Revisionsmodell und erweitertes Demarkationsprotokoll für die arbeitsteilige Produktentwicklung

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften/Naturwissenschaften
der Fakultät für Informatik
der Universität Karlsruhe (TH)

vorgelegte

Dissertation

von

Dietmar Posselt

aus

Stuttgart

Tag der mündlichen Prüfung:	19.12.2002
Erster Gutachter:	Prof. Dr. Peter C. Lockemann
Zweiter Gutachter:	Prof. Dr. Guido Moerkotte

Danksagung

Auf der fachlichen Seite gilt mein größter Dank Prof. Lockemann für die Betreuung der Arbeit. Trotz seiner zahlreichen Verpflichtungen setzte er sich sehr intensiv mit meiner Arbeit auseinander. Weiteren Dank schulde ich Prof. Moerkotte, der das Korreferat übernahm. Für das Grundkonzept meiner Arbeit will ich mich vor allem bei Dr. Gerd Hillebrand bedanken. Er hat mir in regelmäßigen Diskussionen immer wieder neue Impulse gegeben und damit meine Arbeit maßgeblich beeinflußt.

Erfolgreiche wissenschaftliche Arbeit kann nur in einem Umfeld gelingen, das Kritik und Anregung gleichermaßen bietet. Meine Kollegen am IPD und FZI waren diesbezüglich immer ansprechbar und standen mir von grundsätzlichen Fragestellungen bis hin zur Korrektur der Ausarbeitung zur Seite. Stellvertretend möchte ich hier meine Bürogenossin Patricia Krakowski erwähnen, mit der ich schon aufgrund der räumlichen Nähe sehr oft diskutieren konnte.

Fachliche Unterstützung ist unbedingt notwendig, aber noch lange nicht ausreichend für die Entstehung einer solchen Arbeit. Ich muß hier in erster Linie meiner Frau Susanne und meinen Kindern Linus, Tristan und Maja für die moralische Unterstützung danken. Meine Eltern, Kurt und Siegrun Posselt, schulde ich besonderen Dank, da sie den Grundstein für meine Karriere gelegt haben, unter anderem, indem sie mir meine Studien in Stuttgart, Massachusetts und Zürich ermöglicht haben.

Karlsruhe, im Januar 2003

Inhaltsverzeichnis

Kapitel 1

Einleitung

Um die Konkurrenzfähigkeit eines Unternehmens in der modernen Produktentwicklung zu sichern, müssen neue Produkte schnellstmöglich auf den Markt gebracht werden. Dies bedarf unter anderem kurzer Produktentwicklungszyklen. Diese wiederum sind undenkbar ohne eine dem Entwicklungsprozeß angepaßte Rechnerunterstützung.

Kurze Entwicklungszeiten bedürfen zumindest bei komplexeren Produkten eines arbeitsteiligen Vorgehens. Das Ziel dieser Arbeit ist die Rechnerunterstützung arbeitsteiliger Produktentwicklungsprozesse. Speziell wird der Entwicklungsprozeß für Bauteile betrachtet, wie er dem Sonderforschungsbereich 346 zugrunde liegt.

1.1 Analyse des Produktentwicklungsprozesses

Übliche Konstruktionsmethodiken teilen den Konstruktionsprozeß in mehrere Phasen ein, in denen das Produkt zunehmend detaillierter spezifiziert wird. Gebräuchlich ist zum Beispiel die Unterscheidung zwischen Anforderungsmodellierung, Funktionsmodellierung, Prinzipmodellierung und Gestaltung. Diese Phasen bauen aufeinander auf: die Produktfunktionen ergeben sich aus den Anforderungen, die Wirkprinzipien wiederum aus den zu realisierenden Produktfunktionen und die Gestalt schließlich aus den Wirkprinzipien. Auf den ersten Blick scheint daher eine strikt sequentielle Abfolge der Phasen geboten. In der Praxis ist das aber weder wünschenswert noch machbar. Wartet nämlich jede Phase, bis alle Ergebnisse der vorherigen Phasen komplett vorliegen, so dauert der Konstruktionsprozeß zu lange für eine Wettbewerbsfähigkeit. Überdies werden Fehlentwicklungen, etwa die Spezifikation von Anforderungen, die sich bei der Ausgestaltung als zu teuer herausstellen, erst sehr spät erkannt und sind entsprechend schwierig zu korrigieren. Die Phasen müssen also überlappend ausgeführt werden, was dazu führt, daß nachfolgende Phasen auf unfertigen und daher unpräzisen Teilergebnissen der vorherigen Phasen aufsetzen.

Diese unfertigen Teilergebnisse können in Form von Spezifikationen ausgedrückt werden, wobei eine Spezifikation eine Menge von Eigenschaften des zu fertigenden Produkts betrachtet. Aufgrund der vorhandenen Unschärfe sind die Spezifikationen in den allermeisten Fällen keine Punktforderungen, sondern Bereichsforderungen. Eine Spezifikation erlaubt also eine Menge von Ausprägungen für die Produkteigenschaften, während umgekehrt ein Produkt einer Spezifikation genügt, wenn die entsprechenden Eigenschaften des Produkts die Forderungen der Spezifikation erfüllt.

In der frühen Produktentwicklung sind die meisten Spezifikationen noch sehr abstrakt und nur mittelbar beeinflußbar (etwa die Nennleistung eines zu konstruierenden Automotors). Diese abstrakten Spezifikationen geben den Entwicklern einigen Freiraum zur konkreten Ausgestaltung durch unmittelbar beeinflußbare Spezifikationen (etwa die Wahl einer bestimmten Zylindergeometrie). Dieser Transformationsprozeß ist keineswegs eindeutig (die Nennleistung kann durch verschiedene Geometrien erreicht werden), so daß es sich um eine Auswahl und Verfeinerung handelt, die nicht automatisch, sondern nur durch menschliches Zutun erreicht werden kann.

Im Lauf der Produktentwicklung wird das zu fertigende Produkt üblicherweise hierarchisch in mehrere Teile zerlegt. Sowohl die Überlappung der Phasen, wie auch die parallele Arbeit an diesen unterschiedlichen Teilen eines Produkts führt zu einem stark arbeitsteiligen Produktentwicklungsprozeß, in dem viele Entwickler gleichzeitig an einem zu fertigenden Produkt arbeiten. Hierbei bekommt die Koordination der Arbeiten und die Kommunikation zwischen den Entwicklern einen entscheidenden Stellenwert.

In den meisten Anwendungen der elektronischen Datenverarbeitung wird die reale Welt im Rechner nachgebildet (etwa die Buchhaltung oder Lagerverwaltung). Damit muß der Rechner immer den aktuellsten Stand der realen Welt repräsentieren und ersetzt diesen durch neue Zustände. Es gibt also einen Bezug zwischen der realen Welt und der im Rechner abgebildeten Miniwelt, wobei Konsistenz oft in Termen dieses Realweltbezuges definiert ist. Im Gegensatz dazu liegt der Fokus in der Produktentwicklung immer auf dem fertigen Produkt, wobei die Beschreibung des Produkts und damit das „Wissen“ über das Produkt in der Regel immer genauer wird. Dieser Kenntnisstand ist zu Beginn der Produktentwicklung praktisch gleich Null und wird durch immer mehr Spezifikationen so weit präzisiert, daß das Produkt in unzweideutiger Weise gefertigt werden kann. Es gibt also in der Produktentwicklung nur einen gesuchten Zustand, der im Lauf des Entwicklungsprozesses herausgearbeitet wird. Alle Zwischenzustände sind unvollständige Approximationen des endgültigen gesuchten Zustandes, die nicht ersetzt, sondern weiter verfeinert werden müssen.

Zu jedem Zeitpunkt der Entwicklung definiert die Menge der bekannten Spezifikationen genau eine Menge von möglichen Produkten, nämlich alle Produkte, deren Eigenschaften alle Spezifikationen erfüllen. Je mehr Spezifikationen im Lauf der Produktentwicklung hinzukommen, umso weniger mögliche Produkte bleiben übrig. Am Ende der Produktentwicklung sollte idealerweise nur noch genau ein Produkt übrig bleiben, welches dann auch gefertigt wird. Die Entwicklung ist also eine fortlaufende Konkretisierung des Produkts durch Ansammlung von Spezifikationen, bis das Produkt hinreichend genau spezifiziert ist.

Es darf keine Spezifikationen geben, die sich widersprechen, da sonst kein Produkt gefertigt werden kann, welches alle Spezifikationen erfüllt. Definiert also ein Entwickler neue Spezifikationen, die mit schon bekannten Spezifikationen im Widerspruch stehen, so muß dieser Widerspruch aufgelöst werden. Da dies nicht automatisch erledigt werden kann, müssen die beteiligten Entwickler über den Konflikt informiert werden, so daß diese den Konflikt miteinander ausdiskutieren und die Spezifikationen so anpassen können, daß kein Widerspruch mehr besteht.

Insbesondere diese Widersprüche, aber auch Änderungen zum Beispiel in den Kundenanforderungen oder im Kenntnisstand eines Entwicklers, können dazu führen, daß eingeführte Spezifikationen abgeändert oder sogar gänzlich aufgegeben werden müssen. Damit wird die idealerweise monotone Verfeinerung der Produktbeschreibung, wie sie durch Hinzufügen neuer Spezifikationen zustande kommt, durchbrochen. Auch diese „Rückschritte“ in der Entwicklung müssen adäquat unterstützt werden.

1.2 Anforderungen an die Rechnerunterstützung des arbeitsteiligen Produktentwicklungsprozesses

Die wichtigste Konsequenz aus der Analyse des Entwicklungsprozesses ist die phänomenologische Vorgehensweise, also die idealerweise monotone Akquisition von Spezifikationen. Dabei dürfen, abgesehen von den zuvor erwähnten Rückschritten in der Entwicklung, zu keinem Zeitpunkt schon bekannte Spezifikationen durch neue ersetzt werden. Vielmehr müssen alle bekannten Spezifikationen als eine große Konjunktion aufgefaßt werden, da das zu fertigende Produkt eben genau durch die Konjunktion aller Spezifikationen beschrieben wird. Auch die Rückschritte ersetzen nicht irgendwelche Spezifikationen, sondern entfernen sie nur.

Als Konsistenzbedingung der Zwischenzustände im Entwurfsprozeß kann nicht der oft verwendete Realweltbezug herhalten, da dieser nur für das fertige Produkt gegeben ist. Alle vorherigen Zustände sind mehr oder weniger gute Beschreibungen dieses einen Zustandes. Konsistenz kann hier also nur den Zwischenzustand im Bezug auf den gesuchten Endzustand betrachten. Die Frage der Konsistenz eines Zwischenzustands reduziert sich damit auf die Frage, ob aus dem Zwischenzustand durch weitere Verfeinerung noch mindestens ein Endzustand erreichbar ist. Dies ist gegeben, wenn sich die im Zwischenzustand bekannten Spezifikationen nicht widersprechen (Widerspruchsfreiheit).

Die besondere Herausforderung besteht nun darin, daß es zu nebenläufigem Zugriff arbeitsteilig agierender Entwickler auf sich überlappende Teile des Entwurfsraums kommt. Dieser sollte durch die Konsistenzsicherung möglichst wenig gestört werden. Wenn Entwickler aufeinander warten müssen, wird der Prozeß unnötig in die Länge gezogen.

In der Praxis kann die Produktentwicklung nicht mehr mit einem großen monolithischen Informationssystem unterstützt werden. Die arbeitsteilig arbeitenden Entwickler können räumlich verteilt sein und sogar verschiedenen Unternehmen angehören. Viele unterschiedliche Applikationen aus den einzelnen Phasen der Produktentwicklung müssen möglicherweise mit externen Datenquellen von Zulieferern oder Kooperationspartnern kombiniert werden. Das bedeutet wiederum, daß die Arbeitsteiligkeit in der Produktentwicklung zeitweise entkoppelt erfolgt, bis hin zu mobilen Arbeitsstationen, die auch längere Zeit abgekoppelt sein können. Im Zuge dieser zeitweisen Abkopplung wird auch die Replikation der Daten unumgänglich.

1.3 Das Modell der Widerspruchsfreien Freiräume

Das in dieser Arbeit vorgestellte Modell der widerspruchsfreien Freiräume orientiert sich direkt an den Anforderungen des Produktentwicklungsprozesses, wie sie hier vorgestellt wurden. Sämtliche Kommunikation und damit auch die Koordination der arbeitsteilig arbeitenden Entwickler wird über Zugriffe auf einen logisch zentralen Datenspeicher abgewickelt, der beliebig viele Spezifikationen aufnehmen kann. Eine Spezifikation ist dabei keine Punktforderung, sondern beschreibt eine Menge von Ausprägungen einer Produkteigenschaft, die aus der Sicht der Spezifikation akzeptabel sind. Jedes Produkt, welches eine der akzeptablen Ausprägungen aufweist, genügt eben dieser Spezifikation. Entwickler können neue Spezifikationen einbringen, alte Spezifikationen löschen und den auf einer Produkteigenschaft verbleibenden Freiraum auslesen, und zwar in der Form aller Ausprägungen dieser Produkteigenschaft, für die es noch Produkte gibt, die allen Spezifikationen genügen.

Als einzige Konsistenzbedingung des Datenspeichers wird die Widerspruchsfreiheit in das Modell aufgenommen. Konzeptionell gesehen ist die Durchsetzung dieser Widerspruchsfreiheit trivial, da der Datenspeicher nur für jede neue Spezifikation auf Widerspruchsfreiheit untersucht werden muß. Im Kern der Rechnerunterstützung muß also ein theoretisch fundiertes Modell des Produktentwicklungsprozesses stehen, derart, daß

darauf aufbauend Methoden der Constraint-Programmierung für die Entscheidung der Widerspruchsfreiheit der Spezifikationen, also der Konsistenz, verwendet werden.

Widerspruchsfreie Freiräume sind somit das zustandsorientierte Pendant zu einem gegebenenfalls arbeitsteiligen Entwicklungsprozeß. Sie unterscheiden sich damit von Arbeiten aus dem Datenbankbereich, die im Grundsatz auf der Ersetzung alter Informationen durch neue und dem ACID-Paradigma beharren, während Arbeiten aus dem Bereich der Produktentwicklung zwar constraintbasierte Modelle vorschlagen, sich aber über die Konsequenzen für die Datenbankunterstützung und die Frage der Datenkonsistenz nicht im klaren sind.

1.4 Konsequenzen der Widerspruchsfreien Freiräume

So einfach und intuitiv das Modell der widerspruchsfreien Freiräume auch ist, in der Umsetzung und der praktischen Anwendung stellen sich einige interessante Herausforderungen. So stellt sich zum Beispiel die Frage, wie die Spezifikationen konkret aussehen. Prinzipiell darf sich hinter einer Spezifikation jede Form der Teilmengenbeschreibung verbergen (zum Beispiel Constraints, Intervalle oder explizite Aufzählungen), wobei die spezifizierte Teilmenge eben genau alle akzeptablen Ausprägungen beschreibt. Fuzzy-Mengen bieten darüber hinaus noch die interessante Möglichkeit, die Teilmengenbeschreibung mit Unschärfe zu versehen.

Durch die große Zahl der beteiligten Entwickler und deren Mobilität können im zunehmend komplexer werdenden Produktentwicklungsprozeß nicht immer alle Entwickler angekoppelt sein. Das heißt, auf dem logisch zentralen Spezifikationsspeicher muß abgekoppelt gearbeitet werden können. Technisch gesehen muß hierfür der Spezifikationsspeicher verteilt und repliziert implementiert werden, wobei Kommunikation zwischen den Replikaten nicht immer möglich ist. Trotzdem soll auf allen Replikaten gearbeitet werden können und dabei die Widerspruchsfreiheit nicht verletzt werden. Die üblicherweise in solchen Szenarien verwendete optimistische Vorgehensweise ist hier absolut ungeeignet, da für den Entwickler das Überleben seiner Arbeit essentiell ist. Es darf nicht passieren, daß ein Entwickler neue Spezifikationen einbringt, die erst deutlich später bei einer Zusammenführung der Replikate abgelehnt werden.

In der vorliegenden Arbeit werden noch einige weitere Fragestellungen der praktischen Anwendung der widerspruchsfreien Freiräume behandelt. Insbesondere ist dies die Integration von Alt-Systemen, die üblicherweise auf dem Ersetzungsmodell und auf scharfen Daten basieren. Außerdem werden die Möglichkeiten einer Implementierung der widerspruchsfreien Freiräume auf Basis von bestehenden Datenbanksystemen, oder zumindest von Teilen dieser Systeme, diskutiert.

Kapitel 2

Anforderungsanalyse

Gegenstand der vorliegenden Arbeit ist die Unterstützung arbeitsteiliger Entwurfsprozesse mit den Mitteln der Datenbanktechnik. Dazu bedarf es eines auf die wesentlichen Elemente reduzierten Modells von Entwurfsprozessen. Das Modell erfordert seinerseits eine Analyse konkreter technischer Entwurfsprozesse. Das Kapitel beginnt mit dieser Analyse und entwickelt daraus ein Modell. Aufbauend auf diesem Modell werden die Anforderungen für die Unterstützung arbeitsteiliger Produktentwicklungsprozesse erarbeitet.

2.1 Das Szenario

Die vorliegende Arbeit entstand im Rahmen des *Sonderforschungsbereichs 346* „Rechnerintegrierte Konstruktion und Fertigung von Bauteilen“. Damit ist das Szenario der Arbeit auf den *maschinenbaulichen Entwurfsprozeß* festgelegt. Da aber auch in anderen Gebieten sehr ähnliche Entwurfsprozesse ablaufen, sind die Ergebnisse dieser Arbeit durchaus auf diese Gebiete übertragbar. Als Beispiel sei hier der Entwurfsprozeß in der Architektur genannt. Am Institut für Programmstrukturen und Datenorganisation der Universität Karlsruhe, an dem die Arbeit entstand, hat die Kooperation mit dem Fachbereich der Architektur eine lange Tradition, so daß dem Autor zumindest ein grober Einblick in diesen Fachbereich möglich war. Zudem ist nicht zu vergessen, daß auch in der Informatik selbst, im Rahmen von Softwareentwicklungen, durchaus große und komplexe Entwurfsprozesse ablaufen, die im Fachgebiet des Softwareengineering genauer betrachtet und optimiert werden.

2.1.1 Hintergrund

Der *SFB 346* befaßt sich mit der methodischen Verbesserung der verschiedenen erzeugnisorientierten und planerischen Bereiche im Produktentstehungsprozeß. Die dem Sonderforschungsbereich zugrundeliegende Prämisse ist, daß eine wesentliche Voraussetzung für eine in Preis, Qualität und Lieferzeit wettbewerbsfähige Produktion die Integration der Ingenieurwerkzeuge ist, die in den verschiedenen Funktionsbereichen eingesetzt werden. Diese Integration ist das Hauptanliegen des SFB 346, und zu seiner Umsetzung haben sich Maschinenbau und Informatik unter dem „Dach“ des SFB 346 zusammengeschlossen.

Die datentechnische Integration wurde dadurch vorangetrieben, daß in der ersten Forschungsperiode die Partialmodelle für die einzelnen Bereiche entwickelt wurden, welche in der zweiten Forschungsperiode in einem gemeinschaftlichen *Produkt- und Produktionsmodell (PPM)* konsolidiert wurden. Da sich das PPM in seiner Entwicklung als zu umfangreich und komplex erwies, wurde eine Restrukturierung des PPM in der dritten Forschungsperiode angestoßen. Hierbei wurde das Modell in zwei Schichten unterteilt. In der unteren Schicht ist das sogenannte *Kernmodell* des PPM verankert. Dieses Kernmodell ist generisch gehalten und ermöglicht es, auf eine einfache Art und Weise weitere *Partialmodelle* zu integrieren. In der oberen Schicht des PPM wird die sogenannte *Sicht* der Anwendungen verankert. Sichten erlauben die Definition der für einen Anwendungsbereich charakteristischen Informationsstrukturen und Funktionalität auf einer beliebigen Abstraktionsebene.

Die vorliegende Arbeit entstand im Rahmen des Teilprojekts *A1* „Kooperation in verteilten Objektsystemen“. Im SFB 346 bestehen die Aufgaben des Teilprojekts A1 im Bereich der Datenbankunterstützung für den Entwurfsprozeß. In der dritten Forschungsperiode wurde die technische Realisierung des PPM auf einer CORBA-Architektur implementiert. Dabei entstand im Rahmen des Teilprojekts A1 eine Entwicklungsumgebung, die aus dem Modell per Knopfdruck die Ankopplungsmodule für den Objektbus und Schnittstellen für die Integration von Alt-Datenbeständen generiert. In der aktuellen vierten Periode liegen die Hauptaufgaben des Teilprojekts A1 im Bereich der Prozeßsteuerung, sowie in der effizienten Koordination der Prozeßphasen im Hinblick auf ihre Zugriffe auf die verteilten und replizierten Datenbestände des PPM.

Im SFB 346 sind in der aktuellen Forschungsperiode 13 Teilprojekte von insgesamt sechs Instituten vertreten. Diese 13 Teilprojekte repräsentieren die unterschiedlichsten Phasen des Produktentstehungsprozesses, angefangen von der Anforderungsermittlung, über die Entwicklung und Konstruktion, bis hin zur Planung, Arbeitsvorbereitung und Fertigung. Damit sind alle für den Produktentstehungsprozeß wichtigen Phasen im SFB 346 vertreten.

In den ersten drei Forschungsperioden wurde die Produktentwicklung exemplarisch an einem *Robotergreifer* vollzogen. Für die aktuelle vierte Phase wurde beschlossen, einen neuen Demonstrator zu wählen. Anhand der vielfältigen Möglichkeiten aus den unterschiedlichsten Bereichen sieht man gut die Allgemeinheit des SFB 346 und damit auch die Anwendbarkeit dieser Arbeit auf den unterschiedlichsten Gebieten.

2.1.2 Der Robotergreifer

Die Entwicklung eines Robotergreifers wurde über Jahre hinweg verfeinert, so daß heute genügend Informationen über diesen Prozeß vorhanden sind. Für diese Arbeit soll daher die Entwicklung des Robotergreifers als primäres Szenario dienen, an dem anschaulich die entwickelten Konzepte verdeutlicht werden.

Schnell hat sich ein Greifer nach dem Funktionsprinzip aus Abbildung 2.1 durchgesetzt, in dem per Öldruck ein Zylinder bewegt wird, dessen Bewegung dann über den Keil in die Schließbewegung des Robotergreifers umgesetzt wird. Eine etwas anschaulichere Darstellung aus der Entwicklung des Robotergreifers findet sich in Abbildung 2.2.

Schon an dem relativ einfachen Beispiel des Robotergreifers gibt es eine ganze Reihe von Anforderungen aus den unterschiedlichsten Bereichen, von denen in Abbildung 2.3 nur einige exemplarisch aufgelistet sind.

Im Folgenden wird von diesem stark vereinfachten Beispiel ausgegangen. Eine ausführlichere Beschreibung des Robotergreifers mit allen Anforderungen würde im Grundsatz wenig hinzufügen.

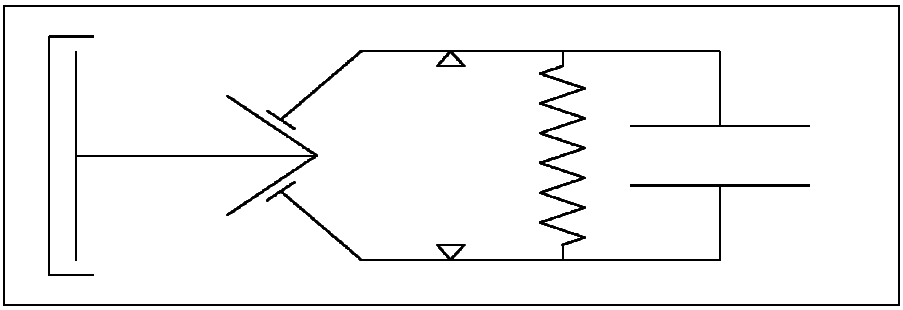


Abbildung 2.1: Funktionsprinzip des Robotergreifers

2.2 Die General Design Theory

Im Bereich der Produktentwicklung sind einige Modelle für den Produktentwicklungsprozeß entstanden ([BM98] und [HV00]). Der SFB 346 orientiert sich dabei in weiten Teilen an der von Yoshikawa vorgestellten General Design Theory ([Yos81] und [Rei95]), die von einem Schüler Yoshikawas weiterentwickelt wurde ([Tom98]).

In der *General Design Theory (GDT)* werden jedem Objekt der realen Welt eine Reihe von Eigenschaften zugeschrieben. Diese Eigenschaften werden unterteilt in funktionale Eigenschaften, also solche, die die Funktion des Objekts beschreiben, und beobachtbare (observable) Eigenschaften, also meßbare Eigenschaften. Dabei entsprechen die funktionalen Eigenschaften genau den gewünschten Anforderungen an ein zu fertigendes Produkt. Auf der anderen Seite beschreiben die beobachtbaren Eigenschaften genau das zu fertigende Produkt. Oder anders ausgedrückt: Das Problem wird durch die funktionalen Eigenschaften beschrieben, die Lösung durch die beobachtbaren Eigenschaften. Für den Entwickler hat dies folgende Konsequenz:

Definition 2.2.1 (Funktionale und beobachtbare Eigenschaften)
Jedes Objekt der realen Welt hat **funktionale und beobachtbare Eigenschaften**. *Die* **beobachtbaren** *Eigenschaften können im Herstellungsprozeß direkt kontrolliert werden, während* **funktionale** *Eigenschaften nur indirekt durch beobachtbare Eigenschaften beeinflußt werden können.*
□

Beispiel 2.2.1 (Funktionale und beobachtbare Eigenschaften)
Beobachtbare Eigenschaften des Robotergreifers sind zum Beispiel die Geometrie der Greiferbacken und die Materialbeschaffenheit. Alle funktionalen Eigenschaften ergeben sich aus diesen beobachtbaren Eigenschaften, also zum Beispiel die maximale Öffnungsweite der Greiferbacken.
□

Außer den funktionalen und beobachtbaren Eigenschaften eines Objekts gibt es keine weiteren Eigenschaften. Jede Eigenschaft eines Objektes ist damit entweder funktional oder beobachtbar.

Nach der GDT ist ein *Entwurfsproblem* statisch gesehen die Abbildung der funktionalen Eigenschaften auf die beobachtbaren Eigenschaften, dynamisch gesehen der Prozeß der Findung dieser Abbildung. Es gibt dabei natürlich für eine Problemstellung im Allgemeinen mehrere Lösungen, so daß die Abbildung der funktionalen Eigenschaften auf die beobachtbaren Eigenschaften nicht eindeutig ist, also eine kreative menschliche Leistung darstellt.

In der Realität kann allerdings nicht davon ausgegangen werden, daß alle funktionalen Eigenschaften als statische Anforderungen vorgegeben sind und ein Entwickler dazu mit

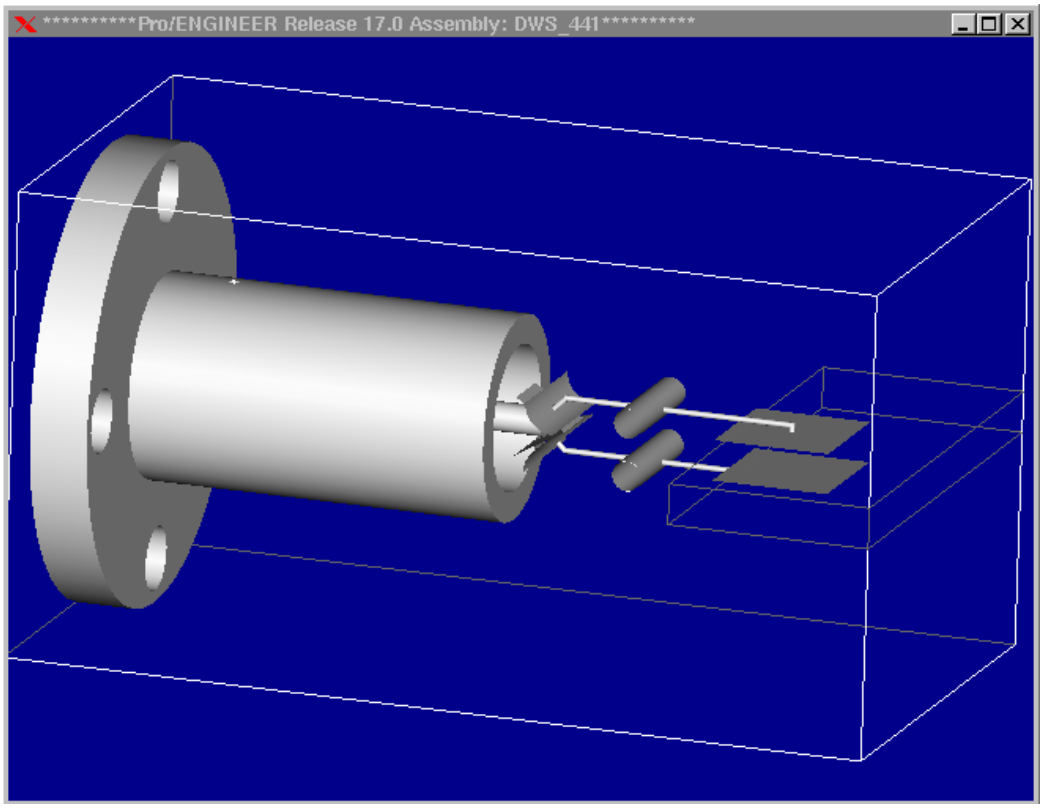


Abbildung 2.2: Der Robotergreifer in Pro/Engineer

Anforderung	Ursache
Mindestanforderung an die Haltekraft	Kundenwunsch
Höchstanforderung an die Baugröße und das Eigengewicht	Kundenwunsch
Höchstanforderung an den Öldruck	Folge der Motorenwahl
Maximalbelastung der Materialien	DIN-Normen

Abbildung 2.3: Anforderungen an den Robotergreifer

einem Schlag alle beobachtbaren Eigenschaften angibt. Die Entwicklung eines Produktes ist vielmehr ein stark iterativer und arbeitsteiliger Prozeß, der zwar auf der GDT basiert, sich aber weitaus komplizierter darstellt.

In den folgenden Abschnitten werden die wichtigsten Eigenschaften des Produktentwicklungsprozesses herausgearbeitet, die gleichzeitig die Anforderungen für eine sinnvolle und brauchbare Rechnerunterstützung darstellen, die also auch als Grundlagen für diese Arbeit dienen. Um den Entwicklungsprozeß sinnvoll behandeln zu können, muß dabei zuerst das gesuchte Produkt betrachtet werden.

2.3 Das Produkt

Nach der GDT ist das gesuchte Produkt ein Objekt der realen Welt mit funktionalen und beobachtbaren Eigenschaften.

Definition 2.3.1 (Produkt und Produkteigenschaften)
Ein **Produkt** *ist ein Objekt der realen Welt.*

Die Menge $\mathbb{G} = \{X_1 \ldots X_p\}$ *der funktionalen und beobachtbaren Eigenschaften des gesuchten Produkts sind die* **Produkteigenschaften**. *Jeder Produkteigenschaft* X *wird eine Domäne* $\mathbb{D}_X$ *zugeordnet. Die* **Domäne** *ist eine Menge von Werten, die die Produkteigenschaft annehmen kann.*

□

Definition 2.3.2 (Entwurfsraum)
Der **Entwurfsraum** *ist ein* p*-dimensionaler Raum, der durch die* p *Produkteigenschaften aufgespannt wird.*

□

Das Ziel der Entwicklung ist genau ein Produkt, also eine physikalisch realisierbare Kombination von Ausprägungen aller p Produkteigenschaften. Im Lauf der Entwicklung konvergiert die Beschreibung des Produkts allerdings langsam. Erst am Ende der Entwicklung ist die Beschreibung eindeutig und daher das zu fertigende Produkt genau bekannt. Während der Entwicklung sind das gesuchte Produkt und damit auch die Ausprägungen der einzelnen Produkteigenschaften noch nicht genau festgelegt. Im Allgemeinen sind zunächst noch nicht einmal alle Eigenschaften selbst bekannt. Es sind also Mittel nötig, mit denen sich Produkte während des gesamten Entwicklungsprozesses einheitlich beschreiben lassen.

2.3.1 Spezifikationen

Definition 2.3.3 (Spezifikation)
Eine **Spezifikation** $\mathcal{S}()$ *auf den Domänen* $\mathbb{D}_1, \ldots \mathbb{D}_d$ *ist eine Funktion von dem kartesischen Produkt der Domänen* $\mathbb{D}_1, \ldots \mathbb{D}_d$ *auf die Menge der booleschen Werte wahr und* $falsch$.

$$\mathcal{S}() : \mathbb{D}_1 \times \mathbb{D}_2 \times \ldots \mathbb{D}_d \rightarrow \{wahr, falsch\}$$

□

Mit der Zuordnung von Domänen zu den Produkteigenschaften sind Spezifikationen auch auf Produkteigenschaften definiert, in dem sie einfach auf die den Produkteigenschaften zugeordneten Domänen angewendet werden.

Definition 2.3.4 (Benutzte Domänen oder Eigenschaften)
Die Spezifikation $\mathcal{S}()$ *über den Produkteigenschaften* $\mathbb{G}$ **benutzt** *genau die Produkteigenschaften* $Var(\mathcal{S}())$ *mit* $Var(\mathcal{S}()) \subseteq \mathbb{G}$, *die einen Einfluß auf das Ergebnis ausüben.*

$$X \in Var(\mathcal{S}()) \Leftrightarrow$$

$$\exists_{x_1,x_2 \in \mathbb{D}_X, x_1 \neq x_2} \exists_{y_1 \in \mathbb{D}_{Y_1} \ldots y_n \in \mathbb{D}_{Y_n}} \textit{mit } \{Y_1 \ldots Y_n\} = \mathbb{G} \setminus X \;\; \mathcal{S}(x_1, y_1 \ldots y_n) \neq \mathcal{S}(x_2, y_1 \ldots y_n)$$

□

Eine Spezifikation $\mathcal{S}()$, die auf den Produkteigenschaften $\mathbb{G}$ definiert ist, beschreibt somit einen *Freiraum*, nämlich genau die Menge aller Produkte, die die Eigenschaften $\mathbb{G}$ aufweisen und deren Ausprägungen von Eigenschaften die Spezifikation $\mathcal{S}()$ den Wert *wahr* zuweist. Die Interpretation einer Spezifikation hat damit zwei Seiten. Einerseits schließt eine Spezifikation eine ganze Reihe von Produkten aus, nämlich genau die, denen die Spezifikation den Wert $falsch$ zuordnet. Auf der anderen Seite stellt die Spezifikation aber auch einen Freiraum für die weitere Produktentwicklung dar, in dem üblicherweise eben nicht nur einer Ausprägung von Eigenschaften der Wert *wahr* zugewiesen wird. Für die weitere Entwicklung entsteht hier also der Freiraum, diese noch möglichen Produkte weiter einzugrenzen.

Die Dynamik der Menge von Produkteigenschaften $\mathbb{G}$ stellt hierbei kein Problem dar, da eine auf $\mathbb{G}$ definierte Spezifikation $\mathcal{S}()$ sehr einfach durch eine Spezifikation $\mathcal{T}()$ ersetzt werden kann, wobei $\mathcal{T}()$ auf $Var(\mathcal{T}()) = \mathbb{G}$ definiert ist, aber nur Eigenschaften aus $Var(\mathcal{S}())$ benutzt. Problematisch wird dabei nur, wenn Produkteigenschaften komplett aufgegeben werden. Daher wird im Folgenden davon ausgegangen, daß immer nur neue Eigenschaften hinzukommen, nie aber bestehende Produkteigenschaften aufgegeben werden.

Um vernünftig mit Spezifikationen und Freiräumen umgehen zu können, sind noch einige Definitionen nötig.

Definition 2.3.5 (Erfüllung, Kompatibilität, Implikation und Gleichheit)
Ein Punkt im Entwurfsraum **erfüllt** *oder* **genügt** *einer Spezifikation, wenn die Spezifikation dem Punkt den Wert wahr zuweist.*

Eine Menge von Spezifikationen ist **kompatibel** *oder* **widerspruchsfrei**, *wenn es mindestens einen Punkt im Entwurfsraum gibt, der alle Spezifikationen erfüllt.*

Zwei Spezifikationen $\mathcal{S}_1()$ und $\mathcal{S}_2()$ auf $\mathbb{G}$ sind **gleich** *($\mathcal{S}_1() = \mathcal{S}_2()$), wenn sie für alle Kombinationen von Ausprägungen von Eigenschaften aus $\mathbb{G}$ das gleiche Ergebnis liefern.*

Eine Spezifikation $\mathcal{S}_1()$ auf $\mathbb{G}$ impliziert eine andere Spezifikation $\mathcal{S}_2()$ auf $\mathbb{G}$, geschrieben als $\mathcal{S}_1() \Rightarrow \mathcal{S}_2()$, wenn der durch $\mathcal{S}_1()$ beschriebene Freiraum eine Teilmenge des von $\mathcal{S}_2()$ beschriebenen Freiraums ist.

□

An dieser Stelle bleibt noch offen, wie die Spezifikationen und die damit verbundenen Freiräume beschrieben werden. Im Entwicklungsprozeß werden aber die Freiräume schrittweise eingeengt werden. Dies geschieht durch Schnittbildung der einzelnen Freiräume, die durch die Konjunktion der entsprechenden Spezifikationen erreicht wird.

Definition 2.3.6 (Konjunktion von Spezifikationen)
Seien den Produkteigenschaften in $\mathbb{G}$ die Domänen $\mathbb{D}_1 \ldots \mathbb{D}_p$ zugeordnet. Die **Konjunktion** *einer Menge von Spezifikationen $\mathcal{S}_1() \ldots \mathcal{S}_n()$ über den Produkteigenschaften $\mathbb{G}$, geschrieben als*

$$\bigwedge_{i=1}^{n} \mathcal{S}_i() \textit{ oder } \mathcal{S}_1() \wedge \mathcal{S}_2() \wedge \ldots \mathcal{S}_n()$$

ist eine Spezifikation über $\mathbb{G}$, die genau den Kombinationen von Eigenschaftsausprägungen den Wert wahr zuweist, denen alle einzelnen Spezifikationen den Wert wahr zuweisen.

$$\forall_{x_1 \in \mathbb{D}_1 \ldots\ldots x_p \in \mathbb{D}_p} \; (\bigwedge_{i=1}^{n} \mathcal{S}_i())(x_1 \ldots x_p) = wahr \Leftrightarrow$$
$$\forall_{i=1..n} \; \mathcal{S}_i(x_1 \ldots x_p) = wahr$$

□

Die Konjunktion einer Menge von Spezifikationen entspricht damit genau der Schnittmenge der durch die Spezifikationen beschriebenen Freiräume. Die Konjunktion beziehungsweise die Schnittmengenbildung ist nach Definition 2.3.6 kommutativ und assoziativ.

Im Produktentwicklungsprozeß werden also immer mehr Spezifikationen gesammelt, die das Produkt immer genauer beschreiben. Eine Sammlung von Spezifikationen ist daher eine Produktbeschreibung.

Definition 2.3.7 (Produktbeschreibung)
Eine **Produktbeschreibung** *ist eine Konjunktion von Spezifikationen. Eine Produktbeschreibung ist* **vollständig**, *wenn es nur genau einen Punkt im Entwurfsraum gibt, der alle Spezifikationen der Produktbeschreibung erfüllt.* □

Da die Konjunktion von Spezifikationen kommutativ und assoziativ ist, spielt aus der Sicht des Ergebnisses die Reihenfolge der Spezifikationen innerhalb der Produktbeschreibung keine Rolle. Die zeitliche Historie von Spezifikationen kann daher dort vernachlässigt werden.

2.3.2 Die Produktentwicklung

Nach der GDT ist die Produktentwicklung eine Abbildung der in Form von Anforderungen vorgegebenen funktionalen Eigenschaften auf die gesuchten beobachtbaren Eigenschaften. In der Praxis kann diese Abbildung nicht mit einem Schlag erstellt werden, sondern wird iterativ erarbeitet. Dabei werden ständig neue Informationen über das Produkt in Form von Spezifikationen eingebracht, die aber durchaus im Widerspruch zu schon bekannten Spezifikationen stehen können. Solche Widersprüche bedeuten, daß es kein Produkt mehr gibt, welches allen Spezifikationen genügt. Offensichtlich sind damit die Anforderungen an das Produkt oder die daraus abgeleiteten Spezifikationen widersprüchlich und müssen daher revidiert werden.

Im einfachsten Fall beginnt ein Entwickler mit den Anforderungen an das gesuchte Produkt und erweitert diese um Spezifikationen, die die gesuchten beobachtbaren Eigenschaften beschreiben. Abgesehen von den nötigen Rückschritten, die die Revisionen in der Entwicklung repräsentieren, wird dabei das gesuchte Produkt durch die Spezifikationen immer genauer beschrieben, bis nur noch ein einziges Produkt existiert, welches alle gesammelten Spezifikationen erfüllt. Dieses einzige Produkt ist genau die Lösung, nämlich das gesuchte Produkt.

2.4 Der arbeitsteilige Produktentwicklungsprozeß

In der Praxis ist die Produktentwicklung allerdings weitaus komplizierter als im vorherigen Abschnitt dargestellt. Die Abbildung der Anforderungen auf die durch Spezifikationen beschriebenen beobachtbaren Eigenschaften des gesuchten Produkts wird in einem langwierigen, arbeitsteiligen und iterativen Prozeß erarbeitet. Dieser Produktentwicklungsprozeß beginnt zum Zeitpunkt t_0 mit den Anforderungen als Eingabe und endet zum Zeitpunkt t_e mit der vollständigen Produktbeschreibung als Ergebnis. Sämtliche Entwicklungsarbeit findet also zwischen den Zeitpunkten t_0 und t_e statt.

Oberflächlich betrachtet ist eine Produktionsumgebung strikt in einzelne *Phasen der Produktionsprozeßkette* eingeteilt. In jeder Phase werden Informationen aus anderen Phasen verwertet und neue Informationen erzeugt. Als Phasen lassen sich in herkömmlichen Produktionsumgebungen die Konzeption eines neuen Produkts, Konstruktion, Fertigungsplanung, Fertigung und Montage, Vertrieb und Wartung identifizieren ([Wie86]), wobei die Phasen noch weiter unterteilt werden können (zum Beispiel Anforderungsmodellierung, Funktionsmodellierung, Prinzipmodellierung und Gestaltung bei der Konstruktion).

2.4.1 Phasenüberlappung im Entwicklungsprozeß

In der Praxis ist die konzeptionell einfache und intuitive sequentielle Ausführung der Phasen weder wünschenswert noch machbar. Wartet nämlich jede Phase, bis alle Ergebnisse der vorherigen Phasen komplett vorliegen, so dauert der Konstruktionsprozeß zu lange für die Wettbewerbsfähigkeit. Überdies werden Fehlentwicklungen, etwa Spezifikationen, die sich bei der Ausgestaltung als zu teuer herausstellen, erst sehr spät erkannt und sind entsprechend schwierig zu korrigieren.

So werden beispielsweise in der Automobilindustrie mit ersten Entwürfen schon umfangreiche Simulationen bezüglich der Verformung bei Unfällen oder einfach nur über das Schwingungsverhalten und den dadurch entstehenden Geräuschen durchgeführt. Da solche Simulationen aufwendig sind, muß gleichzeitig das Produkt weiterentwickelt werden.

Die Phasen der Produktentwicklung müssen damit überlappend ausgeführt werden, so daß nachfolgende Phasen bereits auf den unfertigen Teilergebnissen der vorangegangenen Phasen beginnen. Nachfolgende Phasen müssen damit nicht nur neue Entscheidungen treffen, sondern auch Annahmen über die noch nicht geleisteten Arbeiten der vorangegangenen Phasen anstellen.

Die prinzipielle Vorgehensweise bei der Überlappung von Phasen der Produktentwicklung ist überall dieselbe, es spielt also für diese Analyse und damit für die ganze Arbeit keine Rolle, welche Phasen betrachtet werden. Aufgrund der Ausgeprägtheit des Szenarios in diesem Bereich sollen im Folgenden vor allem die (Teil-)Phasen der Konstruktion betrachtet werden.

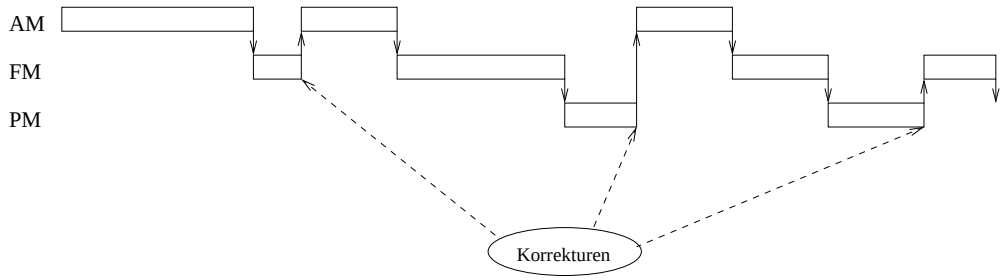


Abbildung 2.4: Der Produktentwicklungsprozeß ohne Phasenüberlappung

In Abbildung 2.4 sind die Phasen Anforderungsmodellierung (AM), Funktionsmodellierung (FM) und Prinzipmodellierung (PM) schematisch abgebildet. Dabei beginnen die jeweils nachfolgenden Phasen erst nach Abschluß der vorangegangenen Phasen. Dies führt aber dazu, daß im Fall von Fehlentwicklungen die Ergebnisse einer bereits abgeschlossenen Phase nachträglich geändert werden müssen. Abgeschlossene Phasen müssen also unter Umständen reaktiviert werden. Dieses iterative Vorgehen ist sehr ineffizient und zieht den Produktentwicklungsprozeß entsprechend in die Länge.

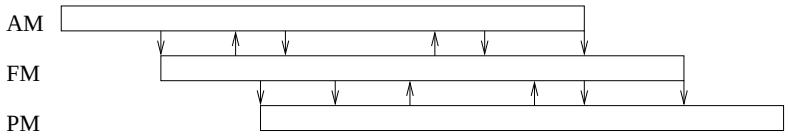


Abbildung 2.5: Der Produktentwicklungsprozeß mit Phasenüberlappung

In Abbildung 2.5 sieht man die überlappende Ausführung der Phasen Anforderungsmodellierung, Funktionsmodellierung und Prinzipmodellierung. Da nachfolgende Phasen sehr früh Zugriff auf die Teilergebnisse der vorgelagerten Phasen haben, können

sie Fehlentwicklungen sofort erkennen und die zuständigen Entwickler darauf aufmerksam machen. Rückläufe von nachfolgenden Phasen kommen also früher und verkürzen die notwendigen Iterationszeiten. Der gesamte Produktentwicklungsprozeß wird damit deutlich schneller.

2.4.2 Arbeitsteiligkeit

Am Produktentwicklungsprozeß ist eine Menge von Entwicklern $E_1 \dots E_e$ beteiligt, wobei jeder Entwickler genau einer Phase zugeteilt ist. Da im Lauf der Entwicklung das gesuchte Produkt üblicherweise in mehrere Teile zerlegt wird, können mehrere Entwickler in derselben Phase an unterschiedlichen Teilen des gesuchten Produkts arbeiten. Diese Aufteilung der Arbeit innerhalb einer Phasen an mehrere Entwickler nennt man *phasenlokale Arbeitsteiligkeit.*

Jeder Entwickler arbeitet damit nur auf einem Teil des zu fertigenden Produkts, also auf einem *Teilentwurfsraum.* Diese Teilentwurfsraum ist ein q-dimensionaler Raum, der durch q Produkteigenschaften aufgespannt wird und ein Unterraum des p-dimensionalen Entwurfsraums darstellt ($q \leq p$).

Beispiel 2.4.1 (Teilentwurfsraum)
Im Lauf der Entwicklung kann der Robotergreifer zum Beispiel in die Teilbereiche Greiferbacken und Druck-Kraft-Wandlung aufgeteilt werden. Entwickler aus dem Bereich Greiferbacken dürfen damit nur noch die Backengeometrie ändern. Deren Teilentwurfsraum wird also nur durch die Eigenschaften der Greiferbacken aufgespannt. Andere Entwickler arbeiten im Bereich der Druck-Kraft-Wandlung. Deren Teilentwurfsraum wird durch die Eigenschaften der Druck-Kraft-Wandlung, also zum Beispiel Zylinderdicke und Keilgeometrie, aufgespannt.

Da die Greiferbacken aber gerade durch den Keil bewegt werden, muß dieser Teil der Greiferbacken, nämlich die den Keil berührende Schräge, von beiden Entwicklern bearbeitet werden können. Diese Schräge ist dann also eine Produkteigenschaft, die einen eindimensionalen Teilentwurfsraum aufspannt. Dieser Raum ist genau die Schnittmenge der Teilentwurfsräume der einzelnen Entwickler. □

Arbeiten innerhalb einer Phase mehrere Entwickler parallel, so sind deren Teilentwurfsräume zu weiten Teilen disjunkt, da sie auf unterschiedlichen Produktteilen arbeiten. Gemeinsam sind aber zumindest die Schnittstellen zwischen den Produktteilen, sie liegen also in der Schnittmenge der Teilentwurfsräume. Darüber hinaus entsteht durch die Phasenüberlappung allerdings auch die sogenannte *phasenübergreifende Arbeitsteiligkeit* auf denselben Produktteilen. Dort ist zu erwarten, daß die Teilentwurfsräume der Entwickler sehr große Überlappungen haben, wobei die Entwickler in unterschiedlichen Phasen tätig sind.

Für das Entstehen der gesuchten Produktbeschreibung ist es unwesentlich, in welcher Phase und an welchen Produktteilen die Entwickler arbeiten. Für die Modellierung des Prozesses ist nur entscheidend, daß zur selben Zeit mehrere Entwickler arbeitsteilig an sich überlappenden Teilentwurfsräumen arbeiten. Die mit dieser Arbeitsteiligkeit einhergehenden Probleme der Datenkonsistenz sind eine der größten Herausforderungen für die Rechnerunterstützung dieses arbeitsteiligen Entwicklungsprozesses.

2.4.3 Der Konsens als Kompromiß der Entwickler

Aufgrund der unterschiedlichen Aufgaben und Zuständigkeiten hat jeder Entwickler eine eigene Perspektive, die zu einer persönlichen „Vorstellung“ des zu fertigenden Produkts führt. Diese Vorstellung ist nur in den Köpfen vorhanden und daher sehr vage und sicherlich nicht formal bearbeitbar. Im Lauf des Entwicklungsprozesses werden sich

diese Vorstellungen daher sowohl durch die Interaktion mit anderen Entwicklern, wie auch durch das Überdenken des Entwicklers selbst präzisieren und dabei immer wieder ändern. Da das Ziel des Entwicklungsprozesses ein eindeutiges Produkt ist, müssen die Vorstellungen aller Entwickler zu einem *Konsens* kombiniert werden, mit dem alle Entwickler zufrieden sind. Die persönlichen Vorstellungen müssen also im Lauf des Entwicklungsprozesses so weit angepaßt werden, daß sie kompatibel sind. Da die persönlichen Vorstellungen einer maschinellen Bearbeitung nicht zugänglich sind, muß sich die Rechnerunterstützung auf die Verwaltung der beobachtbaren Manifestierungen der Vorstellungen in Form von Spezifikationen beschränken. Dazu gehört es, den Entwicklern Möglichkeiten zu geben, ihre persönlichen Vorstellungen in Form von Spezifikationen in einen Konsens einzubringen (siehe Abbildung 2.6). Dieser Konsens ist eine Produktbeschreibung, die kontinuierlich verfeinert wird, bis sie das gesuchte Produkt eindeutig beschreibt.

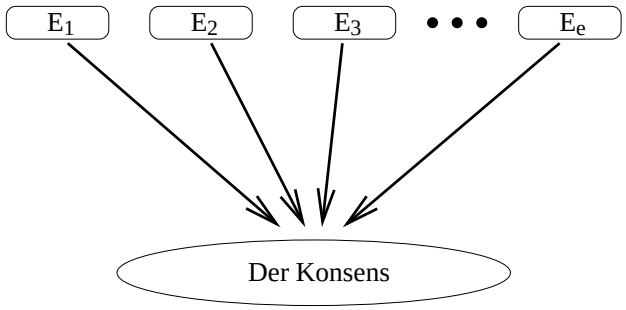


Abbildung 2.6: Die persönlichen Vorstellungen und der Konsens

Wenn sich die Vorstellungen der Entwickler widersprechen (das heißt, eine in den Konsens einzubringende Spezifikation ist inkompatibel mit den bereits im Konsens enthaltenen Spezifikationen), so muß dieser Widerspruch erkannt und die beteiligten Entwickler müssen darauf hingewiesen werden, so daß diese, eventuell auch in einem persönlichen Gespräch, ihre Vorstellungen überdenken, bis die neue Spezifikation widerspruchsfrei in den Konsens eingebracht werden kann. Dabei gibt es keinerlei Prioritäten der sich widersprechenden Spezifikationen, etwa im Hinblick auf den Zeitpunkt der Einbringung in den Konsens. Es wird also des öfteren vorkommen, daß schon im Konsens enthaltene Spezifikationen aufgrund eines Widerspruchs wieder entfernt und revidiert werden müssen.

