Versionierung schließt bereits das Auftreten von Konflikten zwischen DAs weitgehend aus, da aufgrund der Propagierung aller Änderungen als neue Versionen keine gegenseitige Behinderung auftreten kann. Der Nachteil hierbei ist jedoch, daß verschiedene Ablaufeinheiten sich unter Umständen auf jeweils eigenen Pfaden auseinanderbewegen, falls sie keinen Gebrauch von den bereitgestellten Kooperationsprimitiven machen. Solche Entwicklungen müssen (über die Konzepte der AC-Ebene) nachträglich wieder zusammengeführt werden, um zu einer gemeinsamen Lösung zu gelangen (Synthese). Ein *Merge* von Versionen ist i. allg. nicht automatisierbar und nur unter sehr großem Aufwand durch den Benutzer durchführbar. Daher ermöglicht der CONCORD-Ansatz die Dekomposition von Aufträgen bis zur Ebene der versionierbaren Komplexobjekte und unterstützt die Konfliktlösung auf zwei Arten: einerseits durch die Unterstützung der direkten Kooperation (*Usage, Negotiation, Delegation*) und andererseits durch die nachträgliche Konfliktlösung durch Konfigurierung.

Die aktuellen Implementierungsarbeiten an unserem PRIMA-Framework [19] haben das Ziel, das Zusammenwirken von Ablaufsteuerung und OVM-Datenhaltungskomponente praktisch und im Rahmen konkreter Entwurfsanwendungen (VLSI-Entwurf und CAD im Maschinenbau sowie Software-Entwurf) zu untersuchen. Die Arbeiten an OVM-PRIMA [KM93] sind schon abgeschlossen und die Realisierung des CONCORD-Systems schon in Angriff genom-

*) Daneben setzen wir den Schutz des neuanzulegenden Versionsrepräsentanten und der neuanzulegenden Elementarobjekte für die Dauer des Checkin voraus.

men. Für den Bereich VLSI-Entwurf sind bereits Werkzeuge mit Hilfe des angesprochenen Verarbeitungskonzeptes implementiert worden, die mit OVM-PRIMA zusammenarbeiten und dabei einen ersten Prototyp des Transaktions-Managers der TE-Ebene benutzen. Sobald dann weitere Komponenten des CONCORD-Systems verfügbar sind, können erste praktische Erfahrungen mit dem PRIMA-Framework gewonnen werden.

Als weitere zukünftige Arbeiten wollen wir die Integration von "Fremdwerkzeugen" untersuchen. Hier interessiert uns insbesondere die sog. Black-Box-Integration von Werkzeugen, d.h. die Integration von Werkzeugen, auf deren interne Struktur kein Einfluß genommen werden kann. Die Einbindung solcher Fremdwerkzeuge in die CONCORD-Abläufe kann in einfacher Weise als DOPs geschehen. Damit ist die Integration mit den Konzepten der DC- und der AC-Ebene problemlos. Lediglich die Datenversorgung und die Weitergabe der Ergebnisse verlangen eine entsprechende Anpassung. Auf der TE-Ebene ist zu beachten, daß die Datenversorgung und die Propagierung der Ergebnisse dieser Werkzeugläufe entweder durch werkzeugeigene Datenhaltungssysteme geschieht oder daß eine Anbindung an die gemeinsame Datenhaltungskomponente möglich wird. Letztere Alternative bedeutet für die Black-Box-Integration die Bereitstellung von entsprechenden Konvertierungsmodulen in das gemeinsame (Datenaustausch-)Format. Hier bieten sich z.B. standardisierte Austauschformate wie z.B. STEP/EXPRESS [1] an. Diese Austauschformate eignen sich dann zudem auch zur Weitergabe der Ergebnisse eines Werkzeuglaufs (DOP) an das Nachfolgewerkzeug. Durch weitere Untersuchungen wollen wir zeigen, daß die Konzeption und Flexibilität des CONCORD-Ansatzes (Ablaufmodell und Entwurfsdatenkontexte) auch zur Integration von Fremdwerkzeugen geeignet ist.

8. Literatur

[1] R. Anderl: CAD-Schnittstellen - Methoden und Werkzeuge zur CA-Integration, Carl Hanser Verlag, München, 1993.

[2] D. Beech, B. Mahbod: Generalized Version Control in an Object-Oriented Database, in: Proc. of the 4th Int. Conf. on Data Engineering, 1988, S. 14-22.

[3] J. Gray, A. Reuter: Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publ., San Mateo, CA, 1993.

[4] C. Hübel, W. Käfer, B. Sutter: Controlling Cooperation Through Design-Object Specification - a Database-oriented Approach, in: Proc. of the European Design Automation Conference, Brüssel, Belgien, März 1992.

[5] T. Härder, B. Mitschang, U. Nink, N. Ritter: Workstation/Server-Architekturen für datenbankbasierte Ingenieuranwendungen, Universität Kaiserslautern, 1993, eingereicht zur Veröffentlichung.

[6] T. Härder, A. Reuter: Principles of Transaction-Oriented Database Recovery, in: ACM Computing Surveys, Vol. 15, No.4, 1983.

[7] C. Hübel: Ein Verarbeitungsmodell für datenbankgestützte Ingenieuranwendungen in einer arbeitsplatzrechner-orientierten Ablaufumgebung, Dissertation, Universität Kaiserslautern, 1992.

[8] S. Jablonski: Transaction Support for Activity Management; in: Proc. of the Workshop on High Performance Transaction Processing Systems, Asilomar, CA, September 1993.

[9] R. Katz: Toward a Unified Framework for Version Modeling in Engineering Databases, ACM Computing Surveys, Vol. 22, No. 4, Dezember 1990, S. 375-408.

[10] W. Käfer: A Framework for Version-based Cooperation Control, in Proc. of 2nd Int. Symposium on Database Systems for Advanced Applications (DASFAA), Tokyo, Japan, April 1991.

[11] W. Käfer: Geschichts- und Versionsmodellierung komplexer Objekte - Anforderungen und Realisierungsmöglichkeiten am Beispiel des NDBS PRIMA, Dissertation, Universität Kaiserslautern, 1992.

[12] W. Käfer, B. Mitschang: Flexible Entwurfsdatenverwaltung für CAD-Frameworks: Konzept, Realisierung und Bewertung, in W. Stucky und A. Oberweis (Hrsg.): Tagungsband der GI-Fachtagung "Datenbanksysteme für Büro, Technik und Wissenschaft", Springer-Verlag, 1993, S. 144-163.

[13] W. Käfer, H. Schöning: Mapping a Version Model to a Complex Object Data Model, Proc. of the 8th Int. Conf. on Data Engineering, Tempe, Arizona, 1992, S. 348-357.

[14] W. Käfer, H. Schöning: Versionierung und Konfigurierung in Entwurfsumgebungen; Universität Kaiserslautern, 1993, in Vorbereitung.

[15] M. Muschiol: Rechnerunterstützte Informationsbereitstellung für den Konstruktionsprozeß am Beispiel montagerelevanter Gestaltungsrichtlinien, Reihe Produktionstechnik Berlin, Bd.68, Forschungsberichte für die Praxis, Hanser-Verlag, 1988.

[16] G. Pahl, W. Beitz: Konstruktionslehre, Springer-Verlag, 1986.

[17] F. J. Rammig, B. Steinmüller: Frameworks und Entwurfsumgebungen; in: Informatik-Spektrum, Heft 1, 92/02, S. 33-43.

[18] Y.V.R. Reddy, V. Jagannathan, R. Karinthi: Computer Support for Concurrent Engineering, in: IEEE Computer, Januar 1993, Vol. 26, No. 1, S. 12-16.

[19] N. Ritter, B. Mitschang, T. Härder, M. Gesmann, H. Schöning: Capturing Design Dynamics - The CONCORD Approach; erscheint in: Proc. of the 10th International Conference on Data Engineering, Houston, Texas, 1994.

[20] B. Sutter: Ansätze zur Integration in technischen Entwurfsanwendungen - angepaßte Modellierungswerkzeuge, durchgängige Entwurfsunterstützung, datenorientierte Integration; Dissertation, Universität Kaiserslautern, 1992.

[21] VDI-Richtlinie 2221: Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte, VDI-Verlag, Düsseldorf, 1986.

[22] H. Wedekind: Stehen die Begriffe "Version" und "Alternative" othogonal zueinander? - Eine kritische Analyse der STEP-Standardisierung -, Universität Erlangen-Nürnberg, 1993.

[23] H. Wedekind, T. Müller: Stücklistenorganisation bei großen Variantenzahlen, in: Angewandte Informatik, Heft 9, Sept. 1981, S. 377-383.