2.4.4 Beständigkeit

Alle gesammelten Spezifikationen repräsentieren Informationen über das gesuchte Produkt. Die Spezifikationen wurden von Entwicklern eingebracht mit dem Ziel, das gesuchte Produkt zu beschreiben. Daher muß das gesuchte Produkt auch allen gesammelten Spezifikationen genügen. Im Normalfall müssen also alle einmal eingebrachten Spezifikationen für immer im System erhalten bleiben. In der Praxis gibt es aber immer wieder Fehlentwicklungen, die so nicht aufrechterhalten werden können, da sie zu Widersprüchen mit anderen Spezifikationen führen. In diesem Fall dürfen die widersprüchlichen Spezifikationen allerdings nicht einfach gelöscht werden, da der Entwickler ja davon ausgeht, daß das gesuchte Produkt seine Spezifikationen erfüllt. Der Entwickler muß also zumindest benachrichtigt werden. Besser noch ist es, seine Zustimmung zur Entfernung einer seiner Spezifikationen einzuholen, oder sogar die Entfernung von Spezifikationen nur von dem Entwickler ausführen zu lassen, der sie auch eingebracht hat. Damit erkennt der Entwickler seine Fehlentwicklung und kann diese auch nachbessern.

Definition 2.4.1 (Beständigkeit einer Spezifikation)
Die **Beständigkeit** *einer eingebrachten Spezifikation heißt, daß diese Spezifikation nur*

noch durch die explizite Aufforderung von autorisierten Personen aus dem Konsens entfernt werden kann.

□

In traditionellen Datenbanken wird stattdessen oft die Definition der *Dauerhaftigkeit* verwendet. Diese Dauerhaftigkeit einer abgeschlossenen Transaktion hat den Effekt, daß die Transaktion nie mehr rückgängig gemacht werden kann, während eingebrachte Spezifikationen wieder aus dem Konsens entfernt werden können. Auf der anderen Seite können in traditionellen Datenbanken nach Abschluß einer Transaktion alle Ergebnisse der Transaktion sofort wieder überschrieben werden. Hier offenbart sich der große Vorteil des spezifikationsbasierten Ansatzes, da hier eine einmal eingebrachte Spezifikation immer erhalten bleibt, also auch für das gesuchte Produkt gilt, es sei denn, sie wurde von dem zuständigen Entwickler explizit wieder entfernt.

2.4.5 Entwurfsentscheidungen

Neue Erkenntnisse über das zu fertigende Produkt entstehen in Form von Entwurfsentscheidungen. Entwurfsentscheidungen bestehen aus Voraussetzungen und Ergebnissen, die nur in einem Verbund Sinn machen.

Definition 2.4.2 (Entwurfsentscheidung)
Eine **Entwurfsentscheidung** *D ist ein Paar von zwei Mengen, Spezifikationen $\mathbb{V}$ (Voraussetzungen) und $\mathbb{R}$ (Ergebnisse).*

□

Üblicherweise sind die Voraussetzungen einer Entwurfsentscheidung schon im Konsens vorhanden und werden somit von einem Entwickler gelesen. Dieser basiert seine Entwurfsentscheidung auf den gelesenen Voraussetzungen, die zu neuen Erkenntnissen in Form von Spezifikationen, den Ergebnissen, führt. Diese Ergebnisse werden dann in den Konsens eingebracht. Damit wird die Entwurfsentscheidung beständig, ist also im Konsens dokumentiert.

Definition 2.4.3 (Beständigkeit einer Entwurfsentscheidung)
Eine Entwurfsentscheidung ist **beständig**, *wenn alle Spezifikationen aus $\mathbb{V}$ und $\mathbb{R}$ beständig sind.*

□

Wird jeder Entwurfsentscheidung ein eindeutiger Zeitpunkt zugewiesen, zum Beispiel das Einbringen ihrer ersten oder letzten Spezifikation, so ist die Produktentwicklung eine Sequenz von Entwurfsentscheidungen. Darüber hinaus gibt es zwischen den Entwurfsentscheidungen allerdings kausale Zusammenhänge der Art, daß die Durchführung einer Entwurfsentscheidung möglicherweise von vorangegangenen Entscheidungen abhängt. Das heißt, eine Entscheidung zieht möglicherweise eine zweite Entscheidung nach sich, die ohne die erste Entscheidung nie zustande gekommen wäre.

Kausale Zusammenhänge zwischen Entwurfsentscheidungen sind sehr schwer zu erkennen und liegen möglicherweise auch außerhalb des Konsenses, etwa in informeller Kommunikation zwischen den Entwicklern. Im schlimmsten Fall muß daher davon ausgegangen werden, daß eine Entwurfsentscheidung kausal von allen vorangegangenen abhängt. Die Rechnerunterstützung kann sich daher nur auf die beobachtbaren Abhängigkeiten beschränken.

Definition 2.4.4 (Liest von)
Wenn die Ergebnisse ($\mathbb{R}_1$) der Entwurfsentscheidung D_1 und die Voraussetzungen ($\mathbb{V}_2$) der Entwurfsentscheidung D_2 nicht disjunkt sind, dann **liest** *D_2 von D_1.*

□

Die Liest-von-Beziehung ist damit klar und eindeutig definiert und kann daher auch anhand der Entwicklung des Konsenses überprüft werden. Eine Rechnerunterstützung

des Produktentwicklungsprozesses kann nur die Liest-von-Beziehung betrachten. Alle anderen kausalen Zusammenhänge zwischen den Entwurfsentscheidungen werden damit ignoriert. Durch die beobachtbare Liest-von-Beziehung auf den Entwurfsentscheidungen wird über allen Entwurfsentscheidungen eine Halbordnung definiert.

Konsequenz 2.4.1 (Liest-von-Halbordnung)
Die **Liest-von-Halbordnung** $L = (\{D_1, D_2, ...\}, \prec)$ *ist eine Halbordnung über der Menge der Entwurfsentscheidungen* $\{D_1, D_2, ...\}$*, wobei zwei Entwurfsentscheidungen* D_i, D_j *vergleichbar sind, wenn* D_i *von* D_j *liest.*
□

2.4.5.1 Semantik von Entwurfsentscheidungen ($\mathbb{V}$, $\mathbb{R}$)

Eine Entwurfsentscheidung repräsentiert ein Arbeitsfragment eines Entwicklers, wobei $\mathbb{V}$ die Voraussetzungen für die Entscheidung und $\mathbb{R}$ die Ergebnisse der Entscheidung darstellen. In der Regel sind die Spezifikationen in $\mathbb{V}$ schon vorgegeben, etwa durch andere Entwurfsentscheidungen. In diesem Fall entscheidet der Entwickler, daß unter der Voraussetzung, daß das Produkt die Spezifikationen in $\mathbb{V}$ erfüllt, das Produkt auch die Spezifikationen in $\mathbb{R}$ erfüllen muß. Die Spezifikationen in $\mathbb{R}$ sind hierbei also neue Informationen über das Produkt und damit ein Fortschritt in der Produktentwicklung.

Aufgrund der Phasenüberlappung müssen Entwickler aber teilweise auf unvollständigen Arbeiten der vorangegangenen Phasen aufsetzen. Dabei kann es durchaus vorkommen, daß die nötigen Voraussetzungen für weitere Entscheidungen noch nicht gegeben sind, der Entwickler aber trotzdem Entscheidungen treffen muß. In diesem Fall bringt der Entwickler selbst die Voraussetzungen ein und nimmt damit die Ergebnisse der vorangegangenen Phasen vorweg. An dieser Stelle wird die Richtung des Informationsflusses im Bezug auf die Phasen der Produktentwicklung umgedreht. Die nachgereichten Ergebnisse der vorangegangenen Phasen müssen mit den schon gemachten Voraussetzungen der nachfolgenden Phasen abgeglichen werden. Dabei können die von nachfolgenden Phasen getroffenen Voraussetzungen durchaus invalidiert werden. In diesem Fall muß die Entwurfsentscheidung komplett zurückgesetzt werden, da deren Annahmen nicht mehr erfüllt sind. Die Umkehrung der Richtung des Informationsflusses bezieht sich dabei nur auf die Phasen der Produktentwicklung. Betrachtet man die zeitliche Sequenz von Entwurfsentscheidungen, so fließen Informationen immer in dieselbe Richtung. Nur im Fall der Rücksetzung fließen tatsächlich Information zurück, in dem neue Entwurfsentscheidungen die Rücksetzungen vorangegangener Entwurfsentscheidungen bewirken können (natürlich nur mit Zustimmung von autorisierten Entwicklern).

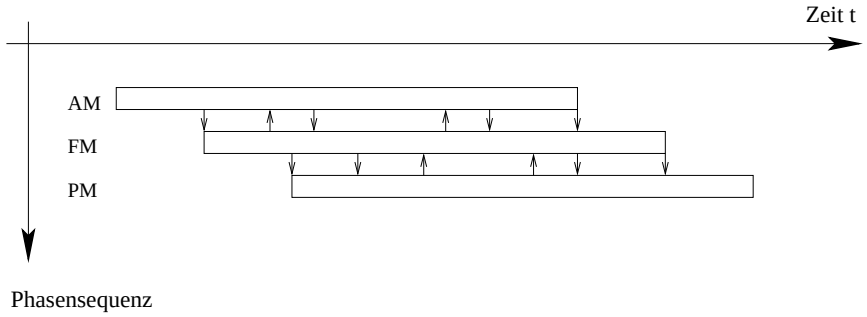


Abbildung 2.7: Umkehr des Informationsflusses

Anschaulich dargestellt ist der Umkehr des Informationsflusses in Abbildung 2.7. Dabei sind die zeitlich überlappenden Phasen der Produktentwicklung (Anforderungsmodellierung, Funktionsmodellierung und Prinzipmodellierung) unter der Zeit aufgetragen. Die nach unten gerichteten Pfeile stellen dabei den normalen Informationsfluß von einer

Phase zur nachfolgenden Phase dar. Die umgekehrte Richtung ist zwar zeitlich gesehen ebenfalls vorwärts gerichtet, fließt aber von nachgelagerten Phasen zu vorangegangenen Phasen.

Eine Entwurfsentscheidung macht nur als ganzes Sinn, ist also eine *atomare Einheit.* Auf der anderen Seite entsteht kein Schaden, wenn zwischenzeitlich nur Teile von Entwurfsentscheidungen im Konsens enthalten sind, solange auf Dauer die Atomizität sichergestellt wird. Das heißt, für das Einbringen oder Entfernen einer Entwurfsentscheidung ist keinerlei Isolation nötig. In Zwischenzuständen dürfen also auch unvollständige Entwurfsentscheidungen vorhanden sein, die im weiteren Verlauf entweder komplettiert oder wieder entfernt werden müssen.

Da Entwurfsentscheidungen also unteilbar sind, sind sie auch die Einheiten für die Rücksetzungen. Wenn eine Spezifikation zurückgesetzt werden muß, dann muß mit dieser die komplette Entwurfsentscheidung zurückgesetzt werden, zu der die rückzusetzende Spezifikation gehört. Daraus resultiert schon der Wunsch nach kleinen Entwurfsentscheidungen. Es liegt also in der Verantwortung der Entwickler, die Entwurfsentscheidungen so klein wie möglich aber so groß wie nötig zu machen.

Mit der *Unteilbarkeit einer Entwurfsentscheidung* verwischen die Unterschiede zwischen $\mathbb{V}$ und $\mathbb{R}$. Oftmals repräsentiert eine Entwurfsentscheidung auch ein physikalisches Gesetz, zum Beispiel die Haltekraft am Robotergreifer, die aus dem Öldruck und der Geometrie hervorgeht. Dabei kann die Haltekraft aus Öldruck und Geometrie berechnet werden, aber auch umgekehrt der nötige Öldruck aus Haltekraft und Geometrie. $\mathbb{V}$ und $\mathbb{R}$ zeigen also das Entstehen einer Entwurfsentscheidung auf. $\mathbb{V}$ war schon bekannt oder wurde angenommen und $\mathbb{R}$ resultiert aus $\mathbb{V}$. Sobald die Entwurfsentscheidung im Konsens enthalten ist, beschreiben alle Spezifikation das Produkt, egal ob sie als Voraussetzung oder Ergebnis einer Entwurfsentscheidung eingebracht wurden.

2.5 Abgekoppeltes Arbeiten

Selbst in kleineren Unternehmen kann der Produktentwicklungsprozeß heute nicht mehr mit einem zentralen System unterstützt werden. In erster Linie liegt dies an der Notwendigkeit, *mobile Arbeitsstationen* einzusetzen, mit denen Entwickler jederzeit und überall auf die Daten zugreifen können, auch wenn keine Netzverbindung möglich ist. Das bedeutet, daß einzelne Entwickler *abgekoppelt*, also ohne Netzverbindung mit den anderen Entwicklern, arbeiten. Hierfür müssen die Daten zumindest teilweise *repliziert* auf den Stationen verwaltet werden.

Ein weiterer Grund für die replizierte Datenhaltung sind akzeptable Antwortzeiten für Datenzugriffe. Betrachtet man zum Beispiel globale Unternehmen mit mehreren Niederlassungen auf unterschiedlichen Kontinenten, so kann ein Zugriff auf einen einzelnen Dienstgeber nicht alle Niederlassung mit akzeptabler Performanz bedienen. Auch hier wird die replizierte Datenhaltung unumgänglich.

Auf der technischen Ebene kommen auch Performanzaspekte in LANs zum Tragen, die auch dort eine replizierte Datenhaltung wünschenswert machen. An dieser Stelle geht es eher um Optimierungen. Das heißt, die Replikation ist hier erwünscht, aber nicht zwingend durch den Produktentwicklungsprozeß vorgegeben.

Definition 2.5.1 (Replikation)
Replikation *bezeichnet die Verwaltung von Teilen desselben Konsenses an physikalisch getrennten Stellen. Jede Ausprägung des Konsenses heißt* **Replikat**.

Wenn ein Replikat genau den kompletten Konsens, also die an allen Replikaten bekannten Spezifikationen, enthält, heißt das Replikat **vollständig**, *andernfalls* **unvollständig**.
□

Wird ein Großteil der Arbeiten abgekoppelt durchgeführt, dann sind Replikate in den wenigsten Fällen vollständig, da meistens irgendwo ein abgekoppeltes Replikat mit neuen Entwurfsentscheidungen existiert, die noch nicht weitergegeben wurden. Es ergeben sich daraus eher Versionen, die sich asynchron immer wieder aneinander annähern und dann doch wieder voneinander entfernen. Diese Versionen sind daher nach Definition 2.5.1 unvollständige Replikate, werden im Folgenden aber - wenn keine Verwechslung möglich ist - nur noch als Replikate bezeichnet.

Jeder Entwickler arbeitet auf einem Replikat. Damit entwickeln sich die Replikate unterschiedlich und die Entwickler sehen auch nur den Zustand ihres Replikats, also auch nur die Entwurfsentscheidungen, die an ihrem Replikat eingebracht wurden. Zu Beginn des Entwicklungsprozesses gibt es nur ein Replikat. Mit der Zeit können neue Replikate als Kopien existierender Replikate erstellt werden, die sich dann unabhängig entwickeln. Ab und zu müssen die Replikate durch Kommunikation miteinander abgeglichen werden, so daß alle seit dem letzten Abgleich eingegangenen Entwurfsentscheidungen ausgetauscht werden. Direkt nach einem solchen Abgleich beinhalten alle am Abgleich beteiligten Replikate damit dieselben Entwurfsentscheidungen. Werden alle existierenden Replikate miteinander abgeglichen, so sind die Replikate vollständig. Anstatt eines Abgleichs können zwei Replikate auch miteinander verschmolzen werden, das heißt, die Replikate werden abgeglichen und eines der beteiligten Replikate wird danach aufgelöst. Durch den vorangegangenen Abgleich gehen die Informationen des Replikats dabei nicht verloren.

Da Spezifikationen erst an einem Replikat eingebracht werden und dann langsam an die anderen Replikate weitergeleitet werden, ergibt sich durch die Replikation eine weitere Halbordnung, die beschreibt, welche Entwurfsentscheidung von welcher anderen Entscheidung lesen kann.

Definition 2.5.2 (Kann-lesen-von-Halbordnung)
Die **Kann-lesen-von-Halbordnung** $L' = (\{D_1, D_2, ...\}, \lhd)$ *ist eine Halbordnung über der Menge der Entwurfsentscheidungen* $\{D_1, D_2, ...\}$*, wobei zwei Entwurfsentscheidungen* $D_i, D_j \in \{D_1, D_2, ...\}$ *vergleichbar sind, wenn* D_j *von* D_i *lesen kann, das heißt die Ergebnisse von* D_i *müssen beim Einbringen von* D_j *auf dem Replikat sichtbar sein, auf dem* D_j *eingebracht wird.*
□

Entwurfsentscheidungen sind also nur vergleichbar, wenn die zeitlich zuerst getroffene zum Zeitpunkt des Einbringens der zweiten Entwurfsentscheidung auf deren Replikat bekannt ist. Zwei Entwurfsentscheidungen, die auf unterschiedlichen Replikaten eingebracht wurden, sind damit erst vergleichbar, wenn die ältere der beiden Entscheidungen vor der zweiten Entscheidung zwischen den beteiligten Replikaten ausgetauscht wurde. So ein Austausch von Entwurfsentscheidungen sollte so oft wie möglich durchgeführt werden, damit die Entscheidungen möglichst schnell auch an anderen Replikaten sichtbar werden.

Während der Abkopplung sind naturgemäß keine Abgleiche der Replikate möglich, so daß diese sich immer weiter voneinander entfernen. Nach der Wiederankopplung sollte daher möglichst sofort ein Abgleich stattfinden. Sofern der laufende Betrieb dadurch nicht beeinträchtigt wird, sollten im angekoppelten Betrieb ständige Abgleiche durchgeführt werden.

Offensichtlich ist die Liest-von-Halbordnung eine Teilmenge der Kann-lesen-von-Halbordnung. Durch volle Synchronisation, also den Abgleich aller Replikate nach jeder eingebrachten Spezifikation, kann die Kann-lesen-von-Halbordnung zu einer totalen Ordnung werden[1], wobei die Liest-von-Halbordnung in der Regel trotzdem partiell bleibt. Die damit einhergehende Transparenz der Replikation wird allerdings durch den hohen Aufwand für die Abgleiche erkauft. Insbesondere bei mobilen Arbeitsstationen ohne Netzverbindung sind solche Abgleiche zumindest zeitweise unmöglich.

[1] Das bedingt allerdings die serielle Durchführung der Entwurfsentscheidungen, da zwei aktive Entwurfsentscheidungen nicht unbedingt voneinander lesen können.

Eine Entwurfsentscheidung kann bei Verwendung von Replikation also nicht von allen vorangegangenen Entwurfsentscheidung lesen und sieht damit nur eine Teilmenge der schon vorhandenen Informationen. Dies stellt allerdings noch kein Problem dar, da zwar nicht alle, aber doch keine falschen Informationen sichtbar sind. Auf der anderen Seite können aber auf unterschiedlichen Replikaten im Rahmen von nicht vergleichbaren Entwurfsentscheidungen widersprüchliche Spezifikationen eingebracht werden. Dieser Widerspruch wird erst bei einem Abgleich explizit und bedarf dann der Zurücksetzung von Entwurfsentscheidungen.

2.5.1 Verhinderung von widersprüchlichen Spezifikationen

In Abschnitt 2.4.4 wurde die Beständigkeit von Spezifikationen als wichtige Forderung für die Unterstützung des Produktentwicklungsprozesses herausgearbeitet und später auf Entwurfsentscheidungen erweitert. Da diese Beständigkeit im replizierten Umfeld nur durchgesetzt werden kann, wenn zu keinem Zeitpunkt widersprüchliche Spezifikationen auf den Replikaten existieren, müssen Widersprüche also verhindert werden. Eine spätere Auflösung von zuvor entstandenen und nicht bemerkten Widersprüchen ist nicht akzeptabel.

Die anzustrebende Lösung ist also eine pessimistische, die eine Absprache vor der Abkopplung erfordert, so daß die Zusammenführung bei der Wiederankopplung widerspruchsfrei erfolgen kann.

Beispiel 2.5.1 (Die Entwicklung eines repliziert verwalteten Konsenses)
Betrachtet man zum Beispiel die Abbildung 2.8, dann kann folgende Matrix für die Kann-Lese-von-Halbordnung aufgestellt werden:

Kann lesen von	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8
D_1				*	*			*
D_2				*	*			*
D_3						*		*
D_4								
D_5								*
D_6								*
D_7								*
D_8								

So kann zum Beispiel D_5 von D_1 lesen, da die Replikate R_{3a} und R_{3b} abgeglichen wurden. Damit sind die Ergebnisse der Entwurfsentscheidung D_1 auch auf dem Replikat R_{3b} vorhanden und D_5 kann diese lesen. Auf der anderen Seite kann D_6 nicht von D_1 lesen, da die Ergebnisse von D_1 in R_{3a} eingeflossen sind, aber von dort nicht zu R_{3c} gekommen sind. Also kann D_6 diese nicht lesen.

Die Sicherung der Widerspruchsfreiheit bezieht sich dabei immer auf die gerade aktuellen Replikate. In Abbildung 2.8 sind dies R_{6a}, R_{6b} und R_{6c}. Die Widerspruchsfreiheit mit älteren Replikaten muß nicht geprüft werden, da diese geprüft wurde, als diese Replikate aktuell waren. Anders ausgedrückt müssen genau die Replikate widerspruchsfrei sein, aus denen noch keine neuen Replikate durch Entwurfsentscheidungen, Abgleiche oder Aufspaltungen hervorgegangen sind.

□

2.6 Rechnerunterstützung des Entwicklungsprozesses

Die Analyse des Entwicklungsprozesses definiert die Anforderungen an die Rechnerunterstützung des Entwicklungsprozesses, die in diesem Abschnitt aufgelistet werden.

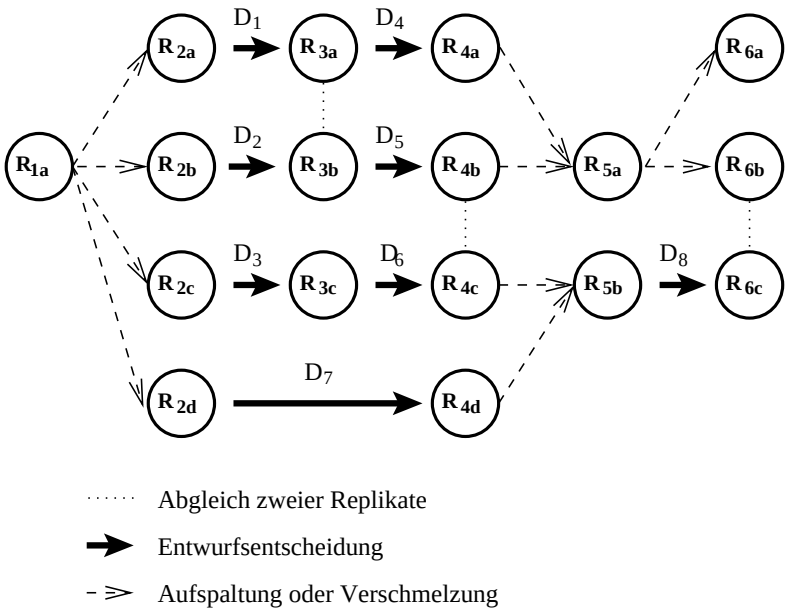


Abbildung 2.8: Die Entwicklung der Replikate

Schon auf den ersten Blick ist zu sehen, daß für eine solche Rechnerunterstützung Datenbanktechniken notwendig sind, da große Mengen von Spezifikationen, also Daten, verwaltet und verarbeitet werden müssen. These dieser Arbeit ist, daß das betrachtete Problem daher vorrangig ein Datenbankproblem ist.

2.6.1 Anforderungen an die Rechnerunterstützung

In erster Linie muß die Rechnerunterstützung des Produktentwicklungsprozesses den *evolvierenden Konsens* verwalten. Hierzu gehört die Speicherung der Spezifikationen mit der Gruppierung zu Entwurfsentscheidungen. Entwickler müssen in der Lage sein, neue Spezifikationen als Teil einer Entwurfsentscheidung einzubringen. Dabei muß die Rechnerunterstützung die Widerspruchsfreiheit der Spezifikationen garantieren und gegebenenfalls neue Spezifikationen ablehnen. Hinzu kommt die Evaluierung der auf den Produkteigenschaften verbleibenden Freiräume, die den Entwicklern auf Anfrage mitgeteilt werden müssen. Einzelne Entwurfsentscheidungen müssen vollständig wieder aus dem Konsens entfernt werden können, ohne dabei die Unteilbarkeit anderer Entwurfsentscheidungen zu verletzen.

Durch die zeitweise Abkopplung einiger Benutzer und der damit verbundenen replizierten Datenhaltung wird die Aufgabe der Rechnerunterstützung um einiges komplizierter. Dies bedeutet die Verwaltung mehrerer Replikate und die Möglichkeit zur Erstellung, Verschmelzung und zum Abgleich der Replikate. Wie schon in Abschnitt 2.5 aufgezeigt, ist das Hauptproblem bei der Replikation die Durchsetzung der Widerspruchsfreiheit der Vereinigungsmenge aller Replikate.

Die hohe Arbeitsteiligkeit führt dazu, daß mehrere Entwickler gleichzeitig auf denselben Daten arbeiten. Hierbei soll die Rechnerunterstützung die Entwickler so wenig wie möglich in ihrer Arbeit behindern. Idealerweise soll jeder Entwickler zu jedem Zeitpunkt auf alle Daten zugreifen können, ohne dabei irgendwelche Wartezeiten in Kauf nehmen zu müssen.

Da es sich hier um ein *Datenbankproblem* handelt, gilt es zu fragen, in wie weit traditionelle Datenbanken für die Rechnerunterstützung des Produktentwicklungsprozesses geeignet sind und wenn nicht, woran das liegt.

2.6.2 Traditionelle Datenbanken

Traditionelle Datenbanken erfüllen die genannten Forderungen nicht und sind daher nicht für die Unterstützung des Produktentwicklungsprozesses geeignet. Die Verwaltung des Konsenses mit Unterstützung von Entwurfsentscheidungen kann noch über ein geeignetes Schema erreicht werden. Auch das Einbringen neuer Spezifikationen und die Berechnung der verbleibenden Freiräume auf einer Produkteigenschaft stellen noch kein Problem dar[2].

Das Problem traditioneller Datenbanken beginnt bei der selektiven Entfernung einzelner Entwurfsschritte, die so in Datenbanken nicht möglich ist - dort kann nur chronologisch zurückgesetzt werden. Desweiteren bieten Datenbanken üblicherweise transaktionale Konsistenzgarantien, die hier nicht gefragt sind. Stattdessen ist die Widerspruchsfreiheit zu sichern, die eher mit der Prüfung von Mengen logischer Konsistenzbedingungen zu tun hat.

Abkopplung und Replikation ist in Datenbanken ein lange diskutiertes Problem. Dabei ist allerdings zu beachten, daß die Forschung in diesem Bereich sich vor allem auf die Konsistenzsicherung replizierter Datenbestände im Sinn der Isolation und unabhängiger Zugriffe auf die Replikate konzentriert. Da im Produktentwicklungsprozeß die Widerspruchsfreiheit als Konsistenzbedingung gänzlich verschieden ist, können hier die Replikationsprotokolle traditioneller Datenbanken keine Lösung bieten.

Die Isolation in Datenbanken wird üblicherweise als Konfliktserialisierbarkeit durch das Zwei-Phasen-Sperrprotokoll durchgesetzt. Damit verbunden ist eine deutliche Einschränkung der Nebenläufigkeit, die im vorliegenden Szenario nicht akzeptabel ist. Erweiterte Transaktionsmodelle bis hin zu Modellen für die Produktentwicklung haben hier versucht Abhilfe zu schaffen. Allerdings wird dabei in irgendeiner Form die höhere Nebenläufigkeit durch Einschränkungen bei den Konsistenzgarantien erkauft.

Das Auseinanderklaffen der Eigenschaften liegt in erster Linie an der Philosophie der Anwendungen. Datenbanken verwirklichen normalerweise eine Miniwelt, die eine reale Welt nachbilden soll. Es gibt also einen Bezug zur einer realen Welt, die sich in Schritten ändert. Diese Schritte werden in der Datenbank durch isoliert ablaufenden Transaktionen nachgebildet. In der Produktentwicklung hingegen wird ein einzelner Zustand der realen Welt konstruiert, in dem sich der Zustand der Datenbank immer weiter an diesen einen gesuchten Zustand annähert. Alle Entwickler arbeiten gemeinsam und gleichzeitig an diesem Zustand. Isolation ist damit unerwünscht. Die Konsistenz muß vielmehr im Hinblick auf den einen gesuchten Endzustand definiert werden.

Traditionelle Datenbanken sind also nur als Basistechnik brauchbar, lösen aber das vorliegende Problem nicht oder nur unzureichend. Daraus ergibt sich ein offener Handlungsbedarf für die vorliegende Arbeit, der im folgenden Abschnitt näher beleuchtet werden soll.

2.6.3 Konsequenzen für die Rechnerunterstützung

Im Wesentlichen lassen sich die folgenden drei Kriterien für eine gute Rechnerunterstützung des Produktentwicklungsprozesses nennen. Diese drei Kriterien allein sind zwar noch nicht ausreichend, stellen aber die größten Herausforderungen dar. Andere Anforderungen sind schon durch bestehende Systeme abgedeckt und ohne größere Probleme auf neue Lösungen übertragbar. Die folgende Liste stellt also die Forschungslücke dar, die von der vorliegenden Arbeit geschlossen werden soll. Die drei Punkte müssen dabei in ihrem Zusammenspiel betrachtet werden. Die Diskussion der Abkopplung macht zum Beispiel nur im Kontext der ersten zwei Punkte Sinn.

[2] Natürlich kann letztere Berechnung sehr aufwendig werden. Dies ist jedoch kein Problem der Datenbank, sondern eine Frage der Wahl der Formalismen zur Darstellung von Spezifikationen.

Widerspruchsfreiheit als Konsistenzbedingung:
Die Widerspruchsfreiheit des Konsenses stellt in der Produktentwicklung die zentrale Konsistenzbedingung dar. Diese Widerspruchsfreiheit muß automatisch von der Rechnerunterstützung durchgesetzt werden, wobei die Nebenläufigkeit der einzelnen Entwickler so wenig wie möglich eingeschränkt werden darf. Weitere Garantien im Sinn von traditionellen Datenbanken sind hierbei nicht nötig, da das Einbringen von Spezifikationen ja kommutativ und assoziativ ist und daher keine Isolation nötig ist (Atomizität und Dauerhaftigkeit wird bereits auf der Ebene der Operationen vorausgesetzt).

Monotone Informationsakquisition mit selektivem Rücksetzen:
Die Datenbasis ist nicht mehr in Datenelemente eingeteilt, sondern besteht aus einer Formel von Spezifikationen über den Produkteigenschaften. Neue Spezifikationen können als Teil einer Entwurfsentscheidung hinzugefügt werden. Ganze Entwurfsentscheidungen müssen von autorisierten Entwicklern auch wieder entfernt werden können, ohne andere Entwurfsentscheidungen zu beeinträchtigen. Ein Rücksetzen auf einen früheren Zustand ist daher nicht akzeptabel. Durch die Rücksetzung einer Entwurfsentscheidung darf die „Liest-von-Beziehung“ nicht so weit beeinträchtigt werden, daß nachfolgenden Entwurfsentscheidungen Teile der Voraussetzungen entzogen werden. Es muß also eine Möglichkeit geben, diese „Liest-von-Beziehung“ aufzulösen, in dem jede Entwurfsentscheidung ein atomares und autonomes Paket von Voraussetzungen und Ergebnissen ist, also nicht mehr von anderen Entwurfsentscheidungen abhängt.

Unterstützung der Abkopplung:
Der Produktentwicklungsprozeß muß in einer verteilten Umgebung mit abgekoppelten Entwicklern unterstützt werden. Das heißt, die Verwaltung des Konsenses mit Durchsetzung der Beständigkeit und selektivem Rücksetzen einzelner Entwurfsentscheidungen muß repliziert erfolgen, wobei die Nebenläufigkeit möglichst wenig eingeschränkt werden soll. Das Hauptproblem hierbei ist die Durchsetzung der Widerspruchsfreiheit der Vereinigungsmenge aller unabhängig evolvierenden Replikate.

Kapitel 3

Stand der Forschung

In diesem Kapitel werden die drei Hauptprobleme bei der Rechnerunterstützung des Produktentwicklungsprozesses separat betrachtet und existierende Ansätze, die damit im Zusammenhang stehen, vorgestellt. Die Ansätze werden insbesondere im Hinblick auf deren Nutzbarkeit für die vorliegende Arbeit untersucht.

Die Dissertation von Rose Sturm über Dynamische Regelmengen ([Stu97]) unterstützt den Entwicklungsprozeß in der Architektur. Diese Arbeit ist im Zusammenhang mit mehreren vorliegenden Problemen relevant und soll daher gesondert am Ende des Kapitels diskutiert werden. Gleiches gilt für die Dissertation von Vijay A. Saraswat über Constraint Stores ([Sar93]), die sich zwar nicht mit der Produktentwicklung beschäftigt, trotzdem aber einige interessante Ansätze vereint.

3.1 Widerspruchsfreiheit als Konsistenzbedingung

In diesem Abschnitt sind Arbeiten zu untersuchen, die die Widerspruchsfreiheit als Konsistenzbedingung unterstützen. In erster Linie gibt es keine Datenbanken, die Spezifikationen als Daten verwalten und deren Widerspruchsfreiheit sicherstellen. Am nächsten an der Fragestellung liegen daher die Möglichkeiten der expliziten Angabe von Konsistenzbedingungen für Datenbanksysteme. Aus dem Bereich der Constraintsysteme kennt man allerdings das Problem der Widerspruchsfreiheit sehr gut, so daß auch dieser Bereich untersucht werden muß.

Prädikatssperren sind ein Ansatz aus dem Datenbankbereich, in dem die Konsistenzsicherung in Abhängigkeit von der Erfüllung von Prädikaten durchgeführt wird. Damit können auch Prädikatssperren Ansätze für die vorliegende Arbeit liefern.

3.1.1 Forschungsergebnisse

3.1.1.1 Explizite Konsistenzbedingungen

Explizite Konsistenzbedingungen werden in Datenbanken eingesetzt, um Sachverhalte der realen Welt in der Datenbank sicherzustellen ([EGLT76]). Diese Bedingungen müssen explizit spezifiziert werden und werden dann automatisch vom System geprüft. Dabei sind drei Ebenen zu unterscheiden:

1. Der Zustand der Datenbasis, der den Zustand der realen Welt nachbildet.

2. Die expliziten Konsistenzbedingungen, die Sachverhalte in der realen Welt beschreiben und auf den Zustand der Datenbasis angewendet werden können.

3. Die Konsistenz der Konsistenzbedingungen, also die Erfüllbarkeit oder auch Widerspruchsfreiheit. Sind die Konsistenzbedingungen nicht erfüllbar, so gibt es keinen Zustand der Datenbasis, der alle expliziten Konsistenzbedingungen erfüllt ([BM86]).

Explizite Konsistenzbedingungen sind nur als Zusatz zur herkömmlichen Definition der Konsistenz über Transaktionen sinnvoll. Die vollständige Beschreibung aller möglichen Zusammenhänge in der realen Welt ist in den meisten Fällen praktisch unmöglich.

3.1.1.2 Constraintprobleme

Constraintprobleme (Constraint Satisfaction Problems) über endlichen Wertebereichen sind bereits seit den siebziger Jahren im Bereich der Künstlichen Intelligenz erforscht worden. Allgemein betrachtet besteht ein Constraintproblem aus einer Menge von Variablen und Constraints, das heißt Relationen, die Eigenschaften dieser Variablen sowie Beziehungen zwischen den Variablen ausdrücken. Eine Lösung des Constraintproblems ist eine Belegung der Variablen, so daß alle Constraints erfüllt sind ([FA97]).

Eine wesentliche Erkenntnis ist, daß allgemeine Constraintprobleme NP-vollständig sind, was insbesondere in Datenbanken aufgrund der großen Datenmengen ein massives Problem darstellt. Durch die Wahl entsprechender Constraintsysteme kann die Entscheidbarkeit allerdings auch in polynomialer Zeit gelöst werden. Leider trifft dies nur für sehr einfache Constraintsysteme zu und meistens ist auch der polynomiale Aufwand nicht gerade mit kleinen Potenzen versehen.

3.1.1.3 Prädikatssperren

Bei der Argumentation über Transaktionsmodelle wird oft von Zugriffen auf Datenelementen geredet. Viele Anfragen beziehen sich aber nicht genau auf ein Datenelement, sondern betrachten eine ganze Reihe von Datenelementen, so zum Beispiel die folgende SQL-Anfrage:

```
select   *
from     employee
where    salary > 40000
```

Fügt gleichzeitig eine andere Transaktion einen neuen Mitarbeiter in die Relation ein, so kann die SQL-Anfrage davon betroffen sein oder nicht. Ein interessanter Ansatz zur Lösung dieses Problems sind Prädikatssperren ([EGLT76] und [GR93]). Dabei wird eine Sperre in Form einer Bedingung gesetzt, wie sie auch in einer „where“-Klausel vorkommen kann, um das Einfügen und Löschen von Datensätzen zu verhindern, auf die das Prädikat zutrifft.

Leider haben Prädikatssperren einige Nachteile, allen voran die NP-Vollständigkeit der Kompatibilitätsprüfung, die einen realen Einsatz in einem Datenbanksystem verhindern. Alternativ dazu werden Sperren mit unterschiedlichen Granularitäten verwendet, die als ein Spezialfall der Prädikatssperren gesehen werden können.

3.1.2 Erkenntnisse für die vorliegende Arbeit

Betrachtet man die vorliegende Arbeit im Zusammenhang mit den expliziten Konsistenzbedingungen, so stellen genau die Spezifikationen diese expliziten Konsistenzbedingungen dar, die die Konsistenz des Datenbankzustands beschreiben, wobei hier nie ein Datenbankzustand gespeichert wird, sondern immer nur die Konsistenzbedingungen. Die in dieser Arbeit geforderte Widerspruchsfreiheit der Spezifikationen entspricht also der Erfüllbarkeit der expliziten Konsistenzbedingungen, die in [BM86] betrachtet wird. Dort geht es in erster Linie um die Entscheidbarkeit der Erfüllbarkeit, die ja von den Spezifikationsformen abhängt. An dieser Stelle können also Erkenntnisse bei der Entscheidbarkeit der Widerspruchsfreiheit gewonnen werden, die für die Diskussion der verschiedenen Spezifikationsformen interessant sind.

Auf der anderen Seite liefern auch Constraintprobleme Erkenntnisse zur Entscheidbarkeit der Widerspruchsfreiheit. Um die Effizienz der Rechnerunterstützung zu sichern, sind hier Einschränkungen auf zwei Ebenen nötig. Einerseits müssen die verwendeten Constraintsysteme beschränkt werden, wobei die Fachrichtung der Constraintprobleme wesentliche Erkenntnisse über die Komplexität solcher Constraintsysteme liefert. Andererseits muß auch die Größe des Problems reduziert werden. Hierfür bieten die betrachteten Ansätze allerdings keine Hilfe.

3.2 Monotone Informationsakquisition mit selektivem Rücksetzen

Diese Problemstellung kann zweigeteilt werden. Auf der einen Seite steht die monotone Informationsakquisition, also die Verwaltung einer Spezifikationsmenge, die durch neue Spezifikationen angereichert werden kann. Hierzu sind erweiterte Datenmodelle zu betrachten. Auf der anderen Seite steht das selektive Rücksetzen einzelner Entwurfsentscheidungen. In Datenbanksystemen werden Rücksetzungen auf Basis von Transaktionen durchgeführt, so daß hier Transaktionsmodelle untersucht werden müssen. Ein weiterer Ansatz, der auch als eine Art Transaktionsmodell angesehen werden kann, ist die Versionierung, die hier aber gesondert diskutiert werden soll.

Im Bereich der Fuzzy-Informationssysteme gibt es Ansätze zur Verfeinerung einer Lösungsbeschreibung durch Revision. Auch diese Fuzzy-Revisionsmodelle müssen als Ansatz für das vorliegende Problem betrachtet werden.

3.2.1 Forschungsergebnisse

3.2.1.1 Erweiterte Datenmodelle

Datenbanken wurden entwickelt, um ein Abbild der realen Welt zu speichern, das idealerweise zu jedem Zeitpunkt konsistent ist. Das von Codd entwickelte relationale Datenmodell kann daher nur mit scharfen Daten umgehen, so daß auch immer ein voll definierten Zustand der Miniwelt verwaltet wird. In der Praxis ist das Wissen über die reale Welt aber oftmals unvollkommen, so daß Datenbanken diese unvollkommenen Informationen über die reale Welt speichern können müssen.

Der erste Ansatz einer Erweiterung des relationalen Datenmodells für die Speicherung von unvollkommenen Daten sind NULL-Werte, die ausdrücken, daß der Wert entweder gar nicht oder nur unzureichend bekannt ist. In den frühen achtziger Jahren gab es dann mehrere Ansätze, das relationale Datenmodell um unvollkommene Daten zu erweitern, zum Beispiel mit Hilfe der Fuzzy-Mengen. Einen guten Überblick über Ansätze in diesem Gebiet bieten [BK95] und [Che98].

3.2.1.2 Transaktionsmodelle

Transaktionsmodelle erlauben die Gruppierung mehrerer Operationen zu Transaktionen ([GR93]). Für eine Transaktion gibt es dabei die Garantien der atomaren, isolierten und dauerhaften Ausführung, die kombiniert mit der Anforderung der Konsistenzerhaltung an die Transaktionen selbst als ACID-Garantien bezeichnet werden. Die Dauerhaftigkeit hat hierbei allerdings eine gänzlich andere Bedeutung als die im Produktentwicklungsprozeß nötige Beständigkeit. Die Auswirkungen einer dauerhaften Transaktion bleiben einerseits für immer in der Datenbasis erhalten, können aber von allen anderen Benutzern beliebig abgeändert werden, sind also nicht beständig. Wenn die Effekte einer Transaktion allerdings nicht einfach überschrieben, sondern wieder rückgängig gemacht werden sollen, dann führt dies zu kaskadierenden Rücksetzungen. Dieses Problem wird oft durch kompensierende Transaktionen behandelt. Diese kompensierenden Transaktionen müssen dann die mit der Rücksetzung entstandenen Inkonsistenzen reparieren.

3.2.1.3 Versionierung

Die Versionierung von Daten kann auf unterschiedliche Art erfolgen. Der Ansatz, unabhängige Versionen zu erzeugen, die später wieder zusammengeführt werden, wird nicht betrachtet. Hier geht es um die Speicherung der Historie in Form einer Sequenz von Zuständen der Datenbasis. Abgesehen von dem deutlich erhöhten Speicherplatzbedarf, der insbesondere mit der Zeit steigt, hat die Versionierung einige Vorteile. Das Rücksetzen auf frühere Zustände wird sehr viel einfacher, ja ist fast sogar trivial. Die Versionen können auch für mehr Flexibilität genutzt werden, in dem Benutzer auf unterschiedlichen Versionen arbeiten. Hierbei muß allerdings die Konsistenz sichergestellt werden.

3.2.1.4 Das Revisionsmodell für Fuzzy-Mengen

In [Wit02b] wird ein Revisionsmodell für Fuzzy-Mengen vorgestellt, mit dem Lösungen durch kontinuierliche Sammlung von möglicherweise unvollkommenen und inkonsistenten Informationen gefunden werden. Dabei beschreibt eine Menge von Fuzzy-Mengen die Lösung. Diese Beschreibung wird durch neue Fuzzy-Mengen angereichert, bis die Lösungsbeschreibung eindeutig ist. Das Hinzufügen neuer Informationen kann entweder über eine Expansion geschehen, die abgelehnt wird, wenn die entstehende Produktbeschreibung inkonsistent ist. Alternativ wird eine Revision angeboten, die neue Informationen auf jeden Fall einbringt, unter Umständen aber alte Informationen automatisch entfernt, um die neue entstehende Lösungsbeschreibung konsistent zu machen. Über Abkopplung, explizite Rücksetzungen und alternative Spezifikationsformen wird allerdings nicht diskutiert.

3.2.2 Erkenntnisse für die vorliegende Arbeit

Die Speicherung von unvollkommenen Informationen, zum Beispiel in Form von Fuzzy-Mengen oder Constraints, kann heute nicht mehr als Problem angesehen werden. Allerdings weist die Verwaltung des Konsenses die Besonderheit auf, daß Daten gesammelt und nicht überschrieben werden. Diese Besonderheit stellt aber kein ernst zu nehmendes Problem dar, ist vielmehr ein bisher noch nicht verfolgter Lösungsansatz für den arbeitsteiligen Produktentwicklungsprozeß.

Das selektive Rücksetzen einzelner Entwurfsschritte birgt da schon viel größere Herausforderungen. Transaktionsmodelle bieten hier keinen Ansatz, da die Rücksetzung einer Transaktion zu kaskadierenden Abbrüchen führt oder in irgendeiner Form, zum Beispiel durch kompensierende Transaktionen, wieder repariert werden muß. Auch mit der Versionierung sind nur Rücksetzungen auf alte Versionen möglich, was natürlich

einer Rücksetzung aller nach dieser Version eingebrachten Entwurfsentscheidungen entspricht. An dieser Stelle gibt es also in der aktuellen Forschung keine Ansätze, die für die vorliegende Arbeit hilfreich sind.

Das Revisionsmodell für Fuzzy-Mengen gibt einige Ergebnisse für Fuzzy-Mengen als Spezifikationsformen. So muß dort zum Beispiel die Entscheidbarkeit der Widerspruchsfreiheit effizient durchgeführt werden können und sogar eine möglichst minimale Menge von Fuzzy-Mengen gefunden werden, nach deren Entfernung eine andere Fuzzy-Menge konfliktfrei eingebracht werden kann.

3.3 Unterstützung der Abkopplung

Praktisch alle Arbeiten zum Thema Konsistenzsicherung und Abkopplung beziehungsweise Replikation gehen von der Isolation als Konsistenzbedingung aus und versuchen diese, oder eine abgeschwächte Form der Isolation, in einer replizierten Umgebung durchzusetzen ([Kot96]). Da hier mit der Widerspruchsfreiheit eine grundlegend andere Konsistenzbedingung verwendet wird, können die bekannten Arbeiten zur Replikation damit kaum Anregungen liefern. Interessant sind allerdings Quorenverfahren, die die Synchronisation von Replikaten mit geringstmöglicher Wartezeit erledigen.

Der einzige Ansatz für die replizierte Datenhaltung, der von einem alternativen Konsistenzbegriff ausgeht, ist das Demarkationsprotokoll (im englischen Original: Demarcation Protocol), das nachfolgend vorgestellt wird.

3.3.1 Forschungsergebnisse

3.3.1.1 Quorenverfahren

In vielen Fällen wird in einem replizierten System gefordert, daß alle Operationen an allen Replikaten zwar nicht gleichzeitig, aber doch in derselben Reihenfolge eintreffen. Auf den ersten Blick sind hierfür einige gravierende Einschränkungen nötig, so zum Beispiel volle Synchronisation, also der Zugriff auf alle Replikate gleichzeitig, oder die Auswahl eines speziellen Replikats, auf dem dann alle Operationen ausgeführt und von dort an die anderen Replikate weitergeleitet werden.

Quorenverfahren minimieren die Restriktionen in solchen Fällen durch Definition eines Quorums ([GMB85]). Operationen müssen innerhalb eines Quorums synchron ausgeführt werden. Die Hauptanforderung ist dann, daß zwei aufeinanderfolgende Quoren nicht disjunkt sind. Dies kann auf unterschiedlichste Arten erreicht werden (zum Beispiel mehr als die Hälfte aller existierenden Replikate pro Quorum).

3.3.1.2 Demarkationsprotokoll

Das Demarkationsprotokoll ([BMGM94]) ersetzt globale Bedingungen, die für zwei Datenelemente auf unterschiedlichen Rechnern gelten, durch mehrere lokale Bedingungen. So könnten zum Beispiel die Datenelemente X und Y auf unterschiedlichen Rechnern liegen, wobei global die Bedingung $x+y > 100$ erfüllt sein muß. Anstatt für jeden Zugriff auf X gleichzeitig einen Zugriff auf Y zu machen, wird die globale Bedingung durch die beiden Bedingungen $x > 50$ und $y > 50$ ersetzt, die jeweils lokal überprüft werden können. Gegebenenfalls werden die beiden einzelnen Bedingungen angepaßt (zum Beispiel $x > 30$ und $y > 70$). Der gesamte Freiraum ($x + y > 100$) wird also durch einen kleineren Freiraum ($x > 50 \wedge y > 50$) so ersetzt, daß der kleinere Freiraum jeweils lokal an den Datenelementen ohne Kommunikation mit anderen Datenelementen überprüft werden kann. Erst wenn der kleine Freiraum überschritten werden soll, muß durch Kommunikation geprüft werden, ob eine Anpassung der kleineren Freiräume so möglich ist, daß der ursprüngliche große Freiraum gewahrt bleibt und trotzdem die geplante Änderung eingebracht werden kann.

3.3.2 Erkenntnisse für die vorliegende Arbeit

Quorenverfahren bieten einen interessanten Ansatz zur asynchronen Durchsetzung einer globalen Historie. Dafür müssen Operationen immer in einem Quorum ausgeführt werden und zwei aufeinanderfolgende Quoren dürfen nicht disjunkt sein. Für die vorliegende Arbeit bedeutet dies, daß im Fall einer Abkopplung Quoren nur noch auf dem abkoppelnden Replikat oder auf anderen, damit nicht verbundenen Replikaten zustande kommen können. Das bedeutet wiederum, daß Operationen entweder ausschließlich auf dem abgekoppelten Replikat ausgeführt werden dürfen, oder daß das abgekoppelte Replikat eben gar keine Operationen ausführen darf. Quoren sind daher für die Durchführung von Entwurfsentscheidungen uninteressant, müssen allerdings für den Abgleich der Replikate wieder betrachtet werden.

Das Demarkationsprotokoll ist hierbei schon viel interessanter. Ein ähnliches Vorgehen kann zur Aufteilung der Freiräume an den einzelnen Replikaten genutzt werden, so daß an den Replikaten lokal im Rahmen der dort verfügbaren Freiräume neue Entwurfsentscheidungen eingebracht werden können, ohne Kommunikation mit anderen Replikaten. Bei geeigneter Wahl der Aufteilung kann dadurch die Widerspruchsfreiheit aller Spezifikationen an allen Replikaten garantiert werden. Reichen lokal verfügbare Freiräume nicht mehr aus, so ist wie im Demarkationsprotokoll Kommunikation zwischen den Replikaten nötig, um die verfügbaren Freiräume entsprechend anzupassen.

3.4 Sonstige Relevante Arbeiten

Schon bei der Durchsetzung der Widerspruchsfreiheit als Konsistenzbedingung wird möglichst uneingeschränkte Nebenläufigkeit gefordert. Beim abgekoppelten Arbeiten kann keine uneingeschränkte Nebenläufigkeit mehr gefordert werden, die Einschränkungen sollen sich aber auf ein Minimum reduzieren. Hierzu gibt es einige Arbeiten unter dem Begriff CSCW (Computer Supported Cooperative Work), die hier betrachtet werden sollen.

Die schon in der Einleitung dieses Kapitels angesprochenen Arbeiten von Sturm und Saraswat lassen sich nicht in eine der drei obigen Kategorien einordnen und sollen daher hier separat diskutiert werden. Darüber hinaus werden in diesem Abschnitt noch Constraint Databases diskutiert, da auch diese keinem der drei Hauptprobleme zugeordnet werden können.

3.4.1 Forschungsergebnisse

3.4.1.1 Computer Supported Cooperative Work (CSCW)

Kooperatives Arbeiten mehrerer Benutzer kann sicherlich nicht durch Isolation im Sinn von Transaktionen unterstützt werden. Der Bereich des CSCW widmet sich daher der Unterstützung des kooperativen Arbeitens mehrerer Benutzer. Der Schlüsselbegriff in diesem Zusammenhang ist das Gruppenbewußtsein (Awareness). Darunter wird verstanden, daß jeder Benutzer Informationen über die Aktivitäten der anderen Benutzer erhält und somit einen Eindruck des Gesamtsystems mit allen beteiligten Benutzern und deren Aktivitäten erhält und sein Verhalten darauf einstellen kann ([DB92]). Die Anwender beeinflussen sich also gegenseitig und sind sich dessen auch bewußt, so daß die Isolation überflüssig wird. Eine einfache Form der Awareness wurde bereits vor mehreren Jahren in das Entwicklungssystem des SFB 346 eingebaut, so daß die vorliegende Arbeit darauf zurückgreifen kann ([HKLP98]).

3.4.1.2 Constraint Databases

Die Erkenntnis, daß in relationalen Datenbanken immer nur explizit aufgezählte Tupel enthalten sind und damit die Beschreibung von unendlichen Datenmengen praktisch unmöglich ist, hat zur Entwicklung von Constraint Databases ([Rev98], [KLP00]) geführt. Hierbei beschreibt ein Tupel in einer Relation einen Constraint. Damit kann ein Tupel mehrere Objekte beschreiben bis hin zu unendlich vielen.

Ein Objekt ist in einer Constraint Database enthalten, wenn es ein Tupel gibt, welches das Objekt beschreibt. Damit sind Constraint Databases Disjunktionen von Tupeln.

3.4.1.3 Dynamische Regelmengen

Im Jahr 1997 wurde die Dissertation von Rose Sturm ([Stu97], [SML95]) veröffentlicht, die in einem Kooperationsprojekt mit dem Fachbereich Architektur entstand. Auch in dieser Arbeit wurde die Bedeutung von unvollkommenen Daten erkannt und in Form von Constraints verwaltet. Als Szenario dient die Konstruktion eines Schulgebäudes, dessen wichtige Eigenschaften als Constraints verwaltet werden und so die Aufdeckung von Widersprüchen ermöglichen. Abgesehen von temporär geduldeten Inkonsistenzen, die später aufgelöst werden müssen, ist in dieser Arbeit die Widerspruchsfreiheit von Constraints ein Teil der Konsistenz. Die Constraints stellen dabei eine zusätzliche (semantische) Bedingung dar, die von Entwicklern spezifiziert und vom System geprüft werden.

Das Kooperationsmodell entspricht jedoch nicht den hier vorliegenden Voraussetzungen. Es wird von weitgehend disjunkten Entwurfsräumen ausgegangen. Überlappungen können nur in den Konsistenzbedingungen auftreten. Verstößt eine Entscheidung eines Entwicklers gegen die Konsistenzbedingungen eines zweiten Entwicklers, so werden beide benachrichtigt. Was dann geschieht wird nicht weiter betrachtet. Auch die Abkopplung wird nicht behandelt.

3.4.1.4 Constraint Stores

Eine interessante Weiterentwicklung der Constraint-Logikprogrammierung in verteilten und replizierten Systemen verfolgt Saraswat in seiner Dissertation ([Sar93]) mit Constraint Stores. Ein Constraint Store besteht aus eine Menge von Variablen und Constraints über diesen Variablen. In der Arbeit wird die Widerspruchsfreiheit beziehungsweise die Erfüllbarkeit aller Constraints als Konsistenzbedingung verwendet und neue Constraints dürfen nur eingefügt werden, wenn diese Konsistenzbedingung gewährleistet bleibt. Alte Constraints können nicht entfernt werden, so daß Rückschritte, wie sie im Produktentwicklungsprozeß auftreten, damit in den Constraint Stores nicht durchgeführt werden können.

Die Dissertation von Saraswat ist eine recht theoretische Arbeit, die daher auch ohne Szenario auskommt. Die Arbeit ist auf Constraints festgelegt und schließt damit andere Repräsentationsformen der Freiräume, insbesondere auch Fuzzy-Mengen, aus. Hinzu kommt noch, daß die Constraints mehrdimensional sind, also möglicherweise mehrere Variablen überspannen, und daher im Allgemeinen die Entscheidung der Widerspruchsfreiheit NP-vollständig in der Anzahl der Variablen ist.

Für die replizierte Implementierung der Constraint Stores schlägt Saraswat eine pessimistische und eine optimistische Alternative vor. Die pessimistische Version führt das Einbringen neuer Constraints prinzipiell synchron mit allen Replikaten durch und ist damit, auch nach Saraswat selbst, viel zu aufwendig und ineffizient. Die optimistische Version hingegen erlaubt das Einbringen neuer Constraints, wenn diese an einem Replikat konfliktfrei eingebracht werden können. Im Lauf der Zeit werden diese neuen Constraints dann an die anderen Replikate verteilt. Damit ist die Beständigkeit der

Constraints erst garantiert, wenn diese alle Replikate erreicht haben. Es kann also passieren, daß ein Benutzer neue Constraints einbringen darf, die möglicherweise sehr viel später doch abgelehnt werden müssen.

3.4.2 Erkenntnisse für die vorliegende Arbeit

CSCW hat gerade die Entkoppelung der Arbeitsteiligkeit zum Ziel. Die dabei unterlaufene Konsistenzsicherung wird auf den Benutzer abgewälzt, wobei diesen durch das Gruppenbewußtsein die dazu nötigen Informationen gegeben werden. Da in der vorliegenden Arbeit die Konsistenzsicherung aber nicht auf den Benutzer abgewälzt wird, sondern in Form von Widerspruchsfreiheit sichergestellt werden soll, sind die Ansätze aus dem Bereich CSCW wenig interessant. Insbesondere kann das Gruppenbewußtsein nicht in einem hier angenommenen System mit zeitweiser Abkopplung unterstützt werden.

Constraint Databases speichern zwar Constraints in einer Datenbank, allerdings nur mit dem Zweck der Beschreibung von unendlichen Tupelmengen. Damit ist eine Constraint Database eine Disjunktion von Constraints. Im Gegensatz dazu sind Entwurfszustände Konjunktionen aller Spezifikationen. Das in der Produktentwicklung auftretende Problem der Widerspruchsfreiheit ist damit in den Constraint Databases zum Beispiel nicht gegeben. Interessant sind Constraint Databases im Zusammenhang mit dieser Arbeit daher höchstens im Bereich der Anfrage, die ja auch wieder Constraints enthalten kann. Allerdings wird in praktisch allen Constraint Databases von Intervallen ausgegangen ([KLP00]), so daß eine einfache Fallunterscheidung für die Anfragebearbeitung genügt.

Die Dynamischen Regelmengen von Sturm verwalten eine explizite Produktbeschreibung und Regelmengen, die explizite Konsistenzbedingungen für die Produktbeschreibung darstellen. Damit werden Freiräume nur in den Konsistenzbedingungen ermöglicht und der verfeinernde Charakter der Produktentwicklung nicht ausreichend unterstützt. Die vollständige Ersetzung der expliziten Produktbeschreibung durch Konsistenzbedingungen ist die logische Folge, die in dieser Arbeit behandelt wird. Die Dynamischen Regelmengen erlauben temporäre Inkonsistenz, die später nur durch Rücksetzungen einzelner Spezifikationen beseitigt werden kann. Damit wird die wichtige Beständigkeit von Spezifikationen beziehungsweise Entwurfsentscheidungen verletzt.

Die ständige Informationsakquisition wird in der Arbeit zwar nicht explizit gefordert, kann aber mit dem Modell erreicht werden. Allerdings wird nur das Rücksetzen auf einen bestimmten früheren Zustand (Backtracking) unterstützt und nicht das selektive Rücksetzen einzelner Spezifikationen. Die Abkopplung beziehungsweise Replikation wird in der Arbeit von Sturm erst gar nicht betrachtet, so daß auch hier keine Erkenntnisse für die vorliegende Arbeit anfallen.

Constraint Stores kommen der gesuchten Lösung wohl am nächsten. Bei der ständigen Informationsakquisition bieten sie ein gutes Verhalten, leiden aber an der fehlenden Rücksetzungsmöglichkeit. Eine einmal eingebrachte Spezifikation kann also nie wieder entfernt werden. Aufgrund der fehlenden Rücksetzung wird die Autonomie der Entwurfsentscheidungen sicherlich nicht verletzt, allerdings um den Preis, daß Entwurfsentscheidungen nie revidiert werden können. Die Forderung nach Widerspruchsfreiheit als Konsistenzbedingung wird in den Constraint Stores erfüllt. Abstriche müssen hier allerdings gemacht werden, da die Evaluierung der Widerspruchsfreiheit in den Constraint Stores im Allgemeinen NP-vollständig ist. An dieser Stelle sind daher Einschränkungen für die Praxis nötig, um das Modell in der Produktentwicklung anwenden zu können. Constraint Stores bieten unbegrenzte Nebenläufigkeit, haben aber massive Einschränkungen bei der Abkopplung. Der Benutzer kann zwischen einem pessimistischen Verfahren wählen, welches selbst Saraswat für viel zu langsam und damit nicht praktikabel ansieht, und einem optimistischen Verfahren, in dem aber die Beständigkeit aufgegeben wird.

3.5 Zusammenfassung

Verfeinerungen von impliziten Lösungsbeschreibungen durch Constraints gibt es bei Saraswat (Constraint Store) und Sturm (Dynamische Regelmengen). Die Widerspruchsfreiheit an sich wird auch in den Constraintsystemen oft diskutiert. Von dort können Erkenntnisse über die Entscheidbarkeit der Widerspruchsfreiheit bei verschiedenen Constraintsystemen übernommen werden. Unzureichend sind die Ansätze aber in der Einschränkung auf Constraints. In der vorliegenden Arbeit werden die konkreten Spezifikationsformen möglichst offen gelassen und können auch durch andere Konzepte ausgefüllt werden.

Erweiterte Datenmodelle ermöglichen die Speicherung vieler verschiedener Spezifikationsformen und können einfach auf die monotone Informationsakquisition erweitert werden. Auf der anderen Seite gibt es für das selektive Rücksetzen einzelner Entwurfsentscheidungen keine Ansätze, die für die vorliegende Arbeit relevant sind, da praktisch alle existierenden Ansätze von der Ersetzung alter Informationen durch neue ausgehen. Gerade weil neue Spezifikationen die bekannten ergänzen und nicht ersetzen, stehen hier neue Möglichkeiten offen. Einzig das Revisionsmodell für Fuzzy-Mengen ([Wit02b]) bietet einen Ansatz zur Verfeinerung einer Lösungsbeschreibung. Dabei wird allerdings die Beständigkeit durch automatisches Entfernen von schon eingebrachten Informationen verletzt. Für Fuzzy-Mengen als Spezifikationsform kann dieses Revisionsmodell allerdings einige nützliche Erkenntnisse bringen.

Viele Arbeiten zur Abkopplung und Replikation bieten gute Ansätze, betrachten allerdings nicht die Durchsetzung der Widerspruchsfreiheit in einem replizierten System. Einzig Constraint Stores bieten hier eine Lösung, die jedoch im betrachteten Szenario nicht akzeptabel ist. Das Demarkationsprotokoll kann hier als Ansatz und Idee für eine neue Lösung benutzt werden, allerdings sind hier noch erhebliche Erweiterungen und Ausarbeitungen in Richtung Widerspruchsfreier Freiräume nötig.

Kapitel 4

Handlungsbedarf

Aus dem Stand der Forschung kann gefolgert werden, daß bisher keine Arbeiten existieren, die das vorliegende Problem behandeln und deren Lösungen alle Forderungen erfüllen. Es liegt also in diesem Bereich noch Handlungsbedarf vor, der von der vorliegenden Arbeit geschlossen werden soll. Die wesentlichen drei Kriterien für eine gute Rechnerunterstützung für den arbeitsteiligen Entwicklungsprozeß sind die Durchsetzung der *Datenkonsistenz* bei Erhaltung der vollen Nebenläufigkeit, die *ständige Informationsakquisition mit selektivem Rücksetzen* bei gleichzeitiger Erhaltung der *Autonomie der Entwurfsentscheidungen* und die Unterstützung des *abgekoppelten Arbeitens* auf mehreren *Replikaten*.

Da die Entscheidbarkeit der Widerspruchsfreiheit von Spezifikationen in der vollen Allgemeinheit nicht garantiert ist, müssen hier gewisse Einschränkungen gemacht werden. Der Formalismus hinter den Spezifikationen muß so weit beschränkt werden, daß die Widerspruchsfreiheit sehr einfach zu entscheiden ist. Hierzu werden im direkt nachfolgenden Kapitel 5 Intervalle als einfache aber für die Produktentwicklung ausreichende Spezifikationsformen vorgestellt, auf denen sich dann im Wesentlichen der Rest der Arbeit beschränkt.

Bei der nachfolgenden Entwicklung der Datenbankunterstützung für den Produktentwicklungsprozeß wird analog zu der Anforderungsanalyse in Schritten vorgegangen:

Kapitel 6: Das Revisionsmodell für die Produktentwicklung
Der erste Teil widmet sich den Eigenheiten der konvergierenden Produktbeschreibung, wie sie in Abschnitt 2.3 (insbesondere in 2.3.2) analysiert wurden. Um nicht durch andere Probleme abgelenkt zu werden, wird dabei von einem zentralen System und einem einzelnen Benutzer ausgegangen, so daß die Probleme nur im Bereich der Verwaltung und Bearbeitung der evolvierende Produktbeschreibung liegen.

Kapitel 7: Widerspruchsfreiheit unter Arbeitsteiligkeit
Die aus der Arbeitsteiligkeit resultierenden erweiterten Anforderungen aus Abschnitt 2.4 werden im zweiten Teil betrachtet und das zuvor entwickelte Revisionsmodell für die Produktentwicklung entsprechend erweitert. Die Erweiterungen beziehen sich in erster Linie auf die Gruppierung von Spezifikationen zu Entwurfsentscheidungen und eine Zuständigkeitsverwaltung, so daß die Beständigkeit überprüft und durchgesetzt werden kann.

Kapitel 8: Erweitertes Demarkationsprotokoll für die Abkopplung
Die in größeren Entwicklungsprozessen nötige Abkopplung ist Thema des dritten Teils, der vor allem auf Abschnitt 2.5 der Anforderungsanalyse aufbaut. Auf der Idee des Demarkationsprotokoll aufbauend werden Mechanismen entwickelt, mit denen vor der Abkopplung Vereinbarungen getroffen werden, nach denen auf den abgekoppelten Replikaten unabhängige Entwurfsentscheidungen eingebracht werden können, die später garantiert widerspruchsfrei zusammengeführt werden können.

Mit den Kapiteln 6 bis 8 ist das grundlegende Modell für eine Datenbasis zur Unterstützung der arbeitsteiligen und entkoppelten Produktentwicklung vorgestellt. Die nachfolgenden Kapitel befassen sich mit den Konsequenzen dieses Modells, also zum Beispiel möglichen Erweiterungen, alternativen Spezifikationsformen und natürlich auch der Evaluierung.

Kapitel 9: Erweiterungen der Widerspruchsfreien Freiräume
In Kapitel 9 sollen mögliche Erweiterungen für das Revisionsmodell mit dem erweiterten Demarkationsprotokoll aus Kapitel 8 vorgestellt werden. So ist zum Beispiel das unbedingte Festhalten an der garantierten Beständigkeit bei der Abkopplung nicht unbedingt immer das optimale Verhalten. Es kann durchaus vorkommen, daß ein Entwickler seine Arbeit lieber optimistisch, also ohne garantierte Beständigkeit, einbringt, als gar nichts zu tun, wenn vom System momentan keine Beständigkeit garantiert werden kann. Eine andere mögliche Erweiterung ist die Verwaltung von Inkonsistenzen im Modell. Hier muß untersucht werden, ob inkonsistente, also widersprüchliche Spezifikationen im System gespeichert werden können. Falls ja, muß geklärt werden, wie lange die Inkonsistenz im System bleiben darf und wenn sie entfernt werden muß, welche Mechanismen hierfür notwendig sind.

Kapitel 10: Abstraktes Modell
Die Motivation des Revisionsmodells für die Produktentwicklung entsteht aus der ständigen Verfeinerung der Produktbeschreibung durch Hinzufügen neuer Spezifikationen. Die Beschreibung des Produkts ist also eine Konjunktion von Spezifikationen. Auf der anderen Seite kann durchaus in Teilbereichen eine Disjunktion von Spezifikationen interessant sein, zum Beispiel bei mengenwertigen Produkteigenschaften (etwa verschiedene Teile eines Zulieferers oder Motorvarianten des zu konstruierende Robotergreifers).

In diesem Kapitel sollen die Spezifikationen als abstrakte Datentypen aufgefaßt werden, wobei jede Spezifikation eine Operation anbietet, die diese Spezifikation mit einer anderen kombiniert. Da die Implementierung dieser Kombinationsoperationen spezifikationsspezifisch ist und damit dem Spezifikationsprogrammierer überlassen bleibt, ist das Modell daher nicht mehr auf Konjunktionen festgelegt, sondern kann auch auf Disjunktionen oder andere Operatoren erweitert werden. Das Ziel des Kapitels ist die Spezifikation des abstrakten Datentyps und die Formalisierung des Modells aus Kapitel 8 auf der Basis dieses abstrakten Datentyps.

Kapitel 11: Spezifikationen
Das Revisionsmodell für die Produktentwicklung wurde auf der Basis von Intervallen als Spezifikationen entwickelt. Das allgemeine Konzept der Spezifikationen aus der Anforderungsanalyse und Kapitel 6 kann aber auch durch andere konkrete Formen ausgefüllt werden. In diesem Kapitel geht es darum, einige dieser Formen zu diskutieren, deren Vor- und Nachteile zu beleuchten und Besonderheiten bei der Anwendung für das entwickelte Modell vorzustellen.

Kapitel 12: Realisierung
Das Ziel von Kapitel 12 ist die Vorstellung eines im Rahmen der Arbeit entwickelten Demonstrators. Dabei sind Architekturfragen ebenso zu beleuchten, wie das

Schema und eine anschauliche Darstellung der Funktionsweise des Revisionsmodells für die Produktentwicklung.

Kapitel 13: Evaluierung

Für die Evaluierung des Modells wird der Demonstrator aus dem vorangegangenen Kapitel herangezogen. Die Evaluierung selbst kann hier nicht im Vergleich mit bestehenden Systemen geführt werden, da ein Vergleich nur im Kontext eines kompletten Entwurfsprozesses Sinn macht und die Simulation geschweige denn Durchführung eines solchen Prozesses im Rahmen dieser Arbeit nicht möglich ist. Damit kann nur das Laufzeitverhalten des Demonstrators mit derzeit gängigen Produkten im Hinblick auf die Antwortzeiten im interaktiven Betrieb untersucht werden.

Kapitel 5

Einfache und effiziente Spezifikationsform

Die zentrale Konsistenzbedingung im Produktentwicklungsprozeß ist die Widerspruchsfreiheit von Spezifikationen. Um diese effizient entscheiden zu können, sind Einschränkungen in den zugelassenen Spezifikationsformen nötig. Hierzu gilt es zuerst die Anforderungen an die Spezifikationen zu beschreiben, also die Berechnungen, die mit den Spezifikationen durchgeführt werden müssen. Auf der Basis dieser Anforderungen können dann einfache Spezifikationsformen diskutiert und auf deren Eignung für die vorliegende Arbeit hin untersucht werden.

5.1 Eigenschaften von Spezifikationen

Definition 5.1.1 (Dimension einer Spezifikation)
Die **Dimension** *einer Spezifikation $\mathcal{S}()$ ist die Anzahl der in $\mathcal{S}()$ benutzten Produkteigenschaften.*
□

Eine wichtige Eigenschaft von Spezifikationen ist deren Dimension, also die Anzahl der in der Menge benutzten Produkteigenschaften. Ist diese Anzahl größer als eins, so können Produkteigenschaften nicht unabhängig betrachtet werden, da es möglicherweise zwischen allen Paaren von Produkteigenschaften Zusammenhänge gibt. Die Entscheidbarkeit der Widerspruchsfreiheit wird in diesem Fall also mindestens linear in der Anzahl der Produkteigenschaften. Je nach konkretem System von Spezifikationen wird die Entscheidbarkeit sogar sehr schnell NP-vollständig in der Anzahl der Spezifikationen oder ist gar nicht mehr entscheidbar. Diese Arbeit beschränkt sich daher auf *eindimensionale Spezifikationen*, also solche, die nur eine Produkteigenschaft beschreiben. Damit kann jede Produkteigenschaft separat betrachtet werden und der Aufwand für die Berechnungen ist nur noch eine Funktion der Anzahl der Spezifikationen pro Produkteigenschaft. Durch diese Eindimensionalität können in der weiteren Arbeit Produkteigenschaften und die Spezifikationen auf diesen isoliert und unabhängig von anderen Produkteigenschaften betrachtet werden. Die Widerspruchsfreiheit aller Spezifikationen ergibt sich damit automatisch aus der Widerspruchsfreiheit der Spezifikationen an den einzelnen Produkteigenschaften.

Die Eindimensionalität von Spezifikationen bezieht sich auf die Anzahl der in der Spezifikation benutzten Produkteigenschaften. Dennoch können die Produkteigenschaften

selbst einen mehrdimensionalen Charakter haben. So sind gerade Entwurfsdaten oftmals geometrische Positionen im dreidimensionalen Raum. Das heißt, einer Produkteigenschaft liegt dann eine dreidimensionale Domäne zugrunde. Trotzdem ist die Spezifikation, die nur diese eine Produkteigenschaft betrachtet eindimensional.

Als Abkürzung gilt im Folgenden $\mathcal{S}(x)$ als wahr, wenn die Spezifikation $\mathcal{S}()$ dem Wert x den Wert *wahr* zuweist.

Definition 5.1.2 (Erfüllbarkeit)
Die Spezifikation $\mathcal{S}()$ über der Domäne $\mathbb{D}$ ist **erfüllbar**, *geschrieben als $Sat(\mathcal{S}())$, wenn mindestens ein x aus $\mathbb{D}$ existiert, so daß $\mathcal{S}(x)$ gilt.*

$$Sat(\mathcal{S}()) \Leftrightarrow \exists_{x \in \mathbb{D}}\ \mathcal{S}(x)$$

□

Oft wird nicht nur nach der Erfüllbarkeit einer Spezifikation, sondern auch nach der Teilmenge der Domäne gefragt, für die die Spezifikation den Wert *wahr* ergibt.

Definition 5.1.3 (Erfüllungsmenge)
Zu einer gegebenen Spezifikation $\mathcal{S}()$ über der Domäne $\mathbb{D}$ definiert sich die **Erfüllungsmenge** *$\mathbb{P}_{\mathcal{S}()}$ als die Teilmenge der Domäne, für die die Spezifikation $\mathcal{S}()$ den Wert wahr liefert:*

$$\mathbb{P}_{\mathcal{S}()} = \{x \in \mathbb{D} \mid \mathcal{S}(x)\}$$

Die Erfüllungsmenge ist somit der Freiraum einer eindimensionalen Spezifikation. □

Die Definition der Spezifikationen ist immer noch sehr abstrakt und muß hier auch noch nicht weiter ausgefüllt werden. Konkrete Formalismen bieten *Spezifikationsfamilien* über einer Domäne an, wobei eine Spezifikationsfamilie über der Domäne $\mathbb{D}$ eine Menge von Spezifikationen über $\mathbb{D}$ ist. Eine wichtige Eigenschaft dieser Spezifikationsfamilien ist die Abgeschlossenheit bezüglich Konjunktion.

Definition 5.1.4 (Abgeschlossenheit von Spezifikationsfamilien)
Sei $\mathbb{S}$ eine Menge von Spezifikationen über der Domäne $\mathbb{D}$. $\mathbb{S}$ heißt **abgeschlossen** *bezüglich der Konjunktion, wenn die Konjunktion von beliebigen Spezifikationspaaren aus $\mathbb{S}$ selbst wieder eine Spezifikation aus $\mathbb{S}$ ist.*

$$\mathbb{S} \textit{ ist abgeschlossen} \Leftrightarrow \forall_{\mathcal{S}_1(),\mathcal{S}_2() \in \mathbb{S}}\ \exists_{\mathcal{S}() \in \mathbb{S}}\ \mathcal{S}() = \mathcal{S}_1() \wedge \mathcal{S}_2()$$

□

Leider kann nicht für alle sinnvollen Spezifikationsformen die Abgeschlossenheit vorausgesetzt werden. So sind zum Beispiel Kugeln über mehrdimensionalen Domänen nicht abgeschlossen.

Beispiel 5.1.1 (Keine Abgeschlossenheit von Kugeln)
Betrachtet man zum Beispiel Zahlenpaare (x, y) mit $x, y \in \mathbb{R}$ und dem üblichen euklidischen Abstandsmaß, dann stellen Zahlenpaare in Verbindung mit einem ϵ eine Kugel $\overline{B_\epsilon(x, y)}$, beziehungsweise im zweidimensionalen Raum einen Kreis dar[1]*. Sei $\mathbb{B}$ die Menge aller Kreise um einen Punkt im zweidimensionalen Raum ($\mathbb{B} = \{\overline{B_\epsilon(x, y)} | \epsilon, x, y \in \mathbb{R} \wedge \epsilon \geq 0\}$). Wenn die Schnittmenge von zwei Kreisen eine Linse ist, dann ist diese nicht mehr Element von $\mathbb{B}$ (siehe zum Beispiel Abbildung 5.1).*

$$\nexists_{\mathcal{S}() \in \mathbb{B}}\ \mathcal{S}() = \overline{B_2(1, 0)} \wedge \overline{B_2(0, 1)}$$

□

[1] Die Zahlenpaare stellen also eine zweidimensionale Domäne dar. Trotzdem können die Spezifikationen eindimensional bleiben, in dem sie nur ein Datenelement spezifizieren und keine Zusammenhänge zwischen den Datenelementen.

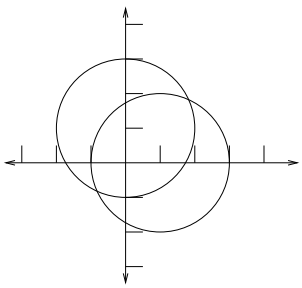


Abbildung 5.1: Die Schnittmenge von $\overline{B_2(1,0)}$ und $\overline{B_2(0,1)}$

5.2 Anforderungen an die Spezifikationen

Aus der Anforderungsanalyse geht in erster Linie die Entscheidbarkeit der Widerspruchsfreiheit von Spezifikationen als Anforderung hervor. Auf der anderen Seite ist eine Produktbeschreibung genau die Konjunktion der eingebrachten Spezifikationen, so daß die explizite Verwaltung dieser Konjunktion wünschenswert ist. So wollen zum Beispiel die Entwickler zu einem gegebenen Zeitpunkt die noch akzeptablen Ausprägungen einer Produkteigenschaft erfahren. Diese können am besten mit einer Spezifikation beschrieben werden, die eben genau die Konjunktion aller einzelnen Spezifikationen auf dieser Produkteigenschaft darstellt. Geht man also davon aus, daß zu jedem Zeitpunkt für jede Produkteigenschaft die Konjunktion aller einzelnen Spezifikationen verwaltet wird, dann gibt es vier Anforderungen an die Spezifikationen:

- Die Berechnung der Konjunktion von Spezifikationen. Hierzu wird für die Entwicklung des Modells von einer abgeschlossenen Spezifikationsfamilie ausgegangen,das heißt, zu jeder Menge von Spezifikationen gibt es eine Spezifikation, die genau die Konjunktion der Menge von Spezifikationen beschreibt.
- Mit der Verwaltung der Konjunktion von Spezifikationen reduziert sich die Entscheidbarkeit der Widerspruchsfreiheit auf die Entscheidung der Erfüllbarkeit einer einzelnen Spezifikation.
- Für die in Abschnitt 3.3.2 angedeutete Lösung des Abkopplungsproblems über das Demarkationsprotokoll muß entschieden werden können, ob eine Spezifikation eine andere impliziert.
- Die Berechnung einer minimalen Konfliktmenge.

Definition 5.2.1 (Konfliktmenge)
Sei $\mathcal{S}()$ eine erfüllbare Spezifikation $\mathbb{S}$ eine Menge von Spezifikationen, deren Konjunktion erfüllbar ist. Eine **Konfliktmenge** *$\mathbb{C}$ für das Paar $(\mathbb{S}, \mathcal{S}())$, geschrieben als $\mathbb{C}(\mathbb{S}, \mathcal{S}())$, ist eine Teilmenge von $\mathbb{S}$, nach deren Entfernung aus $\mathbb{S}$ die Konjunktion von $\mathbb{S}$ und $\mathcal{S}()$ erfüllbar ist.*

$$\mathbb{C} \text{ ist Konfliktmenge für } (\mathbb{S}, \mathcal{S}()) \Leftrightarrow$$

$$\mathbb{P}_{\mathcal{S}()} \cap \mathbb{P}_{\mathbb{S}\setminus\mathbb{C}} \neq \emptyset$$

Eine Konfliktmenge $\mathbb{C}(\mathbb{S}, \mathcal{S}())$ ist eine **minimale Konfliktmenge** *$\mathbb{C}_{min}(\mathbb{S}, \mathcal{S}())$, wenn es keine andere Konfliktmenge mit einer kleineren Mächtigkeit gibt.*

$$\mathbb{C}(\mathbb{S}, \mathcal{S}()) \text{ ist minimale Konfliktmenge für } (\mathbb{S}, \mathcal{S}()) \Leftrightarrow$$

$$\forall_{\mathbb{C}'}\ \mathbb{C}' \text{ ist Konfliktmenge für } (\mathbb{S}, \mathcal{S}()) \Rightarrow |\mathbb{C}(\mathbb{S}, \mathcal{S}())| \leq |\mathbb{C}'|)$$

□

Die Berechnung dieser minimalen Konfliktmenge ist nicht eindeutig. Betrachtet man zum Beispiel die folgenden drei Constraints, so sind diese zwar paarweise kompatibel, aber alle drei zusammen sind nicht kompatibel.

$$C_1 : x = 3 \vee x = 5 \qquad C_2 : x = 3 \vee x = 4 \qquad C_3 : x = 4 \vee x = 5$$

Sind die Constraints C_1 und C_2 von anderen Benutzern aufgeprägt worden und C_3 soll nun neu aufgeprägt werden, so genügt das Entfernen von C_1 oder C_2. Es gibt damit also zwei minimale Konfliktmengen, nämlich $\{C_1\}$ und $\{C_2\}$.

5.3 Einfache Spezifikationsformen

Beschreibungsformen für eindimensionale Spezifikationen sind einfache Beschreibungen einer Teilmenge einer Domäne, nämlich genau der Domäne, die der von der Spezifikation benutzten Produkteigenschaft zugeordnet ist. Unterschiedliche Formalismen eignen sich dabei mehr oder weniger gut für die Repräsentation dieser Teilmengen. Einige verschieden mächtige Möglichkeiten sollen hier kurz am Beispiel der Domäne der ganzen Zahlen dargestellt werden. Dabei bezeichnet $\mathbb{P}$ immer die durch die Spezifikation beschriebene Teilmenge der Domäne.

Scharfe Werte

Scharfe Werte sind genau die bekannte und übliche Beschreibungsweise für einelementige Teilmengen der Domänen.

$$x = 5 \quad \Rightarrow \quad \mathbb{P} = \{5\}$$

Null-Werte

Spezielle Null-Werte bezeichnen die leere Teilmenge einer Domäne und drücken damit auf sehr einfache Art eine Unvollkommenheit aus. Allerdings gibt es hier eben nur die Möglichkeit eines scharfen Wertes oder der vollkommenen Unwissenheit, die je nach Definition alle oder kein Element zulassen kann.

$$x = \text{Null} \quad \Rightarrow \quad \begin{cases} \mathbb{P} = \emptyset \\ \mathbb{P} = \mathbb{U}(\mathit{niversum}) \end{cases} \quad \textit{je nach Interpretation}$$

Kugeln

Kugeln beschreiben zusammenhängende konvexe Teilmengen der Domäne über einen Mittelpunkt und einen Radius. Zu der beschriebenen Teilmenge gehören alle Werte, deren Abstand vom Mittelpunkt kleiner oder gleich dem Radius ist. Für diese Art der Beschreibung ist allerdings ein Abstandsmaß auf den Domänen erforderlich.

$$x = 5, \epsilon = 2 \quad \Rightarrow \quad \mathbb{P} = \{3, 4, 5, 6, 7\}$$

Constraints

Allgemein gesagt sind Constraints Aussagen über einer Menge von Variablen. Je nach Besetzung der Variablen kann die Aussage wahr oder falsch werden. Im Rahmen dieser Arbeit sind insbesondere Aussagen über eine einzelne Variable von Interesse. Diese Variable stellt dann genau eine Produkteigenschaft dar, so daß mit einem Constraint genau eine Menge von akzeptablen Ausprägungen dieser Produkteigenschaft beschrieben werden kann.

$$x > 3 \wedge x < 8 \quad \Rightarrow \quad \mathbb{P} = \{4, 5, 6, 7\}$$

Explizite Aufzählung

Natürlich kann die Teilmenge einer Domäne auch über eine explizite Aufzählung beschrieben werden. Allerdings sind solche Aufzählungen insbesondere bei großen Mengen nicht mehr praktikabel.

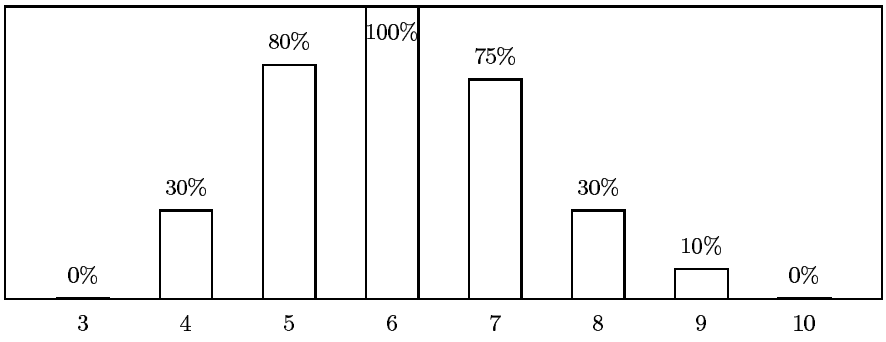
$$x = 3 \vee x = 5 \vee x = 42 \quad \Rightarrow \quad \mathbb{P} = \{3, 5, 42\}$$

Fuzzy-Mengen

Fuzzy-Mengen sind Zuordnungen von einer Basismenge Ω in das Einheitsintervall $[0, 1]$.

$$\mu : \Omega \mapsto [0, 1].$$

Der einem Element der Basismenge zugeordnete Wert spezifiziert dabei die Zugehörigkeit des Elements zur Fuzzy-Menge. Eine Fuzzy-Menge über einer Domäne beschreibt somit für jedes Element den Zugehörigkeitsgrad zu der zu beschreibenden Teilmenge. Dabei ist der unscharfe Zugehörigkeitsgrad kein Problem, sondern vielmehr eine Erweiterung der Teilmengenzugehörigkeit um Unvollkommenheit. An dieser Stelle ist leicht zu sehen, daß alle anderen Beschreibungen durch eine Fuzzy-Menge ersetzt werden können, wobei alle Zugehörigkeitsgrade genau 0 oder 1 sind. Fuzzy-Mengen sind damit die allgemeinste der hier aufgelisteten Beschreibungen.



$$\Rightarrow \quad \mathbb{P} = ?$$

Oftmals muß aufgrund von Fuzzy-Mengen eine binäre Entscheidung getroffen werden. In diesem Fall ist natürlich die gegebene Fuzzy-Menge in eine scharfe Menge zu *defuzzyfizieren*. Dies geschieht üblicherweise über einen Schwellenwert α:

$$\mathbb{P} = \{x : \mu(x) \geq \alpha\}$$

Wenn eine Fuzzy-Menge als Spezifikation verwendet werden soll, dann geht dies nur in Verbindung mit dem zugehörigen Schwellenwert α, da nur durch α festgelegt werden kann, welchen Elementen der Basismenge die entsprechende Spezifikation *wahr* oder $falsch$ zuordnet.

Im Sinn einer einfachen Benutzbarkeit darf eine Rechnerunterstützung für den Produktentwicklungsprozeß keine komplizierten Spezifikationsformen verwenden. Am besten geeignet ist zum Beispiel die Verwendung oder Erweiterung schon bekannter Möglichkeiten. Dies kann zum Beispiel die Verwendung von scharfen Werte sein, wie sie die Entwickler gewohnt sind. Um dann die Freiräume zu schaffen werden diese scharfen Werte x einfach mit maximalen Abweichungen ϵ versehen. Im einfachsten Fall führt dies zu Intervallen $[x - \epsilon, x + \epsilon]$. Betrachtet man zum Beispiel eine vorgegebene Haltekraft für den Robotergreifer von 100 N mit einer maximalen Abweichung von 5 N, so ergibt sich als Spezifikation das Intervall $[95, 105]$ N.

Leider sind die Domänen, mit denen Entwickler umgehen, nicht immer die rationalen Zahlen, sondern oft auch records mit mehreren Feldern, wie dies zum Beispiel bei Geometriedaten der Fall ist. Dabei gibt es natürlich trotzdem die Möglichkeit, im dreidimensionalen Raum einen Punkt mit einer maximalen Abweichung zu spezifizieren. Das Problem dabei ist, daß dadurch eine Kugel um diesen einen Punkt entsteht und der Umgang mit solchen Kugeln nicht gerade einfach ist.

Eine einfachere Version als Kugeln sind Quader, in denen einfach für jede einzelne Dimension eine separate maximale Abweichung angenommen wird. Der Entwickler muß dann also für einen Punkt im dreidimensionalen Raum drei maximale Abweichungen angeben. Die notwendigen Berechnungen auf Quadern sind dann im Gegensatz zu Kugeln kein Problem mehr.

Wenn für die Unterstützung des Produktentwicklungsprozesses Fuzzy-Mengen verwendet werden, dann ist es sehr interessant, diese Fuzzy-Mengen auch als Spezifikationsformen anzusehen. Leider gibt es viele unterschiedliche Repräsentationen von Fuzzy-Mengen mit verschiedenen Eigenschaften (zum Beispiel endlich, unendlich, diskret oder kontinuierlich), so daß eine allgemeine Diskussion von Fuzzy-Mengen hier nicht sinnvoll erscheint.

Da abgesehen von den geometrischen Daten die Mehrzahl der Produkteigenschaften eindimensionale Domänen haben, beschränkt sich die vorliegende Arbeit auf Intervalle. Andere Spezifikationsformen fügen im Kern keinen Neuigkeitswert hinzu, können aber in der Praxis leicht eingesetzt werden. Einzig die oben genannten Anforderungen an die Spezifikationsformen müssen erfüllt werden und es gilt zu beachten, daß die Effizienz des zu entwickelnden Modells direkt von der Spezifikationsform abhängt.

5.4 Intervalle

Intervalle als ausgewählte Spezifikationsform sind nun noch auf die Erfüllung der oben genannten Anforderungen zu untersuchen. Dabei ist auch die Effizienz der notwendigen Berechnungen zu diskutieren, da diese grundlegend die gesuchte Rechnerunterstützung des Produktentwicklungsprozesses beeinflußt. Nebenbei kann hier noch bemerkt werden, daß die Speicherung von Intervallen mit konstantem Platzbedarf möglich ist, da nur die Unter- und Obergrenzen, also zwei Zahlen, zu verwalten sind.

- Die Konjunktion einer Menge von Intervallen ist abgeschlossen, kommutativ und assoziativ und kann sehr einfach mit linearem Aufwand berechnet werden.

  Die Konjunktion der Intervalle $[x_1, y_1]$ bis $[x_n, y_n]$ ist das Intervall $[MAX(x_1, ...x_n), MIN(y_1, ...y_n)]$. Die MAX- und MIN-Berechnungen können durch $n-1$ Vergleiche erledigt werden und sind somit linear in der Anzahl der Intervalle.

- Die Erfüllbarkeit eines Intervalls $[x, y]$ ist gegeben, wenn gilt: $x \leq y$

- Das Intervall $[x_1, y_1]$ impliziert $[x_2, y_2]$, wenn gilt: $x_1 \geq x_2 \wedge y_1 \leq y_2$

- Die minimale Konfliktmenge $\mathbb{C}$ für $(\mathbb{S}, \mathcal{S}())$ kann berechnet werden, in dem jedes einzelne Intervall aus $\mathbb{S}$, das mit $\mathcal{S}()$ eine leere Schnittmenge hat, in $\mathbb{C}$ aufgenommen wird. Diese Berechnung ist mit linearem Zeitaufwand möglich und liefert eine eindeutige minimale Konfliktmenge.

Intervalle bieten zusätzlich die Möglichkeit, einfache Ungleichungen auszudrücken. So kann die Ungleichung $x \geq 5$ beispielsweise durch das Intervall $[5, \perp]$ dargestellt werden. Da nur eine Untergrenze definiert ist, wird die Obergrenze einfach ignoriert oder als unendlich angenommen.

Kapitel 6

Das Revisionsmodell für die Produktentwicklung

Das Ziel dieses Kapitels ist die Entwicklung eines Modells für die Rechnerunterstützung des Produktentwicklungsprozesses für einen einzelnen Entwickler. Damit gibt es in diesem Kapitel keine Interaktionen zwischen verschiedenen Entwicklern. Zusätzlich kann davon ausgegangen werden, daß der einzelne Entwickler einen Überblick über seine Arbeit hat. Er weiß also, welche Spezifikation aus welchem Grund eingebracht wurden. Ein einzelner Entwickler arbeitet immer nur an einer physikalischen Stelle, so daß hier auch von einem zentralen System ausgegangen werden kann.

Die Entwicklung des Modells der Widerspruchsfreien Freiräume ist in vier Schritte eingeteilt. Im ersten Schritt wird eine Datenbasis mit Zugriffsoperationen entwickelt, so daß die Spezifikationen dort verwaltet werden können. Im zweiten Schritt wird die Widerspruchsfreiheit als Konsistenzbedingung für den zuvor vorgestellten Datenspeicher definiert. Das Modell wird durch die Vorstellung des Protokolls und dem Beweis, daß das Protokoll die geforderte Datenkonsistenz garantiert, komplettiert. Das Kapitel wird durch eine Betrachtung des Revisionsmodells für die Produktentwicklung in der Praxis abgeschlossen.

6.1 Datenbasis

Da in der Arbeit nur eindimensionale Spezifikationen betrachtet werden, wird der Informationsspeicher in Datenelemente eingeteilt, wobei jede Produkteigenschaft genau einem Datenelement zugeordnet wird.

Definition 6.1.1 (Datenelement)
Ein **Datenelement** X *ist eine Menge von Spezifikationen. Alle Spezifikationen in einem Datenelement müssen auf derselben, dem Datenelement statisch zugeordneten, Domäne definiert sein. Diese Domäne wird als* $\mathbb{D}_X$ *bezeichnet.* □

Ein Datenelement X beschreibt damit eine Teilmenge der ihm zugeordneten Domäne, nämlich genau alle Elemente, die alle Spezifikationen in dem Datenelement erfüllen, also die Schnittmenge der Erfüllungsmenge aller Spezifikationen in X. Die Menge von Spezifikationen ist bei neu angelegten Datenelementen initial leer, so daß die beschriebene Teilmenge der Domäne genau die Domäne selbst ist. Im Lauf der Zeit können

theoretisch beliebig viele Spezifikationen eingebracht werden, die dann die Teilmenge der Domäne einschränken können. Da gerade die von allen Spezifikationen eines Datenelements gemeinsam beschriebene Teilmenge den verbleibenden Freiraum auf der Produkteigenschaft darstellt, soll diese Konjunktion als Erfüllungsmenge des Datenelements bezeichnet werden.

Definition 6.1.2 (Erfüllungsmenge eines Datenelements)
Die **Erfüllungsmenge** $\mathbb{P}_X$ *des Datenelements* X *ist genau die Erfüllungsmenge der Konjunktion aller Spezifikationen in* X*.*

$$\mathbb{P}_X = \{x \in \mathbb{D}_X \mid \forall_{\mathcal{S}() \in X}\, \mathcal{S}(x)\}$$

□

Auf der Basis von Datenelementen kann eine *Datenbasis* wie üblich einfach als Menge von Datenelementen aufgefaßt werden. Allerdings gibt es nicht einen einzelnen Datenbasiszustand im Sinn eines einzelnen scharfen Wertes für jedes Datenelement, da ein Datenelement zu einem bestimmten Zeitpunkt nicht mehr einen eindeutigen Wert hat, sondern eine Wertemenge beschreibt. Vielmehr gibt es eine möglicherweise leere Menge von Zuständen, die sich aus dem kartesischen Produkt der Erfüllungsmengen der Datenelemente errechnet. Jeder Zustand aus dieser Menge beschreibt damit einen Punkt im Entwurfsraum, der alle Spezifikationen in der Datenbasis erfüllt.

Nach der derzeitigen Definition kann das kartesische Produkt der Erfüllungsmengen der Datenelemente durchaus leer sein. Damit ist die Datenbank allerdings inkonsistent (es gibt keinen Punkt im Entwurfsraum, der alle Spezifikationen in der Datenbasis erfüllt), was aber vermieden werden sollte. Durch die Definition der Operationen und des Protokolls wird dieser „inkonsistente“ Zustand gerade verhindert.

6.2 Operationen

Der Zugriff des Entwicklers auf die Datenbasis wird über die Schnittstelle der Datenbasis abgewickelt, die folgende drei Operationen zur Verfügung stellt. Die erste Operation bringt eine neue Spezifikation in ein Datenelement ein. Dabei ist zu beachten, daß neue Spezifikationen nur eingebracht werden dürfen, wenn die Erfüllungsmenge des entsprechenden Datenelements nicht leer wird. Die zweite Operation „liest“ die aktuell verbleibenden Freiräume aus der Datenbasis aus. Die dritte Operation entfernt fälschlicherweise eingebrachte Spezifikationen wieder aus der Datenbasis, revidiert also die Produktbeschreibung.

constrain **($\mathcal{S}()$, X)**
Die Operation *constrain* bringt die Spezifikation $\mathcal{S}()$ in das Datenelement X ein. Die neue Spezifikation muß dafür mit allen auf dem Datenelement bestehenden Spezifikationen kompatibel sein.

$$\mathbb{P}_{\mathcal{S}()} \cap \mathbb{P}_X \neq \emptyset$$

Ist dies der Fall, so wird sie eingebracht, ansonsten wird die minimale Konfliktmenge zurückgegeben.

```
Wenn der aktuelle Peek nicht gespeichert ist
  Berechne den aktuellen Peek
End
Berechne die Konjunktion aus Peek und neuer Spezifikation
Wenn diese Konjunktion erfüllbar ist
  Bringe die neue Spezifikation in X ein
  Setze den aktuellen Peek auf die Konjunktion
```

```
Sonst
  Berechne eine minimale Konfliktmenge
  Gebe die minimale Konfliktmenge zurück
  FEHLER: Widerspruch
End
FERTIG
```

peek **(X)**
Die Operation *peek* (X) gibt die Konjunktion aller Spezifikationen auf X zurück.

```
Wenn der aktuelle Peek nicht gespeichert ist
  Berechne den aktuellen Peek
End
Gebe den aktuellen Peek zurück
FERTIG
```

release **(X, $\mathcal{S}()$)**
Die Operation *release* (X, $\mathcal{S}()$) löscht die zuvor durch *constrain* eingebrachte Spezifikation $\mathcal{S}()$ von dem Datenelement X. Die Operation *release* setzt damit genau diese *constrain*-Operation zurück, als ob diese nie ausgeführt worden wäre.

```
Wenn die Spezifikation in X enthalten ist
  Entferne die Spezifikation aus X
  Invalidiere den gespeicherten aktuellen Peek
  FERTIG
End
FEHLER: Spezifikation nicht vorhanden
```

Mit den Operationen werden also nicht alte Werte und damit auch alte Zustände durch neue ersetzt. Vielmehr wird die gespeicherte Produktbeschreibung durch neue Spezifikationen verfeinert und durch die nachträgliche Rücksetzung dieser Verfeinerungen auch revidiert. Das Modell der entwickelten spezifikationsbasierten Datenbasis mit den Operationen und der noch zu definierenden Konsistenz und dem Durchsetzungsprotokoll ist damit ein *Revisionsmodell* oder *RV-Modell.*

6.3 Konsistenz der Datenbasis

Schon in den Forderungen aus Kapitel 2 wird die Widerspruchsfreiheit von Spezifikationen als Konsistenzbedingung genannt. Hier gilt es nur noch, diese Konsistenzbedingung auf die Datenbasis und damit auf die Widerspruchsfreiheit von Datenelementen zu übertragen.

Definition 6.3.1 (Konsistenz der Datenbasis)
Eine Datenbasis ist **konsistent**, *wenn es für alle Datenelemente in der Datenbasis mindestens einen Wert aus der dem Datenelement zugeordneten Domäne gibt, der alle Spezifikationen in dem Datenelement erfüllt.*
□

Die Datenbasis gilt also als konsistent, wenn es noch mindestens einen Zustand gibt, der alle in der Datenbasis befindlichen Spezifikationen erfüllt, also analog zu den Forderungen aus Kapitel 2 genau dann, wenn noch mindestens ein Punkt im Entwurfsraum existiert, der allen Spezifikationen genügt.

6.4 Das RV-Protokoll

Bisher wurde das Konzept einer Datenbasis mit Datenelementen definiert, die Spezifikationen speichern können. Die Konsistenz einer solchen Datenbasis ist die Widerspruchsfreiheit, die mit einem geeigneten Protokoll durchgesetzt werden muß. Dieses Protokoll soll hier vorgestellt werden.

Definition 6.4.1 (Das RV-Protokoll)
Die Operationen constrain, peek *und* release *können in beliebiger Reihenfolge gegen die Datenbasis ausgeführt werden. Dabei sind nur die Anforderungen der Operationen selbst zu beachten, insbesondere die Konsistenzerhaltung der* constrain*-Operation. Die einzelnen Operationen werden atomar, isoliert und dauerhaft ausgeführt.*
□

Satz 6.4.1 (Korrektheit des RV-Protokolls)
Das RV-Protokoll garantiert die Konsistenz der Datenbasis.
□

Beweis 6.4.1 (Korrektheit des RV-Protokolls)
Der Beweis der Widerspruchsfreiheit erfolgt durch Induktion über die Operationen:

1. *Initial sind alle Datenelemente leer. Die Erfüllungsmenge jedes Datenelements ist also die dem Datenelement zugeordnete Domäne und damit nicht leer. Anders ausgedrückt sind alle Spezifikationen auf dem Datenelement erfüllbar.*
2. *Die Operation* constrain *darf nur ausgeführt werden, wenn die Spezifikationen auf dem Datenelement nach der Ausführung kompatibel sind, also wenn durch die* constrain*-Operation die Widerspruchsfreiheit nicht verletzt wird.*
3. *Die Operationen* peek *und* release *fügen keine neuen Spezifikationen auf den Datenelementen hinzu und schränken somit die Erfüllungsmenge eines Datenelements nicht weiter ein.*

Damit sieht man sehr einfach per Induktion, daß zu jedem Zeitpunkt auf jedem Datenelement mindestens ein Wert existiert, der alle Spezifikationen erfüllt (Widerspruchsfreiheit).

□

6.5 Das RV-Modell in der Praxis

Die Diskussion des RV-Modells in der Praxis zerfällt in drei Schritte. Im ersten Schritt wird die notwendige technische Umsetzung geklärt, direkt gefolgt von der Betrachtung der Performanz und Skalierbarkeit dieser Umsetzung. Im dritten Teil wird dann die Anwendungsseite betrachtet, also der Umgang eines Entwicklers mit dem RV-Modell. Hierbei wird auch der Robotergreifer als praktisches Szenario aufgegriffen.

6.5.1 Technische Umsetzung des RV-Modells

Die Diskussion der technischen Umsetzung orientiert sich an dem Aufbau des Modells, beginnt also mit den Datenstrukturen und folgt darauf aufbauend mit den Operationen, die im Rahmen der *constrain*-Operation auch die Konsistenzsicherung miteinbezieht.

Bei der Verwendung von Intervallen als Spezifikationen wird die Verwaltung der Intervalle als Paar von Untergrenze und Obergrenze trivial. Datenelemente können darauf aufbauend einfach Listen von Spezifikationen sein. Einzig für die Entfernung einzelner Spezifikationen durch die Operation *release* bietet sich eine Verwaltung als Hash-Tabelle an. Der Platzbedarf ist damit linear in der Anzahl der Spezifikationen.

Sowohl für die Operation *peek*, wie auch für die Operation *constrain* bietet sich die zusätzliche Verwaltung des Peeks eines Datenelements an. Damit ist die Operation *peek* trivial und auch die *constrain*-Operation muß nur noch eine Konjunktion bilden, nämlich die mit dem Peek und der neuen Spezifikation. Ist diese Konjunktion widerspruchsfrei, so kann die *constrain*-Operation durchgeführt werden, andernfalls wird sie abgelehnt.

Da die Konjunktion zweier Intervalle in konstanter Zeit möglich ist, haben somit die Operationen *peek* und *constrain* einen konstanten Zeitaufwand. Auch die *release*-Operation ist sehr einfach, da nur die entsprechende Spezifikation aus dem Datenelement entfernt werden muß. Allerdings wird dadurch der gespeicherte Peek ungültig und muß spätestens beim nächsten Zugriff neu berechnet werden. Diese Neuberechnung ist mit linearem Aufwand möglich (vergleiche Abschnitt 5.4).

Auch die Berechnung einer minimalen Konfliktmenge für abzulehnende *constrain*-Operationen ist mit Intervallen sehr einfach mit linearem Aufwand zu berechnen.

Sei $\mathbb{L}$ die Liste aller Spezifikationen und $\mathcal{S}()$ genau die einzubringende Speifikation. Dann könnte der Algorithmus zur Berechnung der Konfliktmenge $\mathbb{K}$ folgendermaßen aussehen:

```
Setze 𝕂 = ∅
Für alle Spezifikationen 𝒮()' in 𝕃
  Wenn 𝒮() und 𝒮()' nicht kompatibel sind
    𝕂 = 𝕂 ∪ { 𝒮()' }
  End
End
Fertig: 𝕂 ist die gesuchte Konfliktmenge
```

Im Folgenden sei die Skalierbarkeit des entwickelten RV-Modells in Kürze zusammengefaßt.

Platzbedarf: Linear in der Anzahl der Spezifikationen.

Aufwand für *constrain* und *peek*: Konstant beziehungsweise linear bei einer notwendigen Neuberechnung des Peeks oder der minimalen Konfliktmenge.

Die Neuberechnung des Peeks kann zum Beispiel über folgenden Algorithmus erledigt werden (sei $\mathbb{L}$ die Liste aller Spezifikationen):

```
Setze 𝒮() auf eine beliebige Spezifikation in 𝕃
Für alle Spezifikationen 𝒮()' in 𝕃
  Setze 𝒮() = 𝒮() ∩ 𝒮()'
End
Fertig: 𝒮() ist der gesuchte Peek
```

Aufwand für *release*: Bei Verwendung einer geeigneten Hash-Tabelle konstant.

6.5.2 Anwendung des RV-Modells

Das Revisionsmodell für die Produktentwicklung unterscheidet sich grundsätzlich von den traditionellen Datenbanken. Daher gibt es auch in der Benutzung durch den Entwickler große Unterschiede. Im RV-Modell werden als primitive Operationen *peek* und *constrain* statt lesen und schreiben angeboten. Die Funktionen sind durchaus ähnlich: *peek* „liest“ die Daten und damit die Voraussetzungen aus der Datenbasis, während *constrain* die Ergebnisse der Entwicklungsarbeit in die Datenbasis einbringt oder „schreibt“.

Allerdings gibt es zwei wichtige Unterschiede bei den Operationen im RV-Modell. Zum Einen überschreibt die Operation *constrain* nicht etwa alle vorherigen Entscheidungen,

sondern ergänzt diese. Daher ist auch der Ausdruck „einbringen“ deutlich besser geeignet als „schreiben“. Kann das System die Informationen nicht einbringen, da diese mit den bisherigen Spezifikationen nicht vereinbar sind, so erhält der Entwickler eine Liste aller in Konflikt stehenden Spezifikationen. Der Entwickler wird damit auf den Widerspruch hingewiesen, muß diesen aber selbst auflösen.

Betrachtet man das Beispiel aus Abschnitt 2.1.2, so können die einzelnen Anforderungen als Spezifikationen in die Datenbank eingebracht werden. Dies sind insbesondere:

- Mindestanforderung für die Haltekraft ($F_h \geq x_1 N$).
- Maximum für den Kolbendurchmesser ($d \leq x_2 mm$).
- Maximum für den Öldruck ($p \leq x_3 bar$).
- Daraus resultierend eine maximale Kolbenkraft ($F_k = f(d, p)$).

Der Entwickler entwirft nun die Geometrie des Greifers und kann über das Hebelgesetz das Verhältnis von Haltekraft und Kolbenkraft ermitteln. Über die Mindestanforderung an die Haltekraft kann durch das Verhältnis auch eine Mindestanforderung für die Kolbenkraft berechnet und als Ergebnis wieder in die Datenbasis eingebracht werden.

Wenn die Mindestanforderung an die Kolbenkraft mit der schon eingebrachten Maximalanforderung für die Kolbenkraft vereinbar ist, dann können die Ergebnisse eingebracht werden. Andernfalls bekommt der Entwickler die Rückmeldung, daß ein Konflikt ausgelöst wurde, den er zu bereinigen hat. Hierzu erhält der Entwickler auch gleich die Informationen über die Art des Konflikts, also die widersprüchlichen Spezifikation. Auf Basis dieser Information muß der Entwickler nun seine Arbeit und damit die Spezifikationen so abändern, daß der Widerspruch beseitigt wird.

Wenn die alte Anforderung an die maximale Kolbenkraft revidiert werden kann (zum Beispiel wenn der Öldruck erhöht werden kann), dann können die neuen Spezifikationen doch noch eingebracht werden. Sind wie im hier vorliegenden Fall die Anforderungen an den Kolbendurchmesser und den Öldruck und damit auch die Anforderung an die maximale Kolbenkraft sehr wichtig, so können die Spezifikationen nicht eingebracht werden. Der Entwickler weiß nun also, daß er die Geometrie entsprechend abändern muß, um die maximale Kolbenkraft nicht zu übersteigen.

Kapitel 7

Widerspruchsfreiheit unter Arbeitsteiligkeit

In diesem Kapitel wird das Szenario auf mehrere arbeitsteilig agierende Entwickler erweitert. Trotzdem wird noch von einem zentralen System ausgegangen, so daß die Arbeitsteiligkeit eng gekoppelt werden kann. Hierbei stellt sich die Frage, inwieweit das RV-Modell für die Arbeitsteiligkeit erweitert werden kann.

Die daraus resultierenden Probleme sind in erster Linie, daß die Arbeit eines Entwicklers in übersichtliche atomare Einheiten, die Entwurfsentscheidungen, eingeteilt wird. Eine Entwurfsentscheidung eines Entwicklers basiert in der Regel auf Entwurfsentscheidungen anderer Entwickler, die aber jederzeit zurückgesetzt werden können. Hierzu sind einerseits Rücksetzungen von Entwurfsentscheidungen zu unterstützen und andererseits geeignete Mechanismen nötig, die die Entwurfsentscheidungen so absichern, daß sie möglichst autonom sind.

Für die Diskussion der Arbeitsteiligkeit muß zuerst die in Definition 2.4.1 verwendete autorisierte Person näher betrachtet werden. Darauf aufbauend können die in diesem Kapitel zusätzlich nötigen Operationen vorgestellt werden. Diese dienen nur der Gruppierung von Spezifikationen zu Entwurfsentscheidungen und ergänzen die Operationen aus dem vorangegangenen Kapitel. Nur die Revision von Spezifikationen wird durch die Revision von ganzen Entwurfsentscheidungen ersetzt.

Die Dynamik des Entwicklungsprozesses und die damit verbundenen Wechselwirkungen zwischen den Entwurfsentscheidungen werden in zwei Schritten diskutiert. Im ersten Schritt wird von einer seriellen Durchführung der Entwurfsentscheidungen ausgegangen. Dabei werden die Wechselwirkungen zwischen abgeschlossenen Entwurfsentscheidungen und nachfolgenden Entwurfsentscheidungen beziehungsweise der Rücksetzung von vorangegangenen Entwurfsentscheidungen betrachtet. Im zweiten Schritt wird dann untersucht, inwieweit sich die Wechselwirkungen zwischen Entwurfsentscheidungen verändern, wenn Entwurfsentscheidungen auch nebenläufig eingebracht werden können.

Auf diese Erkenntnisse aufbauend kann dann das Revisionsmodell für die Arbeitsteiligkeit vorgestellt werden, gefolgt von der Betrachtung dieses Modells in der Praxis.

7.1 Zuständigkeitsgruppen

Im Gegensatz zum einfachen RV-Modell arbeiten hier mehrere Entwickler gleichzeitig an dem zu fertigenden Produkt. In diesem Fall darf es nicht passieren, daß ein Entwickler die Entwurfsentscheidungen eines anderen Entwicklers zurücksetzt. Um die Beständigkeit einer Entwurfsentscheidung aufrecht zu erhalten, dürfen nur autorisierte Entwickler Entwurfsentscheidungen zurücksetzen.

Prinzipiell kann diese Autorisierung zwar durch abstrakte Konzepte realisiert werden, im Hintergrund muß aber immer ein Mensch stehen, der die Verantwortlichkeit dann wahrnimmt. Für die Realisierung einer solchen verantwortlichen Instanz sollen hier verschiedene Konzepte vorgestellt werden:

Eine Person allein
Eine Person allein ist sicher die direkteste Möglichkeit und funktioniert auch gut, solange die Person anwesend ist. Im Fall von Krankheit oder Urlaub kann es hier zu massiven Problemen kommen. Bei der Kündigung eines Mitarbeiters ist die verantwortliche Instanz sogar nie wieder zu erreichen.

Eine Person mit Vertretung
In diesem Fall gibt es immer noch eine Person, die die Verantwortlichkeit trägt, aber zusätzlich noch einige andere Personen, die im Problemfall einspringen können. Die Asymmetrie zwischen der Person und ihrer Vertretung ist dabei nicht unbedingt nötig und eher verwirrend.

Eine Hierarchie von Personen
Denkbar wäre die Zuständigkeit an eine Person zu geben und damit automatisch den direkten und alle indirekten Vorgesetzten mit einzubeziehen. Damit hat aber der Vorstand eines Unternehmens sämtliche Verantwortlichkeiten für alle Daten, aber überhaupt keine Ahnung von der Semantik der Daten und den getroffenen Entscheidungen.

Eine Gruppe
Der allgemeinste Ansatz ist wohl die Definition einer Gruppe von Personen, die dann für die Entscheidungen zuständig ist. Dabei können alle obigen Konzepte beliebig kombiniert werden. Die Person, die die Entscheidung getroffen hat, sollte natürlich auch in der Gruppe enthalten sein und kann speziell vermerkt sein. Somit hat eine Gruppe die Verantwortung für die Entscheidung, bekommt aber gleichzeitig mitgeteilt, welche Person in der Gruppe die Entscheidung getroffen hat.

Aufgrund der Allgemeinheit wird im Folgenden von einer Gruppe von Personen als zuständige Instanz ausgegangen.

Definition 7.1.1 (Zuständigkeitsgruppe)
Eine **Zuständigkeitsgruppe** *ist eine Menge von Entwicklern. Statt Zuständigkeitsgruppe wird auch* **Z-Gruppe** *oder einfach* **ZG** *geschrieben.* □

Eine Möglichkeit für die Definition der Gruppe ist zum Beispiel die *Rolle*. So kann eine Rolle als für die Entwurfsentscheidung zuständige Instanz benutzt werden und alle Personen, die in dieser Rolle arbeiten, sind automatisch in der entsprechenden Z-Gruppe enthalten. Eine Rolle ist üblicherweise so definiert, daß alle Personen in dieser Rolle in denselben Teilbereichen arbeiten und somit auch über alle innerhalb der Rolle getroffenen Entscheidungen zumindest ein Grundwissen haben müssen. Insbesondere durch dieses Wissen macht die Verwendung von Rollen als Z-Gruppen Sinn. Im Folgenden wird aber nur vorausgesetzt, daß eine Menge von nicht leeren Z-Gruppen definiert wurde und daß jedem Entwurfsschritt genau eine Z-Gruppe zugeordnet werden kann. Wenn die Spezifikation $\mathcal{S}()$ in einem Entwurfsschritt eingebracht wird, dem die Z-Gruppe ZG_i zugeordnet ist, dann wird der Spezifikation dieselbe Z-Gruppe zugeordnet (geschrieben als $ZG(\mathcal{S}()) = ZG_i$).

7.2 Operationen

Die Operationen *constrain* und *peek* aus dem einfachen RV-Modell können direkt übernommen werden, müssen aber mit Möglichkeiten zur Gruppierung von Spezifikationen erweitert werden. Hierzu bieten sich äquivalent zur Spezifikation von Transaktionen die Operationen *begin*, *abort* und *commit* an. Hinzu kommt noch eine Operation, mit der ganze Entwurfsentscheidungen wieder entfernt werden können (*undo*), die die Operation *release* ersetzt. Die Operation *abort* (der Abbruch einer Entwurfsentscheidung) ist aus Anwendersicht äquivalent zum Festschreiben und sofortigen Zurücksetzen (*commit* und *undo*). Trotzdem soll die Operation *abort* hier eingeführt werden, da sie für den Entwickler etwas intuitiver ist als das Festschreiben mit nachfolgendem Löschen. Außerdem kann je nach Implementierung eine *abort*-Operation unter Umständen deutlich effizienter sein als die Folge *commit* und *undo*, etwa wenn vor dem *commit* noch alle Daten im flüchtigen Speicher stehen, mit dem *commit* aber von dort verdrängt werden.

begin ()

Die Operation *begin* markiert den Anfang einer Entwurfsentscheidung.

```
Generiere einen eindeutigen Bezeichner für die Entwurfsentsch.
Lege die Entwurfsentscheidung an
Markiere die Entwurfsentscheidung als aktiv
Gebe den Bezeichner zurück
FERTIG
```

abort ()

Die Operation *abort* schließt die Entwurfsentscheidung ab und entfernt sie sofort wieder aus der Datenbasis.

```
Wenn die Entwurfsentscheidung nicht aktiv ist
  FEHLER: nicht aktiv
End
Für alle Spezifikationen dieser Entwurfsentscheidung
  Lösche die Spezifikation (mit release)
End
Lösche die Entwurfsentscheidungsinformationen
FERTIG
```

commit ()

Die Operation *commit* schließt die Entwurfsentscheidung ab und beläßt sie in der Datenbasis.

```
Markiere die Entwurfsentscheidung als abgeschlossen
FERTIG
```

undo ()

Die Operation *undo* entfernt eine durch *commit* abgeschlossene Entwurfsentscheidung wieder aus der Datenbasis. Diese Operation darf nur von autorisierten Entwicklern ausgeführt werden.

```
Wenn die Entwurfsentscheidung aktiv ist
  FEHLER: ist aktiv
End
Wenn der Aufrufer nicht für die Entwurfsentscheidung
                                          zuständig ist
  FEHLER: nicht zuständig
End
Für alle Spezifikationen dieser Entwurfsentscheidung
  Lösche die Spezifikation (mit release)
```

```
End
Lösche die Entwurfsentscheidungsinformationen
FERTIG
```

Mit jeder *constrain*-Operation muß zusätzlich die verantwortliche Z-Gruppe übergeben und gespeichert werden. Auch mit dem abschließenden *commit* sollte dies geschehen, da dann sofort ersichtlich ist, wer eine Entwurfsentscheidung zu verantworten hat.

Die Operationen *begin*, *abort* und *commit* müssen von den Entwicklern genau wie die Transaktionsoperationen in traditionellen Datenbanken verwendet werden und bieten die gewohnte Atomizität. Damit wird auch eine Prüfung der Autorisierung hinfällig, da *abort* und *commit* nur von dem Entwickler ausgeführt werden kann, der die Entwurfsentscheidung eingebracht hat. Die durch *begin* und *commit* angelegten Entwurfsentscheidungen sind beständig nach Definition 2.4.1, aber nicht dauerhaft im Sinne des ACID-Paradigmas, da sie durch *undo* wieder entfernt werden können. Außerdem bieten die Gruppierungsoperationen keinerlei Isolation; sie dienen lediglich der Zusammenfassung von Spezifikationen zu Entwurfsentscheidungen.

Analog zur Terminologie bei Transaktionen sind *aktive Entwurfsentscheidungen* solche, die noch nicht durch *commit* abgeschlossen wurden. Alle anderen Entwurfsentscheidungen heißen abgeschlossen.

7.3 Die Liest-von-Beziehung

Nach Definition 2.4.4 liest eine Entwurfsentscheidung D_1 von einer anderen Entwurfsentscheidung D_2, wenn die Ergebnisse von D_2 und die Voraussetzungen von D_1 nicht disjunkt sind. Dabei stellt sich nun die Frage, wie die Rechnerunterstützung diese Liest-von-Beziehung erkennen kann. Die Ergebnisse einer Entwurfsentscheidung sind genau die im Rahmen der Entwurfsentscheidung eingebrachten Spezifikationen. Auf der anderen Seite stellt die Erkennung der Voraussetzungen einer Entwurfsentscheidung ein Problem dar. Hierfür können bisher nur die durchgeführten *peek*-Operationen herangezogen werden. Allerdings werden im Rahmen einer Entwurfsentscheidung möglicherweise sehr viele andere Spezifikationen gelesen und nur einige wenige als Voraussetzungen verwendet.

Im Folgenden wird daher davon ausgegangen, daß jeder Entwickler für jede Entwurfsentscheidung explizit spezifiziert, welches die Voraussetzungen für die Entwurfsentscheidung sind. Hierzu gibt es zwei weitere Operationen:

premise **($\mathcal{S}()$, X)**
: Die Operation *premise* vermerkt die Spezifikation $\mathcal{S}()$ als Voraussetzung für die aktuelle Entwurfsentscheidung. Die Voraussetzung muß damit erfüllt sein, also vom aktuellen Peek impliziert werden. Andernfalls wird die Operation *premise* abgelehnt.

assume **($\mathcal{S}()$, X)**
: Wenn die Konjunktion der Spezifikation $\mathcal{S}()$ und dem Peek des Datenelements X erfüllbar ist, vermerkt die Operation *assume* die Spezifikation $\mathcal{S}()$ als Annahme der aktuellen Entwurfsentscheidung. Damit ist $\mathcal{S}()$ eine angenommene Voraussetzung der aktuellen Entwurfsentscheidung. Wenn $\mathcal{S}()$ schon durch den aktuellen Peek impliziert wird, dann ist der Effekt von *assume* identisch mit dem von *premise*.

Entwurfsentscheidungen kommen damit zustande, in dem sich Entwickler durch *peek*-Operationen ein Bild von einem Ausschnitt der Datenbasis machen und daraufhin eine Entwurfsentscheidung mit Voraussetzungen und Ergebnissen erarbeiten. Die Voraussetzungen dieser Entwurfsentscheidung werden dann mit *premise* beziehungsweise *assume* dem System mitgeteilt. Die Ergebnisse der Entwurfsentscheidungen werden dann wie üblich durch *constrain*-Operationen eingebracht.

7.4 Wechselwirkungen zwischen Entwurfsentscheidungen

In diesem ersten Schritt werden die Wechselwirkungen zwischen seriell einzubringenden Entwurfsentscheidungen diskutiert. Dabei sind zwei Fälle zu unterscheiden. Einerseits können die Voraussetzungen oder Ergebnisse einer festgeschriebenen Entwurfsentscheidung durch eine nachfolgende Entwurfsentscheidung weiter verfeinert werden, während andererseits die Rücksetzung einer Entwurfsentscheidung nachfolgende Entwurfsentscheidungen betreffen kann. Die beiden Fälle werden im Folgenden getrennt diskutiert.

7.4.1 Rücksetzung von vorangegangenen Entwurfsentscheidungen

Nach Definition 2.4.2 aus Kapitel 2 besteht eine Entwurfsentscheidung aus Voraussetzungen und Ergebnissen. Ergebnisse sind damit die Folge der Entwurfsentscheidung und sind gültig, solange die Entwurfsentscheidung gültig ist. Diese hängt allerdings von den Voraussetzungen ab, die noch weiter unterteilt werden können in gültige, ungültige und angenommene Voraussetzungen.

7.4.1.1 Voraussetzungen von Entwurfsentscheidungen

Gültige Voraussetzungen
Gültige Voraussetzungen sind durch Ergebnisse oder auch Voraussetzungen von anderen Entwurfsentscheidungen vorgegeben und das Ergebnis von *premise* oder eventuell auch *assume*. Das heißt, solange die anderen Entwurfsentscheidungen gültig bleiben, sind auch die gültigen Voraussetzungen erfüllt.

Angenommene Voraussetzungen
Im Zuge der Phasenüberlappung in der Produktentwicklung müssen manchmal Annahmen über noch nicht erfolgte Entwicklungsarbeiten der vorangegangenen Phasen gemacht werden. Sie sind das Ergebnis von *assume*. Voraussetzungen einer Entwurfsentscheidung, die solche Annahmen repräsentieren, sind angenommene Voraussetzungen. Diese angenommenen Voraussetzungen sollten später durch die Arbeit der konzeptionell vorangegangenen Phasen ausgefüllt werden und damit zu gültigen Voraussetzungen werden.

Ungültige Voraussetzungen
Fallen die Entwurfsentscheidungen, auf denen eine gültige Voraussetzung beruht, weg, dann wird die gültige Voraussetzung ungültig. In diesem Fall muß der zuständige Entwickler benachrichtigt werden. Dieser kann dann die ganze Entwurfsentscheidung aufgrund der Ungültigkeit der Voraussetzungen selbst zurücksetzen, andererseits aber auch die ungültige Entwurfsentscheidung zu einer angenommenen Voraussetzung erklären und somit die Entwurfsentscheidung erhalten.

Entwurfsentscheidungen dürfen damit also nur auf gültigen und angenommenen Voraussetzungen aufbauen. Danach können sich Voraussetzungen nach dem Übergangsdiagramm in Abbildung 7.1 ändern.

7.4.1.2 Bedeutungen der unterschiedlichen Voraussetzungen für den Entwicklungsprozeß

Ungültige oder angenommene Voraussetzungen verursachen keine Inkonsistenzen im Sinn der Widerspruchsfreiheit. Das heißt, alle Spezifikationen einschließlich den ungültigen oder angenommenen Voraussetzungen sind erfüllbar und die Widerspruchsfreiheit

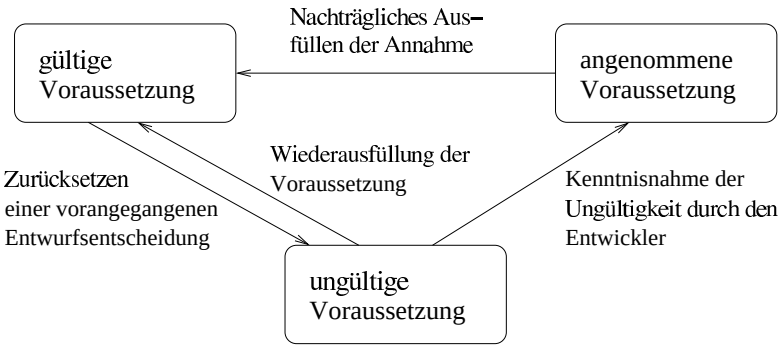


Abbildung 7.1: Typen von Voraussetzungen

ist damit gegeben. Auf der anderen Seite schränken ungültige oder angenommene Voraussetzungen die noch möglichen Produkte unnötig ein und verhindern damit möglicherweise die Entwicklung des „idealen Produkts“. Das heißt, eine Eigenschaftsausprägung dieses idealen Produkts wird durch eine ungültige oder angenommene und damit unmotivierte Spezifikation ausgeschlossen. Wenn allerdings die Entwicklung auf das ideale Produkt hin konvergiert, dann entsteht ein Widerspruch mit der unmotivierten Entwurfsentscheidung, die ja gerade dieses ideale Produkt ausschließt. Dieser Widerspruch tritt auf, wenn ein Entwickler eine Spezifikation einbringen will, die zum Beispiel nur genau die Ausprägung der Produkteigenschaft zuläßt, die von der unmotivierten Spezifikation ausgeschlossen wurde. In diesem Fall werden die am Konflikt beteiligten Entwickler benachrichtigt und erwartungsgemäß sollte der für die unmotivierte Spezifikation zuständige Entwickler die entsprechende Entwurfsentscheidung zurücksetzen, so daß wieder die Entwicklung des idealen Produkts möglich ist.

Beispiel 7.4.1 (Angenommene Voraussetzungen)
In der folgenden Tabelle hat die Entwurfsentscheidung D_1 eine Annahme über die Motorleistung gemacht, auf der dann das Ergebnis für den Öldruck beruht. D_1 ist damit eine „hypothetische“ Entscheidung, da sie auf der Annahmen beruht. Trotzdem erzeugt D_1 keinen Widerspruch, verhindert aber unter Umständen die Entwicklung des idealen Produkts. Die Entwurfsentscheidung D_2 liefert nun nachträglich genauere Informationen über die Motorleistung, die der Annahme von D_1 widerspricht. Damit muß D_2 modifiziert werden oder D_1 zurückgesetzt werden. Wenn das ideale Produkt eben mit einer Motorleistung $\leq 7kW$ auskommt, dann wird D_1 zurückgesetzt. Insofern hat sich das ideale Produkt trotzdem durchgesetzt, in dem die Entwurfsentscheidung D_1 verdrängt wurde.

	Motorleistung	*Öldruck*
D_1	assume $\geq 10kW$	constrain $\geq 5bar$
D_2	constrain $\leq 7kW$	

□

Im Fall von angenommenen Spezifikationen ist dies sogar die normale Vorgehensweise. Zu einem späteren Zeitpunkt muß die Arbeit der vorgelagerten Phasen nachgereicht werden. Dabei kann diese Arbeit die angenommenen Voraussetzungen zu gültigen machen oder einen Konflikt erzeugen. Im letzteren Fall muß der Konflikt beseitigt werden, zum Beispiel durch das Rücksetzen der auf angenommenen Voraussetzungen basierenden Entwurfsentscheidungen.

Wenn also am Ende der Entwicklung eine Reihe von angenommenen oder sogar ungültigen Voraussetzungen in der Datenbasis verbleiben, dann verursacht dies keine Probleme,

da alle anderen Spezifikationen ebenfalls erfüllt sein müssen. Es gibt damit offensichtlich mehrere Produkte, die alle im Entwicklungsprozeß als nötig gefundene Eigenschaften aufweisen. Einige dieser Produkte werden unnötigerweise durch die angenommenen und ungültigen Voraussetzungen ausgeschlossen. Trotzdem liefern die verbleibenden Produkte ein ebenso gutes Ergebnis, in dem Sinn, daß alle Anforderungen erfüllt werden.

Nichtsdestotrotz besteht die gesuchte Produktbeschreibung idealerweise nur aus gültigen Voraussetzungen und Ergebnissen. Dabei sind angenommene Voraussetzungen zwar nicht erwünscht, werden aber auch nicht bestraft, da sie durch Entwickler aufgestellt oder zumindest anerkannt wurden. Auf der anderen Seite sind ungültige Voraussetzungen nicht tragbar, da sie auf anderen Entwurfsentscheidungen basieren, die zwischenzeitlich weggefallen sind. Zuständige Entwickler müssen über ungültige Voraussetzungen benachrichtigt werden und haben damit zwei Möglichkeiten. Zum Einen können die auf den ungültigen Voraussetzungen basierenden Entwurfsentscheidungen zurückgesetzt werden oder andererseits durch reine Kenntnisnahme die ungültigen Voraussetzungen zu angenommenen gemacht werden.

Für die Rechnerunterstützung ergeben sich hier also zwei Probleme. Einerseits dürfen gültige Voraussetzungen beim Wegfallen der Entwurfsentscheidung, von der die gültigen Voraussetzungen gelesen wurden, nicht verloren gehen. Sie müssen weiterhin als ungültige Voraussetzungen gespeichert bleiben. Zum Anderen muß aber auch die Z-Gruppe der ungültig gewordenen Voraussetzung benachrichtigt werden, damit diese die Ungültigkeit zur Kenntnis nimmt und damit beseitigt.

7.4.1.3 Autonomie von Entwurfsentscheidungen

Für das erste Problem wird der Mengencharakter der Datenelemente ausgenutzt. Wird eine Spezifikation, die in einem Datenelement enthalten ist, ein zweites mal in das Datenelement eingebracht, so ändert sich dieses Datenelement nicht, da die Konjunktion zweier gleicher Spezifikationen wieder diese Spezifikation ergibt. Damit ist es möglich, daß eine Entwurfsentscheidung ihre Voraussetzungen zwar von anderen Entscheidungen „liest“, dann aber selbst in die Datenbasis einbringt. Eine Entwurfsentscheidung D bringt somit alle zu ihr gehörenden Spezifikationen ($\mathbb{V}$ und $\mathbb{R}$) als atomare Entwurfsentscheidung in die Datenbasis ein. Diese Einbringung kann automatisch vorgenommen werden, in dem alle durch erfolgreiche *premise*- und *assume*-Operationen vermerkten Voraussetzungen durch implizite *constrain*-Operationen in die Datenbasis eingebracht werden. Wenn danach andere Entwurfsentscheidungen zurückgesetzt werden, von denen D gelesen hat, so spielt dies keine Rolle, da die Voraussetzungen von D immer noch in der Datenbasis enthalten sind. Möglicherweise sind einige Voraussetzungen in D ungültig geworden, aber die so wichtige Atomizität der Entwurfsentscheidungen bleibt erhalten. Die Vorgehensweise zur Sicherung der Autonomie von Entwurfsentscheidungen ist anschaulich in Abbildung 7.2 dargestellt.

Die Operationen *premise* und *assume* werden also wie folgt erweitert:

premise **($\mathcal{S}()$, X)**

Die Operation *premise* vermerkt die Spezifikation $\mathcal{S}()$ als Voraussetzung für die aktuelle Entwurfsentscheidung. Die Voraussetzung muß damit erfüllt sein, also vom aktuellen Peek impliziert werden. Andernfalls wird die Operation *premise* abgelehnt. Im Erfolgsfall wird die Spezifikation $\mathcal{S}()$ mit einer *constrain*-Operation in X eingebracht.

```
Wenn der aktuelle Peek nicht gespeichert ist
  Berechne den aktuellen Peek
End
Wenn der aktuelle Peek die neue Spezifikation nicht impliziert
  FEHLER: keine Annahme
End
```

```
Verbinde alle Spezifikationen auf X  mit der einzubringenden
                                              Spezifikation und
                                     deren Entwurfsentscheidung
Bringe die neue Spezifikation ein (mit constrain)
FERTIG
```

assume **($\mathcal{S}()$, X)**
Wenn die Konjunktion der Spezifikation $\mathcal{S}()$ und dem Peek des Datenelements X erfüllbar ist, vermerkt die Operation *assume* die Spezifikation $\mathcal{S}()$ als Annahme der aktuellen Entwurfsentscheidung und bringt $\mathcal{S}()$ durch eine *constrain*-Operation in X ein. Wenn $\mathcal{S}()$ schon durch den Peek von X impliziert wird, dann ist $\mathcal{S}()$ eine gültige Voraussetzung, andernfalls ist es eine angenommene Voraussetzung.

```
Wenn der aktuelle Peek nicht gespeichert ist
  Berechne den aktuellen Peek
End
Wenn der aktuelle Peek die neue Spezifikation impliziert
  Die Spezifikation ist eine gültige Voraussetzung (premise)
  FERTIG
End
Wenn die Konjunktion aus der Spezifikation und dem Peek
                                                 erfüllbar ist
  Markiere die Spezifikation als Annahme
  Bringe die Spezifikation ein (mit constrain)
  FERTIG
End
FEHLER: Widerspruch
```

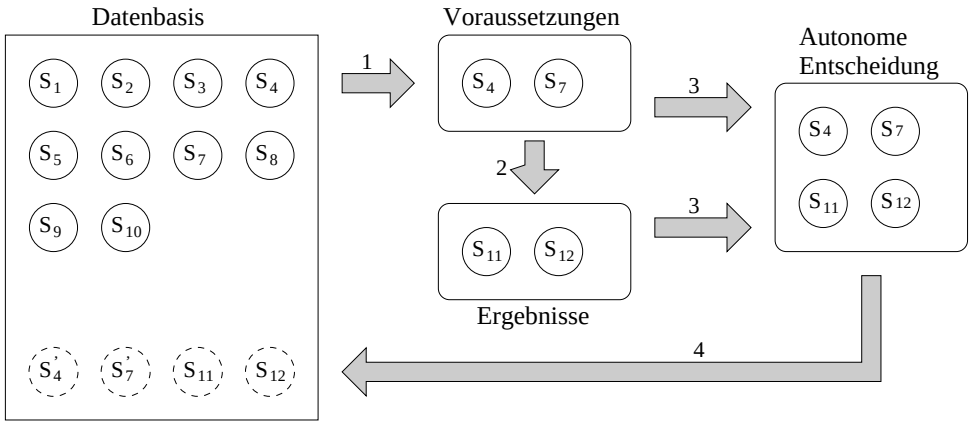


Abbildung 7.2: Autonomie von Entwurfsentscheidungen

In der Datenbasis werden die Entwurfsentscheidungen nicht nach deren Einbringung unterschieden. Es spielt also keinerlei Rolle, ob eine Spezifikation durch *constrain*, *premise* oder *assume* eingebracht wurde. *assume* unterscheidet sich nur von *constrain*, falls die einzubringende angenommene Voraussetzung auch schon gültig ist, also durch bereits existierende Spezifikationen impliziert wird. Wenn dies nicht der Fall ist, dann ist die *assume*-Operation identisch mit der *constrain*-Operation. Die Operation *premise* führt ebenfalls nur eine *constrain*-Operation aus. Gleichzeitig wird damit aber dem System bekanntgegeben, daß die einzubringende Spezifikation eine gültige Voraussetzung ist, die auf bereits in der Datenbasis befindlichen Spezifikationen beruht, also von diesen gelesen hat. Diese Information wirkt sich allerdings nicht auf die Speicherung der Spezifikation selbst aus, sondern wird im Entwicklungsgraph verwendet, der im folgenden Abschnitt vorgestellt wird.

Beispiel 7.4.2 (Autonomie der Entwurfsentscheidungen)
Im Rahmen der Entwurfsentscheidung D_1 wird die Motorleistung auf maximal $10kW$ festgesetzt. Die Entwurfsentscheidung D_2 nimmt diese maximale Motorleistung als Voraussetzung und berechnet daraus den maximalen Öldruck am Zylinder auf $5bar$.

	Motorleistung	*Öldruck*
D_1	constrain $\leq 10kW$	
D_2	premise $\leq 10kW$	constrain $\leq 5bar$

Wenn daraufhin D_1 zurückgesetzt wird, dann bleibt D_2 doch als atomare Einheit in der Datenbasis enthalten, auch wenn D_2 von D_1 gelesen hat. Die Voraussetzung von D_2 ist damit ungültig geworden, geht aber nicht verloren.

□

7.4.1.4 Der Entwicklungsgraph

Um ungültig gewordene Voraussetzungen zu erkennen, muß die Liest-von-Beziehung verwaltet werden. Hierzu wird der Entwicklungsgraph benutzt.

Definition 7.4.1 (Entwicklungsgraph)
Der **Entwicklungsgraph** *ist ein Paar $(\mathbb{K}, \mathbb{E})$, wobei die Knoten $\mathbb{K}$ genau die Entwurfsentscheidungen darstellen. Eine Kante von D nach D' wird gezogen, wenn D' von D gelesen hat.*

□

Beispiel 7.4.3 (Entwicklungsgraph)
Bei der Entwicklung des Robotergreifers könnten zum Beispiel die folgenden Entwurfsentscheidungen ablaufen:

	Motorleistung	*Öldruck*	*Kolbendurchmesser*	*Greiferbackenfläche*	*Haltekraft*
D_1	constrain $\leq 10kW$				
D_2	premise $\leq 10kW$	constrain $\leq 5bar$			
D_3			constrain $> 4500nm$ $< 4700nm$		
D_4		constrain $< 4bar$ $> 3bar$		constrain $< 3975mm^2$ $> 3850mm^2$	
D_5		premise $\leq 5bar$	premise $< 4700nm$	premise $< 3975mm^2$	constrain $\leq 20N$

Durch die Betrachtung der premise*-Operationen können Rückschlüsse über die Voraussetzungen der Entwurfsentscheidungen gezogen werden. So hat zum Beispiel D_2 die Motorleistung ($\leq 10kW$) als gültige Voraussetzung und den Öldruck ($\leq 5bar$) als Ergebnis. Der entsprechende Entwicklungsgraph ist in Abbildung 7.3 dargestellt.*

□

Anhand des Entwicklungsgraphs kann nun sehr gut verfolgt werden, wann eine Voraussetzung ungültig wird. Dies ist genau dann der Fall, wenn eine Entwurfsentscheidung mit ausgehenden Kanten zurückgesetzt wird. Alle direkten Nachfolger dieser zurückzusetzenden Entwurfsentscheidung haben von dieser gelesen und damit möglicherweise ungültig gewordene Voraussetzungen. Es gilt also für die direkten Nachfolger zu überprüfen, ob deren Voraussetzungen noch durch die Vorgänger im Entwicklungsgraph

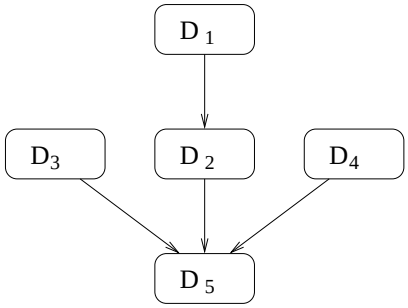


Abbildung 7.3: Der Entwicklungsgraph nach D_1 bis D_5

impliziert werden. Wird im Beispiel 7.4.3 etwa die Entwurfsentscheidung D_2 zurückgesetzt, dann ist nur D_5 direkt betroffen. Allerdings werden die Voraussetzungen von D_5 auch durch D_3 und D_4 impliziert, so daß keine der Voraussetzungen von D_5 durch die Rücksetzung von D_2 ungültig werden.

Indirekt nachfolgende Entwurfsentscheidungen sind nicht betroffen, da die direkt nachfolgende Entwurfsentscheidung noch erhalten bleibt und nur die zuständige Instanz benachrichtigt wird. Wenn diese sich natürlich entscheidet, die Entwurfsentscheidung aufgrund der ungültig gewordenen Voraussetzung zurückzusetzen, dann werden wiederum bei den nächsten betroffenen Entwurfsentscheidungen Voraussetzungen ungültig und deren Z-Gruppen benachrichtigt.

Die Benachrichtigung der zuständigen Instanzen von ungültig gewordenen Voraussetzungen übernehmen die Operationen *undo* und *abort*. Diese werden dann um folgendes Codestück ergänzt (vor der eigentlichen Rücksetzung einzufügen):

```
Für alle im Entwicklungsgraph nachfolgenden
                Entwurfsentscheidungen (D)
  Für alle gültigen Voraussetzungen dieser Entwurfsentscheidungen
    Sei diese Voraussetzung eine Spezifikation auf X
    Hole alle Spezifikationen auf X von den Vorgängern von D
    Berechne die Konjunktion dieser Spezifikationen
    Wenn diese Konjunktion die gültigen Voraussetzungen nicht
                                                    impliziert
      Die für die Entwurfsentscheidungen zuständigen Entwickler
                                                  benachrichtigen
      Gehe zurück zur äußersten Schleife
    End
  End
End
FERTIG
```

7.4.1.5 Nachträgliches Ausfüllen von angenommenen Entwurfsentscheidungen

Das konzeptionelle Ausfüllen von angenommenen Entwurfsentscheidungen kann bei der praktischen Umsetzung zu „selbsterfüllenden Prophezeiungen“ führen. Dies ist der Fall, wenn eine Entwurfsentscheidung D_1 mit der angenommenen Voraussetzung $\mathcal{S}_1()$ auf das Ergebnis $\mathcal{S}_2()$ kommt und die nachfolgende Entwurfsentscheidung D_2 mit $\mathcal{S}_2()$ als gültige Voraussetzung zum Ergebnis $\mathcal{S}_1()$ kommt. Betrachtet man nun die Annahme von D_1 als gültige Voraussetzung, da $\mathcal{S}_2()$ ja mittlerweile als Ergebnis einer Entwurfsentscheidung eingebracht wurde, dann haben D_1 und D_2 nur gültige Voraussetzungen.

Im Entwicklungsgraph bedeutet das, daß D_2 auf jeden Fall von D_1 liest. Allerdings kann die Ausfüllung der angenommenen Voraussetzung von D_1 auch als Lesen von D_2

betrachtet werden. Damit lesen die beiden Entwurfsentscheidungen voneinander und im Entwicklungsgraph entsteht damit ein Zyklus, der eben gerade die „selbsterfüllenden Prophezeiungen“ repräsentiert.

Beispiel 7.4.4 (Nachträgliches Ausfüllen von angenommenen Entwurfsentscheidungen)

Im Beispiel mit Motorleistung und Öldruck kann so ein Zyklus wie folgt aussehen. In der Entwurfsentscheidung D_1 wird die Motorleistung angenommen und daraus der resultierende Öldruck berechnet. In D_2 wird dann das Ergebnis von D_1, also der resultierende Öldruck, als Anforderung gesehen und die dafür notwendige Motorleistung berechne. Damit erfüllt D_2 natürlich genau die Voraussetzungen von D_1.

	Motorleistung	*Öldruck*
D_1	assume $\leq 10kW$	constrain $\leq 5bar$
D_2	constrain $\leq 10kW$	premise $\leq 5bar$

In diesem einfachen Beispiel ist der Zyklus im Entwicklungsgraph sehr offensichtlich. Allerdings können sich auch Zyklen mit sehr vielen beteiligten Entwurfsentscheidungen bilden, die dann natürlich auch deutlich komplexer sind. □

Um dieses Verhalten zu umgehen, dürfen angenommene Voraussetzungen nur gültig werden, wenn dadurch kein Zyklus im Entwicklungsgraph entsteht, also wenn der Grund für die Spezifikation, die die angenommene Spezifikation gültig macht, nicht in der angenommenen Spezifikation selbst liegt.

In der vorliegenden Arbeit wird auf die Ausfüllung von angenommenen Spezifikationen gänzlich verzichtet, da diese keine Vorteile bringt. Der Unterschied in angenommenen und gültigen Voraussetzungen liegt nur in der Benachrichtigung für ungültig werdende Voraussetzungen, die vorher gültig waren. Wenn aber ein Entwickler eine Annahme macht, die später gültig und dann wieder ungültig wird, dann wird der Entwickler diese Ungültigkeit sicherlich nur zur Kenntnis nehmen und damit wieder eine Annahme aus der Spezifikation machen, also genau das, was er ursprünglich schon getan hat. Vielmehr kann der Entwickler sogar irritiert werden, da er eine Annahme gemacht hat und ihm das System dann mitteilt, daß seine mittlerweile gültig gewordene Annahme nun nicht mehr gültig ist.

Im Folgenden wird daher (wie schon in Abschnitt 7.4.1.3 beschrieben) nicht unterschieden, ob Spezifikationen direkt mit *constrain* oder durch *assume* eingebracht werden. Nur wenn die Operation *assume* eine *premise*-Operation auslöst, werden die Spezifikationen anders behandelt. Es liegt also in der Verantwortung des zuständigen Entwicklers zu unterscheiden, ob eine Spezifikation eine angenommene Voraussetzung oder ein Ergebnis repräsentiert.

7.4.2 Nachträgliche Spezialisierung von Voraussetzungen oder Ergebnissen

Ein weit geringeres Problem ist die nachträgliche Verfeinerung einer Spezifikation, die in einer festgeschriebenen Entwurfsentscheidung als Voraussetzung genutzt wurde. Damit die Voraussetzung zwar spezieller geworden, bleibt aber nach wie vor gültig. Es ist zwar gut möglich, daß die Entwurfsentscheidung mit der spezielleren Voraussetzung selbst speziellere Ergebnisse produzieren könnte, diese müssen aber zukünftigen Entwurfsentscheidungen überlassen werden.

Beispiel 7.4.5 (Nachträgliche Spezialisierung von Voraussetzungen)
Geht man wieder vom bekannten Beispiel 7.4.2 aus und verschärft die in D_2 getroffenen Voraussetzungen nachträglich durch D_{13}, dann wird D_2 deswegen nicht falsch. Mit der strengeren Voraussetzung hätte das Ergebnis, also der maximale Öldruck, ebenfalls strenger ausfallen können, aber zum Zeitpunkt der Einbringung von D_2 waren diese noch nicht bekannt. Statt D_2 an die spezielleren Voraussetzungen anzupassen, wird im Rahmen der normalen Weiterentwicklung des Produkts irgendwann eine neue Entwurfsentscheidung, im Beispiel D_{42}, eingebracht, die mit den strengeren Voraussetzungen dann auch zu dem strengeren Ergebnis kommt. D_2 bleibt trotzdem in der Datenbasis, da zum Beispiel nach der möglichen Rücksetzung von D_{13} auch D_{42} zurückgesetzt werden könnte. In diesem Fall repräsentiert D_2 dann wieder den aktuellsten Stand der Produktentwicklung.

	Motorleistung	*Öldruck*
D_1	constrain $\leq 10kW$	
D_2	premise $\leq 10kW$	constrain $\leq 5bar$
...	...	...
D_{13}	constrain $\leq 9.2kW$	
...	...	...
D_{42}	premise $\leq 9.2kW$	constrain $\leq 4.6bar$

□

7.5 Nebenläufige Entwurfsentscheidungen

Nebenläufiges Einbringen von Entwurfsentscheidungen erlaubt zusätzlich die Beeinflussung von aktiven Entwurfsentscheidungen durch andere ebenfalls aktive Entwurfsentscheidungen. Dabei wird davon ausgegangen, daß einzelne Operationen atomar und isoliert ausgeführt werden. Die Wechselwirkungen zwischen aktiven Entwurfsentscheidungen können in zwei Kategorien unterteilt werden:

Wegfallen von Voraussetzungen aktiver Entwurfsentscheidungen:
Dies ist zum Beispiel der Fall, wenn eine Entwurfsentscheidung, von der bereits eine zweite Entwurfsentscheidung gelesen hat, abgebrochen oder zurückgesetzt wird. Damit werden Voraussetzungen der zweiten Entwurfsentscheidung ungültig während sie noch aktiv ist. Wie im seriellen Fall muß hier der zuständige Entwickler benachrichtigt werden. Diese Benachrichtigung kann den Entwickler vor oder nach Abschluß der Entwurfsentscheidung erreichen. Unabhängig davon muß sich dieser überlegen, ob die Entwurfsentscheidung trotz der ungültig gewordenen Voraussetzung erhalten bleiben soll. Wenn ja, dann nimmt der Entwickler die Benachrichtigung zur Kenntnis und vervollständigt die Entwurfsentscheidung (falls sie noch nicht vollständig ist). Wenn der Entwickler die Entwurfsentscheidung nicht aufrecht erhalten will, dann muß er sie zurücksetzen oder die Einbringung abbrechen.

Hierbei gilt allerdings zu beachten, daß der Entwicklungsgraph in diesem Fall auch aktive Entwurfsentscheidungen beinhalten muß, damit die Abhängigkeiten auch erkannt werden können.

Beispiel 7.5.1 (Wegfallende Voraussetzungen aktiver Entscheidungen)

Betrachtet man sich zwei aktive Entwurfsentscheidungen D_1 und D_2, die wie folgt interagieren:

1	D_1	D_2
2	begin *()*	
3		begin *()*
4	constrain $(\mathcal{S}_1(x \leq 10))$	
5		premise $(\mathcal{S}_2(x \leq 20))$
6		constrain $(\mathcal{S}_3(y \geq 5))$
7	abort *()*	

Damit ist die in Schritt 5 *gemachte gültige Voraussetzung von D_2 weggefallen, bevor D_2 abgeschlossen wurde. Da aber im Entwicklungsgraph (rechts) auch die aktiven Entwurfsentscheidungen D_1 und D_2 vertreten sind, kann dies erkannt werden und der für D_2 zuständige Entwickler wird von der ungültig gewordenen Voraussetzung unterrichtet.*

□

Weitere Verfeinerungen:

Hier geht es um die Verfeinerung einer Spezifikation, die eine Voraussetzung einer aktiven Entwurfsentscheidung darstellt. Diese gelesene Voraussetzung kann vor oder nach dem Lesen von einer anderen aktiven Entwurfsentscheidung verfeinert werden. Geschieht die Verfeinerung vorher, dann wird sie automatisch mitgelesen als ob sie von einer festgeschriebenen Entwurfsentscheidung stammen würde. Ein Abbruch der aktiven Entwurfsentscheidung, die die Verfeinerung eingebracht hat, wird in diesem Fall behandelt, als ob die Entwurfsentscheidung festgeschrieben wäre und nachträglich zurückgesetzt wird. Die nachträgliche Verfeinerung einer gelesenen Voraussetzung stellt analog zum seriellen Fall kein Problem dar. Auch hier gilt, daß die „lesende“ Entwurfsentscheidung zwar möglicherweise strengere Ergebnisse hätte erzielen können, aber dennoch nicht zu falschen Ergebnissen führt.

7.6 Das ARV-Protokoll

Bisher wurde das RV-Protokoll, Zuständigkeitsgruppen und die Operationen zur Gruppierung von Spezifikationen zu Entwurfsentscheidungen vorgestellt. Darauf aufbauend soll hier das *arbeitsteilige RV-Protokoll* oder *ARV-Protokoll* vorgestellt werden.

Definition 7.6.1 (Das ARV-Protokoll)
Die Operationen constrain, premise, assume, peek *und* undo *können in beliebiger Reihenfolge gegen die Datenbasis ausgeführt werden, wobei alle* constrain-, premise- *und* assume-*Operation Teil einer Entwurfsentscheidung eines Entwicklers sein müssen, die durch* begin *und* commit *oder* abort *eingerahmt sein muß. Dabei sind nur die Anforderungen der Operationen selbst zu beachten, insbesondere die Konsistenzerhaltung der* constrain-*Operation. Die einzelnen Operationen werden atomar, isoliert und dauerhaft ausgeführt.*

Aus der Sicht eines Benutzers können Operationen damit nach folgendem Schema ausgeführt werden:

$$((peek|undo)^*, begin, (peek|undo|constrain|premise|assume)^*, (abort|commit))^*$$

□

Die Benachrichtigung der entsprechenden Z-Gruppen beim Ungültigwerden von Voraussetzungen wird nicht ins Protokoll aufgenommen, da diese Benachrichtigung keinen Einfluß auf den folgenden Satz hat.

Satz 7.6.1 (Korrektheit des ARV-Protokolls)
Das ARV-Protokoll garantiert die Konsistenz der Datenbasis und die Beständigkeit von constrain-, premise- *und* assume-*Operationen, die durch* commit *zu einer Entwurfsentscheidung zusammengefaßt werden.*

□

Beweis 7.6.1 (Korrektheit des ARV-Protokolls)
Der Beweis der Widerspruchsfreiheit erfolgt durch Induktion über die Operationen:

1. *Initial sind alle Datenelemente leer. Die Erfüllungsmenge jedes Datenelements ist also die dem Datenelement zugeordnete Domäne und damit nicht leer. Anders ausgedrückt sind alle Spezifikationen auf dem Datenelement erfüllbar.*
2. *Die Operation* constrain *darf nur ausgeführt werden, wenn die Spezifikationen auf dem Datenelement nach der Ausführung kompatibel sind, also wenn durch die* constrain*-Operation die Widerspruchsfreiheit nicht verletzt wird.*
3. *Die Operationen* premise *ist nur erfolgreich, wenn die zu vermerkende Voraussetzung schon durch den Peek impliziert wird und löst damit höchstens eine erlaubte* constrain*-Operation aus.*
4. *Die Operation* assume *ist nur erfolgreich, wenn die Voraussetzung keinen Widerspruch erzeugt und löst somit höchstens eine erlaubte* constrain*-Operation aus.*
5. *Die Operationen* begin, abort, commit, peek *und* undo *fügen keine neuen Spezifikationen auf den Datenelementen hinzu und schränken somit die Erfüllungsmenge eines Datenelements nicht weiter ein.*

Damit sieht man sehr einfach per Induktion, daß zu jedem Zeitpunkt auf jedem Datenelement mindestens ein Wert existiert, der alle Spezifikationen erfüllt (Widerspruchsfreiheit). Zu zeigen ist noch die Beständigkeit einer durch commit *eingebrachten Entwurfsentscheidung:*

Nach der Definition der Operation commit *werden alle seit dem letzten* begin *eingebrachten Spezifikationen (Voraussetzungen wie auch Ergebnisse), also genau die Entwurfsentscheidung, in der Datenbasis belassen, können also nur noch durch andere Operationen wieder entfernt werden. Die Operationen* begin, abort, commit, constrain, premise, assume *und* peek *können keine Spezifikationen aus der Datenbasis wieder entfernen. Da* undo*-Operationen aber per Definition nur von der zuständigen Z-Gruppe aufgerufen werden dürfen, ist jede durch* commit *abgeschlossenen Entwurfsentscheidung mit allen darin enthaltenen Spezifikationen beständig.*

□

7.7 Das ARV-Modell in der Praxis

Die hier zu diskutierenden Erweiterungen des RV-Modells sind die Zuständigkeitsgruppen, die atomaren Entwurfsentscheidungen und die Autonomie derselben sowie der Entwicklungsgraph mit der Erkennung ungültig gewordener Voraussetzungen und der Benachrichtigung der entsprechenden Z-Gruppe.

7.7.1 Technische Umsetzung des ARV-Modells

Eine Zuständigkeitsverwaltung ist sehr einfach zu implementieren. So ist zum Beispiel in der Entwicklungsumgebung des SFB 346 bereits eine Benutzer- und Rollenverwaltung realisiert, die für die hier nötigen Anforderungen ausreichend ist. Damit genügt es, jeder Spezifikation die Information über die Entwurfsentscheidung mitzugeben, die dann wiederum die Rolle und damit die Zuständigkeitsgruppe kennt.

Etwas aufwendiger wird die Realisierung der atomaren Entwurfsentscheidungen. Während dem Einbringen genügt dafür ein Tabelle, die allein schon aufgrund der Operationen *begin*, *abort* und *commit* an eine Transaktionstabelle erinnert. Allerdings muß für spätere *undo*-Operationen diese Tabelle persistent gemacht werden, so daß jederzeit zu einer gegebenen Entwurfsentscheidung alle darin enthaltenen Spezifikationen ausfindig

gemacht werden können. Der Platzbedarf hierfür ist linear in der Anzahl der Spezifikationen, während der Zugriff auf diese Tabelle bei geeigneter Struktur (zum Beispiel Hash-Tabelle) praktisch in konstanter Zeit abgewickelt werden kann.

Das wiederholte Einbringen von Voraussetzungen im Rahmen von Entwurfsentscheidungen durch *premise* oder *assume* bedarf aus technischer Sicht keinerlei Änderungen. Einzig der Platzbedarf steigt entsprechend an.

Die für die Benachrichtigung notwendige Verzeigerung der Entwurfsentscheidungen entsprechend dem Entwicklungsgraph sieht komplizierter aus als sie ist. Im Rahmen von *premise*-Operationen (oder von *assume*-Operationen, deren einzubringende Voraussetzung bereits durch den Peek impliziert wird) werden alle Spezifikationen auf dem Datenelement, die nicht durch die einzubringende Voraussetzung impliziert werden, mit der einzubringenden Voraussetzung entsprechend dem Entwicklungsgraph verzeigert.

Fällt im Rahmen einer *undo*-Operation eine Spezifikationen weg, die ausgehende Kanten zu Voraussetzungen anderer Entwurfsentscheidungen hat, so müssen alle diese Voraussetzungen überprüft werden, ob deren verbleibende Vorgänger im Entwicklungsgraph die Voraussetzung weiterhin implizieren.

Der Anzahl der betroffenen Voraussetzungen ist genau die Anzahl der ausgehenden Kanten der rückzusetzenden Spezifikation. Der Aufwand für die Überprüfung der betroffenen Voraussetzung ist die Konjunktion aller verbleibenden eingehenden Kanten und ein Test auf Implikation. Damit ist der Aufwand für das Herausfinden ungültig gewordener Voraussetzungen das Produkt der ausgehenden Kanten der rückzusetzenden Entwurfsentscheidung mit der Anzahl der eingehenden Kanten der möglicherweise betroffenen Entwurfsentscheidungen. Die Behandlung von ungültig gewordenen Voraussetzungen beschränkt sich dann noch auf die Benachrichtigung, die in konstanter Zeit möglich ist.

7.7.2 Anwendung des ARV-Modells

Die Erweiterungen des RV-Modells für die Arbeitsteiligkeit sind moderat. Jeder Entwickler muß seine Spezifikationen zu semantisch zusammengehörigen Entwurfsentscheidungen gruppieren. Damit können auch nur noch komplette Entwurfsentscheidungen und nicht mehr einzelne Spezifikationen zurückgesetzt werden. Innerhalb dieser Entwurfsentscheidungen ist es die Aufgabe des Entwicklers, seine gültigen Voraussetzungen und seine Annahmen zu identifizieren und diese mit *premise* beziehungsweise *assume* statt *constrain* einzubringen. Wenn die Voraussetzungen wirklich gültig sind, dann macht die Verwendung von *premise* und *assume* statt *constrain* für den Entwickler keinen Unterschied.

Im einfachsten Fall „liest“ der Entwickler seine Voraussetzungen mit der Operation *peek* und bringt genau diese Voraussetzungen wieder mit *premise* ein. Dieses Verhalten kann für den Entwickler sogar transparent durch ein Werkzeug erledigt werden.

Wenn die durch *premise* eingebrachten gültigen Voraussetzungen ungültig werden, dann wird der zuständige Entwickler benachrichtigt. Dieser muß nun entscheiden, ob die betroffene Entwurfsentscheidung zurückgesetzt werden muß oder nicht. Einerseits können die ungültigen Voraussetzungen zu angenommenen Voraussetzungen werden, in dem der Entwickler die Entwurfsentscheidung nicht zurücksetzt. Die Entwurfsentscheidung verbleibt somit in der Datenbasis, muß aber unter Umständen später zurückgesetzt werden, falls sich die angenommenen Voraussetzungen als falsch herausstellen.

Betrachtet man wieder den Robotergreifer und die Entwurfsentscheidung D, die mit der Geometrie und dem Öldruck als gültigen Voraussetzungen auf die Haltekraft als Ergebnis kommt, dann kann diese nach Wegfallen der Voraussetzung über den Öldruck durchaus weiter bestehen bleiben. In diesem Fall ist die Spezifikation über den Öldruck einfach eine Annahme, die schon frühzeitig die weitere Arbeit an dem Greifer gestattet,

bevor der Motor spezifiziert wurde. Widerspricht der später durch den Motor aufgebaute Öldruck der angenommenen Spezifikation, dann muß dieser Konflikt beseitigt werden. Dies kann durch Rücksetzung und entsprechende Änderung der Entwurfsentscheidung D geschehen, aber auch durch die Änderungen am Motor, vor allem, wenn der Austausch des Motors einfacher ist als die möglicherweise nötigen umfangreichen Änderungen am Robotergreifer.

Angenommene Spezifikationen entsprechen somit den Annahmen der nachfolgenden Entwicklungsphasen über noch nicht geleistete Vorarbeiten der vorangegangenen Phasen. Bei der späteren Ausfüllung dieser Annahmen können Konflikte auftreten, die sowohl durch Modifikationen der vorangegangenen Phasen, wie auch durch erneutes Ausführen der nachfolgenden Phasen aufgelöst werden können. Es liegt hierbei in der Verantwortung der beteiligten Entwickler zu entscheiden, welche Spezifikationen zurückzusetzen sind und welche erhalten bleiben.

Kapitel 8

Erweitertes Demarkationsprotokoll für die Abkopplung

Um mit Abkopplung umgehen zu können, wird der Konsens in mehreren Replikaten verwaltet. Die Koordination dieser Replikate unter abgekoppelt einzubringenden Entwurfsentscheidungen stellt das Thema dieses Kapitels dar. Die in Kapitel 3 angedeutete Lösung basiert auf dem Demarkationsprotokoll, in dem eine globale Bedingung über mehreren verteilten Datenelementen durch mehrere lokale abprüfbare Bedingungen ersetzt wird.

In diesem Kapitel geht es also darum, das Demarkationsprotokoll so zu erweitern, daß den Replikaten noch näher zu beschreibende Prädikate mitgegeben werden können, die den Replikaten die lokale Einbringung von Spezifikationen erlauben, die später widerspruchsfrei zusammengeführt werden können. Für die Entwicklung des abgekoppelten Protokolls ist allerdings noch ein genaueres Modell der Abkopplung und der lokal einzubringenden Entwurfsentscheidungen nötig, das im folgenden Abschnitt entwickelt werden soll. Der erste Teil des Kapitels beschränkt sich dabei auf die synchrone An- und Abkopplung aller Replikate und wird am Ende des Kapitels auf asynchrone An- und Abkopplungen erweitert.

Da sich diese Arbeit auf eindimensionale Spezifikationen beschränkt, gibt es keine dem System bekannten Abhängigkeiten zwischen den Datenelementen. Einzig die Verbindung von Spezifikationen auf verschiedenen Datenelementen zu atomaren Entwurfsentscheidungen stellen eine Abhängigkeit zwischen verschiedenen Datenelementen dar. Da diese Atomizität aber keine Auswirkungen auf die Widerspruchsfreiheit an sich hat, kann für die Abkopplung ein einzelnes Datenelement isoliert betrachtet werden. Die Anforderungen aus dem vorangegangenen Kapitel bezüglich der Arbeitsteiligkeit werden dadurch nicht verletzt. Im Folgenden wird also die abgekoppelte Arbeit auf einem einzelnen Datenelement X betrachtet und die Widerspruchsfreiheit dieses Datenelements sichergestellt.

8.1 Einfaches Abkopplungsmodell

Das einfache Abkopplungsmodell geht entsprechend dem Demarkationsprotokoll von der synchronen Abstimmung der Replikate vor der Abkopplung aus. Diese Abstimmung mit der Verteilung der Prädikate findet dann zum Zeitpunkt t_1 statt. Danach findet bis zum Zeitpunkt t_2 keinerlei Kommunikation mehr zwischen den Replikaten statt. Die Replikate können also vollständig entkoppelt arbeiten. Spätestens zum Zeitpunkt t_2 müssen wieder alle Replikate verfügbar sein, da dann die Prädikate wieder aufgehoben werden und die lokal durchgeführten Arbeiten an die anderen Replikate verteilt werden.

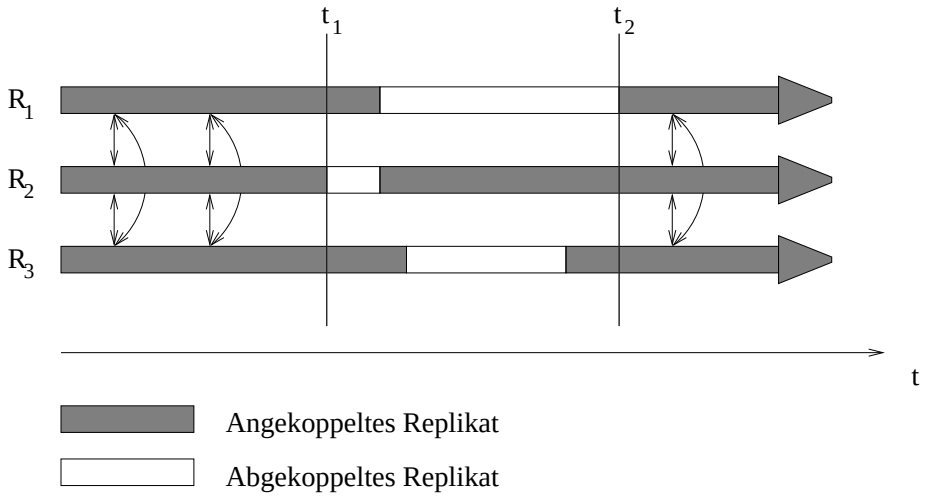


Abbildung 8.1: Drei Replikate bei der Ab- und Wiederankopplung

In Abbildung 8.1 ist das Modell der Abkopplung dargestellt. Die drei Replikate arbeiten angekoppelt bis zum Zeitpunkt t_1, an dem die Prädikate verteilt und damit die Abkopplung vorbereitet wird. Zwischen t_1 und t_2 können die Replikate beliebig abkoppeln, müssen es aber nicht. Kommunikation zwischen den Replikaten gibt es allerdings frühestens wieder zum Zeitpunkt t_2. Davor sind die Replikate völlig auf sich gestellt und dürfen nur im Rahmen ihrer Prädikate arbeiten. Zum Zeitpunkt t_2 werden die Replikate synchronisiert, das heißt es werden alle Prädikate aufgegeben und die zwischenzeitlich lokal durchgeführten Arbeiten „globalisiert“. Nach t_2 arbeiten die Replikate wieder im synchronen und angekoppelten Betrieb.

Da für den Produktentwicklungsprozeß vor allem die Phase der Abkopplung interessant ist, wird das Abkopplungsmodell so erweitert, daß nach jeder Abkopplungsphase sofort wieder eine Abkopplungsphase folgt. Die Ankopplung ist dann nur noch für die zeitweise Synchronisation und für die Neuverteilung der Prädikate erforderlich. Dieses erweiterte Abkopplungsmodell ist in Abbildung 8.2 dargestellt.

Es existieren r Replikate im System $(R_1 \ldots R_r)$, die in dem Sinn passiv sind, daß sie keine Eigeninitiative zeigen. Replikate agieren nur auf direkte Anweisung durch Klienten. Dabei können die Anweisungen entweder im Rahmen der Prädikate lokal abgearbeitet werden oder müssen mit allen anderen Replikaten abgestimmt werden durch globale Operationen. Dies wird zwar von den Replikaten selbst erledigt, allerdings nur auf direkten Anweisung durch Klienten.

Während der Abkopplungsphasen können auf den Replikaten nur lokale Operationen ausgeführt werden, also solche, die das Replikat ohne Kommunikation mit anderen Replikaten ausführen kann. Für die Synchronisation und Neuverteilung der Prädikate sind dann allerdings globale Operationen erforderlich, die an allen Replikaten synchron ausgeführt werden.

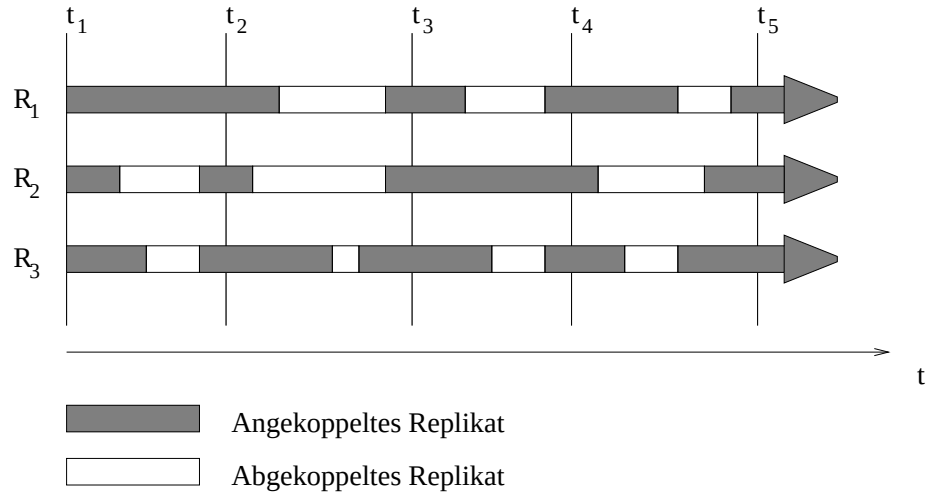


Abbildung 8.2: Drei Replikate im erweiterten Abkopplungsmodell

Definition 8.1.1 (Globale und lokale Operationen)
Eine **globale Operation** *ist eine Operation, die von einem Dienstnehmer auf einem beliebigen Replikat initiiert wird und von dort synchron an alle anderen Replikate weitergeleitet wird.*

Eine **lokale Operation** *wird von einem Dienstgeber auf einem Replikat initiiert und dort vollständig lokal ohne Kommunikation mit anderen Replikaten ausgeführt.* □

Im Folgenden bezeichnet zu jedem Zeitpunkt t die Spezifikation $\mathcal{G}()$ den Peek des betrachteten Datenelements zum Zeitpunkt der letzten Synchronisation vor t. $\mathcal{G}()$ ist damit zu jedem Zeitpunkt an allen Replikaten gleich. Zum Zeitpunkt t sind die *lokalen Spezifikationen* $\mathcal{S}_{1,\omega}()$, $\ldots\mathcal{S}_{s_\omega,\omega}()$ am Replikat R_ω genau die Spezifikationen, die seit der letzten Synchronisation vor t eingebracht wurde und damit noch nicht an die anderen Replikate verteilt wurden.

Der *lokale Peek* $\mathcal{P}_\omega()$ zum Zeitpunkt t ist die Konjunktion von $\mathcal{G}()$ zum Zeitpunkt t mit allen lokalen Spezifikationen an R_ω zum Zeitpunkt t. Dies sind genau die an R_ω zum Zeitpunkt t bekannten Spezifikationen. Der lokale Peek eines Datenelements kann damit ohne Kommunikation mit anderen Replikaten ermittelt werden. Der *globale Peek* $\mathcal{H}()$ zum, Zeitpunkt t ist die Konjunktion der lokalen Peeks aller Replikate. Diese Konjunktion ist nur im Moment einer Synchronisation bekannt.

8.2 Erweitertes Demarkationsprotokoll

Die Idee des Demarkationsprotokolls ist die Aufteilung einer globalen Bedingung, die nur durch Kommunikation geprüft werden kann, in mehrere lokale überprüfbare Bedingungen, deren Konjunktion die globale Bedingung impliziert. Das Problem hierbei ist, daß die globale Bedingung in dieser Arbeit die Widerspruchsfreiheit ist, die nicht so einfach aufgespalten werden kann. Es ist also zuerst eine möglicherweise strengere Zwischenbedingung zu konstruieren, die die globale Widerspruchsfreiheit garantiert und selbst für die Replikate aufspaltbar ist.

Die Widerspruchsfreiheit heißt nun nichts anderes, als daß zu jedem Zeitpunkt t gilt:

$$Sat(\mathcal{H}())$$

Im folgenden Abschnitt gilt es nun, eine alternative Bedingung zu fingen, die die genannte Widerspruchsfreiheit impliziert und sich auf die einzelnen Replikate aufteilen läßt.

8.2.1 Eine alternative und verschärfte Konsistenzbedingung

Im einfachsten Fall sind neue Spezifikationen nur an einem Replikat zugelassen. Damit ergibt sich eine Konsistenzbedingung, die die Widerspruchsfreiheit impliziert und sich sehr gut auf die Replikate aufteilen läßt. Allerdings kann in diesem Fall nur noch auf einem Replikat gearbeitet werden. Daher ist eine feinere Bedingung notwendig.

Der in dieser Arbeit entwickelte Ansatz ist die Einteilung in verfeinernde und fixierende Spezifikationen. Verfeinerungen sind solche Spezifikationen, die den Peek weiter einschränken, während fixierende Spezifikationen schon durch den aktuellen Peek impliziert werden.

Für jedes Replikat gibt es genau eine *Absichtssperre*, die entweder verfeinernd oder fixierend sein kann. Da für alle Replikate zusammen nur eine *verfeinernde Sperre* möglich ist, werden verfeinernde Sperren als *exklusive Sperren* oder *X-Sperren* bezeichnet. Im Gegensatz dazu kann es mehrere *fixierende Sperren* geben. Fixierende Sperren werden daher als *S-Sperren* bezeichnet.

Die Replikate fordern ihre Absichtssperren durch einen Entwickler initiiert selbst an. Da dies durch synchrone Kommunikation mit allen anderen Replikaten geschieht, kennt jedes Replikat seine eigenen Absichtssperren, aber auch die Absichtssperren der anderen Replikate.

X- oder S-Sperren selbst sind nichts anderes als Spezifikationen, allerdings mit unterschiedlicher Semantik. Jedem Replikat ist genau eine Absichtssperre zugeordnet. Es gibt also r Absichtssperren $(\mathcal{A}_1() \ldots \mathcal{A}_r())$.

Sei $\mathbb{X}$ die Menge aller X-Sperren und $\mathbb{Y}$ die Menge aller S-Sperren auf einem Datenelement. Zusätzlich sei $\mathbb{A} = \mathbb{X} \cup \mathbb{Y}$ die Menge aller Absichtssperren. Für diese Absichtssperren müssen folgende Bedingungen gelten:

- Es existiert maximal eine X-Sperre

$$|\mathbb{X}| \leq 1 \tag{$A_{8.1}$}$$

- Die Konjunktion aller S-Sperren ist kompatibel mit dem globalen Spezifikationen

$$Sat(\mathcal{G}() \wedge \bigwedge_{\mathcal{Y}() \in \mathbb{Y}} \mathcal{Y}()) \tag{$A_{8.2}$}$$

- Eine mögliche X-Sperre impliziert die Konjunktion aller S-Sperren

$$\forall_{\mathcal{X}() \in \mathbb{X}}\, \mathcal{X}() \Rightarrow \bigwedge_{\mathcal{Y}() \in \mathbb{Y}} \mathcal{Y}() \tag{$A_{8.3}$}$$

8.2.2 Bedeutung der Absichtssperren für die Replikate

Im Rahmen der an jedem Replikat vorhandene Absichtssperre können die Replikate nun neue Spezifikationen als Teil von Entwurfsentscheidungen einbringen. Dabei sind die an den Replikaten erlaubten Spezifikationen zu unterscheiden nach Replikaten mit X-Sperren und solchen mit S-Sperren:

Replikate mit X-Sperren
Auf einem Replikat mit einer X-Sperre dürfen Spezifikationen eingebracht werden, die mit der X-Sperre und dem aktuellen lokalen Peek kompatibel sind. Sei R_ω ein Replikat mit einer X-Sperre, auf dem die Spezifikation $\mathcal{S}()$ neu eingebracht werden soll. Dann muß gelten:

$$Sat(\mathcal{X}() \wedge \mathcal{S}() \wedge \bigwedge_{i=1..s_\omega} \mathcal{S}_{i,\omega}()) \tag{$A_{8.4}$}$$

Replikate mit S-Sperren

Auf einem Replikat mit einer S-Sperre dürfen Spezifikationen eingebracht werden, die von der S-Sperre impliziert werden. Sei R_ω ein Replikat mit der S-Sperre $\mathcal{Y}()$, auf dem die Spezifikation $\mathcal{S}()$ neu eingebracht werden soll. Dann muß gelten:

$$\mathcal{Y}() \Rightarrow \mathcal{S}() \qquad (A_{8.5})$$

Beispiel 8.2.1 (X-Sperren und S-Sperren)
Sei X ein Datenelement, das in zwei Replikaten X_1 und X_2 existiert. X kann zum Beispiel die maximale Öffnungsweite des Robotergreifers repräsentieren. Initial haben X_1 und X_2 denselben Peek ($[2,5]$). Vor der Abkopplung werden auf den Replikaten folgenden Absichtssperren gesetzt:

- *S-Sperre $\mathcal{Y}()$ auf X_1 mit $[3,5]$.*
- *X-Sperre $\mathcal{X}()$ auf X_2 mit $[3,4]$.*

Damit dürfen lokal auf X_1 nur Intervalle eingebracht werden, die $[3,5]$ umfassen, während auf X_2 im Rahmen von $\mathcal{X}()$ nur solche Intervalle eingebracht werden können, die in ihrer Kombination mindestens einen Wert aus $[3,4]$ akzeptieren, also eine nicht leere Schnittmenge mit $[3,4]$ haben.

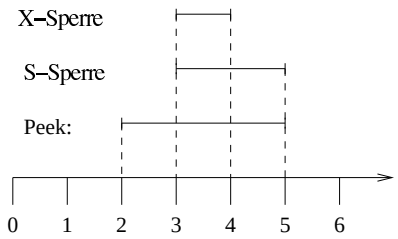


Abbildung 8.3: Das Datenelement mit Peek und Absichtssperren

Bei der Wiederankopplung von X_1 und X_2 existiert damit mindestens ein Wert x, der alle Spezifikationen erfüllt. Dieser Wert x wird vorgegeben durch die Spezifikationen, die im Rahmen der X-Sperre $\mathcal{X}()$ eingebracht wurden, da diese auf jeden Fall einen Wert im Intervall $[3,4]$ akzeptieren. Da aber sowohl der alte Peek ($[2,5]$), wie auch alle Spezifikationen im Rahmen von S-Sperren alle Werte aus dem Intervall $[3,4]$ akzeptieren, erfüllt der Wert x alle Spezifikationen, die auf den Replikaten X_1 und X_2 vorhanden sind. □

Definition 8.2.1 (Maskierungen)
Eine Spezifikation, die nach den vorangegangenen Regeln im Rahmen der Absichtssperre $\mathcal{A}()$ eingebracht wurde, ist von dieser Absichtssperre **maskiert**. *Eine Entwurfsentscheidung wird von einer Absichtssperre* **maskiert**, *wenn mindestens eine ihrer Spezifikationen von der Absichtssperre maskiert wird.* □

8.2.3 Aufteilung der Absichtssperren

Während der Synchronisation können die Replikate nacheinander ihre Absichtssperren für die nächste Abkopplungsperiode anfordern. Die Vergabe geschieht nach dem Prinzip: „Wer zuerst kommt, mahlt zuerst“ (first come - first serve). Für jede neu angeforderte Absichtssperre wird dann geprüft, ob sie mit den bereits vergebenen Absichtssperren kompatibel ist. Wenn dies nicht der Fall ist, dann wird die Absichtssperre so nicht vergeben. Wenn also das erste Replikat eine X-Sperre anfordert, dann können alle weiteren Replikate nur noch S-Sperren erhalten. Fordert ein Replikat keine Absichtssperre an, so hat es automatisch eine S-Sperre mit der wahren Spezifikation, das heißt eine Spezifikation, die allen Elementen der Domäne den Wert *wahr* zuordnet.

Beispiel 8.2.2 (Aufteilung der Absichtssperren)
Das Datenelement X repräsentiert die maximale Öffnungsweite des Robotergreifers und existiert in drei Replikaten, die alle eine wahre S-Sperre als Absichtssperre haben (eine Spezifikation, die allen Elementen der Domäne den Wert wahr zuordnet). Wenn nun ein beliebiges Replikat seine Absichtssperre ändern will, dann ist diese Operation in jedem Fall erfolgreich. Angenommen R_1 ändert seine Absichtssperre auf die S-Sperre $x \in [3,5]$ (Schritt 1). Wenn nun allerdings R_2 seine Absichtssperre in die X-Sperre $x \in [1,6]$ ändern will, dann ist dies nach Formel $A_{8.3}$ nicht möglich und wird abgelehnt (Schritt 2). R_2 darf also höchstens die X-Sperre mit der Spezifikation $x \in [3,5]$ setzen (Schritt 3). So wird auch die Anforderung des dritten Replikats nach einer S-Sperre mit der Spezifikation $x \in [2,4]$ abgelehnt (Schritt 4). Akzeptabel wäre sie noch gewesen, wenn R_3 diese S-Sperre vor R_2 angefordert hätte. Nachdem aber R_2 schon die X-Sperre gesetzt hat, muß R_3 seine geplante S-Sperre weiter abschwächen und wird möglicherweise die Spezifikation $x \in [2,5]$ als S-Sperre setzen (Schritt 5). Die angeforderten und erfolgreich gesetzten Absichtssperren dieses Beispiels sind in der Abbildung 8.4 graphisch aufgetragen.

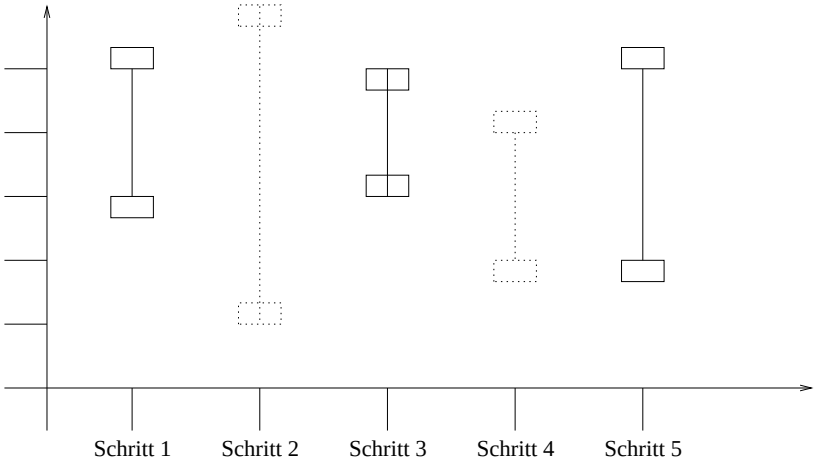


Abbildung 8.4: Erfolgreiche und abgelehnte Modifikationen der Absichtssperren

□

Wenn für eine angeforderte Absichtssperre bereits eine inkompatible Absichtssperre aktiv ist, so kann über eine Benachrichtigung der am Konflikt beteiligten Parteien nachgedacht werden. Diese Vorgehensweise entspricht aber in weiten Teilen den Überlegungen aus Kapitel 6 und soll hier nicht weiter diskutiert werden.

8.3 Das erweiterte Demarkationsprotokoll

Definition 8.3.1 (Das erweiterte Demarkationsprotokoll)
Das **erweiterte Demarkationsprotokoll** *oder* **ED-Protokoll** *ist wie folgt definiert:*

1. *Initial haben alle Replikate eine S-Sperre mit der wahren Spezifikation.*
2. *Zu jedem Synchronisationspunkt können die Replikate Absichtssperren verändern, das heißt eine Absichtssperre durch eine andere ersetzen. Dieser Ersetzung ist nur erlaubt, wenn die Absichtssperren kompatibel nach den Formeln $A_{8.1}$, $A_{8.2}$ und $A_{8.3}$ sind.*
3. *Zwischen den Synchronisationspunkten kann an den Replikaten mit dem arbeitsteiligen RV-Protokoll aus Abschnitt 7.6 gearbeitet werden, wobei neue Spezifikationen nur eingebracht werden dürfen, wenn diese durch lokale Absichtssperren nach den Formeln $A_{8.4}$ und $A_{8.5}$ erlaubt sind*

4. *Zu den Synchronisationspunkten darf es keine aktiven Entwurfsentscheidungen geben. Mit der Synchronisation werden alle seit der letzten Synchronisation lokal ausgeführten Operationen an alle anderen Replikate weitergeleitet. Da* peek*-Operationen keine Veränderungen der Datenbasis zur Folge haben, müssen diese nicht ausgetauscht werden. Der Austausch beschränkt sich also auf die eingebrachten Entwurfsentscheidungen und die* undo*-Operationen.*

□

Der zweite Schritt entspricht dabei der Aufteilung der globalen Bedingung in mehrere lokal prüfbare Bedingungen im Demarkationsprotokoll. Der dritte Schritt sind dann die lokalen Arbeiten im Rahmen der für das Replikat geltenden Bedingung. Die Einhaltung aller Voraussetzungen führt dann zur garantierten Widerspruchsfreiheit des globalen Peeks.

8.3.1 Korrektheit des ED-Protokolls

Die Korrektheit des erweiterten Demarkationsprotokolls hat mehrere Aspekte. Die Beständigkeit von Spezifikationen wird dabei nicht mehr betrachtet, da sich hier im Vergleich zum angekoppelten Modell nichts ändern. Spezifikationen können nach wie vor nur von autorisierten Entwicklern zurückgesetzt werden und gehen auch sonst nicht verloren. Die Korrektheit ist damit in erster Linie die Frage der Widerspruchsfreiheit des globalen Peeks. Die Auswirkungen der Abkopplung auf die Liest-von-Beziehung sind nicht unmittelbar eine Frage der Korrektheit und werden daher im Anschluß behandelt.

Satz 8.3.1 (Korrektheit des erweiterten Demarkationsprotokolls)
Mit dem erweiterten Demarkationsprotokoll gilt zu jedem Zeitpunkt die globale Widerspruchsfreiheit, also $Sat(\mathcal{H}())$.

□

Beweis 8.3.1 (Korrektheit des erweiterten Demarkationsprotokolls)
Durch das Protokoll werden die Formeln zur Kompatibilität von Absichtssperren ($A_{8.1}$, $A_{8.2}$ *und* $A_{8.3}$*) und die Formeln für die neuen Spezifikationen (*$\mathcal{T}$ *und* $A_{8.5}$*) durchgesetzt. Jetzt gilt es noch zu zeigen, daß durch diese Formeln die Widerspruchsfreiheit des globalen Peeks folgt. Dabei sind zwei Fälle nach der Anzahl der X-Sperren zu unterscheiden.*

Es existiert eine X-Sperren ($|\mathbb{X}| = 1$)
Sei R_ω *das Replikat mit der X-Sperre. Damit sind alle an* R_ω *eingebrachten Spezifikationen mit* $\mathcal{G}()$ *und allen lokalen Spezifikationen kompatibel. Sei*

$$\mathcal{T}() = \mathcal{X}() \wedge \mathcal{S}() \wedge \bigwedge_{i=1..s_\omega} \mathcal{S}_{i,\omega}()$$

Damit gilt $Sat(\mathcal{T}())$*. Da* $\mathcal{X}()$ *aber alle S-Sperren der anderen Replikate impliziert und diese wiederum alle lokalen Bedingungen implizieren gilt:*

$$\mathcal{T}() \Rightarrow \bigwedge_{\tau=1..r} \bigwedge_{i=1..s_\tau} \mathcal{S}_{i,\tau}()$$

und damit automatisch

$$Sat(\mathcal{T}() \wedge \bigwedge_{\tau=1..r} \bigwedge_{i=1..s_\tau} \mathcal{S}_{i,\tau}())$$

Es existiert keine X-Sperren ($|\mathbb{X}| = 0$)
Sei

$$\mathcal{T}() = \bigwedge_{\tau=1...r} \mathcal{Y}_{R_\tau}()$$

Damit gilt $Sat(\mathcal{T}())$ *und zusätzlich*

$$Sat(\mathcal{T}() \wedge \mathcal{G}())$$

Da alle lokalen Spezifikationen aber durch $\mathcal{T}()$ *impliziert werden gilt auch*

$$Sat(\mathcal{T}() \wedge \mathcal{G}() \wedge \bigwedge_{\tau=1..r} \bigwedge_{i=1..s_\tau} \mathcal{S}_{i,\tau}())$$

□

Das heißt, alle im Rahmen der Absichtssperren eingebrachten Spezifikationen können widerspruchsfrei zusammengeführt werden.

8.4 Die Liest-von-Beziehung im abgekoppelten Betrieb

Die zwei Aspekte der Liest-von-Beziehung aus dem arbeitsteiligen RV-Modell in Kapitel 7 sind auch die hier zu betrachtenden. Dies sind Rücksetzungen von Entwurfsentscheidungen, die Voraussetzungen für andere Entwurfsentscheidungen liefern, und verschärfte Voraussetzungen von anderen Replikaten, die an einem Replikat noch nicht verfügbar sind.

8.4.1 Verzögerte *undo*-Operationen

Eine verzögerte *undo*-Operation bedeutet, daß am Replikat R_ω eine Entwurfsentscheidung D_i eine gültige Voraussetzung auf die Ergebnisse der Entwurfsentscheidung D_j basiert, wobei D_j auf einem andere Replikat in derselben Abkopplungsphase zurückgesetzt wird. Die Rücksetzung von D_j macht mindestens eine Voraussetzung von D_i ungültig. Allerdings kann dies erst bei der nächsten Synchronisation erkannt werden. Probleme im Sinn der Korrektheit können hierbei nicht auftreten. Die Verspätung der Benachrichtigung des Benutzers hat keinerlei Einfluß auf die Konsistenz der replizierten Datenbasis. Nur wenn die Rücksetzung von D_j zeitlich gesehen vor der Durchführung von D_i stattgefunden hat, dann wäre im angekoppelten Betrieb D_i nie so zustande gekommen. Trotzdem wird der zuständige Entwickler benachrichtigt und kann D_i gegebenenfalls wieder zurücksetzen.

Beispiel 8.4.1 (Verzögerte *undo*-Operationen)
In Abbildung 8.5 sind zwei Replikate zu sehen. Vor der ersten Synchronisation wird auf

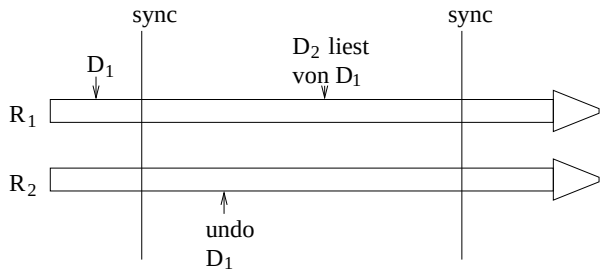


Abbildung 8.5: Eine verzögerte *undo*-Operation

dem Replikat R_1 *die Entwurfsentscheidung* D_1 *eingebracht, die durch die Synchronisation an* R_2 *weitergeleitet wird. Zwischen den beiden Synchronisationen wird am Replikat* R_2 *die Entwurfsentscheidung* D_1 *gelöscht. Danach liest am Replikat* R_1 *die Entwurfsentscheidung* D_2 *von* D_1*, obwohl* D_1 *zu diesem Zeitpunkt bereits an* R_2 *gelöscht wurde. Erst mit der zweiten Synchronisation wird diese* undo*-Operationen an* R_1 *weitergeleitet und die gültigen Voraussetzungen von* D_2 *werden ungültig. Dadurch entsteht aber kein Widerspruch, sondern nur ungültige Voraussetzungen, über die der zuständige Entwickler informiert werden muß.*

□

8.4.2 Verschärfte Voraussetzungen an anderen Replikaten

Auf der anderen Seite können Voraussetzungen einer Entwurfsentscheidung D_i am Replikat R_ω durch die Ergebnisse einer anderen Entwurfsentscheidung D_j auf einem anderen Replikat R_τ schon vor D_i verschärft worden sein, wobei D_i von dieser Verschärfung nichts mitbekommt. Werden die Voraussetzungen erst nach dem Einbringen von D_i strenger gemacht, so entspricht dies dem normalen Vorgehen und muß nicht weiter betrachtet werden. Wird D_j allerdings zeitlich gesehen vor D_i durchgeführt, dann könnte D_i im angekoppelten Betrieb möglicherweise zu strengeren Ergebnissen kommen. Trotzdem sind die Ergebnisse von D_i nach den Diskussionen aus Kapitel 7 nicht falsch und führen vor allem nicht zu Widersprüchen.

8.5 Asynchrones Abkopplungsmodell

Das Abkopplungsmodell aus Abschnitt 8.1 diente der einfachen Einführung des erweiterten Demarkationsprotokolls, ist aber für die Praxis wenig geeignet. Dies liegt in erster Linie daran, daß regelmäßige Synchronisationen nötig sind, zu denen alle Replikate angekoppelt sein müssen. In einem großen Produktentwicklungsprozeß über vielen Abteilungen eines Unternehmens ist diese Vorstellung allerdings kaum realistisch. So wird es möglicherweise immer mindestens ein abgekoppeltes Replikat geben und die notwendige Synchronisation kann nie zustande kommen.

In diesem Abschnitt wird ein asynchrones Abkopplungsmodell vorgestellt, in dem das erweiterte Demarkationsprotokoll benutzt werden kann, ohne daß jemals alle Replikate gleichzeitig angekoppelt sein müssen. Die Idee dabei ist, daß die bei der Synchronisation ausgetauschten Informationen einzelne Replikate durchaus zeitverzögert erreichen dürfen. Wichtig ist allerdings, daß alle zwischen den Replikaten ausgetauschten Informationen an allen Replikaten in derselben Reihenfolge eintreffen. Dies kann über eine globale Historie erreicht werden, die durch ein Quorumverfahren durchgesetzt wird. Das asynchrone Abkopplungsmodell wird in drei Schritten entwickelt, die anschaulich in Abbildung 8.6 dargestellt sind.

Im ersten Schritt werden die Synchronisationspunkte auf einzelne globale Operationen reduziert. Damit wird ein Synchronisationspunkt immer von einem Replikat ausgelöst, nämlich genau von dem Replikat, auf dem die globale Operation initiiert wird. Ein Synchronisationspunkt aus dem einfachen Abkopplungsmodell wird dann hier durch mehrere synchron auszuführende globale Operationen ersetzt.

Im zweiten Schritt werden die starken Anforderungen an die synchrone Ausführung der globalen Operationen gelockert. Nur die Entwurfsentscheidungen, die von einer abzuschwächenden Absichtssperre maskiert wurden, müssen abgeschlossen sein und werden, falls dies noch nicht erledigt wurde, mit der Abschwächung der Absichtssperre an die anderen Replikate weitergeleitet. Damit dürfen alle anderen Entwurfsentscheidungen, insbesondere auch alle Entwurfsentscheidungen an den anderen Replikaten, während der globalen Operation aktiv bleiben. Außerdem werden mit einer globalen Operation sämtliche auf diesem Replikat seit der letzten von diesem Replikat initiierten globalen Operation durchgeführten *undo*-Operationen an die anderen Replikate verteilt.

Im dritten Schritt wird eine Historie von globalen Operationen gefordert, die an allen Replikaten gleich ist. Alle globalen Operationen müssen also an allen Replikaten in derselben Reihenfolge eintreffen. Geht man von Netzpartitionen aus, so dürfen damit immer nur in einer Partition globale Operationen ausgeführt werden. Diese Anforderung kann durch ein Quoren-Verfahren erreicht werden. Dabei ist für jede globale Operation ein Quorum nötig und zusätzlich dürfen zwei aufeinanderfolgende Quoren nicht disjunkt sein.

Hinzu kommt im dritten Schritt noch, daß vor jeder globalen Operation im Quorum untersucht werden muß, welches Replikat das aktuellste ist. Alle anderen Replikate

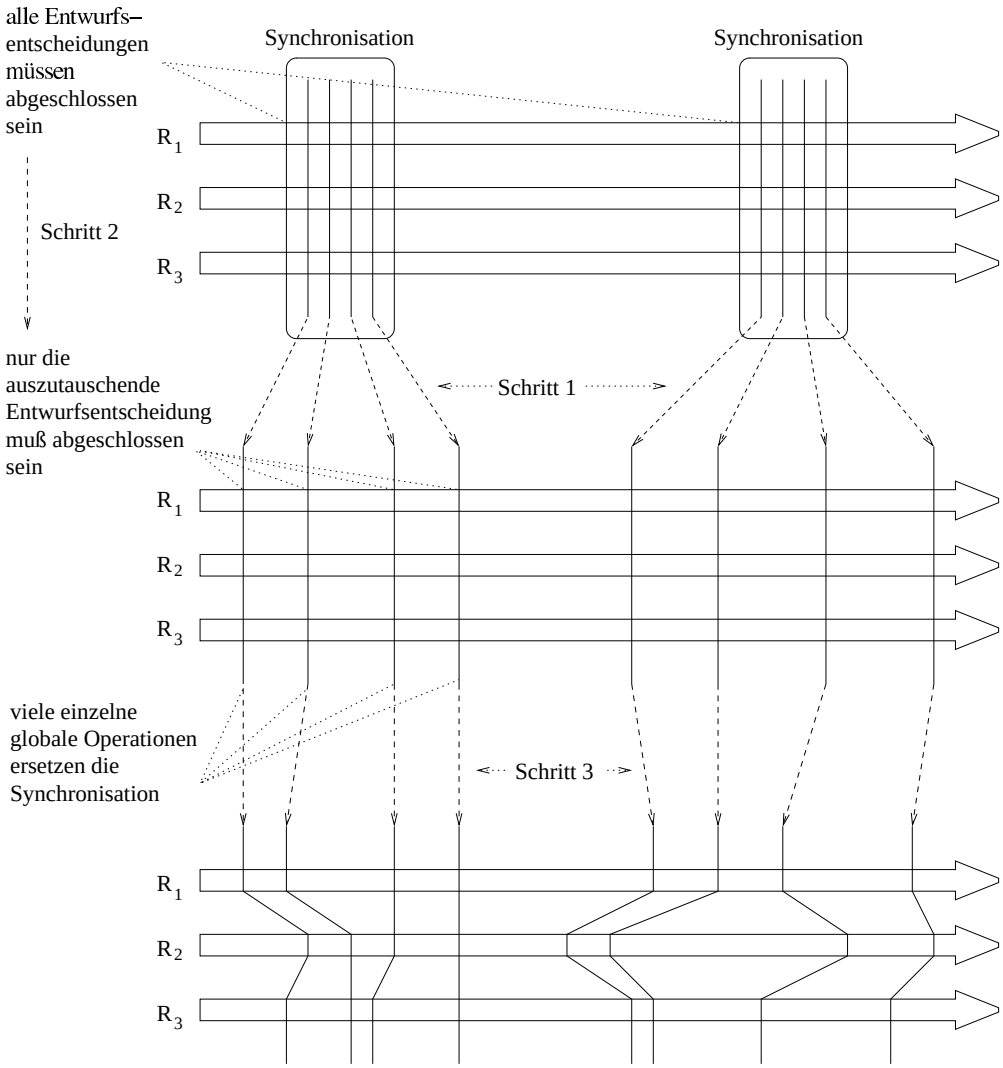


Abbildung 8.6: Die Entwicklung des asynchronen Abkopplungsmodells

müssen sich von diesem dann die neueren globalen Operationen besorgen und sich selbst damit auf den aktuellsten Stand bringen. Damit ist garantiert, daß jedes Replikat, auf dem eine globale Operation ausgeführt wird, alle vor dieser globalen Operation ausgeführten anderen globalen Operationen schon kennt. Eine globale Operation wird damit zwar asynchron ausgeführt, trifft aber jedes Replikat im selben Zustand bezüglich der Verteilung der Absichtssperren an.

Definition 8.5.1 (Quorum)
Ein **Quorum** *ist eine Menge gekoppelter Replikate.*

In einer Folge von Quoren ist ein einzelnes Quorum gültig, *wenn das Quorum mit dem direkt vorangegangenen Quorum nicht disjunkt ist.*
□

Für die weitere Diskussion bleibt noch festzuhalten, daß jegliche Kommunikation zwischen den Replikaten durch globale Operationen zum Ändern von Absichtssperren durchgeführt wird. Für die im Folgenden vorgestellte und in den nachfolgenden Kapiteln verwendete asynchrone Abkopplung wird eine Folge von gültigen Quoren erforderlich. Ungültige Quoren dürfen dabei nicht enstehen. Wenn Mißverständnisse ausgeschlossen sind, wird daher nachfolgend für jedes Quorum unterstellt, daß es gültig ist.

8.5.1 Das asynchrone Abkopplungsprotokoll

Für das asynchrone Abkopplungsprotokoll wird von einem einfachen Quorenverfahren ausgegangen, das auch in der Literatur häufig zu finden ist. Dazu wird die Gültigkeit von Quoren durchgesetzt, in dem an jedem Quorum mehr als die Hälfte der existierenden Replikate beteiligt sind. Damit kann auf sehr einfache Weise garantiert werden, daß zwei beliebige Quoren, also auch zwei direkt aufeinanderfolgende, nicht disjunkt sein können.

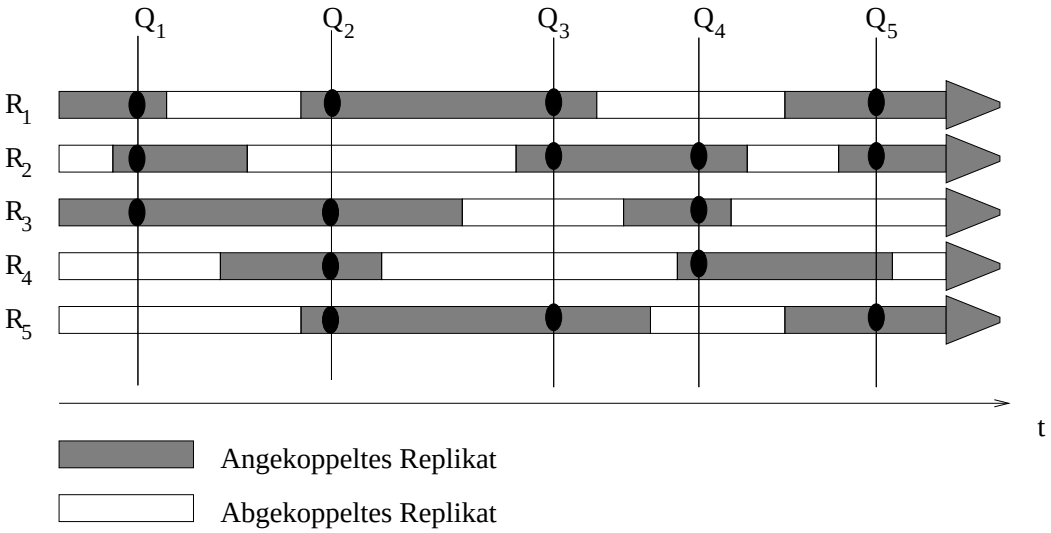


Abbildung 8.7: Ein einfaches Quorenverfahren im asynchronen Abkopplungsmodell

Abbildung 8.7 macht das Modell der asynchronen Abkopplung deutlich. So sind zum Beispiel beim Quorum Q_1 nur die Replikate R_1, R_2 und R_3 beteiligt. Die anderen beiden Replikate können an diesem Quorum gar nicht teilnehmen, da sie abgekoppelt sind. Damit wissen R_4 und R_5 auch nichts über die geänderten Absichtssperren der anderen Replikat. Trotzdem muß aber die Kompatibilität der Absichtssperren weiterhin gelten, da dies ja eine Grundvoraussetzung für die Änderung ist. Alle Replikate haben somit gültige Absichtssperren und können in deren Rahmen lokal arbeiten.

Im Demarkationsprotokoll mit der globalen Bedingung $x+y < 100$ (im Revisionsmodell etwa vergleichbar mit der globalen Widerspruchsfreiheit), die durch die beiden lokalen Bedingungen $x < 50 \wedge y < 50$ (zum Beispiel zwei Absichtssperren) ersetzt wurden, entspricht dieses Vorgehen der Abschwächung einer der beiden Bedingungen wie zum Beispiel $x < 30$ (also etwa der Freigabe einer der beiden Absichtssperren). Die andere Bedingung ist unabhängig von dieser Änderung korrekt und braucht daher von der Änderung auch nicht unterrichtet zu werden.

Erst wenn in Abbildung 8.7 das Replikat R_5 selbst seine Absichtssperren ändern will, muß R_5 ein gültiges Quorum suchen und im Rahmen dieser Quorumsbildung müssen dann alle beteiligten Replikate abgeglichen werden. Beim Quorum Q_2 in Abbildung 8.7 erhalten dann die Replikate R_4 und R_5 die Information über die bis dahin durchgeführten globalen Operationen und können sich diesen Informationen anpassen. Danach kann R_5 wie gehabt die globale Operation ausführen und damit eine seiner Absichtssperren ändern.

Definition 8.5.2 (Das asynchrone Abkopplungsprotokoll)
Das asynchrone Abkopplungsprotokoll ist definiert wie folgt:

- *Lokale Operationen können wie im synchronen Abkopplungsprotokoll ausgeführt werden.*
- *Globale Operationen müssen in einem gültigen Quorum synchron ausgeführt werden. Dabei werden alle am Quorum beteiligten Replikate vor der Ausführung aktualisiert.*
- *Für globale Operationen gelten dieselben Voraussetzungen wie im synchronen Abkopplungsprotokoll.*

□

8.5.2 Korrektheit des asynchronen Abkopplungsprotokolls

Auch das asynchrone Abkopplungsprotokoll soll in erster Linie die Widerspruchsfreiheit der an den Replikaten eingebrachten Entwurfsentscheidungen garantieren. Weitere Garantien, wie zum Beispiel die Beständigkeit der Spezifikationen beziehungsweise der Entwurfsentscheidungen, sind hier nicht mehr zu betrachten, da diese nicht abkopplungsspezifisch sind. Um zu zeigen, daß das asynchrone Abkopplungsprotokoll die Widerspruchsfreiheit des globalen Peeks garantiert, sind zwei Sätze nötig, die im Folgenden vorgestellt und bewiesen werden. Aufbauend auf diesen zwei Sätzen gilt der Satz 8.3.1, der hier nicht nochmal aufgeführt werden soll.

Satz 8.5.1 (Reihenfolge der globalen Operationen)
Alle globalen Operationen treffen an allen Replikaten in derselben Reihenfolge ein. □

Beweis 8.5.1 (Reihenfolge der globalen Operationen)
Das asynchrone Abkopplungsprotokoll generiert eine Sequenz von gültigen Quoren für die Ausführung von globalen Operationen. Da in jedem gültigen Quorum mindestens ein Replikat auch Teil des direkt vorangegangenen Quorums war, ist jedem Quorum der Zustand des vorangegangenen Quorums bekannt. Da aber vor einer globalen Operation erst ein Abgleich durchgeführt wird, sind allen am Quorum beteiligten Replikaten alle „alten" globalen Operationen bekannt, bevor die neue ausgeführt wird.

Replikate, die an den letzten Quoren nicht beteiligt waren erhalten somit bei der nächsten Beteiligung an einem Quorum zuerst alle dem aktuellsten am Quorum beteiligten Replikat bekannten globalen Operationen, die sie noch nicht kennen, in der richtigen Reihenfolge.

□

Da gültige Quoren nur direkt aus dem letzten gültigen Quorum hervorgehen können, ensteht hier eine Folge von Quoren, wobei jedes neue Quorum direkt aus dem aktuellsten Quorum hervorgeht. Damit sind in jedem Quorum alle bis dahin ausgeführten globalen Operationen bekannt. Jedes Replikat, welches eine globale Operation ausführt, kennt also alle zuvor gelaufenen globalen Operationen und damit auch die aktuellen Absichtssperren aller anderen Replikate.

Satz 8.5.2 (Kompatibilität der Absichtssperren)
Zu jedem Zeitpunkt sind alle Absichtssperren kompatibel, das heißt, die Formeln $A_{8.1}$, $A_{8.2}$ und $A_{8.3}$ sind erfüllt.
□

Beweis 8.5.2 (Kompatibilität der Absichtssperren)
Da jedes Replikat, welches eine globale Operation initiiert, ein aktuelles gültiges Quorum benötigt, werden vor dieser globalen Operation alle bisher gelaufenen globalen Operationen ausgetauscht. Damit sind dem Replikat auch alle Absichtssperren der anderen Replikate bekannt und im Rahmen der Ausführung der globalen Operation wird überprüft, ob die genannten Formeln erfüllt bleiben.
□

Mit der Kompatibilität der Absichtssperren und der unveränderten lokalen Restriktion der einzelnen Operationen folgt direkt die Korrektheit des asynchronen Abkopplungsprotokolls nach Satz 8.3.1.

8.5.3 Quorumsdefinition für die asynchrone Abkopplung

In hochgradig abgekoppelten Netzen ist es durchaus möglich, daß meistens mehr als die Hälfte der Replikate abgekoppelt sind. Damit können nach der bisherigen Festsetzung von gültigen Quoren (bisher mehr als die Hälfte der existierenden Replikate) globale Operationen eher selten ausgeführt werden. Es ist also eine weniger restriktive Durchsetzung der Gültigkeit von Quoren wünschenswert, die trotzdem noch Definition 8.5.1 erfüllt. Für die Arbeit wird ein sogenanntes *dynamisches Quorenverfahren* propagiert, in dem für ein gültiges Quorum immer mehr als die Hälfte der Replikate des letzten gültigen Quorums nötig sind.

Damit können Quoren im Lauf der Zeit immer kleiner werden bis hin zu zweielementigen Quoren. Danach ist eine weitere Aufspaltung in einelementige Quoren nicht mehr möglich, da sonst aus einem zweielementigen Quorum zwei einelementige Quoren parallel hervorgehen könnten. Durch die Wiederankopplung können neue Replikate zu Quoren hinzustoßen, so daß sich die Quorumsgröße auch wieder erhöhen kann. Durch diese Festsetzung ist garantiert, daß aus jedem Quorum höchstens genau ein neues Quorum hervorgeht und daß aufeinanderfolgende Quoren nicht disjunkt sind. Damit können selbst wenn fast alle Replikate abgekoppelt sind von den angekoppelten Replikaten noch globale Operationen ausgeführt werden.

Beispiel 8.5.1 (Folgen gültiger Quoren für die asynchrone Abkopplung)
Für die konservative Durchsetzung der Gültigkeit von Quoren genügt schon ein Blick auf Abbildung 8.7. Damit ist ersichtlich, daß keine zwei disjunkten Quoren zustande kommen können.

Etwa komplizierter ist dieses Verhalten bei der vorgeschlagenen alternativen Durchsetzung von gültigen Quoren durch das dynamische Quorenverfahren. In Abbildung 8.8 sind vier aufeinanderfolgende gültige Quoren zustande gekommen. Zweimal wurden Quorenbildungen versucht, die jedoch abgelehnt wurden, weil die notwendige Bedingung nicht erfüllt war.

Damit ergibt sich durch beide Quorenverfahren eine serielle Folge von gültigen Quoren (in Abbildung 8.8 sind es die Quoren Q_1, Q_2, Q_3 und Q_4). Wenn nun ein Replikat eine Absichtssperre modifizieren will, dann muß dies mit einer globalen Operation in einem

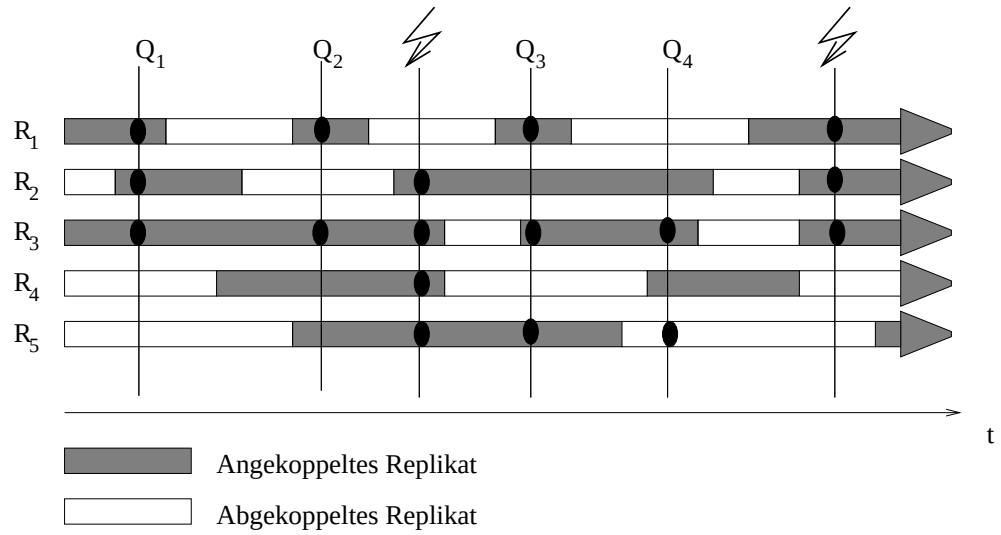


Abbildung 8.8: Eine flexiblere Durchsetzung von gültigen Quoren

neuen Quorum durchgeführt werden. Durch die Gültigkeit aller vorangegangenen Quoren und die daraus resultierende serielle Folge ist gewährleistet, daß in dem nächsten gültigen Quorum Q_5 alle bisherigen globalen Operationen und damit alle Absichtssperren bekannt sind. Alle an Q_5 beteiligten Replikate kennen also die Absichtssperren der einzelnen Replikate und können damit entscheiden, ob die neue globale Operation zugelassen wird oder nicht. □

8.6 Allgemeines Demarkationsprotokoll

Das bisherige erweiterte Demarkationsprotokoll verwaltet für jedes Replikat eines Datenelements genau eine Absichtssperre. In diesem Abschnitt wird nun ein verallgemeinertes Demarkationsprotokoll vorgestellt, welches an jedem Replikat eines Datenelements eine beliebige Anzahl von Absichtssperren erlaubt. Hierzu wird davon ausgegangen, daß initial alle Replikate frei von Absichtssperren sind. Jedes Replikat kann dann auf ein Datenelement eine Absichtssperre mit den bekannten globalen Operationen und einem gültigen Quorum anfordern. Diese Anfrage ist erfolgreich, wenn die zu setzende Absichtssperre und alle zu diesem Zeitpunkt gesetzten Absichtssperren kompatibel sind. Auf einem Replikat dürfen nach wie vor nur Spezifikationen eingebracht werden, die durch Absichtssperren abgesichert sind.

Im *allgemeinen Demarkationsprotokoll* oder *AD-Protokoll* gibt es die folgenden vier Operationen, die von den Entwicklern als Teil ihrer Entwicklungsarbeiten aufgerufen werden.

setX **($\mathcal{X}()$, X)**
Die Operation *setX* setzt eine X-Sperre auf einem Datenelement falls folgende Voraussetzungen erfüllt sind:

1. Kein Replikat hält eine X-Sperre auf X

$$\mathbb{X}(X) = \emptyset$$

2. Die X-Sperre impliziert alle gesetzten S-Sperren

$$\forall_{\mathcal{Y}() \in \mathbb{Y}(X)} \; \mathcal{X}() \Rightarrow \mathcal{Y}()$$

3. Die X-Sperre ist erfüllbar
$$Sat(\mathcal{X}())$$

Der Code für die Überprüfung der drei Bedingungen sieht wie folgt aus:

```
Wenn keine globale Operation ausgeführt werden kann
  FEHLER: Kein Quorum
End
REMARK: Hier muß unter Umständen auch eine Aktualisierung des
                                              Replikats erfolgen
REMARK: Prüfe Bedingung 1
Wenn eine X-Sperre auf X gesetzt ist
  FEHLER: kann nicht gesetzt werden
End
REMARK: Prüfe Bedingung 2
Berechne die Konjunktion aller S-Sperren
Wenn die X-Sperre diese Konjunktion nicht impliziert
  FEHLER: kann nicht gesetzt werden
End
REMARK: Prüfe Bedingung 3
Wenn die X-Sperre nicht erfüllbar ist
  FEHLER: kann nicht gesetzt werden
End
Setze die Sperre
FERTIG
```

setS **($\mathcal{Y}()$, X)**

Die Operation *setS* setzt die S-Sperre $\mathcal{Y}()$ auf dem Datenelement X, falls folgende Voraussetzungen erfüllt sind.

1. Eine mögliche gesetzte X-Sperre impliziert die S-Sperre
$$\forall_{\mathcal{X}() \in \mathbb{X}(X)} \mathcal{X}() \Rightarrow \mathcal{Y}()$$

2. Die S-Sperre ist mit allen aktiven S-Sperren und dem lokalen Peek kompatibel
$$Sat(\mathcal{Y}() \wedge \mathcal{P}_X() \wedge \bigwedge_{\mathcal{Y}() \in \mathbb{Y}(X)} \mathcal{Y}())$$

Der Code für die Überprüfung dieser Bedingungen sieht wie folgt aus:

```
Wenn keine globale Operation ausgeführt werden kann
  FEHLER: Kein Quorum
End
REMARK: Hier muß unter Umständen auch eine Aktualisierung des
                                              Replikats erfolgen
REMARK: Prüfe Bedingung 1
Wenn eine X-Sperre auf X gesetzt ist
  Wenn die X-Sperre die S-Sperre nicht impliziert
    FEHLER: kann nicht gesetzt werden
  End
End
REMARK: Prüfe Bedingung 2
Berechne den lokalen Peek
Berechne die Konjunktion aller S-Sperren
Wenn die Konjunktion dieser Konjunktion und des Peeks nicht
                                                 erfüllbar ist
  FEHLER: kann nicht gesetzt werden
End
Setze die Sperre in einer globalen Operation
FERTIG
```

releaseX **($\mathcal{X}()$, X) und** ***releaseS*** **($\mathcal{Y}()$, X)**

Die Operationen *releaseX* und *releaseS* geben die zuvor auf dem Datenelement X gesetzten X- beziehungsweise S-Sperre wieder frei. Die Operationen dürfen nur auf dem Replikat ausgeführt werden, auf dem die Sperre auch angefordert wurde. Die Globalisierung der Entwurfsentscheidungen funktioniert dabei genau wie bei der zuvor behandelten Abschwächung von Absichtssperren

```
Markiere die Absichtssperre als freizugeben
Wenn keine globale Operation ausgeführt werden kann
  FEHLER: Kein Quorum
End
REMARK: Hier muß unter Umständen auch eine Aktualisierung des
                                                  Replikats erfolgen
Für alle Entwurfsentscheidungen, die durch die Absichtssperre
                                                       maskiert sind
  Wenn die Entscheidung aktiv ist
    FERTIG
  End
End
L = leere Liste von Entwurfsentscheidungen
Für alle Entwurfsentscheidungen, die durch die Absichtssperre
                                                       maskiert sind
  Wenn die Entwurfsentscheidung noch nicht globalisiert wurde
    Markiere die Entwurfsentscheidung als globalisiert
    Füge die Entwurfsentscheidung zu L hinzu
  End
End
Gebe die Absichtssperre in einer globalen Operation frei und
                                                  globalisiere die
                                      Entwurfsentscheidungen in L
FERTIG
```

Über diese vier neuen Operationen hinaus sind noch kleinere Änderungen an den bestehenden Operationen aus den Kapiteln 6 und 7 nötig. Einerseits muß die *constrain*-Operation (wie auch *premise* und *assume*) eine Maskierung der einzubringenden Spezifikation durch Absichtssperren finden und andererseits muß beim Abschluß einer Entwurfsentscheidung überprüft werden, ob dadurch Absichtssperren freizugeben sind.

constrain, ***premise*** **und** ***assume***

Beim Einbringen von Spezifikationen ist die Maskierung durch Absichtssperren zu überprüfen. Dazu ist folgendes Codefragment in *constrain*, *premise* und *assume* zu integrieren:

```
REMARK: Möglichkeit 1: Einbringen durch S-Sperre
Für alle lokalen S-Sperren des Datenelements
  Impliziert die S-Sperre die einzubringende Spezifikation
    Verbinde die Spezifikation mit der S-Sperre
    Verbinde die Entwurfsentscheidung mit der S-Sperre
    FERTIG
  End
End
REMARK: Möglichkeit 2: Einbringen durch X-Sperre
Gibt es eine lokale X-Sperren auf dem Datenelements
Wenn ja
  Berechne Kombination aus Spezifikation, X-Sperre und Peek
  Wenn Kombination erfüllbar
    Verbinde die Spezifikation mit der X-Sperre
    Verbinde die Entwurfsentscheidung mit der X-Sperre
    FERTIG
  End
End
FEHLER: keine ausreichende Absichtssperre
```

commit* beziehungsweise *abort

Nach dem Abschluß (*commit* beziehungsweise *abort*) ist folgendes Codefragment zu durchlaufen:

```
Für alle Absichtssperren, die die abzuschließende Entwurfs-
                                        entscheidung maskiert haben
  Wenn die Absichtssperre als freizugeben markiert ist
    Rufe auf dieser Absichtssperre releaseX oder releaseS auf.
  End
End
FERTIG
```

Da die von den Operationen *setX*, *setS*, *releaseX* und *releaseS* durchgesetzten Bedingungen dieselben bleiben, kann der Satz 8.3.1 mit dem zugehörigen Beweis direkt übernommen werden, soll hier aber nicht nochmal aufgeführt werden. Es bleibt aber festzuhalten, daß das allgemeine Demarkationsprotokoll ebenfalls die Widerspruchsfreiheit des globalen Peeks garantiert.

Ein repliziertes Datenelement kann damit gar keine oder genau eine X-Sperre, aber beliebig viele S-Sperren haben. Besteht an einem Replikat keine Absichtssperre, dann können keine Spezifikationen eingebracht werden. Bestehen eine oder mehrere Absichtssperren, dann können die Spezifikationen wie im Abschnitt 8.2.2 gesetzt werden. Darüber hinaus kann auch die Einbringung einer Spezifikation im Rahmen mehrerer S-Sperren erlaubt werden. Diese Möglichkeit soll aber hier nicht weiter betrachtet werden, da statt den mehreren S-Sperren immer auch die Konjunktion dieser S-Sperren selbst gesetzt werden kann. Das heißt nicht, daß nicht mehrere S-Sperren gesetzt werden können. Nur für jede einzelne einzubringende Spezifikation darf nur eine S-Sperre verwendet werden.

Im Folgenden wird von dem allgemeinen Demarkationsprotokoll ausgegangen, da sich die beiden einfacheren Varianten darauf abbilden lassen.

8.7 Abkopplung in der Praxis

Die Unterschiede des abgekoppelten Modells zum angekoppelten Modell aus Kapitel 7 besteht in erster Linie in den Absichtssperren, die lokal an jedem Replikat verwaltet werden müssen. Hinzu kommt, daß für die Operation *constrain* zusätzliche Bedingungen gelten, nämlich daß die einzubringende Spezifikation durch lokale Absichtssperren maskiert ist.

Die globalen Operationen können zweigeteilt betrachtet werden. Einerseits muß jedes Replikat lokal entscheiden, ob eine Absichtssperre gesetzt oder freigegeben werden darf. Dies ist möglich, da jedes Replikat die Absichtssperren der anderen Replikate kennt und somit die Kompatibilität der Absichtssperren überprüfen kann. Darüber hinaus muß dann aber auch die Kommunikation zwischen den Replikaten unterstützt werden. Dies ist weniger ein reines Kommunikationsproblem. Vielmehr geht es darum, die globale Reihenfolge von Absichtssperroperationen durch das beschriebene Quorenverfahren durchzusetzen.

Für die Umsetzung des Modells und die Performanz sind alle aufgeführten Unterschiede zu betrachten, wobei für den Benutzer und damit die Anwendung nur die Absichtssperren interessant sind. Die Kommunikation mit anderen Replikaten soll ja für die Entwickler gerade transparent ablaufen.

8.7.1 Technische Umsetzung des abgekoppelten RV-Modells

Die Verwaltung der Absichtssperren an einem Datenelement ist fast schon trivial. Da es immer höchsten eine X-Sperre geben kann, kann diese explizit gespeichert werden,

während die S-Sperren in einer Hash-Tabelle verwaltet werden, damit der direkte Zugriff möglichst schnell geht.

Für die Implementierung von *constrain* muß geprüft werden, ob die einzubringende Spezifikation durch eine lokale X- oder S-Sperre abgesichert ist. Diese Prüfung ist für Intervalle sehr einfach durchzuführen. Danach wird noch die übliche Kompatibilität mit dem lokalen Peek überprüft. Ist auch diese erfolgreich, dann kann die neue Spezifikation eingebracht werden.

Der lokale Teil der Operationen zum Ändern der Absichtssperren muß weiter aufgeschlüsselt werden:

setX
Hierzu ist folgende Bedingung zu überprüfen:

$$|\mathbb{X}| = 0 \quad \wedge \quad \mathcal{X}() \Rightarrow \bigwedge_{\mathcal{Y}() \in \mathbb{Y}} \mathcal{Y}()$$

setS Hierzu sind folgende Bedingungen zu überprüfen:

$$Sat(\mathcal{G}() \wedge \bigwedge_{\mathcal{Y}() \in \mathbb{Y}} \mathcal{Y}()) \quad \wedge \quad \forall_{\mathcal{X}() \in \mathbb{X}} \mathcal{X}() \Rightarrow \bigwedge_{\mathcal{Y}() \in \mathbb{Y}} \mathcal{Y}()$$

releaseX* und *releaseS
Hierzu sind keine Bedingungen zu überprüfen. Allerdings müssen alle Entwurfsentscheidungen, von denen mindestens eine Spezifikation im Rahmen der abzuschwächenden Absichtssperre eingebracht wurde, bereits abgeschlossen sein und - falls dies nicht schon erledigt wurde - mit der Abschwächung an die anderen Replikate verteilt werden.

Der zweite Teil der globalen Operationen ist die Kommunikation mit anderen Replikaten und damit verbunden die Quorenbildung. Die Kommunikation selbst stellt dabei kein Problem dar. Für die Quorenbildung müssen alle Replikate des letzten Quorums angefragt werden. Wenn mehr als die Hälfte dieser Replikate bereit sind, das neue Quorum zu bilden, dann kommt dieses zustande.

8.7.2 Theoretische Performanz und Skalierbarkeit des abgekoppelten RV-Modells

Die *constrain*-Operation behält den konstanten Aufwand des RV-Modells. Allerdings ist zusätzlich die Maskierung durch Absichtssperren zu überprüfen. Für X-Sperren ist diese Überprüfung konstant, für S-Sperren ist sie allerdings linear. Durch die Verwaltung der Konjunktion aller S-Sperren kann dieser lineare Aufwand für die *constrain*-Operation vermieden werden. Die Neuberechnung dieser Konjunktion (mit linearem Aufwand) wird dann aber bei der Freigabe einer S-Sperre nötig.

Da sich die anderen lokalen Operation *begin*, *abort*, *commit*, *peek* und *undo* nicht ändern, kann lokal mit vernünftigem Aufwand gearbeitet werden. Damit bleibt also noch die Diskussion der Absichtssperroperationen. Hierzu müssen einerseits beim Setzen oder Freigeben von Absichtssperren die zugehörigen Entwurfsentscheidungen abgeschlossen sein und - falls dies noch nicht erledigt wurde - mit der globalen Operation an die anderen Replikate weitergereicht werden. Der Aufwand hierfür ist linear in der Anzahl der Spezifikationen pro Absichtssperre, da für jede Spezifikation die entsprechende Entwurfsentscheidung auf Vollständigkeit hin untersucht werden muß. Für die Kommunikation spielt auch die Entwurfsentscheidung selbst eine Rolle, so daß der Aufwand insgesamt das Produkt der Spezifikationsanzahl pro Absichtssperre mit der Größe der Entwurfsentscheidungen, also quadratisch ist.

Andererseits sind für das Setzen von Absichtssperren die drei Bedingungen zu überprüfen. Der Aufwand hierzu ist wie folgt:

$|\mathbb{X}| \leq 1$
: Diese Prüfung ist trivial und in konstanter Zeit durchführbar.

$\forall_{\mathcal{X}() \in \mathbb{X}}\ \mathcal{X}() \Rightarrow \bigwedge_{\mathcal{Y}() \in \mathbb{Y}}\ \mathcal{Y}()$
: Der Aufwand für die Prüfung dieser Bedingung ist linear in der Anzahl der S-Sperren oder konstant, wenn die Konjunktion aller S-Sperren verwaltet wird.

$Sat(\mathcal{G}() \wedge \bigwedge_{\mathcal{Y}() \in \mathbb{Y}}\ \mathcal{Y}())$
: Auch hier ist der Aufwand konstant, wenn $\mathcal{G}()$ und die Konjunktion aller S-Sperren verwaltet werden.

Der Aufwand für die Bildung eines gültigen Quorums ist linear in der Anzahl der beteiligten Replikate. Etwas mehr Aufwand ist für die Aktualisierung veralteter Replikate zu investieren, da diese eine Reihe von globalen Operationen nacharbeiten müssen, wobei eine globale Operation (im Fall der Freigabe einer Absichtssperre) schon quadratischen Aufwand hat.

8.7.3 Anwendung des abgekoppelten RV-Modells

Im Wesentlichen bleibt die Anwendung des abgekoppelten Modells im Vergleich zum angekoppelten Modell unverändert. Nur die zusätzlichen Absichtssperren müssen vor den Arbeiten angepaßt werden. Dazu ist ein gültiges Quorum erforderlich, das aus der Sicht des Entwicklers üblicherweise nur im angekoppelten Betrieb erreicht werden kann, so daß sämtliche Absichtssperren vor der Abkopplung zu setzen sind.

Der Entwickler braucht dafür eine grobe Vorahnung, welche Arbeiten während der Abkopplung auszuführen sind. Dabei muß unterschieden werden zwischen fixierenden und verfeinernden Zugriffen. Fixierende Zugriffe sind solche, die eine schon bestehende Spezifikation als gültige Voraussetzung benutzen und selbst wieder in Form einer neuen Spezifikation einbringen. Die neue Spezifikation engt die noch möglichen Produkte nicht weiter ein, verhindert aber, daß durch Rücksetzen anderer Entwurfsentscheidungen die benötigten Voraussetzungen invalidiert werden. Für solche fixierende Zugriffe ist eine S-Sperre mit dem aktuellen Peek als Spezifikation sinnvoll. Damit können andere Entwickler im Rahmen einer gesetzten X-Sperre immer noch verfeinern, während der abgekoppelte Entwickler im Rahmen seiner S-Sperre die Voraussetzungen wieder einbringen (also *premise*-Operationen ausführen) kann. Im Gegensatz dazu sind für die Ergebnisse von Entwurfsentscheidungen in der Regel X-Sperren nötig, da mit Ergebnissen üblicherweise neues Wissen in die Datenbasis eingebracht wird, also die neuen Spezifikationen verfeinernd sind. Die Menge der akzeptablen Produkte wird also weiter eingeschränkt. Aufgrund der Exklusivität von X-Sperren können damit auf einem Datenelement immer nur auf einem Replikat neue Ergebnisse eingebracht werden.

In der Praxis werden die Entwickler hierfür die Einteilung des gesuchten Produkts in Teilbereiche nutzen. Ein Entwickler wird üblicherweise auf seinem Teilbereich X-Sperren setzen, da nur dort neue Ergebnisse zu erwarten sind. Auf den anderen Teilbereichen wird der Entwickler aber durchaus S-Sperren setzen, da die dortigen Ergebnisse oft als Voraussetzungen verwendet werden.

Beispiel 8.7.1 (Praktische Verwendung von Absichtssperren)
Der Robotergreifer verwendet drei unterschiedliche Wirkprinzipien: Druck-Kraft-Wandlung, Kraft verteilen und Kraft verstärken. Diese drei Wirkprinzipien sind am Greifer selbst in unterschiedlichen Bereichen zu erkennen. So zeigt Abbildung 8.9 den Robotergreifer in einem 2-D-Schnitt. In diesem Schnitt sind zwei Bereiche den Wirkprinzipien zugeordnet. Der dritte Bereich in der Mitte entspricht dann noch dem Prinzip Kraft-Verteilen.

Mit der Konstruktion der Greiferbacken wird das Wirkprinzip Kraft-Verstärken realisiert. Der zuständige Konstrukteur muß dafür den Bereich 2 verfeinern, da dort

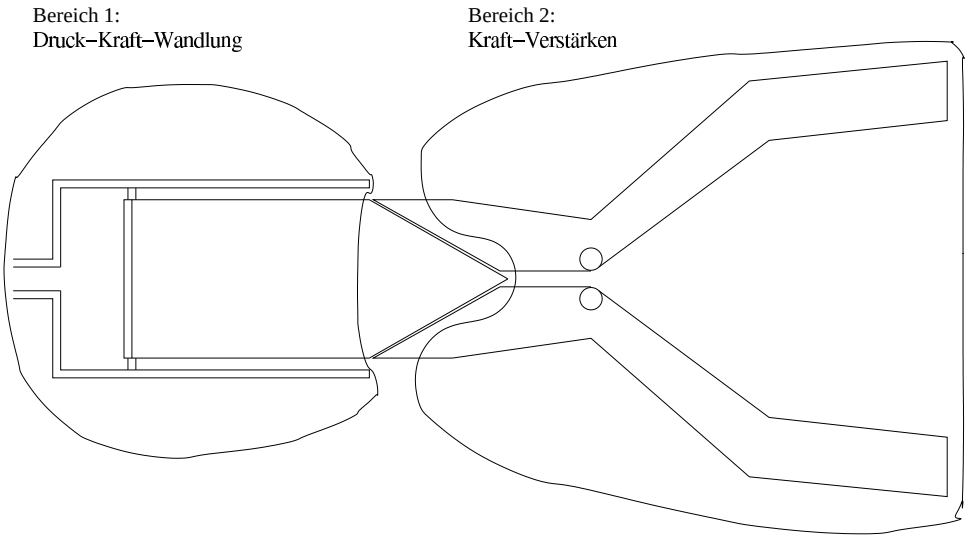


Abbildung 8.9: Die praktische Anwendung von Absichtssperren am Robotergreifer

tatsächlich neue Entwicklungsarbeit von ihm geleistet wird. Auf den anderen Bereichen, wie zum Beispiel dem Bereich 1 „liest" der Konstrukteur nur, da der dort in Kraft umgewandelte und dann verstärkte Druck die Voraussetzung für die Greiferbackengeometrie darstellt.

Der Greiferbackenentwickler muß also auf seinem Bereich X-Sperren setzen, um weitere Verfeinerungen vornehmen zu können. Auf den anderen Bereichen müssen teilweise noch S-Sperren gesetzt werden, da von dort die Voraussetzungen in Form von premise-*Operationen einfließen.*

Auf der anderen Seite wird der für die Druck-Kraft-Wandlung zuständige Entwickler im Bereich 1 X-Sperren setzen, um dort verfeinernd arbeiten zu können. Dieser Entwickler wird allerdings in den anderen Bereichen S-Sperren setzen, falls er Voraussetzungen aus diesen Bereichen hat.

□

Kapitel 9

Erweiterungen des Revisionsmodells

Die unterschiedlichen Varianten des Revisionsmodells für Arbeitsteiligkeit und Abkopplung unterstützen einen oder mehrere Entwickler bei der sukzessiven Verfeinerung einer Produktbeschreibung. Allerdings gibt es zum Teil einige Einschränkungen, wie zum Beispiel der Pessimismus bei der Abkopplung, der zwar aus der Beständigkeit resultiert, unter Umständen aber trotzdem zu restriktiv ist.

Darüber hinaus können Entwickler durch zusätzliche Erweiterungen noch besser unterstützt werden. Hier kann zum Beispiel die temporäre Verwaltung von Inkonsistenzen genannt werden, die einen Konflikt nicht sofort beseitigt, etwa wenn einer der beteiligten Entwickler gerade nicht erreichbar ist.

Einige weitere sinnvolle Erweiterungen des RV-Modells erleichtern eher den praktischen Umgang mit dem RV-Modell, zum Beispiel das automatische Setzen und Freigeben von Absichtssperren, mit dem Entwickler deutlich entlastet werden können.

9.1 Inkonsistenz im RV-Modell

Das Ziel des Produktentwicklungsprozesses ist genau ein gesuchtes Produkt, das die Zustimmung aller beteiligten Entwickler erhält. Dieses Produkt muß also allen Spezifikationen gerecht werden, was die Widerspruchsfreiheit am Ende des Entwicklungsprozesses unverzichtbar macht. Allerdings kann temporär doch die Verwaltung von Inkonsistenz sinnvoll sein, zum Beispiel wenn Konflikte nicht sofort aufgelöst werden können, etwa wegen Abwesenheit von Beteiligten oder wegen fehlenden Kontextinformationen. Diese Möglichkeit betrifft schon das RV-Modell aus Kapitel 6 und damit auch dessen Erweiterungen.

Das Revisionsmodell kann hervorragend mit den im Entwicklungsprozeß vorhandenen Freiräumen umgehen, diese zur Steigerung der Parallelität nutzen und im Lauf des Entwicklungsprozesses hin zum fertigen Produkt minimieren. Im Rahmen diese Freiräume können kleinere Inkonsistenzen im Sinn von unterschiedlichen Werten mit geringen Abweichungen verdeckt werden, so lange es mindestens eine im Sinn des Revisionsmodells „konsistente“ (widerspruchsfreie) Lösung gibt. Dies ist zum Beispiel dann der Fall, wenn Entwickler ihre Ergebnisse mit Freiräumen versehen und sich die Freiräume der Ergebnisse überschneiden. Werden diese Freiräume im Lauf der Entwicklung bei gleichbleibenden Ergebnissen verkleinert, dann werden die Freiräume irgendwann disjunkt und

sind damit nicht mehr widerspruchsfrei. An dieser Stelle werden die beteiligten Entwickler benachrichtigt und müssen ihre Ergebnisse so weit anpassen, daß diese wieder widerspruchsfrei sind.

Beispiel 9.1.1 (Erlaubte Inkonsistenzen im Revisionsmodell)
Im Fall des Robotergreifers können zum Beispiel die Anforderungen an die Haltekraft mit Freiräumen versehen sein (zum Beispiel $[90, 110]N$*). Da der zu entwickelnde Greifer aber erst langsam konkretisiert wird, sind zwischenzeitlich die Ergebnisse für die zu erreichende Haltekraft ebenfalls mit Freiräumen versehen (zum Beispiel* $85N$ *mit einer Ungenauigkeit von* $10N$*). Die voraussichtlich erreichten* $85N$ *sind nicht akzeptabel, aber durch den Freiraum von* $10N$ *ist die Datenbasis noch widerspruchsfrei. Erst wenn die tatsächliche Haltekraft präzisiert wird (zum Beispiel auf genau* $87N$*), dann tritt der Widerspruch zu Tage und muß bereinigt werden.*
□

Durch sehr einfache Erweiterungen kann allerdings auch das Revisionsmodell Inkonsistenzen in Form von momentan nicht vereinbaren Spezifikationen speichern. Da dadurch aber weder die Widerspruchsfreiheit als zentrale Konsistenzbedingung noch die Beständigkeit von Spezifikationen beziehungsweise Entwurfsentscheidungen beeinträchtigt werden darf, können die Inkonsistenzen nur parallel verwaltet werden. Hierzu wird mit jedem Datenelement X eine zweite Menge von Spezifikationen X' verwaltet, die alle derzeit nicht einbringbaren weil widersprüchlichen Spezifikationen beinhaltet.

Spezifikationen in X' haben keinen Einfluß auf die weitere Entwicklung. Damit werden diese Spezifikationen auch nicht von nachfolgenden Entwurfsentscheidungen als Voraussetzungen verwendet und nehmen somit nicht an der Liest-von-Beziehung oder dem Entwicklungsgraph teil. Der Sinn von X' liegt mehr darin, daß diese inkonsistenten Spezifikationen abgespeichert werden können, zum Beispiel bis der Konflikt aufgelöst wird.

Entwickler können nun Spezifikationen auf drei verschiedene Arten einbringen:

Widerspruchsfreie Spezifikationen
Wenn eine *constrain*-Operation eines Entwicklers widerspruchsfrei ausgeführt werden kann, dann wird die Spezifikation auf jeden Fall mit der vollen Beständigkeit in das System eingebracht.

Inkonsistente aber trotzdem durchzusetzende Spezifikationen
Wenn eine *constrain*-Operation fehlschlägt, dann kann die widersprüchliche Spezifikation in X' eingebracht werden. Bei dieser Spezifikation wird vermerkt, daß sie auf jeden Fall zum Ende der Produktentwicklung erfüllt sein muß. Die von dieser Spezifikation verursachte Inkonsistenz muß also auf jeden Fall beseitigt werden.

Inkonsistente und wünschenswerte Spezifikationen
Inkonsistente und damit nicht einbringbare Spezifikationen können auch weniger wichtige Informationen über das Produkt beinhalten, die der Entwickler zwar gerne erfüllt sehen würde („Nice to have“), deren Nichterfüllung aber auch kein echtes Problem darstellt. Auch solchen Spezifikationen können in X' eingebracht werden. Die von solchen Spezifikationen verursachten Inkonsistenzen müssen aber nicht unbedingt beseitigt werden.

Werden per *undo*-Operation Spezifikationen aus der Datenbasis freigegeben, so kann überprüft werden, ob einige der inkonsistenten Spezifikationen nachrücken können. Hierbei kann das System in verschiedenen Freiheitsgraden konfiguriert werden, so zum Beispiel ob Spezifikationen überhaupt nachrücken sollen und wenn ja, welche Spezifikationen nachrücken können. Ein Algorithmus hierfür kann wie folgt aussehen (automatisch aufzurufen nach einer *undo*-Operation auf X):

```
Für alle durchzusetzenden Spezifikationen aus X'
  Bringe die Spezifikation per constrain ein
  Wenn erfolgreich
    Lösche die Spezifikation aus X'
  End
End
Für alle wünschenswerten Spezifikationen aus X'
  Bringe die Spezifikation per constrain ein
  Wenn erfolgreich
    Lösche die Spezifikation aus X'
  End
End
FERTIG
```

Der Aufwand hierfür ist offensichtlich linear in der Anzahl der inkonsistenten Spezifikationen. Durch dieses Nachrücken wird auch sichergestellt, daß jede einzelne Spezifikation aus X' immer widersprüchlich mit dem aktuellen Peek ist. Dies kann sehr einfach gezeigt werden, da Spezifikationen nur in X' eingebracht werden, wenn sie nicht in X eingebracht werden können, also widersprüchlich mit dem aktuellen Peek sind. Wenn der Peek durch eine *constrain*-Operation verändert wird, dann wird dadurch sicherlich keine Spezifikation aus X' kompatibel mit dem Peek. Die umgekehrte Freigabe einer Spezifikation aus X verursacht das angeführte Nachrückverfahren, nach dessen Durchführung wieder alle Spezifikationen in X' inkompatibel mit dem Peek sind, da sie sonst nachgerückt wären.

Spätestens mit dem Abschluß der Entwicklung ist zu überprüfen, ob in X' noch Spezifikationen enthalten sind, deren Erfüllung unbedingt notwendig ist. Wenn ja, dann müssen die an dem Konflikt beteiligten Entwickler darauf hingewiesen werden, so daß diese den Konflikt beseitigen und die Spezifikation einbringen können. Da die Beseitigung dieser Konflikte aber durchaus aufwendige Rückschritte in der Entwicklung bedeuten, sollten sie so früh wie möglich durchgeführt werden. Daher ist eine regelmäßige Überprüfung von X', zumindest zu bestimmten Meilensteinen, sinnvoll, so daß die Konflikte schnell beseitigt werden können.

9.2 Optimistische Erweiterung

Diese Erweiterung durch Optimismus betrifft nur das erweiterte und allgemeine Demarkationsprotokoll für die Abkopplung. Dort können nicht in allen Fällen vor der Abkopplung eines Replikates die Absichtssperren ausreichend abgeändert werden, zum Beispiel wenn die Abkopplung unerwartet eingetroffen ist, länger dauert als erwartet oder wenn die Arbeit sehr spontan und damit nicht vorhersehbar ist. In diesen Fällen muß zwischen zwei Möglichkeiten entschieden werden:

- Keine neuen Spezifikationen können in die Datenbasis eingebracht werden. Dieses verhalten entspricht genau dem aus Kapitel 8, in dem neue Spezifikationen nur mit Absichtssperren erlaubt sind.
- Neue Spezifikationen können zwar eingebracht werden, allerdings ohne Garantie für deren Beständigkeit. Dieser Fall ist eine Erweiterung, die so im bisherigen Modell noch nicht möglich ist. Sie verletzt allerdings die Beständigkeit und muß daher mit extremer Vorsicht angewendet werden.

Da der Entwickler selbst am besten entscheiden kann, welche der beiden Möglichkeiten er wählen will, sollte eine entsprechende Meldung über die nicht zu garantierende Beständigkeit ausgegeben werden. Der Benutzer kann dann entscheiden, ob er die Arbeiten trotzdem ausführen will oder nicht. Dabei ist zu beachten, daß eine Entwurfsentscheidung durch eine optimistische und damit nicht beständige Spezifikation ihre

komplette Beständigkeit verliert, da im Falle eines Konflikts die komplette Entwurfsentscheidung zurückgesetzt werden muß.

Definition 9.2.1 (Optimismus und Pessimismus)
Eine Spezifikation heißt **pessimistisch**, *wenn sie im Rahmen einer Absichtssperre eingebracht wird und damit garantierte Beständigkeit hat. Wird eine Spezifikation ohne Absichtssperre eingebracht, dann heißt sie* **optimistisch**. *Optimistische Spezifikationen haben keine garantierte Beständigkeit.*

Eine Entwurfsentscheidung heißt **pessimistisch**, *wenn alle Spezifikationen in der Entwurfsentscheidung pessimistisch sind. eine nicht pessimistische Entwurfsentscheidung ist* **optimistisch**.
□

Wenn Spezifikationen optimistisch, also ohne Beständigkeit, eingebracht werden, so werden diese möglicherweise zu einem späteren Zeitpunkt wieder zurückgesetzt. Im Gegensatz zu den traditionellen ACID-Transaktionen entstehen hier allerdings keine Probleme wie zum Beispiel kaskadierende Abbrüche, da der Einfluß von Rücksetzung von Entwurfsentscheidungen auf nachfolgende Entwurfsentscheidungen höchstens in Form von ungültig gewordenen Voraussetzungen auftritt (siehe Kapitel 7). Aus der Sicht der Datenkonsistenz ist die fehlende Beständigkeit einzelner Entwurfsentscheidungen damit kein Problem.

Zusammenführungsalgorithmus

Um eine optimistische Entwurfsentscheidung beständig zu machen, muß diese mit allen anderen Entwurfsentscheidungen widerspruchsfrei vereinbar sein. Hierzu werden die optimistischen Entwurfsentscheidung über globale Operationen zwischen den beteiligten Replikaten ausgetauscht. Erst wenn die optimistischen Entwurfsentscheidungen alle Replikate erreicht haben, kann entschieden werden, ob sie überleben werden oder nicht. Dieses Verfahren ist aufwendig, da erst alle Replikate die Entwurfsentscheidungen erhalten müssen und danach erst deren Beständigkeit garantiert werden kann. Wenn wenige Replikate sehr lange abgekoppelt sind, dann dauert dieses Verfahren dementsprechend lange. Dabei ist es wünschenswert, die optimistischen Entwurfsentscheidungen so schnell wie möglich und mit wenig Aufwand beständig zu machen.

Beispiel 9.2.1 (Propagierung von optimistischen Entwurfsentscheidungen)
In Abbildung 9.1 sind zwei optimistische Entwurfsentscheidungen D_1 auf dem Replikat R_3 und D_2 auf dem Replikat R_4 zu sehen. Dabei sind die optimistischen Entwurfsentscheidungen D_1 und D_2 auf jeden Fall mit allen anderen Spezifikationen des Replikats, auf dem sie eingebracht wurden, widerspruchsfrei. D_1 (D_2) ist also widerspruchsfrei mit allen Spezifikationen an R_3 (R_4). Wenn D_2 auch mit den Spezifikationen an allen anderen Replikaten widerspruchsfrei ist (bis auf D_1), dann kann D_2 über das Quorum Q_2 an R_4 und R_1 weitergeleitet werden und erreicht dann durch Q_3 auch das Replikat R_2. Erst mit dem Quorum Q_4 gibt es ein Replikat, auf dem beide optimistischen Entwurfsentscheidungen bekannt sind und ein möglicher Widerspruch festgestellt wird.
□

9.2.1 Nachträgliches Setzen von Absichtssperren

Ein Ansatz zur schnelleren Garantie der Beständigkeit von optimistischen Entwurfsentscheidung ist das nachträgliche Setzen der Absichtssperren. In Abbildung 9.2 ist das pessimistische Verfahren aus Kapitel 8 und das neue optimistische Verfahren im Vergleich vorgestellt. Der große Unterschied liegt darin, daß Spezifikationen eingebracht werden können, bevor die dafür nötigen Absichtssperren gesetzt werden. Mit der nachträglichen erfolgreichen Anforderung der Absichtssperren werden die optimistischen Spezifikationen dann beständig. Wenn nachträglich keine Absichtssperre gesetzt werden kann,

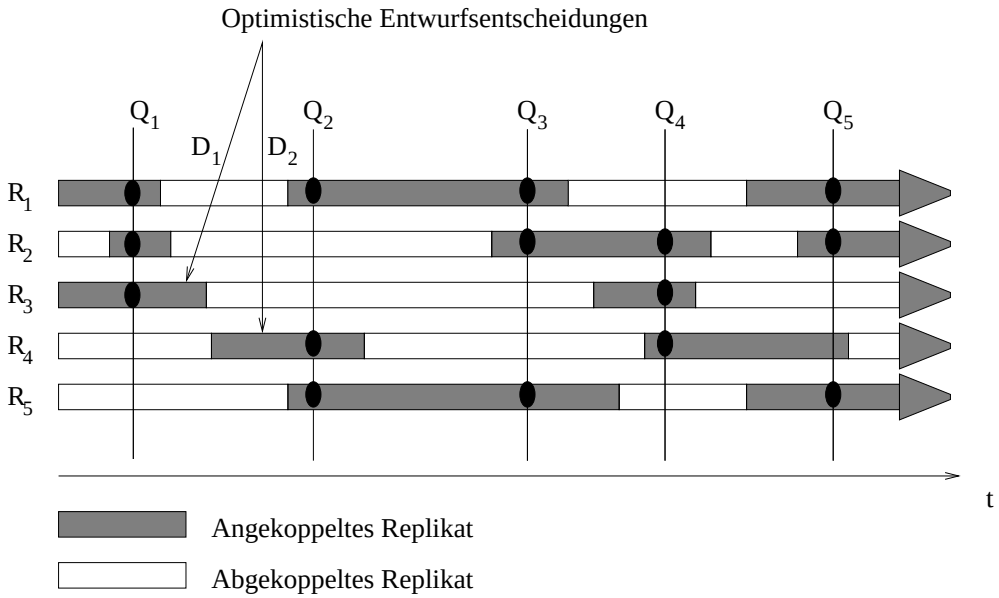


Abbildung 9.1: Propagierung von optimistischen Entwurfsentscheidungen

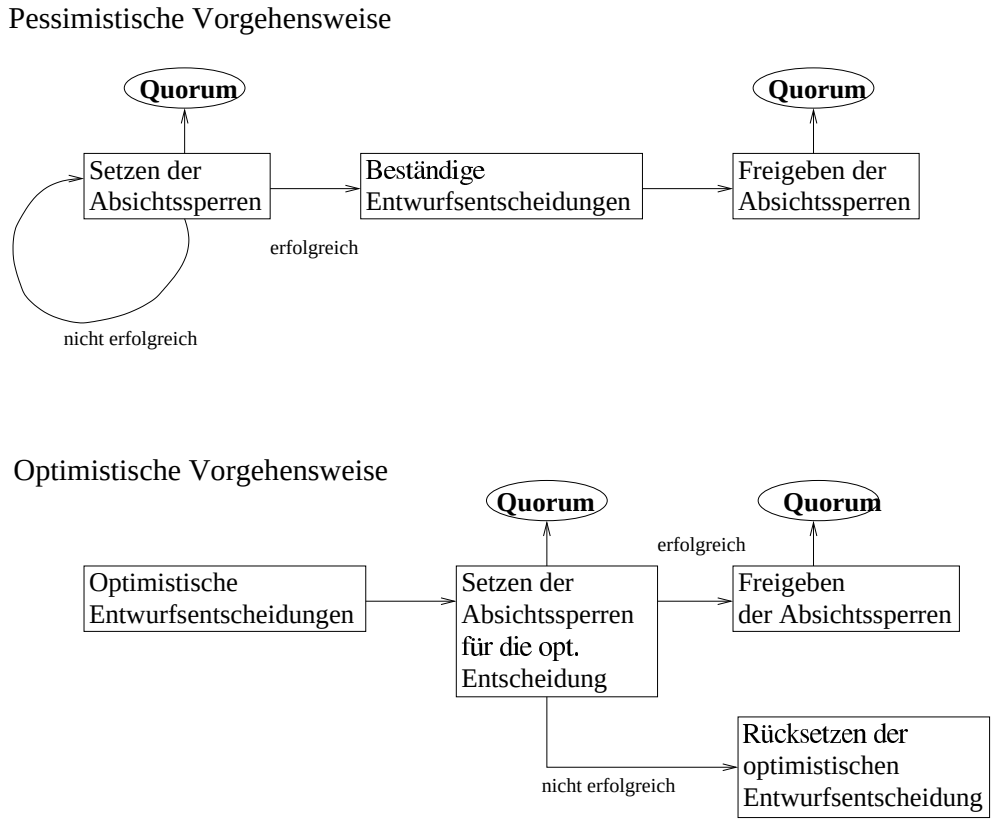


Abbildung 9.2: Pessimistische und optimistische Abkopplung

dann muß die optimistische Spezifikation und damit die gesammte Entwurfsentscheidung zurückgesetzt werden.

Die Änderungen am Protokoll selbst sind geringer als es auf den ersten Blick aussieht.

Einbringen einer Spezifikation (*constrain*, *premise* und *assume*)
Neue Spezifikationen können wie gehabt pessimistisch eingebracht werden. Wenn dies aufgrund fehlender Absichtssperren nicht möglich ist (siehe Abschnitt 8.6), dann kann die Spezifikation mit Zustimmung des Entwicklers optimistisch eingebracht werden. Hierzu muß nur die *constrain*-, *premise*- oder *assume*-Operation aus dem angekoppelten Fall in den Kapiteln 6 und 7 ausgeführt werden. Die Spezifikation wird im System gesondert in einer Liste von optimistischen Spezifikationen vermerkt.

Abschließen einer Entwurfsentscheidung
Wenn eine Entwurfsentscheidung mit *commit* abgeschlossen wird, dann wird sie nur als abgeschlossen markiert, wenn sie beständig ist. Andernfalls wird sie als vollständig markiert, damit aber noch nicht durch die Freigabe von Absichtssperren an andere Replikate globalisiert.

Bei Ausführung einer globalen Operation
Wird eine globale Operation ausgeführt, dann wird vor oder nach dieser globalen Operation versucht, für die optimistischen Spezifikationen Absichtssperren nachträglich anzufordern. Wenn dies nicht gelingt, dann wird die Entwurfsentscheidung abgebrochen.

Wenn eine Absichtssperre nachträglich angefordert werden kann, dann wird folgendes Codefragment ausgeführt:

```
Wenn die Entwurfsentscheidung als vollständig markiert ist
  Wenn alle Spezifikationen der Entwurfsentscheidung durch
                              Absichtssperren maskiert sind
    Markiere die Entwurfsentscheidung als abgeschlossen
    Gebe die nachträglich angeforderte Absichtssperre wieder
                                                        frei
  End
End
FERTIG
```

Wenn also noch weitere optimistische Spezifikationen an der Entwurfsentscheidung beteiligt sind, dann muß auch für diese versucht werden, eine Absichtssperre nachträglich anzufordern. Andernfalls ist die Entwurfsentscheidung pessimistisch geworden und kann damit abgeschlossen und globalisiert werden. Dies geschieht durch die Freigabe der nachträglich angeforderten Absichtssperre, da diese automatisch vom Replikat nur für diese abgeschlossene Entwurfsentscheidung angefordert wurde.

Dieser Ansatz vereint zwei wichtige Vorteile.

- Die Beständigkeit einer Spezifikation beziehungsweise Entwurfsentscheidung kann unter Umständen schon mit dem nächsten Quorum garantiert werden. Wenn in Beispiel 9.2.1 das Replikat R_4 mit dem Quorum Q_2 schon die für D_2 nötigen Absichtssperren setzen kann, dann wird D_2 sofort mit Q_2 beständig. Die Erkennung des Konflikts mit D_1 kann dann zwar trotzdem erst mit Q_4 stattfinden und wird dann auch erst dort beseitigt. Durch die für D_2 nachträglich gesetzten Absichtssperren ist D_2 dann mittlerweile beständig und D_1 muß zurückgesetzt werden. Dies entspricht genau der intuitiven Erwartung, da das Replikat, auf dem D_1 eingebracht wurde, lange abgekoppelt war und erst mit Q_4 wieder Kontakt zu den anderen Replikaten hatte und damit auch erst dann den Widerspruch bemerken konnte.

- Optimistische Spezifikationen beziehungsweise Entwurfsentscheidungen können vollständig lokal an einem Replikat behandelt werden. Für alle anderen Replikate sieht es so aus, als wären diese Spezifikationen ganz normal im Rahmen einer Absichtssperre eingebracht worden. Außerdem ist die Anforderung der Absichtssperre nicht aufwendiger als für normale pessimistische Zugriffe. Der Mehraufwand für dieses Verfahren liegt nur in der lokalen Zwischenspeicherung von optimistischen Spezifikationen, bis wieder ein Quorum verfügbar ist, mit dem dann die Absichtssperren nachgereicht werden können. Mit der Freigabe dieser Absichtssperren werden dann auch ganz automatisch die zwischenzeitlich beständigen Entwurfsentscheidungen verteilt.

Abbildung 9.2 zeigt die Unterschiede des pessimistischen Modells aus Kapitel 8 und dem hier vorgestellten optimistischen Ansatz. Die Unterschiede beschränken sich darauf, daß ein Replikat lokal schon vor dem nötigen Setzen der Absichtssperren neue Spezifikationen einbringen darf, die dann aber nur überleben können, wenn das nachträgliche Setzen der Absichtssperren erfolgreich ist.

9.2.2 Rücksetzungen nur bei unvermeidlichen Konflikten

Kann für eine optimistische Spezifikation nachträglich keine Absichtssperre erworben werden, so ist der Konflikt noch nicht garantiert. Möglicherweise wurde die in Konflikt stehende Absichtssperre zu streng gewählt und die Freiräume dieser Absichtssperre nicht genutzt. Allerdings kann dies erst nach Freigabe der Absichtssperren abschließend geklärt werden. Die optimistische Entwurfsentscheidung kann bis zur Freigabe der anderen Absichtssperre „auf Eis gelegt“ werden, um die letzte Chance für ein Überleben der Daten auszunützen. Wird die optimistische Entwurfsentscheidung sofort beendet, so entstehen keine unnötigen Wartezeiten, aber möglicherweise unnötige Rücksetzungen. Wenn die Entwurfsentscheidung nicht sofort zurückgesetzt wird, dann werden Entwurfsentscheidungen nur zurückgesetzt, wenn dies unvermeidlich ist. Der Nachteil liegt in längeren und zum Teil unnötigen Wartezeiten. Eine Konfiguration des Systems eventuell auch im Einzelfall im Hinblick auf die Entscheidung zwischen früherem aber eventuell unnötigem Rücksetzen und längerem aber möglicherweise erfolglosem Warten ist hier wünschenswert.

In Abbildung 9.3 ist das verlängerte Warten als Ablaufdiagramm dargestellt. Ausgehend von der optimistischen Entwurfsentscheidung (links oben) wird nachträglich versucht, eine Absichtssperre anzufordern. Wenn dies erfolgreich ist, dann wird diese Absichtssperre wieder freigegeben und damit die optimistische Entwurfsentscheidung globalisiert. Wenn dies nicht möglich ist, dann wird untersucht, ob mittlerweile durch Entwurfsentscheidungen von anderen Replikaten, die durch globale Operationen verteilt wurden, ein lokaler Widerspruch entstanden ist. Wenn dies der Fall ist, dann muß die optimistische Entwurfsentscheidung zurückgesetzt werden und das Replikat verfährt im normalen Betrieb weiter. Wenn noch kein lokaler Widerspruch entstanden ist, dann geht das Replikat ebenfalls in den normalen Betrieb über, allerdings in den rechten oberen Kasten. In diesem Kasten „Normaler Betrieb“ ist immer noch die optimistische Entwurfsentscheidung im Hintergrund, für die nach der nächsten globalen Operation wieder überprüft wird, ob mittlerweile die nötigen Absichtssperren gesetzt werden können. Damit ist eine Schleife entstanden, die so oft durchlaufen wird, bis der Normale Betrieb unten links erreicht wird.

Hält allerdings ein Replikat eine optimistische Entwurfsentscheidung und wartet auf die Freigabe einer anderen Absichtssperre, um dann doch noch die Entwurfsentscheidung einbringen zu können, so müssen alle anderen Absichtssperren, von denen die Entwurfsentscheidung abhängig ist, gehalten werden. Geschieht dies an mehreren Replikaten gleichzeitig, kann eine *Verklemmung (Deadlock)* die Folge sein, in dem mehrere Replikate auf die gegenseitige Freigabe der Absichtssperren warten. In diesem Fall sind geeignete Verfahren zur Deadlockerkennung und Behandlung erforderlich (zum Beispiel Zyklenerkennung in Graphen oder einfach Zeitüberschreitungen).

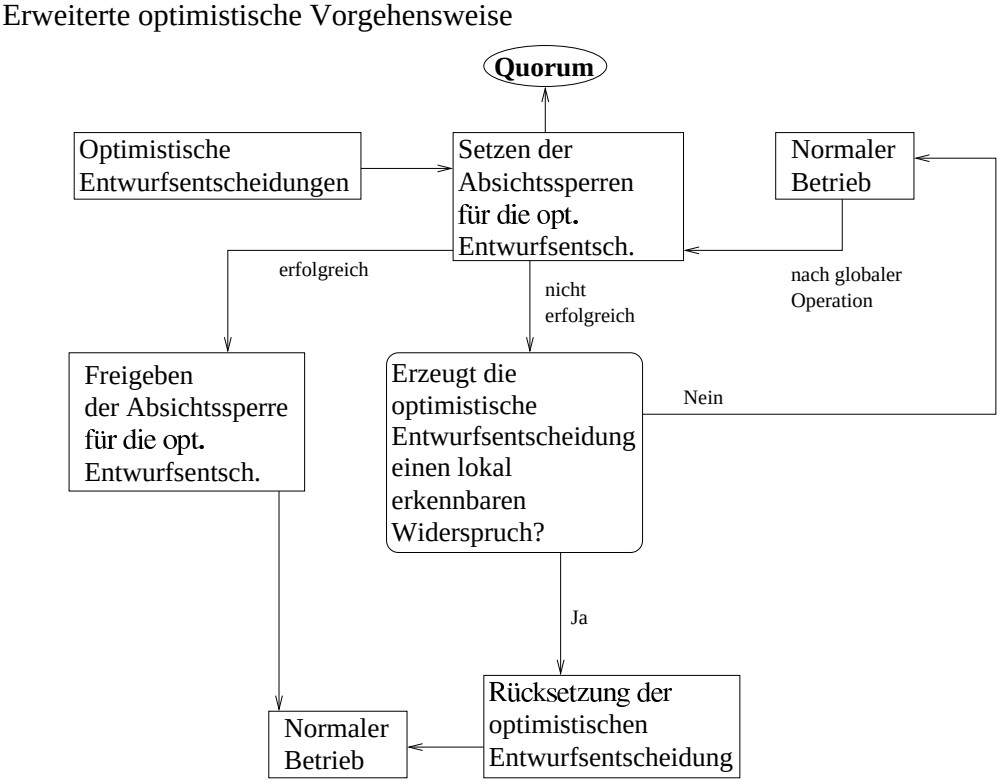


Abbildung 9.3: Pessimistische und optimistische Abkopplung

Beispiel 9.2.2 (Verklemmung zwischen Replikaten)
Geben sind zwei Replikate R_1 und R_2. R_1 hält die X-Sperre $\mathcal{X}_1(x \in [0,200])$ auf dem Datenelement x und R_2 hält die X-Sperre $\mathcal{X}_2(y \in [0,200])$ auf dem Datenelement y.

An R_1 wird die Entwurfsentscheidung D_1 mit der optimistischen Spezifikation $y \in [10,20]$ und der pessimistischen Spezifikation $x \in [100,110]$ eingebracht. Gleichzeitig wird an R_2 die Entwurfsentscheidung D_2 mit der optimistischen Spezifikation $x \in [10,20]$ und der pessimistischen Spezifikation $y \in [100,110]$ eingebracht.

Damit wartet R_1 für D_1 auf die Freigabe der Absichtssperre $\mathcal{X}_2(y \in [0,200])$ während R_2 für D_2 auf die Freigabe der Absichtssperre $\mathcal{X}_1(x \in [0,200])$ wartet. Die beiden Absichtssperren werden aber jeweils erst freigegeben, wenn D_1 beziehungsweise D_2 abgeschlossen ist. □

9.3 Setzen und Freigeben von Absichtssperren in der Praxis

Findet eine Abkopplung intendiert statt, so kann, will und muß der Entwickler die Absichtssperren selbstverständlich vorher explizit setzen. Wenn aber gar keine Abkopplung erfolgt, sondern die Zugriffe im angekoppelten Betrieb erfolgen, so will der Entwickler natürlich keine Absichtssperren setzen, sondern die Spezifikationen direkt einbringen. In diesem Fall müssen Absichtssperren automatisch vom System gesetzt werden, da im erweiterten Demarkationsprotokoll die Synchronisation der Replikate ausschließlich durch Absichtssperren erfolgt, auch wenn Replikate gar nicht abgekoppelt sind.

Ein naiver Algorithmus könnte die Absichtssperren gerade in der unbedingt nötigen Strenge direkt vor dem Zugriff setzen und nach Abschluß der Entwurfsentscheidung wieder freigeben. Diese Variante ist einfach zu implementieren und liefert sicherlich auch das gewünschte Ergebnis. Darüber hinaus bietet sich allerdings ein „Caching“ der

Absichtssperren an. Das heißt, daß Absichtssperren möglicherweise anhand des untersuchten Entwicklerverhaltens vom System im Voraus gesetzt werden und nicht direkt nach dem Einbringen der Entwurfsentscheidung freigegeben werden. Insbesondere wenn der Grund für die replizierte Datenhaltung eben gerade das Caching ist, dann sollte mit dem Vorausladen der Daten auch gleich eine entsprechende Absichtssperre gesetzt werden, da sonst das Vorausladen überhaupt keine Vorteile bringt. Auch das Halten der Absichtssperre über Entwurfsentscheidungen hinaus entspricht dann genau dem Halten der im Cache befindlichen Daten, bis entweder ein Verdrängungsalgorithmus die Daten freigibt oder ein anderer Klient die Daten benötigt und diese mit einem Rückruf anfordert. Da das automatische Setzen und Freigeben von Absichtssperren aber nur Operationen ausführt, die sonst vom Benutzer explizit auszuführen sind, und nichts am allgemeinen Demarkationsprotokoll oder der internen Ausführung der Operationen selbst verändert wird, hat dieses Vorgehen keinerlei Auswirkungen auf das Prinzip des Revisionsmodells und kann damit als Zusatzeigenschaft angesehen werden, die dem Entwickler seine Arbeit erleichtert. Die entsprechenden Überlegungen könnten auch wie beim Caching in einem Klienten verwirklicht werden und nutzen damit nur die Funktionalität der Datenbasis, ändern sie aber nicht.

9.3.1 Automatisches Setzen von Absichtssperren

Für das automatische Setzen einer Absichtssperre gibt es prinzipiell drei Freiheitsgrade:

Der Zeitpunkt zu dem die Absichtssperren angefordert werden
Wenn das System keine zutreffende Voraussage machen kann, so muß die Absichtssperre natürlich im Moment der aufgerufenen *constrain*-Operation angefordert werden. Sind allerdings bestimmte Abläufe immer gleich, so können die Absichtssperren schon im Voraus angefordert werden. Wenn zum Beispiel ein Entwickler zu Beginn seiner Arbeitssitzung immer im Rahmen seiner Anmeldung im System auf denselben Datenelementen Spezifikationen einbringt, dann können schon mit der Anmeldung dieses Entwicklers die entsprechenden Absichtssperren angefordert werden, so daß die folgenden *constrain*-Operationen ohne Verzögerung ausgeführt werden können.

Allerdings ist die Entwicklung eines Produktes hochgradig dynamisch und daher kaum vorhersehbar. Trotzdem können Rückschlüsse auf künftige einzubringende Spezifikationen gezogen werden, etwa durch die Betrachtung der bereits durchgeführten Entwurfsentscheidungen. Wenn zum Beispiel auf dem Datenelement X vom Entwickler E_i eine Spezifikation eingebracht wurde und in der Vergangenheit in der Entwurfsentscheidung D_j vom Entwickler E_i ebenfalls Spezifikationen auf X eingebracht wurden, dann liegt die Vermutung nahe, daß auch die anderen in D_j mit Spezifikationen belegten Datenelemente nun genauer spezifiziert werden sollen. So kann zum Beispiel der Motorenentwickler nach jeder Verfeinerung der Motorleistung in derselben Entwurfsentscheidung auch den Öldruck verfeinern. Das Setzen einer Absichtssperre auf dem Öldruck nach dem Zugriff des Motorenentwicklers auf die Motorleistung ist hier also ein sinnvolles Vorgehen.

Um diese Vorausladung von Absichtssperren allerdings erfolgreich durchzuführen, ist ein Lernalgorithmus nötig, der die abgelaufenen Sperranforderungen analysiert und daraus Rückschlüsse für künftige Arbeiten zieht.

Die Menge der mit Absichtssperren belegten Datenelemente
Ruft ein Entwickler eine *constrain*-Operation auf einem Datenelement auf, so kann das System möglicherweise „Vermutungen“ über die folgenden Zugriffe anstellen. Gründe für solche Vermutungen können vielfältiger Art sein. Eine Möglichkeit sind zum Beispiel referenzierte Datenelemente, für die eine hohe Wahrscheinlichkeit für schnell folgende Zugriffe besteht. In diesem Fall können Methoden aus dem Caching durchaus interessant sein und mit kleineren Anpassungen für das Setzen von Absichtssperren verwendet werden.

Die Art der Absichtssperren selbst

Wenn ein Entwickler eine *constrain*-Operation auf einem Datenelement ausführt, so besteht durchaus eine gesteigerte Wahrscheinlichkeit, daß derselbe Entwickler innerhalb kurzer Zeit danach weitere Spezifikationen auf dieses Datenelement aufprägen will. In diesem Fall ist es wünschenswert, wenn statt der unbedingt notwendigen Absichtssperre, nämlich genau der einzubringenden Spezifikation, eine stärkere Absichtssperre, so zum Beispiel die X-Sperre $\mathcal{X}_{true}()$ (die wahre Spezifikation, die allen Elementen der Domäne den Wert *wahr* zuordnet) gesetzt wird. Damit kann der Entwickler beliebige Spezifikationen im Rahmen dieser einen Absichtssperre einbringen, solange diese mit den bestehenden Spezifikationen kompatibel sind. Im Gegenzug können allerdings Entwickler auf anderen Replikaten für die Dauer des Bestehens dieser Absichtssperre überhaupt keine Spezifikationen mehr einbringen.

Um die drei Freiheitsgrade zum automatischen Setzen von Absichtssperren besser bewerten zu können, sind genauere Kenntnisse des Benutzerverhaltens erforderlich. Diese können aber erst mit der Anwendung des Revisionsmodells in der Praxis gewonnen werden. Wenn dabei vernünftige Voraussagen über künftige Datenzugriffe getroffen werden können, dann können diese drei Möglichkeiten durchaus eine große Rolle spielen. Insbesondere die zweite Alternativ mit den verbundenen Datenelementen scheint bei Entwicklungsarbeit mit starker Lokalität enorme Potentiale zu haben.

9.3.2 Automatische Freigabe von Absichtssperren

Die Freiheitsgrade für die Freigabe von Absichtssperren bestehen dabei nur in dem Zeitpunkt, da die Sperre selbst ja schon vorgegeben ist. Es gibt sehr unterschiedliche Möglichkeiten, den Zeitpunkt für die Freigabe einer Absichtssperre festzulegen. Für alle genannten Möglichkeiten gilt allerdings die Mindestanforderung: Eine Absichtssperre muß mindestens so lange gesetzt bleiben, bis alle Entwurfsentscheidungen, die im Rahmen dieser Absichtssperre Spezifikationen eingebracht haben, abgeschlossen sind. Wenn die automatische Freigabe auch für die optimistische Erweiterung benutzt werden soll, dann kann eine Absichtssperre frühestens freigegeben werden, wenn alle durch die Absichtssperre maskierten Entwurfsentscheidungen beständig oder abgebrochen sind.

Sofortige Freigabe

Nach dieser Möglichkeit wird eine Absichtssperre sofort nach Erfüllung der zeitlichen Mindestanforderung, also des Abschlusses und der garantierten Beständigkeit aller Entwurfsentscheidungen, die durch die Absichtssperre maskiert sind, freigegeben. Diese Möglichkeit ist besonders einfach, da bei der Beendigung einer Entwurfsentscheidung nur alle Absichtssperren überprüft werden müssen. Alle Absichtssperren, denen keine aktiven Entwurfsentscheidungen mehr zugeordnet sind, werden sofort freigegeben.

Durch Verdrängung oder Zeitablauf

Auch diese Strategie der Freigabe von Absichtssperren entspringt dem Caching. Im Fall der Verdrängung gibt es eine fest vorgegebene maximale Zahl von unbenötigten gesetzten Absichtssperren und mit jeder neu dazukommenden muß eine alte freigegeben werden. Eine etwas einfachere Alternative ist die Freigabe nach Ablauf einer bestimmten Zeitspanne, so daß jede Absichtssperre prinzipiell eine feste Zahl von Zeiteinheiten länger gehalten wird als unbedingt notwendig.

Damit werden Absichtssperren nicht sofort nach der *constrain*-Operation wieder freigegeben und sind bei eventuell nachfolgenden wiederholten Zugriffen auf dasselbe Datenelement noch gesetzt. Die Wahrscheinlichkeit für solche Zugriffe ist bei interaktiven Arbeiten an geometrischen Daten, wie zum Beispiel dem Robotergreifer, durchaus hoch.

Auf Wunsch (Callback)

Wenn ein Entwickler eine Absichtssperre nicht mehr benötigt, so kann er diese behalten bis ein anderer Benutzer ebenfalls auf den Daten arbeiten will. Die Absichtssperre darüberhinaus zu behalten, macht dem anderen Benutzer die Arbeit unmöglich und ist nur zu vertreten, wenn der erste Benutzer die Absichtssperre noch für weitere Arbeiten in direkter Zukunft braucht. Andernfalls sollten wie beim Caching die per Callback angefragten Absichtssperren freigegeben werden, so daß die anderen Entwickler selbst Absichtssperren setzen und im Rahmen dieser ihre Arbeiten ausführen können.

Diese Variante entspricht in der Idee dem ursprünglichen Demarkationsprotokoll, in dem die Bedingungen immer gesetzt sind, aber bei Bedarf angepaßt werden. Wenn eine Bedingung daher verstärkt werden soll, dann muß im Gegenzug eine andere Bedingung reduziert werden (entsprechend dem Callback).

Bei geplanter Abkopplung

Im Fall der geplanten Abkopplung ist die Freigabe einer nicht mehr benötigten Absichtssperre sehr wichtig. Da eine Abkopplung sehr lange dauern kann, im Extremfall mehrere Tage bis Wochen, kann durch eine „mitgenommene" Absichtssperre die Arbeit von Kollegen unnötigerweise behindert werden. Die Mitnahme von Absichtssperren macht nur Sinn, wenn auf den gesperrten mobilen Daten auch gearbeitet wird.

Auch Kombinationen der unterschiedlichen Möglichkeiten sind durchaus denkbar und auch sinnvoll. So sollten alle Strategien, bei denen die Sperre länger als unbedingt nötig gesetzt bleibt, die Callback-Möglichkeit und die Freigabe bei der Abkopplung implementieren, da sonst durch unnötig gehaltene Absichtssperren die Arbeit von anderen Entwicklern nachhaltig behindert werden kann. Für die Praxis interessant ist zum Beispiel folgender Algorithmus, in dem nur die betroffenen Methoden beschrieben sind.

Seien $\mathbb{A}_1$ und $\mathbb{A}_2$ Mengen von Absichtssperren. In $\mathbb{A}_1$ sind alle automatisch gesetzten Absichtssperren verzeichnet, die noch aktive Entwurfsentscheidungen maskieren. In $\mathbb{A}_2$ sind alle automatisch gesetzten Absichtssperren verzeichnet, zu denen keine aktiven Entwurfsentscheidungen mehr existieren.

Einbringen einer Spezifikation (*constrain*)

```
Wenn keine ausreichende Absichtssperre gesetzt ist
  Suche kompatible Absichtssperre A(), die für
                          constrain  ausreicht
  Wenn erfolgreich
    Setze Absichtssperre A()
    Trage Absichtssperre A() in A1 ein
    Bringe neue Spezifikation ein
  Sonst
    FEHLER: neue Spezifikation kann nicht eingebracht werden
  End
Sonst
  Wenn die Absichtssperre in A2 steht
    REMARK: Hier liegt der Vorteil der Vorgehensweise
    Entferne die Absichtssperre aus A2
    Trage die Absichtssperre in A1 ein
  Bringe neue Spezifikation ein
End
FERTIG
```

Geplante Abkopplung

```
Für alle Sperren A() in A2
  Entferne A() aus A2
  Gebe A() frei
End
FERTIG
```

Abschluß einer Entwurfsentscheidung (*commit* oder *abort*)

```
Schließe die Entwurfsentscheidung ab
Für alle Sperren A() in 𝔸1
  Wenn A() die abzuschließende Entwurfsentscheidung maskiert
                                                          hat
    Wenn A() keine aktiven Entwurfsentscheidungen mehr
                                                  maskiert
      Entferne A() aus 𝔸1
      Trage A() in 𝔸2 ein
      Wenn 𝔸2 größer als MAX_SPERREN
        B() = älteste Sperre in 𝔸2
        Entferne B() aus 𝔸2
        Gebe B() frei
      End
    End
  End
End
FERTIG
```

Das verzögerte Freigeben von Absichtssperren verringert in einigen Fällen die Zahl der nötigen globalen Operationen, da globale Operationen für die Freigabe und das erneute Anfordern vermieden werden. Allerdings besteht die Gefahr, daß aufgrund einer unerwarteten Abkopplung des Replikats mit der Absichtssperre auf den restlichen Replikaten nicht mehr gearbeitet werden kann, da die Absichtssperre noch nicht freigegeben wurde. Die verzögerte Freigabe sollte daher nur dann genutzt werden, wenn unerwartete Abkopplungen, also Netzausfälle, sehr selten sind.

9.4 Absichtssperren auf Gruppen von Datenelementen

Vor einer intendierten Abkopplung sollten von dem abkoppelnden Entwickler genügend Absichtssperren für die geplante isolierte Arbeit gesetzt werden. Absichtssperren für jedes einzelne Datenelement zu setzen ist zwar prinzipiell möglich, in der Praxis allerdings zu aufwendig. Lösungen außerhalb des RV-Modells können hier zum Beispiel in der Benutzeroberfläche den Entwickler darin unterstützen, durch einen Befehl mehrere Absichtssperren zu setzen.

Auf der anderen Seite kann auch das System die Möglichkeit anbieten, mit einem Befehl mehrere Absichtssperren gleichzeitig zu setzen. Auf jeden Fall muß so eine Möglichkeit sehr flexibel sein, da für einen angekoppelten Zugriff auf ein Datenelement sicherlich nicht jedesmal eine ganze Reihe von Absichtssperre gesetzt werden sollen. Nur wenn der Benutzer dies explizit wünscht, sollten mehrere Absichtssperren auf einmal gesetzt werden. Es gibt mehrere Möglichkeiten zur Gruppierung von Datenelementen, so daß Absichtssperren auf allen Datenelementen einer Gruppe für den Anwender sinnvoll sind. Einige Beispiele sollen hier vorgestellt werden.

Alle Instanzen einer Klasse (Objektorientierung)
Arbeitet ein Entwickler auf einem Objekt, so kann er durchaus mehrere Instanzen dieser Klasse bearbeiten wollen. So gibt es beim Robotergreifer zum Beispiel Lager für beide Greiferbacken. Wenn ein Entwickler nun das eine Lager modifiziert, dann wird er vermutlich auch das zweite Lager modifizieren. In diesem Fall ist dann eine Absichtssperre auf der Klasse der Lager sinnvoll.

Ganze Relationen statt einem Tupel (Relationales Modell)
Im relationalen Modell könnte für einen Entwickler eine Absichtssperre auf einer ganzen Relation interessant sein, wenn er mehrere oder alle Tupel dieser Relation ändern will.

Beliebige vordefinierte Gruppen
Bestimmte Datenelemente können zu einer Gruppe zusammengefaßt werden. Oftmals werden solche Gruppen als Konfigurationen für workspacebasierte Modelle verwendet (zum Beispiel in ObjectStore [Obj95]). Benutzer können hierbei eine komplette Konfiguration ausbuchen (check out), in ihrem privaten Workspace bearbeiten und später wieder einbuchen (check in). Im erweiterten beziehungsweise allgemeinen Demarkationsprotokoll können zum Beispiel Absichtssperren auf ganzen Konfigurationen gesetzt werden.

Bestimmte durch Prädikate ausgezeichnete Datenelemente
Es gibt auch noch die Möglichkeit, Absichtssperren völlig losgelöst von Datenelementen zu setzen, wobei die Absichtssperre selbst über gewisse Prädikate spezifiziert, welche Objekte sie sperrt. Ein Prädikat könnte zum Beispiel eine SQL-Anfrage sein, so daß alle Datenelemente, für die die Anfrage zutrifft, gesperrt werden. Da die Datenelemente selbst aber schon Prädikate sind, müssen hier also Prädikate über Prädikaten ausgewertet werden. Ein großes Problem dieses Ansatzes ist daher sicherlich der hohe Aufwand für die Evaluierung dieser Prädikate. Sowohl wenn eine Absichtssperre gesetzt werden soll, als auch wenn eine neue Spezifikation eingebracht werden soll, müssen im schlimmsten Fall die Prädikate aller aktiven Absichtssperren evaluiert werden.

Viele weiter Möglichkeiten sind noch denkbar, so zum Beispiel nach zuständiger Instanz oder nach Zuordnung der Datenelemente zu einzelnen Abteilungen eines Unternehmens. Auch hierarchische Strukturen können sehr gut für die Sperrung eines Teilbaums ausgenutzt werden.

Steht eine Gruppe von Datenelementen fest, die alle gemeinsam gesperrt werden sollen, so stellt sich noch die Frage, welche Spezifikationen als Absichtssperren verwendet werden sollen. Hierzu bieten sich drei Varianten an:

S-Sperre mit aktuellem peek
Diese Möglichkeit liefert die geringsten Freiräume für den Eigentümer der Absichtssperre, da keinerlei Verfeinerungen möglich sind. Dafür kann auf anderen Replikaten dieses Datenelements noch verfeinert und damit konstruktiv gearbeitet werden.

X-Sperre mit aktuellem peek
Hierbei kann der Eigentümer der X-Sperre das Datenelement weiter verfeinern, während auf allen anderen Replikaten nur fixierende Zugriffe möglich sind. Dort können also nur noch Spezifikationen eingebracht werden, die von der X-Sperre (dem aktuellen Peek) impliziert werden.

X-Sperre mit der wahren Spezifikation
Mit dieser radikalen Variante wird das Datenelement vollständig für die anderen Replikate gesperrt. Nur noch der Eigentümer der X-Sperre kann auf diesem Datenelement arbeiten, hat allerdings dabei alle nur denkbaren Freiheiten. X-Sperren mit der wahren Spezifikation sollten nur in gut überlegten Spezialfällen (zum Beispiel in sehr frühen Phasen der Entwicklung) eingesetzt werden.

Kapitel 10

Spezifikationen als abstrakte Datentypen

Das Revisionsmodell verbunden mit dem allgemeinen Demarkationsprotokoll, wie es bisher in dieser Arbeit vorgestellt wurde, hat sich stark an dem vorgestellten Szenario und damit an der Motivation und den verbundenen Anforderungen orientiert. In diesem Zusammenhang beschreiben Spezifikationen eine Teilmenge der Domäne, die für ein gegebenes Datenelement als akzeptable Werte gelten. In diesem Kapitel soll das RV-Modell losgelöst vom Szenario und der praktischen Anwendung aus einer mehr abstrakten Sicht dargestellt werden. Spezifikationen haben damit keine festgelegte Bedeutung mehr, sondern sind nur noch abstrakte Datentypen, an deren Methoden gewisse Voraussetzungen gestellt werden.

Im Prinzip unterscheidet sich dieses abstrakte RV-Modell nicht vom bisher vorgestellten Modell. Die Unterschiede liegen in den abstrakten Spezifikationen und den damit verbundenen geringen Anforderungen an diese Spezifikationen. Damit muß der Beweis der Korrektheit des Protokolls auf diesen vorausgesetzten Eigenschaften der Spezifikationen aufsetzen, die im direkt folgenden Abschnitt vorgestellt werden.

Da die Entwicklung des allgemeinen Demarkationsprotokolls in drei großen Schritten in den Kapiteln 6 bis 8 sehr ausführlich betrieben wurde, kann das abstrakte RV-Modell direkt auf diese Erkenntnisse aufsetzen und sich auf die wesentlichen Neuerung, nämlich die geänderten Garantien und damit verbunden auch den modifizierten Beweis beschränken.

10.1 Spezifikationen

Eine Spezifikation ist ein abstrakter Datentyp. Die Menge aller Spezifikationen wird mit $\mathbb{S}$ bezeichnet und hat zwei ausgezeichnete Elemente, $\mathcal{S}_{True}()$ und $\mathcal{S}_{False}()$. Jede Spezifikation aus $\mathbb{S}$ realisiert die folgenden Methoden:

- boolean *equals* (Spezifikation)
- Spezifikation *combine* (Spezifikation)
- boolean *implies* (Spezifikation)

An die Implementierung der Methoden werden folgende Anforderungen gestellt:

- *equals* definiert eine Äquivalenzrelation (reflexiv, symmetrisch und transitiv).

$$\mathcal{S}_1().equals(\mathcal{S}_1()) = true$$

$$\mathcal{S}_1().equals(\mathcal{S}_2()) \Leftrightarrow \mathcal{S}_2().equals(\mathcal{S}_1())$$

$$\mathcal{S}_1().equals(\mathcal{S}_2()) \wedge \mathcal{S}_2().equals(\mathcal{S}_3()) \Rightarrow \mathcal{S}_1().equals(\mathcal{S}_3())$$

- *combine* ist abgeschlossen.

$$\forall_{\mathcal{S}_1(),\mathcal{S}_2()\in\mathbb{S}}\ \mathcal{S}_1().combine(\mathcal{S}_2()) \in \mathbb{S}$$

- *combine* ist reflexiv

$$\mathcal{S}_1().equals(\mathcal{S}_1().combine(\mathcal{S}_1()))$$

- *combine* ist symmetrisch

$$\mathcal{S}_1().combine(\mathcal{S}_2()).equals(\mathcal{S}_2().combine(\mathcal{S}_1()))$$

- *combine* ist assoziativ

$$\mathcal{S}_1().combine(\mathcal{S}_2().combine(\mathcal{S}_3())).equals(\mathcal{S}_1().combine(\mathcal{S}_2()).combine(\mathcal{S}_3()))$$

- $\mathcal{S}_{True}()$ ist das neutrale Element.

$$\mathcal{S}_1()combine(\mathcal{S}_{True}()).equals(\mathcal{S}_1())$$

- $\mathcal{S}_{False}()$ ist das Null-Element.

$$\mathcal{S}_1().combine(\mathcal{S}_{False}()).equals(\mathcal{S}_{False}())$$

- $\mathcal{S}_1().implies(\mathcal{S}_2()) \Leftrightarrow \mathcal{S}_1().combine(\mathcal{S}_2()).equals(\mathcal{S}_1())$.

Der Einfachheit wegen wird noch die Kombination mehrerer Spezifikationen definiert:

Definition 10.1.1 (Kombination mehrerer Spezifikationen)
Die Kombination $\mathcal{S}() = Komb(\mathbb{S})$ der Menge $\mathbb{S} = \{\mathcal{S}_1() \ldots \mathcal{S}_n()\}$, bestehend aus den Spezifikationen $\mathcal{S}_1()$ bis $\mathcal{S}_n()$, sei gleich der wahren Spezifikation $\mathcal{S}_{True}()$, wenn $\mathbb{S}$ die leere Menge ist, ansonsten sei $\mathcal{S}() = Komb(\mathbb{S})$ gleich der Spezifikation $\mathcal{S}_1()$ kombiniert mit der Kombination von $\mathbb{S}$ ohne $\mathcal{S}_1()$.

$$Komb(\mathbb{S}) = \begin{cases} \mathcal{S}_{True}() & falls \quad \mathbb{S} = \emptyset \\ \mathcal{S}_1().combine(Komb(\mathbb{S} \setminus \mathcal{S}_1())) & falls \quad \mathbb{S} = \{\mathcal{S}_1() \ldots \mathcal{S}_n()\} \end{cases}$$

□

Das System verwaltet eine Menge von Spezifikationsspeichern. Jeder dieser Spezifikationsspeicher kann eine beliebige Menge von Spezifikationen aufnehmen. Dabei muß das System zu jedem Zeitpunkt die Widerspruchsfreiheit garantieren, das heißt, die Kombination aller Spezifikationen eines Speichers darf nicht das Null-Element ($\mathcal{S}_{False}()$) sein. Jeder Spezifikationsspeicher enthält zu jedem Zeitpunkt das neutrale Element ($\mathcal{S}_{True}()$). Sind also in einem Speicher keine Spezifikationen explizit gespeichert, so ist die Kombination aller Spezifikationen in diesem Speicher gleich dem neutralen Element ($\mathcal{S}_{True}()$).

10.2 Die Operationen

Zu jedem Spezifikationsspeicher gibt es drei Operationen:

peek
Die Operation *peek* gibt genau die Kombination aller momentan im Spezifikationsspeicher enthaltenen Spezifikationen zurück. Diese Kombination kann einfach durch Aufruf der folgenden Methoden erhalten werden:

$$\mathcal{S}_1().combine(\mathcal{S}_2()).... combine(\mathcal{S}_s())$$

Das Ergebnis einer *peek*-Operation wird auch hier Peek genannt.

constrain
Die Operation *constrain* fügt die neue Spezifikation $\mathcal{S}()$ zu dem Spezifikationsspeicher hinzu. Die Operation ist nur erlaubt, wenn die Kombination aller Spezifikationen im Spezifikationsspeicher inklusive der neuen Spezifikation nicht das Null-Element ergibt.

$$\neg(\mathcal{S}_1().combine(\mathcal{S}_2()).... combine(\mathcal{S}_s()).combine(\mathcal{S}())).equals(\mathcal{S}_{False}())$$

release
Die Operation *release* entfernt eine Spezifikation aus dem Spezifikationsspeicher.

10.3 Abkopplung

Das in diesem Kapitel vorgestellte abstrakte RV-Modell mit dem abstrakten Datentyp Spezifikation baut auf das asynchrone Abkopplungsmodell aus Abschnitt 8.5 auf. Damit gilt auch Satz 8.5.1 und abgewandelt auf die hier noch zu spezifizierenden Voraussetzungen der Operationen zum Setzen und Freigeben der Absichtssperren auch Satz 8.5.2.

10.4 Vorbedingungen der Operationen

Die Vorbedingungen der einzelnen Operationen bleiben im wesentlichen dieselben wie in Kapitel 8. Der Unterschied liegt nur in der Anwendung der Formeln auf den abstrakten Datentyp der Spezifikation und die damit verbundene Methode *combine* statt der Konjunktion. Bei der Entwicklung des RV-Modells in den Kapiteln 6 bis 8 wurden nur die Eigenschaften der Konjunktion verwendet, die auch der abstrakte Datentyp Spezifikation nach Abschnitt 10.1 erfüllt. Damit sind alle Überlegungen aus den vorangegangenen Kapiteln übertragbar. Zusätzlich wird bei der Maskierung von Spezifikationen durch S-Sperren die Möglichkeit eingeführt, eine Spezifikation durch mehrere Absichtssperren zu maskieren.

Auch hier gelten die Bezeichnungen aus den vorangegangenen Kapiteln, insbesondere ist

R_1 ... R_r die Replikate

$\mathbb{X}$ die Menge aller X-Sperren

$\mathbb{Y}$ die Menge aller S-Sperren

$\mathcal{G}()$ der Peek zum Zeitpunkt der letzten globalen Operation

$\mathcal{P}_\tau()$ der lokale Peek am Replikat R_τ

$\mathcal{H}()$ der aktuelle globale Peek oder die Kombination der lokalen Peeks aller Replikate

$\mathcal{S}_{\tau,i}()$ die lokalen durch S-Sperren maskierten Spezifikationen. Dabei bezeichnet $\tau = 1..r$ das Replikat, an dem die Spezifikation bekannt ist und $i = 1..s_\tau$ numeriert diese lokalen Spezifikationen am Replikat R_τ durch.

- Eine neue X-Sperre $\mathcal{X}()$ kann gesetzt werden, wenn gilt:
  1. Die X-Sperre ist erfüllbar.
  $$\neg\mathcal{X}().equals(\mathcal{S}_{False}()) \qquad (A_{10.1})$$
  2. Es ist momentan keine andere X-Sperre gesetzt.
  $$\mathbb{X} = \emptyset \qquad (A_{10.2})$$
  3. Die X-Sperre impliziert die Kombination aller S-Sperren und aller globalen Spezifikationen.
  $$\mathcal{X}().implies(Komb(\mathbb{Y}, \mathcal{G}())) \qquad (A_{10.3})$$
- Eine neue S-Sperre $\mathcal{Y}()$ kann gesetzt werden, wenn gilt:
  1. Die neue S-Sperre ist kompatibel mit allen bestehenden S-Sperren und allen globalen Spezifikationen.
  $$\neg Komb(\mathbb{Y}, \mathcal{G}(), \mathcal{Y}()).equals(\mathcal{S}_{False}()) \qquad (A_{10.4})$$
  2. Wenn eine X-Sperre gesetzt ist, so impliziert diese die Kombination alle S-Sperren und alle globalen Spezifikationen.
  $$\mathcal{X}().implies(Komb(\mathbb{Y}, \mathcal{Y}(), \mathcal{G}())) \qquad (A_{10.5})$$
- Die Spezifikation $\mathcal{S}()$ darf im Rahmen einer Menge von lokalen S-Sperren $\mathbb{Y}'$ eingebracht werden, wenn die Kombination aller S-Sperren aus $\mathbb{Y}'$ die einzubringende Spezifikation impliziert.
$$Komb(\mathbb{Y}').implies(\mathcal{S}()) \quad \wedge \quad \mathbb{Y}' \subseteq \mathbb{Y} \qquad (A_{10.6})$$
- Die Spezifikation $\mathcal{S}()$ darf im Rahmen einer lokalen X-Sperre $\mathcal{X}()$ eingebracht werden, wenn die Kombination aus dem lokalen Peek, der einzubringenden Spezifikation und der X-Sperre selbst nicht die falsche Spezifikation ergibt.
$$\neg Komb(\mathcal{P}(), \mathcal{S}(), \mathcal{X}()).equals(\mathcal{S}_{False}()) \qquad (A_{10.7})$$

10.4.1 Korrektheit des abstrakten RV-Protokolls

Nachdem Satz 8.5.2 äquivalent auf die Kompatibilitätsbedingungen der Absichtssperren übertragen werden kann, sind auch im abstrakten RV-Modell immer alle Absichtssperren kompatibel. Da auch Spezifikationen nur durch entsprechende Absichtssperren eingebracht werden können, gelten auch die diesbezüglichen Voraussetzungen. Darauf aufbauend kann der folgende Satz bewiesen werden.

Satz 10.4.1 (Korrektheit des Protokolls für abstrakte Spezifikationen)
Aus den Formeln $A_{10.1}$ bis $A_{10.7}$ folgt die Widerspruchsfreiheit aller lokalen und globalen Spezifikationen.
$$\neg\mathcal{H}().equals(\mathcal{S}_{False}())$$

□

Beweis 10.4.1 (Korrektheit des Protokolls für abstrakte Spezifikationen)
Der Beweis gliedert sich in eine Fallunterscheidung nach dem Vorhandensein von Absichtssperren.

Zu zeigen ist: $\neg\mathcal{H}().equals(\mathcal{S}_{False}())$

Es existiert eine aktive X-Sperre an R_ω und eine Menge von S-Sperren.
Nach Formel $A_{10.7}$ *gilt*

$$\neg Komb(\mathcal{P}_\omega(), \mathcal{X}()).equals(\mathcal{S}_{False}())$$

Mit $\mathcal{X}().implies(Komb(\mathbb{Y}, \mathcal{S}_1(), \ldots \mathcal{S}_g()))$ *(Formel* $A_{10.5}$*) gilt*

$$Komb(\mathcal{X}(), \mathbb{Y}, \mathcal{S}_1(), \ldots \mathcal{S}_g()).equals(\mathcal{X}())$$

und damit auch

$$\neg Komb(\mathcal{P}_\omega(), \mathcal{X}(), \mathbb{Y}, \mathcal{S}_1(), \ldots \mathcal{S}_g()).equals(\mathcal{S}_{False}())$$

Aus Formel $A_{10.6}$ *folgt*

$$\forall_{\tau=1..r}\ \forall_{i=1..s_\tau}\ Komb(\mathbb{Y}).implies(\mathcal{S}_{\tau,i}())$$

und damit

$$Komb(\mathbb{Y}).equals(Komb(\mathbb{Y}, \mathcal{S}_{1,1}() \ldots \mathcal{S}_{r,s_r}()))$$

Zusammen ergibt sich

$$\neg Komb(\mathcal{P}_\omega(), \mathcal{X}(), \mathbb{Y}, \mathcal{S}_{1,1}() \ldots \mathcal{S}_{r,s_r}(), \mathcal{S}_1(), \ldots \mathcal{S}_g()) = \mathcal{S}_{False}()$$

Es existieren keine S-Sperren, aber eine aktive X-Sperre an R_ω.
Nach Formel $A_{10.7}$ *gilt*

$$\neg Komb(\mathcal{P}_\omega(), \mathcal{X}()).equals(\mathcal{S}_{False}())$$

Mit $\mathcal{X}().implies(Komb(\mathbb{Y}, \mathcal{S}_1(), \ldots \mathcal{S}_g()))$ *(Formel* $A_{10.3}$*) gilt:*

$$Komb(\mathcal{X}(), \mathcal{S}_1(), \ldots \mathcal{S}_g()).equals(\mathcal{X}())$$

und damit auch

$$\neg Komb(\mathcal{P}_\omega(), \mathcal{X}(), \mathcal{S}_1(), \ldots \mathcal{S}_g()).equals(\mathcal{S}_{False}())$$

Es existieren S-Sperren, aber keine aktive X-Sperre.
Nach Formel $A_{10.6}$ *gilt*

$$\forall_{\tau=1..r}\ \forall_{i=1..s_\tau}\ \exists_{\mathbb{S}'}\ Komb(\mathbb{S}').implies(\mathcal{S}_{\tau,i}())$$

Daraus folgt

$$Komb(\mathbb{Y}).equals(Komb(\mathbb{Y}, \mathcal{S}_{1,1}() \ldots \mathcal{S}_{r,s_r}()))$$

und

$$\neg Komb(\mathbb{Y}, \mathcal{S}_{1,1}() \ldots \mathcal{S}_{r,s_r}(), \mathcal{S}_1(), \ldots \mathcal{S}_g()).equals(\mathcal{S}_{False}())$$

Es existieren keine Absichtssperren.
In diesem Fall sind in allen Replikaten nur globale Spezifikationen vorhanden, deren Widerspruchsfreiheit aus dem Einbringen der Spezifikationen als lokale Spezifikationen und der anschließenden Umwandlung in globale Spezifikationen folgt.

□

Kapitel 11

Spezifikationen

Schon in Kapitel 5 wurden Intervalle als Spezifikationsformen diskutiert und bei der folgenden Entwicklung der Widerspruchsfreien Freiräume mit den Erweiterungen für Arbeitsteiligkeit und Abkopplung vorausgesetzt. In diesem Kapitel sollen nun alternative Spezifikationsformen untersucht werden. Interessant sind hierfür insbesondere Kugeln, Quader und Fuzzy-Mengen.

Da zum Beispiel Kugeln bezüglich der Konjunktion nicht abgeschlossen sind, kann hier nicht mit voller Genauigkeit, sondern nur mit Approximationen gearbeitet werden. Hinzu kommt, daß je nach Spezifikation die Berechnung der exakten Konjunktion oder zumindest der besten darstellbaren Approximation derselben sehr aufwendig ist, so daß Annäherungen durchaus akzeptabel sind. Dabei werden kleinere Fehler toleriert zugunsten einer deutlich schnelleren Berechnung. In diesem Kapitel müssen daher auch die möglichen Fehler klassifiziert und auf Tolerierbarkeit hin untersucht werden.

11.1 Voraussetzungen

Vor der Diskussion der Spezifikationsformen müssen noch einige Grundlagen bezüglich der Domänen diskutiert werden. Die bisherige Einschränkung auf rationale Zahlen kann für die allgemeine Anwendung des vorgestellten RV-Modells nicht ausreichend sein.

11.1.1 Abstandsmaß und Ordnung

Eine wichtige Einschränkung ergibt sich dadurch, daß die meisten Spezifikationsformen ein Abstandsmaß oder zumindest eine Ordnung auf den Domänen voraussetzt. Von den in dieser Arbeit betrachteten Spezifikationsformen benötigen Intervalle, Kugeln und Quader ein Abstandsmaß auf den Domänen. Fuzzy-Mengen benötigen an sich noch keinerlei Struktur auf der Domäne. Allerdings wird diese in einigen Fällen durch die interne Verwaltung der Fuzzy-Mengen erforderlich und muß daher im Zusammenhang mit diesen diskutiert werden. Abgesehen von den Fuzzy-Mengen benötigen also alle hier betrachteten Spezifikationsformen ein Abstandsmaß auf den Domänen.

Auf vielen atomaren Typen besteht bereits ein Abstandsmaß, das sehr gut verwendet werden kann. Allerdings ist hier zu beachten, daß die Abstände dabei nicht immer einen sinnvollen semantischen Abstand definieren, wie zum Beispiel bei Zeichen oder

Aufzählungstypen. Dort muß entweder dem Benutzer die Möglichkeit gegeben werden, den Abstand selbst zu definieren (etwa bei Enums im Schema) oder der Abstand sollte eher groß bis unendlich definiert werden, damit nicht fälschlicherweise zwei Spezifikationen als widerspruchsfrei gelten, die eigentlich sehr unterschiedlich sind.

Ein Abstandsmaß auf zusammengesetzten Typen wie records oder arrays funktioniert gut, solange die Länge des zusammengesetzten Typs im Schema festgeschrieben ist und auf den einzelnen Feldern vernünftige Abstandsmaße vorhanden sind. In diesem Fall kann der zusammengesetzte Typ als Vektor von einzelnen Typen betrachtet und der euklidische Abstand berechnet werden. Sind die zusammengesetzten Typen ungleich lang, dann müssen zuerst die Längen angeglichen werden, zum Beispiel durch Auffüllen von Null-Elementen, und dann kann der metrische Abstand berechnet werden.

Zeichenketten spielen hier eine besondere Rolle, da deren Semantik nichts mit der Syntax zu tun hat und daher ein syntaktischer Abstand irreführend ist. Ein sinnvolles Abstandsmaß auf Zeichenketten läßt sich daher nur mit einer semantischen Analyse, zum Beispiel mit einem Wörterbuch, definieren. In [Wit02a] wird zum Beispiel ein Wörterbuch verwendet, um Begriffen Fuzzy-Mengen zuzuordnen, auf deren Basis dann eine syntaktische Bewertung der Semantik der Zeichenketten möglich ist.

11.1.2 Typen und Domänen

Das Revisionsmodell ist sehr gut für die Domäne der Zahlen geeignet. Dabei spielt es keine Rolle, ob Fließ-Komma-Zahlen oder ganze Zahlen benutzt werden. Es geht vor allem um den Abstand zwischen den Elementen der Domäne. Für andere Domänen ist das Revisionsmodell weniger gut geeignet oder benötigt zumindest einige Hilfsmittel wie zum Beispiel Wörterbücher im Fall von Zeichenketten.

Eine Beschränkung der Arbeit auf Zahlenwerte verbindet sich aber auch mit der Betrachtung des Szenarios. Wichtige Entwicklungsdaten wie zum Beispiel geometrische Daten oder viele Spezifikationen eines Produkts (Haltekraft beim Robotergreifer oder Spritverbrauch beim PKW) sind Zahlenwerte. Aufzählungstypen oder Zeichenketten werden oft für Produkteigenschaften benutzt, auf denen das Revisionsmodell wenig Sinn macht. Betrachtet man zum Beispiel einen Aufzählungstyp Farbe mit den Werten rot, grün und blau, dann machen Spezifikationen auf diesen Werten keinen Sinn. Gerade Daten in Form von Zeichenketten, wie zum Beispiel „leise“ oder „kann schnell zupacken“ sind grobe Beschreibungen, die erst in Spezifikationen umgewandelt werden müssen, wie etwa ein Maximalwert für die Geräuschentwicklung des Greifers oder einen Mindestwert für die Geschwindigkeit des Zupackens.

Fuzzy-Mengen stellen hier wieder eine Besonderheit dar, da deren Semantik durch die Syntax erkennbar ist und semantische Operationen zwischen zwei Fuzzy-Mengen auf syntaktische Operationen abgebildet werden können. Das Problem bei den Fuzzy-Mengen stellt sich daher mehr in der Benutzerführung, da einzelnen Benutzern der direkte Umgang mit Fuzzy-Mengen nicht zugemutet werden kann. Hierfür ist zum Beispiel die Verknüpfung von linguistischen Begriffen mit Fuzzy-Mengen interessant ([Lan97]), so daß der Benutzer mit ihm bekannten Begriffen arbeiten kann und das System die dem Begriff zugehörige Fuzzy-Menge verarbeitet.

11.2 Anforderungen an die Spezifikationen

Betrachtet man das RV-Modell, so fallen zwei Verknüpfungen auf, die immer wieder benötigt werden, nämlich die Konjunktion zweier Spezifikationen und die Frage, ob eine Spezifikation eine andere impliziert. Da im RV-Modell eine Spezifikation als Beschreibung einer akzeptablen Teilmenge der Domäne betrachtet wird, muß die Konjunktion die Schnittmenge von zwei solchen Teilmengen ermitteln, während bei der Implikation getestet wird, ob eine Menge Teilmenge einer anderen ist.

Insbesondere die Berechnung der Schnittmengen ist nicht immer ganz einfach. So ist zum Beispiel die Schnittmengenbildung bei Kugeln nicht abgeschlossen, da die Schnittmenge von zwei Kugeln nicht wieder eine Kugel ergibt. In solchen Fällen muß die maximal darstellbare Schnittmenge berechnet werden (vergleiche Definition 5.1.4 und Beispiel 5.1.1).

Definition 11.2.1 (Maximale darstellbare Konjunktion)
Seien $\mathbb{S}$ *eine Menge von Spezifikationen und* $\mathbb{S}'$ *eine Teilmenge von* $\mathbb{S}$. $\mathcal{S}_{\mathbb{S}'}() \in \mathbb{S}$ *heißt* **maximale in** $\mathbb{S}$ **darstellbare Konjunktion von** $\mathbb{S}'$*, wenn gilt:*

$$\mathbb{P}_{\mathcal{S}_{\mathbb{S}'}()} \subseteq \mathbb{P}_{\mathbb{S}'} \wedge \forall_{\mathcal{S}() \in \mathbb{S}} \, \mathbb{P}_{\mathcal{S}()} \subseteq \mathbb{P}_{\mathbb{S}'} \Rightarrow |\mathbb{P}_{\mathcal{S}()}| \leq |\mathbb{P}_{\mathcal{S}_{\mathbb{S}'}()}|$$

□

Wenn $\mathbb{S}$ abgeschlossen ist, dann ist die maximale in $\mathbb{S}$ darstellbare Konjunktion einer Teilmenge von $\mathbb{S}$ immer genau die Konjunktion selbst und damit auch eindeutig.

Im Folgenden bezeichnet $\wedge_{real}$ die reale Schnittmengenbildung, die durchaus von der korrekten Schnittmengenbildung $\wedge$ abweichen kann. Zum einen liegt das daran, daß die eigentlich korrekte Schnittmenge mit der betrachteten Spezifikationsform nicht darstellbar ist, in anderen Fällen ist die Berechnung der maximalen darstellbaren Schnittmenge so aufwendig, daß in einer realen Implementierung besser Approximationen verwendet werden. Im Prinzip ist es egal, ob die von der Implementierung ermittelte Schnittmenge wegen der fehlenden Abgeschlossenheit der Spezifikationsfamilie oder der Approximation durch die Implementierung von der theoretischen Schnittmengenbildung abweicht. Daher wird im Folgenden nicht zwischen diesen Fällen unterschieden. In jedem Fall gibt es für das Verhältnis von $\wedge_{real}$ und $\wedge$ auf den Spezifikationen über $\mathbb{D}$ zwei Möglichkeiten:

Die reale Berechnung ist strenger

$$(\mathcal{S}_1() \wedge_{real} \mathcal{S}_2()) \Rightarrow (\mathcal{S}_1() \wedge \mathcal{S}_2())$$

In diesem Fall ist die reale Schnittmengenbildung „strenger“ als die theoretisch korrekte Berechnung der Konjunktion. Die im RV-Modell geforderten Konsistenzbedingungen, Widerspruchsfreiheit und Beständigkeit, werden in diesem Fall sicher nicht verletzt, da alle Kombinationen von Spezifikationen, die real eingebracht werden können, auch nach der theoretischen genauen Berechnung eingebracht werden können.

Auf der anderen Seite werden möglicherweise einige theoretisch korrekte Kombinationen von Spezifikationen durch die reale Schnittmengenbildung ausgeschlossen. Das System verhindert das Einbringen von Spezifikationen, die theoretisch korrekt wären, da eine genaue Evaluierung aller zulässigen Spezifikationen zu aufwendig oder sogar unmöglich ist (zum Beispiel wenn die Schnittmengenbildung nicht abgeschlossen ist).

Dieses Verhalten des Systems kann problematisch werden, wenn ein Entwickler sich nicht mit der einfachen Abweisung des Systems zufrieden gibt, sondern den theoretisch gar nicht existierenden Konflikt beheben will. Der Entwickler läßt sich die Spezifikationen geben, an denen das Einbringen der neuen Spezifikation scheitert und nimmt Kontakt mit deren verantwortlichen Instanzen auf, um den Konflikt zu lösen. Allerdings ist der Konflikt theoretisch gar nicht vorhanden, das bedeutet, die Entwickler finden eine Lösung, zum Beispiel ein fertiges Produkt, die alle Spezifikationen erfüllt, während das System trotzdem das Einbringen aller Spezifikationen ablehnt, da die reale Implementierung zufällig gerade diese Lösung ausschließt.

Beispiel 11.2.1 (Theoretisch unnötiger Konflikt)
In Abbildung 11.1 ist ein zweidimensionaler Graph dargestellt, in dem die geometrische Position des Lagers für einen Arm des Robotergreifers dargestellt ist. Für

die Position dieses Lagers gibt es drei Spezifikationen, die in Form von durchgezogenen Kreisen dargestellt sind. Im einzelnen sind diese Spezifikationen $\mathcal{S}_1((x,y) \in \overline{B_{1.25}((2.5,1.5))})$, $\mathcal{S}_2((x,y) \in \overline{B_{1.25}((2.5,3.25))})$ *und* $\mathcal{S}_3((x,y) \in \overline{B_{1.25}((4.3,2.4))})$.

Wenn die Schnittmenge dieser drei Spezifikationen inkrementell berechnet wird, dann ergibt sich im ersten Schritt je nach Reihenfolge eine der drei gepunkteten Spezifikationen, die mit der jeweils dritten Spezifikation im Konflikt stehen. In diesem Fall akzeptiert das System die drei Spezifikationen nicht, obwohl es einen Punkt gibt, der alle drei Spezifikationen erfüllt. Wenn sich die Entwickler also an einen Tisch setzen, so stellen sie fest, daß der Punkt $(3.2, 2.4)$ *alle drei Spezifikationen erfüllt, obwohl das System die drei Spezifikationen für nicht kompatibel erklärt.*

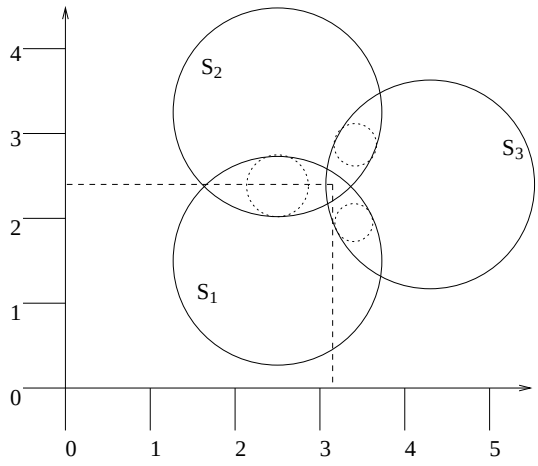


Abbildung 11.1: Beispiel für einen theoretischen nicht vorhandenen Konflikt

Die drei Entwickler können aber durch einfache Anpassungen ihre Spezifikationen so verändern, daß diese auch vom System als widerspruchsfrei erkannt werden. Im Beispiel würde schon die Vergrößerung der drei Epsilons auf 1.4 *dazu führen, daß das System die drei Spezifikationen als widerspruchsfrei erkennt.* □

Die theoretische Berechnung ist strenger

$$(\mathcal{S}_1() \wedge \mathcal{S}_2()) \Rightarrow (\mathcal{S}_1() \wedge_{real} \mathcal{S}_2())$$

Im Gegensatz zum ersten Fall werden hier alle theoretisch korrekten Kombinationen von Spezifikationen akzeptiert, allerdings auch einige die nicht korrekt sind. Das obige Problem der unnötigen und für den Entwickler unverständlichen Konflikte gibt es hier nicht, allerdings ist die Widerspruchsfreiheit gefährdet. Es ist also möglich, daß eine Menge von Spezifikationen eingebracht wird, die in der realen Kombination erfüllbar ist, für die es aber keine korrekte Lösung gibt.

Spätestens mit dem gesuchten Ergebnis, nämlich genau einem konkreten Produkt, kann dieses nochmal mit allen Spezifikationen abgeglichen werden, wobei der bisher nicht sichtbare Konflikt aufgedeckt werden könnte. Die Herstellung eines Produkts, welches nicht den Anforderungen genügt, wird damit zwar verhindert, aber die Erkennung des Konflikts findet sehr spät statt und führt damit möglicherweise zu umfangreichen Revisionen des Produkts. Dieses Verhalten entspricht genau dem von traditionellen Systemen und soll durch das RV-Modell gerade verhindert werden.

Approximationen von Konjunktionen führen zu ungewollten Resultaten und sollten daher vermieden werden. Allerdings ist die genaue Darstellung der gesuchten Konjunktion

mit den vorhandenen Spezifikationen nicht immer möglich (bei fehlender Abgeschlossenheit), oder eine deutlich effizientere Berechnung wird trotz der hier beschriebenen Folgen eingesetzt (im Beispiel 11.2.1 wäre die genaue Berechnung der Schnittmenge sehr aufwendig). Dabei ist zu beachten, daß eine kleine Abweichung zwischen Implementierung und korrekter Berechnung auch die ungewünschten Folgen minimiert. Wenn die Berechnung zum Beispiel nur für ganz wenige Elemente der Domäne fehlerhaft ist, dann ist das Auftreten einer Anomalie sehr selten und kann daher eher toleriert werden.

Der zweite Fall ist zwar durchaus denkbar, repräsentiert aber eher einen optimistischen Ansatz, der am Ende der Produktentwicklung zu einem Konflikt und damit möglicherweise zu einem großen Rückschritt in der Entwicklung führen kann. Der erste Fall ist am ehesten praktikabel, da keine vollbrachte Arbeit invalidiert wird. Dieser entspricht auch genau dem in Definition 11.2.1 geforderten Verhalten nach dem maximalen mit S darstellbaren Peek. Im Fall von Approximationen wird dieser zwar noch weiter abgeschwächt, was aber im Prinzip keine grundlegenden Änderungen zur Folge hat.

Unnötige Konflikte könnten zwar gerade die ideale Lösung der Produktentwicklung ausschließen, wenn aber die Approximation gut ist, dann kann durch eine minimale Anpassung der in Konflikt stehenden Spezifikationen dasselbe Ergebnis erhalten werden. Im Folgenden werden also speziell im Fall von Kugeln, deren Schnittmenge eben gerade keine Kugel ist, Approximationen vorgestellt und diskutiert. Dabei wird der erste Fall aufgrund der garantierten Beständigkeit vorgezogen. Das heißt, die reale Implementierung muß mindestens so streng sein wie die theoretische Berechnung.

Die logische Konjunktion von drei Spezifikationen $\mathcal{S}_1()$, $\mathcal{S}_2()$ und $\mathcal{S}_3()$ auf dem Datenelement X ist unabhängig von der Reihenfolge:

$$\mathcal{S}_1() \wedge \mathcal{S}_2() \wedge \mathcal{S}_3() = \mathcal{S}_2() \wedge \mathcal{S}_3() \wedge \mathcal{S}_1()$$

Bei Approximationen ist dies nicht unbedingt erfüllt. Wenn die approximative Konjunktion $\wedge_{real}$ nicht assoziativ ist, dann muß das System sicherstellen, daß eine Konjunktion von mehreren Spezifikationen immer in der gleichen Reihenfolge evaluiert wird, damit das System immer zu denselben Ergebnissen kommt. Trotzdem sind einige eigenartige Verhaltensweisen vorprogrammiert. So kann zum Beispiel das erneute Einbringen einer bereits im System befindlichen Spezifikation einen Konflikt auslösen.

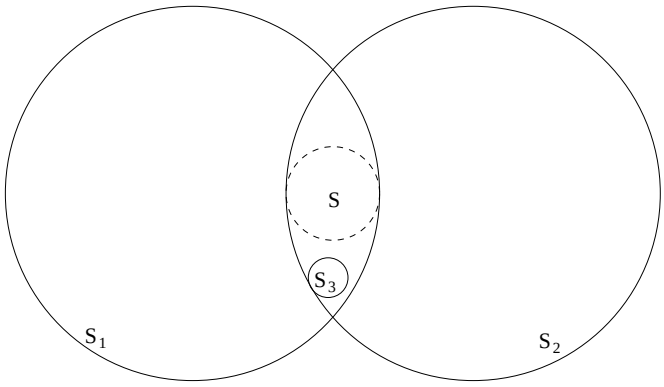


Abbildung 11.2: Fehlende Assoziativität der Approximationen

Beispiel 11.2.2 (Fehlende Assoziativität der Approximationen)
Die inkrementelle Konjunktion der drei Spezifikationen $\mathcal{S}_1()$, $\mathcal{S}_2()$ und $\mathcal{S}_3()$ in Abbildung 11.2 ist abhängig von der Reihenfolge. Wird zuerst die Konjunktion aus $\mathcal{S}_1()$ und $\mathcal{S}_2()$ berechnet, so entsteht $\mathcal{S}()$ und $\mathcal{S}_3()$ kann nicht mehr eingebracht werden. Die anderen beiden Reihenfolgen erhalten $\mathcal{S}_3()$ als Zwischen- und als Endergebnis.

Wenn nun im System die Spezifikationen $\mathcal{S}_1()$, $\mathcal{S}_3()$ und $\mathcal{S}_2()$ in genau dieser Reihenfolge eingebracht wurden und in umgekehrter Reihenfolge evaluiert werden, dann gibt es keine Probleme. Wird dann die Spezifikation $\mathcal{S}_1()$ ein zweites mal eingebracht und an das Ende der Liste angehängt, dann ensteht die Reihenfolge $\mathcal{S}_1()$, $\mathcal{S}_3()$, $\mathcal{S}_2()$ und $\mathcal{S}_1()$, die bei der Evaluierung in umgekehrter Reihenfolge zu einem Widerspruch kommt. □

Um das System nicht vollständig durcheinander zu bringen, müssen noch drei Bedingungen an die Approximation der Konjunktion gestellt werden. Die erste Bedingung verlangt nur die Symmetrie und daß die reale Konjunktion von zwei Spezifikationen, deren theoretischen Konjunktion erfüllbar ist, ebenfalls erfüllbar ist. Man beachte, daß diese Eigenschaft nur für die Konjunktion von zwei Spezifikationen gefordert wird (das Assoziativgesetz gilt also nicht). Eine Erweiterung auf mehrere Spezifikationen würde gerade die Approximation zur theoretisch korrekten Implementierung machen. Die anderen beiden Eigenschaften sind nötig, damit eine Absichtssperre auch weiterhin eine Spezifikation maskieren kann.

1. Für zwei gegebene Spezifikationen $\mathcal{S}_1()$ und $\mathcal{S}_2()$ muß immer gelten:

   $$\mathcal{S}_1() \wedge_{real} \mathcal{S}_2() = \mathcal{S}_2() \wedge_{real} \mathcal{S}_1()$$

   und

   $$Sat(\mathcal{S}_1() \wedge \mathcal{S}_2()) \Leftrightarrow Sat(\mathcal{S}_1() \wedge_{real} \mathcal{S}_2())$$

2. Wenn eine gegebene Spezifikationen $\mathcal{X}()$ eine andere Spezifikation $\mathcal{Y}()$ impliziert

   $$\mathcal{X}() \Rightarrow \mathcal{Y}()$$

   und die beiden Spezifikationen $\mathcal{S}_1()$ und $\mathcal{S}_2()$ die folgenden Eigenschaften erfüllen,

   $$Sat(\mathcal{S}_1() \wedge_{real} \mathcal{X}()) \quad \wedge \quad \mathcal{Y}() \Rightarrow \mathcal{S}_2()$$

   dann muß folgendes gelten:

   $$Sat(\mathcal{S}_1() \wedge_{real} \mathcal{S}_2())$$

3. Wenn die zwei Spezifikationen $\mathcal{Y}_1()$ und $\mathcal{Y}_2()$ kompatibel sind,

   $$Sat(\mathcal{Y}_1() \wedge \mathcal{Y}_2())$$

   und die beiden Spezifikationen $\mathcal{S}_1()$ und $\mathcal{S}_2()$ die folgenden Eigenschaften erfüllen,

   $$\mathcal{Y}_1() \Rightarrow \mathcal{S}_1() \quad \wedge \quad \mathcal{Y}_2() \Rightarrow \mathcal{S}_2()$$

   dann muß folgendes gelten:

   $$Sat(\mathcal{S}_1() \wedge_{real} \mathcal{S}_2())$$

11.3 Kugeln

Eine einfache Möglichkeit zur Beschreibung von Teilmengen einer Domäne ist die Benennung eines Elements aus der Domäne mit einer maximalen Abweichung. Alle Elemente der Domäne, deren Abstand von dem spezifizierten Wert kleiner als die maximalen Abweichung ist, gelten in diesem Fall als akzeptabel.

Der einfachste Fall einer Kugel ist über eindimensionalen Domänen, in denen die Kugel zu einem Intervall wird. Die Schnittmengenbildung für Intervalle ist trivial über eine einfache Fallunterscheidung zu erledigen. Betrachtet man aber mehrdimensionale

Domänen (zum Beispiel geometrische Daten), wie sie in der Produktentwicklung häufiger vorkommen, so ergeben sich schon Kreise oder Kugeln, deren Schnittmengen nicht trivial zu ermitteln sind und auch selbst keine Kreise oder Kugeln sind. Im Folgenden soll daher die Schnittmengenbildung von n-dimensionalen Kugeln ($n \geq 2$) betrachtet werden. Intervalle sind dabei nicht explizit ausgenommen, können aber sehr viel einfacher als mit den hier beschriebenen Methoden implementiert werden.

Eine Kugel ist eine Umgebung $\overline{B_\epsilon(x)}$ mit $x \in \mathbb{D}_X$ und beschreibt die nicht leere Menge von Elementen aus $\mathbb{D}_X$, so daß gilt: $\overline{B_\epsilon(x)} = \{y \in \mathbb{D}_X : \delta(x,y) \leq \epsilon\}$. Hierbei ist $\delta(x,y)$ der Abstand zwischen den Werten x und y gemessen an dem vorausgesetzten metrischen Abstandsmaß.

Um das Wissen über die Domäne aus der eigentlichen Konjunktionsberechnung herauszuhalten, bietet sich eine Schichtenarchitektur (siehe Abbildung 11.3) an. Die Konvertierungsschicht wandelt Elemente der Domäne in Vektoren um und umgekehrt, damit die unterliegende Berechnungsschicht domänenunabhängig in Termen von Vektoren arbeiten kann. Die beiden Schichten werden im Folgenden getrennt behandelt.

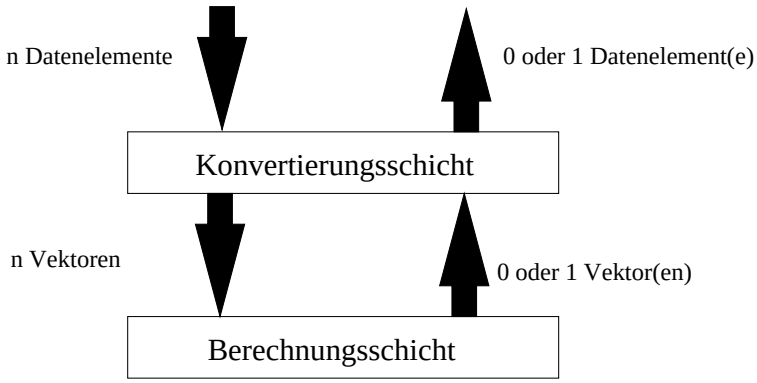


Abbildung 11.3: Schichten der Schnittmengenberechnung

11.3.1 Die Konvertierungsschicht

Die Aufgabe der Konvertierungsschicht ist eine Abbildung von einer Domäne auf die Menge der n-dimensionalen Vektoren von rationalen Zahlen. Dabei müssen die euklidischen Abstände der Vektoren genau die semantischen Abstände der Elemente der Domänen widerspiegeln, so daß Berechnungen auf der Basis der Vektoren durchgeführt werden können. Diese Zuordnung von Datenelementen zu Vektoren und umgekehrt läuft rekursiv ab. Zuerst werden alle elementaren Datentypen auf einen Vektor (im Normalfall 1-dimensional) abgebildet. Zusammengesetzte Typen, wie zum Beispiel Strukturen, werden dann aus den Vektoren der einzelnen Datenelemente zusammengesetzt. Problematisch wird diese Vorgehensweise nur bei Typen mit dynamischer Länge wie zum Beispiel Mengen oder Sequenzen.

Abbildungen zwischen Datentypen und Vektoren

Gesucht wird eine im Idealfall bijektive Abbildung h zwischen den einzelnen Datentypen und Vektoren. Die eindeutige Abbildung von den Datentypen auf die Vektoren muß dabei auf jeden Fall injektiv sein. Die Umkehrabbildung kann allerdings partiell sein; es gibt also zu einem berechneten Ergebnisvektor kein Element der Domäne. Dann muß das nächstliegende Element der Domäne gewählt werden und es muß geprüft werden, ob das Ergebnis trotzdem noch die geforderten Eigenschaften erfüllt.

Die gesuchte Abbildung der Domäne $\mathbb{D}$ auf die Menge der m-dimensionalen Vektoren $\mathbb{V}_m$ muß auf jeden Fall folgende Kriterien erfüllen:

$$\forall_{x\in\mathbb{D}}\ h(x) \in \mathbb{V}_m \wedge \forall_{\vec{v}\in\mathbb{V}_m}\ \exists_{x\in\mathbb{D}}\ h^{-1}(\vec{v}) = x \wedge \forall_{y\in\mathbb{D}}\ \delta(h(x),\vec{v}) \leq \delta(h(y),\vec{v})$$

Die Umkehrfunktion $h^{-1}()$ ist dadurch nicht eindeutig vorgegeben. Es wird nur gefordert, daß es keinen anderen Wert in der Domäne gibt, der ein „besseres“ Resultat abgeben würde.

Die Forderungen an die Abbildungen sollen an dem folgenden Beispiel verdeutlicht werden:

Beispiel 11.3.1 (Abbildung von Ganzen Zahlen auf Vektoren)
Gesucht ist eine Abbildung von der Menge der ganzen Zahlen $\mathbb{Z}$ auf die Menge der eindimensionalen Vektoren $\mathbb{V}_1$ und umgekehrt.

$$h : \mathbb{Z} \rightarrow \mathbb{V}_1, h(i) = [i]$$

Die Abbildung ist einfach und intuitiv aber nicht surjektiv. Damit die Umkehrabbildung vollständig ist, muß daher eine Approximation gefunden werden.

$$h^{-1} : \mathbb{V}_1 \rightarrow \mathbb{Z}, h^{-1}([v]) = round(v)$$

In dieser Variante von h^{-1} wird der Wert einfach auf die nächste Integer-Zahl gerundet. Ein Abschneiden nach dem Komma ist hier nicht erlaubt , da sonst zum Beispiel der Vektor [6.9] *auf* 6 *abgebildet wird, wobei die Zahl* 7 *deutlich näher ist.*

Es sei noch darauf hingewiesen, daß durchaus andere Abbildungen möglich sind. So können zum Beispiel Faktoren in die Funktionen eingebaut werden, oder andere bijektive Funktionen. □

In praktisch allen Fällen ist eine automatische Abbildung von Domänen auf Vektoren zwar möglich, die Frage der „semantischen Nähe“ kann dabei aber oft nicht ausreichend berücksichtigt werden. Allerdings lassen sich für ganze und rationale Zahlen einfache und sinnvolle Abbildungen finden und damit der weitaus größte Teil der Produkteigenschaften bearbeiten. Für die verbleibenden Produkteigenschaften ist eine Herangehensweise mit dem Revisionsmodell zumindest fraglich, so daß diese hier nicht weiter betrachtet werden sollen. Die Konvertierungsschicht ist dennoch sinnvoll. Einerseits können die Domänen auch wie in Beispiel 11.3.1 ganze Zahlen sein, die auf Fließkommazahlen abgebildet werden sollten. Andererseits ist die Schnittmengenbildung für spezielle Fälle auch auf andere Domänen einfach erweiterbar.

11.3.2 Die Berechnungsschicht

Die Berechnungsschicht hat zwei Probleme zu lösen. Für die Konjunktion muß die Schnittmenge von zwei Kugeln berechnet werden, während für die Implikation auf Teilmengenverhältnis getestet werden muß.

Die Lösung des zweiten Problems ist relativ einfach und effizient lösbar: $\overline{B_{\epsilon_1}(x_1)}$ ist Teilmenge von $\overline{B_{\epsilon_2}(x_2)}$, wenn gilt:

$$\epsilon_2 \geq \delta(x_1, x_2) + \epsilon_1$$

Da x_1 und x_2 schon durch die Konvertierungsschicht auf n-dimensionale Vektoren abgebildet wurden, muß hier nur noch die euklidische Distanz der Vektoren berechnet werden.

Die Berechnung der Schnittmengen von zwei Kugeln ist schon deutlich schwieriger, insbesondere sind nur Schnittmengen von Intervallen (eindimensionalen Kugeln) selbst

wieder Intervalle und damit Kugeln. Ab zwei oder mehr Dimensionen ist die Schnittmenge von zwei Kugeln im Allgemeinen nicht wieder eine Kugel. Wie oben schon diskutiert wird eine approximierte Kugel gesucht, so daß diese eine möglichst große Teilmenge der Schnittmenge der beiden ursprünglichen Kugeln beschreibt. Drei mögliche Verfahren zur Berechnung dieser gesuchten Kugeln werden im Folgenden vorgestellt:

Genaue Ergebnisberechnung (GEB)
Zumindest in einigen Fällen ist die genaue Ergebnisberechnung (das heißt die Berechnung der größten in der Schnittmenge einbeschriebenen Kugel) durchaus möglich. Mit Hilfe der vorgegebenen Umgebungen können einige Aussagen in Form von Gleichungen über die neue gesuchte Umgebung getroffen werden. So können zum Beispiel folgende Ungleichungen aufgelöst werden:

$$\forall_{i=1..m}\ \delta(\vec{x},\vec{x}_i) \leq \epsilon_i - \epsilon$$

Alle $\vec{x}_i$ und ϵ_i in der Gleichung sind bekannt, während $\vec{x}$ und ϵ genau die gesuchte Umgebung beschreiben. Das Arbeiten mit Ungleichungen ist leider etwas unschön, so daß nur die Umgebungen zur Berechnung herangezogen werden sollten, die auch einen Einfluß auf die Größe oder Lage der gesuchten Schnittmenge haben.

Um eine n-dimensionale Kugel eindeutig darzustellen, sind $n+1$ Punkte und damit $n+1$ sich schneidende Umgebungen nötig. Aus diesen $n+1$ Umgebungen können $n+1$ quadratische Gleichungen mit $n+1$ Unbekannten erstellt werden. Im allgemeinen Fall steigt die Anzahl der Lösungen mit der Anzahl der Unbekannten. In diesem speziellen Fall sind jedoch die quadratischen Anteile der Unbekannten immer genormt, so daß das entstehende Gleichungssystem höchstens zwei Lösungen hat.

Ein Problem bei dieser Vorgehensweise ergibt sich für Sonderfälle und degenerierte Fälle, die die Implementierung komplexer und damit auch aufwendiger und fehleranfälliger machen. Ein häßlicher Sonderfall soll verdeutlichen, warum so eine genaue Berechnung kaum durchführbar ist.

Beispiel 11.3.2 (Schwer berechenbare Schnittmenge)
Im linken Beispiel in Bild 11.4 wird die gesuchte Umgebung nur von $\mathcal{S}_1()$ und $\mathcal{S}_2()$

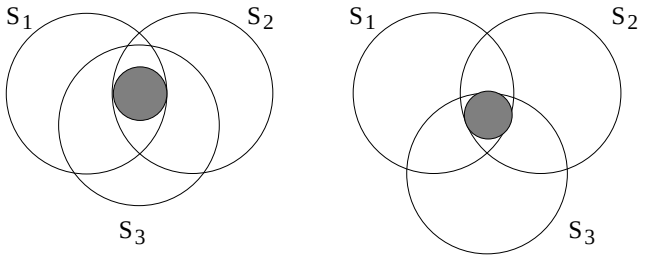


Abbildung 11.4: Schwer berechenbares Beispiel

begrenzt, während im rechten Fall auch $\mathcal{S}_3()$ die gesuchte Umgebung begrenzt. □

Eine rechnerische Unterscheidung der beiden Fälle aus Beispiel 11.3.2 scheint kaum möglich zu sein. Das Problem der Schnittmengensuche kann also nur durch Berechnung beider Fälle und anschließendem Ausprobieren gelöst werden. Im n-dimensionalen Fall müssen dazu die Lösungen für alle Teilmengen aller Mengen mit $n+1$ Umgebungen berechnet werden. Für die Fälle mit weniger Umgebungen (zum Beispiel weniger als vier Kugeln im dreidimensionalen Raum) funktioniert das Verfahren leider gar nicht mehr und ist damit praktisch nicht benutzbar.

Polytop-Approximation der Umgebungen (PAU)

Berechnungen mit Kugeln im n-dimensionalen Raum sind durchaus komplex. Die Schnittmengenbildung kann durch eine Approximation der Kugeln durch Polytope deutlich vereinfacht werden. Im n-dimensionalen Raum wird dabei eine Kugel durch ein Polytop mit 2^n Eckpunkten ersetzt. Alle Kanten des Polytops sind dabei parallel zu einer der Koordinatenachsen. Die Berechnung des Schnittpunktes wird sehr einfach, da es sich nur noch um eine Fallunterscheidung handelt.

Leider ist die Approximation einer Kugel durch einen einbeschriebenen Würfel (siehe Beispiel 11.3.3) relativ schlecht. Eine Verbesserung läßt sich durch eine Erhöhung der Anzahl der Punkte erreichen. So kann im 2-dimensionalen Fall ein Kreis durch ein Sechseck approximiert werden. Dabei wird aber die Schnittmenge komplexer und muß wieder auf ein Kreis oder Sechseck heruntergerechnet werden.

Beispiel 11.3.3 (Schlechte Polytop-Approximation der Umgebungen)
Die Schnittmenge der beiden Quadrate in Bild 11.5, die ja die Kreise approximieren sollen, ist leer, während die Schnittmenge der Kreise nicht leer ist.

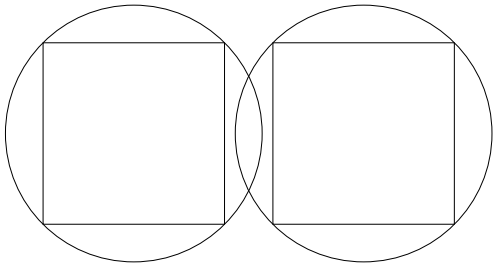


Abbildung 11.5: Schlechte Approximation

□

Im Beispiel ist die reale Schnittmenge von zwei Umgebungen leer, obwohl die eigentlich korrekte Schnittmenge der Umgebungen nicht leer ist. Damit widerspricht dieses Verfahren den Anforderungen an die Spezifikationen und kann nicht verwendet werden.

Einfache Berechnung durch Iteration (EBI)

Im Fall von zwei Kugeln kann die beste Approximation der Schnittmenge durch eine Kugel recht einfach berechnet werden. Wenn $\delta(x_1, x_2) > \epsilon_1 + \epsilon_2$, dann ist die Schnittmenge auf jeden Fall leer. Sei also $\delta(x_1, x_2) \leq \epsilon_1 + \epsilon_2$, $\Delta = \epsilon_1 + \epsilon_2 - \delta(x_1, x_2)$ und $\epsilon_d = \Delta/2$. Wenn x' berechnet werden kann, so daß x' auf der direkten Verbindung zwischen x_1 und x_2 liegt und daß $\delta(x_1, x') = \epsilon_1 - \epsilon_d$ und $\delta(x_2, x') = \epsilon_2 - \epsilon_d$ gilt, dann ist die Umgebung $\overline{B_{\epsilon_d}(x')}$ eine Teilmenge der Schnittmenge. In diesem Fall ist die berechnete Umgebung genau dann die leere Menge, wenn auch die Schnittmenge leer ist. Außerdem gibt es keine andere Umgebung, deren Epsilon größer ist als das von $\overline{B_{\epsilon_d}(x')}$ und die trotzdem eine Teilmenge der Schnittmenge ist.

$$\vec{x} = \left\{ \begin{array}{ll} \mathit{nil} & \delta(\vec{x}_1, \vec{x}_2) > \epsilon_1 + \epsilon_2 \\ \vec{x}_1 & \overline{B_{e_1}(\vec{x}_1)} \subseteq \overline{B_{e_2}(\vec{x}_2)} \quad (\Leftrightarrow \delta(\vec{x}_1, \vec{x}_2) + \epsilon_1 \leq \epsilon_2) \\ \vec{x}_2 & \overline{B_{e_2}(\vec{x}_2)} \subseteq \overline{B_{e_1}(\vec{x}_1)} \quad (\Leftrightarrow \delta(\vec{x}_1, \vec{x}_2) + \epsilon_2 \leq \epsilon_1) \\ \frac{\epsilon_1 + \epsilon_2 - \delta(\vec{x}_2, \vec{x}_1)}{2 * \delta(\vec{x}_2, \vec{x}_1)} * (\vec{x}_2 - \vec{x}_1) & \mathit{sonst} \end{array} \right\}$$

$$\epsilon = \frac{\epsilon_1 + \epsilon_2 - \delta(\vec{x}_2, \vec{x}_1)}{2} = \epsilon_1 - \delta(\vec{x}, \vec{x}_1) = \epsilon_2 - \delta(\vec{x}, \vec{x}_2)$$

Für mehr als zwei Mengen kann die Schnittmenge nun durch mehrere Iterationen berechnet werden, in dem immer zwei der Mengen durch ihre Schnittmengen ersetzt werden, bis nur noch eine Menge vorhanden ist.

Da die einfache Berechnung durch Iteration (EBI) im Fall von zwei Kugeln immer das beste Ergebnis liefert, kann dieses Verfahren hier sehr gut eingesetzt werden. Eine Verbesserung durch andere kompliziertere Verfahren ist nur interessant, wenn bei jeder neu eingebrachten Spezifikation der Peek neu berechnet wird. In der Praxis ist diese Neuberechnung nach jeder *constrain*-Operation allerdings kaum realistisch.

Die inkrementelle Berechnung der Schnittmenge ist durchaus abhängig von der Reihenfolge, in der die Spezifikationen eingebracht wurden. Im normalen Betrieb kann dies zu unerwarteten Situation führen. So kann zum Beispiel auf einer widerspruchsfreien Spezifikationsmenge, also auch auf einem widerspruchsfreien Datenelement, durch Änderung der Evaluierungsreihenfolge ein Widerspruch entstehen (vergleiche Beispiel 11.2.2). Es ist also sicherzustellen, daß alle im System befindlichen Spezifikationen immer in derselben Reihenfolge evaluiert werden.

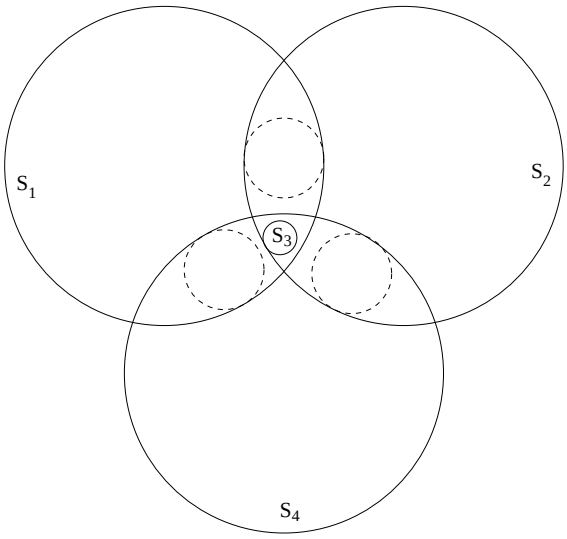


Abbildung 11.6: Widerspruch durch *release*

Leider geht die Relevanz der Reihenfolge so weit, daß möglicherweise das Entfernen einer Spezifikation zu einem Widerspruch führt. Betrachtet man zum Beispiel Abbildung 11.6, so kann mit der Spezifikation $\mathcal{S}_3()$ die Widerspruchsfreiheit erreicht werden, wenn $\mathcal{S}_3()$ in der ersten Berechnung enthalten ist, also zum Beispiel

$$\mathcal{S}_3() \wedge_{real} \mathcal{S}_1() \wedge_{real} \mathcal{S}_2() \wedge_{real} \mathcal{S}_4()$$

Wenn $\mathcal{S}_3()$ aber durch *release* entfernt wird, dann kann durch keine Berechnung eine widerspruchsfreie Spezifikation erhalten werden.

In diesem Fall muß sich das System den berechneten Peek vor der *release*-Operation merken und neue Operationen nur zulassen, wenn diese mit dem gespeicherten Peek kompatibel sind, oder wenn der Peek nach Ausführung der Operation durch die EBI-Berechnung widerspruchsfrei ist. In jedem Fall ist dieses Verhalten ein Nachteil der EBI-Berechnung, der aber mit konstantem Zusatzaufwand für den Entwickler verborgen werden kann.

11.4 Quader

Kugeln haben eine sehr einfache Bedeutung für den Benutzer bei teilweise aufwendiger und ungenauer Berechnung. Eine für den Benutzer ähnlich gute Anwendbarkeit bieten Quader, die zudem viel einfacher zu berechnen sind. Der Gedanke hierbei ist, daß wie bei den Kugeln ein fester Wert vorgegeben ist. Allerdings wird nicht nur eine maximale Abweichung in Form des ϵ spezifiziert, sondern für jede Dimension ein eigenes ϵ. Damit kann zum Beispiel im zweidimensionalen Raum eine Ellipse oder ein Rechteck beschrieben werden. Im Fall der Ellipse wird die Ergebnismengenberechnung nicht gerade einfacher, die Schnittmenge von zwei Rechtecken läßt sich allerdings sehr einfach durch eine Fallunterscheidung berechnen und ist vor allem selbst wieder ein Rechteck.

Ausgedehnt auf den n-dimensionalen Raum wird sozusagen auf jeder Dimension separat ein Intervall spezifiziert, welches dann einen Quader beschreibt. Die Schnittmenge von zwei solchen Quadern kann sehr einfach berechnet werden und ist selbst wieder ein Quader.

Der Nachteil für den Benutzer ist die Spezifikation mehrerer Epsilons und die Tatsache, daß die maximale Abweichung in allen Dimensionen natürlich deutlich größer ist als die maximale Abweichung in einer Dimension. Wird zum Beispiel der Punkt $P_1 = (x, y)$ mit den Epsilons $\epsilon = (\epsilon_x, \epsilon_y)$ angegeben, so befindet sich auch der Punkt $P_2 = (x', y')$ mit $x' = x + \epsilon_x$ und $y' = x + \epsilon_y$ in der Umgebung, obwohl der Abstand von P_2 zu P_1 gleich $\sqrt{\epsilon_x^2 + \epsilon_y^2}$ durchaus größer als $MAX(\epsilon_x, \epsilon_y)$ ist.

Insbesondere wenn die Epsilons für die unterschiedlichen Dimensionen einfach anzugeben sind, wie dies zum Beispiel bei räumlichen Daten der Fall ist, bieten Quader eine gute Möglichkeit von Spezifikationen auf Datenelementen. Wenn die Epsilons schlecht anzugeben sind (zum Beispiel bei Sequenzen mit variabler Länge), ist ein anderes Verfahren durchaus sinnvoller.

Da zweidimensionale Quader, also Rechtecke, im später zu diskutierenden Demonstrator verwendet werden, müssen hier noch genauere Algorithmen für die in Abschnitt 5.2 identifizierten Operationen der Konjunktion, Implikations- und Widerspruchsfreiheitsentscheidung angegeben werden. Zweidimensionale Quader werden im Folgenden mit zwei Punkten $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ bezeichnet (mit $x_1 < x_2$ und $y_1 < y_2$). Das aus diesen Punkten resultierende Rechteck ist dann das Quadrupel $[x_1, y_1, x_2, y_2]$

- Die Erfüllbarkeit eines Rechtecks $[x, y, z, w]$ ist gegeben, wenn gilt: $x \leq z \wedge y \leq w$
- Die Konjunktion einer Menge von Rechtecken ist abgeschlossen, kommutativ und assoziativ und kann sehr einfach mit linearem Aufwand berechnet werden.

  Die Konjunktion der Rechtecke $[x_1, y_1, z_1, w_1]$ bis $[x_n, y_n, z_n, w_n]$ ist das Rechteck $[MAX(x_1, ...x_n), MAX(y_1, ...y_n), MIN(z_1, ...z_n), MIN(w_1, ...w_n)]$. Die MAX- und MIN-Berechnungen können durch $n-1$ Vergleiche erledigt werden und sind somit linear in der Anzahl der Rechtecke.
- Das Rechteck $[x_1, y_1, z_1, w_1]$ impliziert $[x_2, y_2, z_2, w_2]$, wenn gilt: $x_1 \geq x_2 \wedge y_1 \leq y_2 \wedge z_1 \geq z_2 \wedge w_1 \leq w_2$
- Die minimale Konfliktmenge $\mathbb{C}$ für $(\mathbb{S}, \mathcal{S}())$ kann analog zu der von Intervallen (siehe Kapitel 5) berechnet werden, in dem jedes einzelne Rechteck, das mit $\mathcal{S}()$ eine leere Schnittmenge hat, in $\mathbb{C}$ aufgenommen wird. Diese Berechnung ist mit linearem Zeitaufwand möglich und liefert eine eindeutige minimale Konfliktmenge.

Beispiel 11.4.1 (Schnittmengenberechnung von Rechtecken)
In Abbildung 11.7 ist die Schnittmenge von drei Rechtecken dargestellt. Dabei ist zum Beispiel die X-Koordinate von P_a genau das Maximum der X-Koordinaten von P_{1a}, P_{2a} und P_{3a}.
□

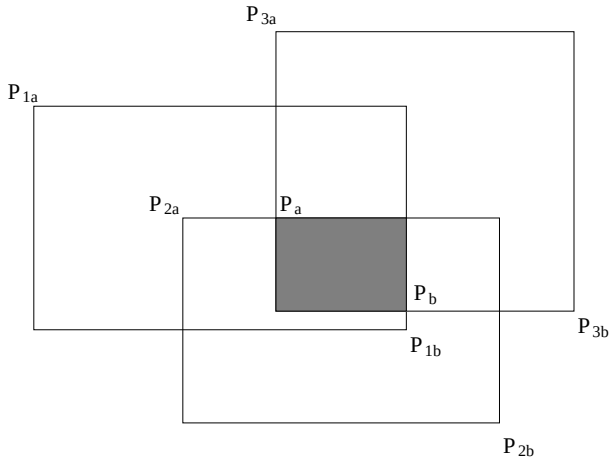


Abbildung 11.7: Die Schnittmenge von drei Rechtecken

Der Aufwand für die Berechnungen mit den Rechtecken ist also nur um den konstanten Faktor 2 höher als die Berechnungen für Intervalle (siehe Abschnitt 5.4). Diese Tatsache kann analog zu der geführten Diskussion auf n-dimensionale Quader ausgedehnt werden, soll aber hier nicht explizit gezeigt werden.

11.5 Fuzzy-Mengen

Fuzzy-Mengen bieten eine gute Möglichkeit der Beschreibung von Mengen und damit natürlich auch von akzeptablen Teilmengen der Domänen. Zusätzlich bieten Fuzzy-Mengen die Möglichkeit, für jedes Element einen Zugehörigkeitsgrad festzulegen. Damit kann der Grad festgelegt werden, zu dem ein Element in der Menge enthalten ist. Die Beschreibung kann somit viel mehr Informationen als eine bloße Teilmenge, nämlich auch unvollkommene Daten, enthalten. Im SFB 346 treten unvollkommene Daten des öfteren auf und werden über Fuzzy-Mengen repräsentiert und weiterverarbeitet. In der Literatur sind Fuzzy-Mengen und deren Schnitte üblicherweise wie folgt definiert ([Bie99], [Zad65], [KGK93] und [KY95]):

Definition 11.5.1 (Fuzzy-Menge)
Gegeben sei eine Funktion μ von der Referenzmenge Ω in das Einheitsintervall:

$$\mu : \Omega \mapsto [0, 1].$$

Die durch diese Funktion definierte Menge $\mathbb{H}$ ist eine **Fuzzy-Menge** *oder unscharfe Menge auf Ω.*

□

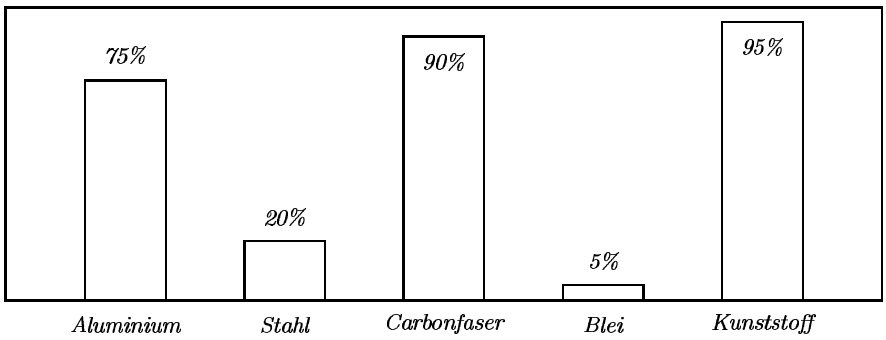
Die Definition der Fuzzy-Mengen basiert auf der Referenzmenge Ω, über die nicht allzuviele Aussagen gemacht wird, insbesondere wird nichts über die Kardinalität von Ω gesagt. So kann Ω eine kontinuierliche oder eine diskrete Menge sein. Wenn Ω eine diskrete Menge ist, so kann Ω endlich oder unendlich sein, während kontinuierliche Referenzmengen immer unendlich sind. Wenn Ω eine endliche Menge ist, so wird auf Ω keine Ordnung vorausgesetzt.

In einigen Arbeiten wird dem Benutzer der Umgang mit Fuzzy-Mengen durch *linguistische Begriffe* vereinfacht. So kann zum Beispiel der Begriff „groß“ im Zusammenhang mit Menschen durch eine Fuzzy-Menge dargestellt werden. Ein spezielles *Wörterbuch* ordnet dabei jedem Begriff der Miniwelt eine Fuzzy-Menge zu und umgekehrt. Auf solchen Begriffen können nun logische Verknüpfungen, genannt Formeln, aufgebaut werden

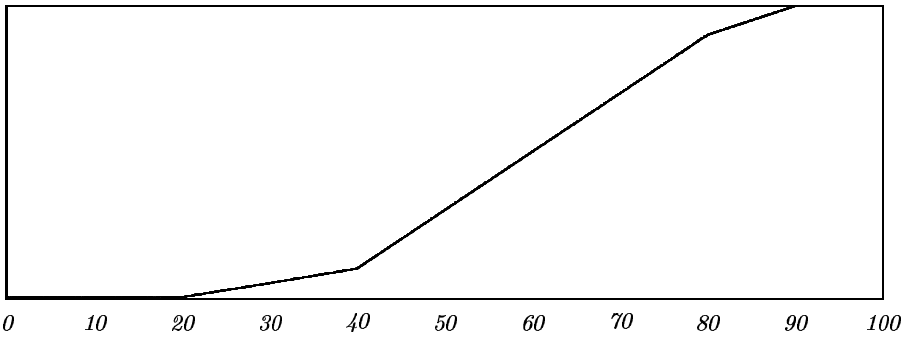
(zum Beispiel [ERZD95], [Wit95] und [Bos96]). Da der Umgang mit Fuzzy-Mengen, insbesondere die Eingabe einer Fuzzy-Menge, für den Benutzer einigermaßen kompliziert erscheint, ist der beschriebene Ansatz sicherlich sinnvoll. Allerdings ist die Schnittmengenbildung im RV-Modell auf einer sehr tiefen Ebene angesiedelt und kann daher nicht auf ein externes Wörterbuch zugreifen. Für die Schnittmengenbildung können also einzig die Fuzzy-Mengen selbst verwendet werden.

Aufgrund der großen Unterschiedlichkeiten muß hier zwischen den endlichen und unendlichen Fuzzy-Mengen unterschieden werden. In vielen Arbeiten (zum Beispiel [Wit95], [Bos96]) werden endliche diskrete Fuzzy-Mengen, nicht zuletzt wegen der einfachen Verarbeitung als α-Schnitte, deutlich bevorzugt. Unendliche Fuzzy-Mengen werden üblicherweise durch parametrisierte Funktionen repräsentiert ([KGK93]), deren weitere Verarbeitung nicht trivial ist.

Beispiel 11.5.1 (Endliche und unendliche Fuzzy-Mengen)
Als Beispiel für eine endliche Fuzzy-Menge soll das Konzept eines leichten Materials für die Greiferbacken dienen. Dabei gibt es eine Reihe von Materialien, die jeweils im Hinblick auf ihr Gewicht bewertet werden. Hierbei wird recht schnell klar, daß der Vorteil einer Fuzzy-Menge in den unterschiedlichen Zugehörigkeitsgraden liegt. Scharfe Mengen würden in diesem Beispiel sicherlich keinen Sinn machen.



Das zweite Beispiel soll den Begriff „laut“ an sich definieren. Dabei wird die Zugehörigkeit einer bestimmten Lautstärke in dB zu der Menge der lauten Geräusche charakterisiert. Auch hier wird schnell klar, warum die bekannte Mengenlehre da nicht mehr weiter hilft. Eine Grenze wie zum Beispiel die Aussage „Ein Geräusch ist laut, wenn es mehr als 62dB hat“ ist bei weitem zu ungenau. Die scheinbare Obergrenze von 100 dB ist darstellungsbedingt und bedeutet nur, daß für jedes Geräusch über 100 dB der Zugehörigkeitsgrad zur Fuzzy-Menge gleich Eins ist.



□

Das RV-Modell benötigt zum Umgang mit Fuzzy-Mengen als Spezifikationen eine Schnittmengenbildung und die Beantwortung der Frage der Implikation beziehungsweise ob eine Fuzzy-Menge Teilmenge einer anderen ist. Die Schnittmenge von zwei

Fuzzy-Mengen ist üblicherweise als die Fuzzy-Menge definiert, deren Zugehörigkeitsgrade genau das Minimum der Zugehörigkeitsgrade der zu schneidenden Fuzzy-Mengen ist. Damit lassen sich beide Berechnungen im diskreten endlichen Fall sehr einfach durchführen. Fuzzy-Mengen bieten daher von den hier vorgestellten Konzepten die größte Mächtigkeit bei effizienten Bearbeitungsmöglichkeiten. Die Nachteile der Fuzzy-Mengen liegen vielmehr in der Benutzerführung. Viele Benutzer kennen keine Fuzzy-Mengen und müssen im Umgang mit diesen erst geschult werden. Außerdem ist die Eingabe von Fuzzy-Mengen deutlich komplizierter als einfache Werte mit maximaler Abweichung. Linguistische Wörterbücher können dies zwar beheben, verursachen aber einen erhöhten Implementierungs- und Laufzeitaufwand. Fuzzy-Mengen als Spezifikationen sind vor allem dann interessant, wenn die Unterstützung von Fuzzy-Mengen im System sowieso vorhanden oder zumindest erwünscht ist.

11.6 Zusammenfassung

Für eindimensionale Domänen sind die in der Arbeit unterstellten Intervalle eine sehr gute und intuitive Spezifikationsform. Für mehrdimensionale Daten, wie zum Beispiel Geometriedaten, müssen stattdessen Kugeln oder Quader verwendet werden. Kugeln entsprechen eher der gewünschten Semantik im Sinn einer maximalen Abweichung von einer bestimmten geometrischen Position, sind aber dafür sehr komplex in der Berechnung. Quader hingegen sind sehr einfach zu verarbeiten, haben aber den Nachteil, daß die maximale Abweichung von dem Mittelpunkt des Quaders größer als die jeweils für eine Dimension geltenden Beschränkungen ist. Wenn den Entwickler dieser Nachteil aber bekannt ist, dann können die damit umgehen und Quader sind somit eine effizient zu verarbeitende Möglichkeit der Spezifikationsformen für mehrdimensionale Domänen.

Fuzzy-Mengen fügen sich direkt in das Revisionsmodell ein, werden sogar schon in anderen Bereichen für ähnliche Herangehensweisen verwendet (siehe Abschnitt 3.2.1.4 oder [Wit02b]). Der Nachteil von Fuzzy-Mengen liegt in der Benutzerführung. Damit werden Fuzzy-Mengen erst interessant, wenn diese im System sowieso verwendet werden (siehe auch [AO97], [AS99] und [ERY+95]). In diesem Fall ist dann auch eine Entscheidung für die Art der Fuzzy-Mengen gefallen und effiziente Verarbeitungsmöglichkeiten müssen zur Verfügung stehen (siehe zum Beispiel [Lan97]), auf die sich dann das hier entwickelte Revisionsmodell stützen kann.

Kapitel 12

Der Demonstrator

In diesem Kapitel wird der im Rahmen der Arbeit entwickelte Demonstrator beschrieben. Dabei werden zuerst einige grundsätzliche Probleme diskutiert und die verwendete Architektur vorgestellt. Die Ziele bei der Realisierung des Demonstrators waren ein praktischer Beweis für die Implementierbarkeit des entwickelten Modells und die anschauliche Darstellung der Funktionsweise und der Mächtigkeit des Revisionsmodells. Außerdem soll der Demonstrator die Grundlage für die nachfolgende Evaluierung der Performanz bilden.

Das zentrale Ergebnis der Arbeit ist das Revisionsmodell und das darauf aufbauende allgemeine Demarkationsprotokoll für die Abkopplung. Diesem Fokus der Arbeit sollte beim Demonstrator Rechnung getragen werden, in dem vor allem die Datenbasis, also der Dienstgeber, mit viel Funktionalität ausgestattet wurde. Auch in Anbetracht der vielen unterschiedlichen Werkzeuge, die für die Unterstützung des Produktentwicklungsprozesses nötig sind, wurde für den Klient nur ein Mindestmaß an Aufwand investiert.

Da Arbeitsteilung und Abkopplung zwei essentiell wichtige Eigenschaften der Rechnerunterstützung für den Produktentwicklungsprozeß sind, sollte auf jeden Fall das allgemeine Demarkationsprotokoll realisiert werden. Da die Verallgemeinerung zum abstrakten Modell keine größeren Schwierigkeiten bereitet und der abstrakte Datentyp zu einer Implementierung einlädt, wurde das abstrakte RV-Modell aus Kapitel 10 realisiert.

Für die Arbeitsteiligkeit wurden Entwurfsentscheidungen mit garantierter Atomizität und Beständigkeit realisiert. Allerdings wurde die Liest-von-Beziehung außer Acht gelassen. Das heißt, eine ungültig gewordene Spezifikation wird damit automatisch zu einer Annahme. Spezifikationen werden daher nur durch *constrain* eingebracht und nicht durch *premise* oder *assume*.

12.1 Allgemeine Fragen der Realisierung des Revisionsmodells

Bevor der Demonstrator selbst vorgestellt wird, sind noch ein paar allgemeine Anmerkungen zu einer Realisierung des Revisionsmodells mit den Erweiterungen für die Abkopplung zu machen. Dabei fällt insbesondere das Schema auf, das nicht mehr Werte sondern Spezifikationen beinhaltet und zum anderen die internen Datenstrukturen, die gerade mit den Absichtssperren und den Entwurfsentscheidungen etwas komplizierter sind, als die in Standard-Datenbanken.

12.1.1 Das Schema

Das Schema, wie es aus Standard-Datenbanken bekannt ist, muß hier einige Änderungen erfahren, da in den Datenelementen nicht mehr Elemente einer dem Datenelement zugeordneten Domäne, sondern vielmehr Spezifikationen gespeichert werden. Die Domäne eines jeden Datenelements muß somit die Domäne der Spezifikationen sein, die allerdings noch weiter unterscheidbar ist.

Spezifikationen können nach ihrer Struktur (Kugeln, Constraints, Fuzzy-Mengen, et cetera) oder nach ihrem Inhalt beziehungsweise der unterliegenden Domäne unterschieden werden. Aus der Sicht der Korrektheit des Systems und damit des Entwurfsprozesses ist die Struktur der Spezifikationen irrelevant. Die den Spezifikationen zugrundeliegende Domäne ist allerdings sehr wohl interessant und sollte daher statisch festgelegt werden. Die Struktur der Spezifikationen auf den jeweiligen Datenelementen kann dynamisch festgelegt werden. Die erste Spezifikation auf einem Datenelement legt somit die Struktur fest und alle weiteren Spezifikationen sind nur erlaubt, wenn deren Struktur kompatibel ist. Dabei ist es durchaus möglich, daß Spezifikationen mit unterschiedlichen Strukturen kompatibel sind, wenn die Strukturen ineinander konvertiert werden können. Scharfe Werte oder Kugeln können zum Beispiel in Constraints konvertiert werden und sind daher mit diesen kompatibel.

Für den Entwickler soll das Schema nur die den Spezifikationen unterliegenden Domänen, nicht jedoch die Spezifikationen selbst oder sogar deren Struktur enthalten. Ein ganz einfacher Schema-Übersetzer kann das bekannte Datenbankschema, in dem jedem Datenelement genau eine Domäne zugeordnet ist, auf das hier nötige Modell der Spezifikationen umsetzen. Da die Beziehung zwischen den beiden Schemata trivial ist, kann die Umsetzung allerdings auch zur Laufzeit geschehen, so daß das Schema nur die bekannten Domänen enthält und das System eben weiß, daß nicht Werte sondern Spezifikationen abgespeichert werden.

Die interne Speicherung der Spezifikationen ist um einiges aufwendiger als das Schema vermuten läßt. Zum Einen wird nicht nur eine Spezifikation pro Datenelement, sondern gleich eine ganze Menge von Spezifikationen pro Datenelement verwaltet. Zudem bietet sich intern die Verwaltung des aktuellen Peeks für alle Datenelemente an, so daß dieser nicht immer neu berechnet werden muß. Diese interne Speicherung kann dem Anwender allerdings verborgen bleiben und muß daher nicht im Schema auftauchen. Die einzige im Schema notwendige Information ist die Domäne des jeweiligen Datenelements, so daß das Schema zumindest aus Entwicklersicht nicht geändert werden muß.

12.1.2 Operationen

Um die Datenstrukturen festlegen zu können, muß mehr über den Ablauf der Operationen bekannt sein, um die Zugriffspfade auf die Daten erkennen zu können. Daher sind hier die Operationen des abstrakten Revisionsmodells mit dem allgemeinen Demarkationsprotokoll in Pseudo-Code aufgeführt, gefolgt von der Analyse der Zugriffspfade auf die persistenten Daten.

Mit jedem Datenelement wird der aktuelle Peek, also die Kombination aller Spezifikationen auf dem Datenelement verwaltet. Wenn Spezifikationen an einem Datenelement zurückgesetzt werden, dann wird der Peek nur invalidiert und beim nächsten Zugriff neu berechnet. Ein Zugriff auf den Peek kann also immer auch einen Zugriff auf die Spezifikationen des Datenelements verursachen.

combine: Siehe Abschnitt 10.1 und 6.2 (für *constrain*)

```
Wenn es keine aktuelle Entwurfsentscheidung gibt
  FEHLER: fehlende Entwurfsentscheidung
End
Wenn der aktuelle Peek nicht gespeichert ist
```

```
  Berechne den aktuellen Peek
End
Berechne Kombination aus Peek und einzubringender Spezifikation
Wenn Kombination nicht erfüllbar
  Berechne eine minimale Konfliktmenge
  Gebe die minimale Konfliktmenge zurück
  FEHLER: Widerspruch
End
REMARK: Möglichkeit 1: Einbringen durch S-Sperre
Lese alle lokalen S-Sperren des Datenelements
Wenn der SPeek die einzubringende Spezifikation impliziert
  Verbinde die Spezifikation mit den S-Sperren
  Füge Spezifikation zur aktuellen Entwurfsentscheidung hinzu
  Verbinde die Entwurfsentscheidung mit den S-Sperren
  Füge Spezifikation zum Datenelement hinzu
  Aktualisiere den Peek des Datenelements
  FERTIG
End
REMARK: Möglichkeit 2: Einbringen durch X-Sperre
Wenn es eine lokale X-Sperre auf dem Datenelement gibt
  Berechne Kombination aus Spezifikation, X-Sperre und Peek
  Wenn Kombination erfüllbar
    Füge Spezifikation zum Datenelement hinzu
    Aktualisiere den Peek des Datenelements
    Verbinde die Spezifikation mit der X-Sperre
    Verbinde die Entwurfsentscheidung mit der X-Sperre
    FERTIG
  End
End
FEHLER: keine ausreichende Absichtssperre
```

Die Zugriffspfade einer *combine*-Operation gehen also über einen Direktzugriff an die Datenelemente und von dort zum Peek, den Spezifikationen und den Absichtssperren. Zusätzlich ist noch ein Direktzugriff auf die Entwurfsentscheidung nötig.

***peek*:** Siehe Abschnitt 6.2

```
Wenn der aktuelle Peek nicht gespeichert ist
  Berechne den aktuellen Peek
End
Gebe den aktuellen Peek zurück
FERTIG
```

Für die *peek*-Operation gibt es nur einen Direktzugriff auf das Datenelement und von dort auf den Peek desselben.

***undo*:** Siehe Abschnitt 7.2 und 6.2 (für *release*)

```
Lese die Entwurfsentscheidung
Wenn die Entwurfsentscheidung nicht abgeschlossen ist
  FEHLER: nicht abgeschlossen
End
Wenn der Aufrufer nicht für die Entwurfsentscheidung
                                        zuständig ist
  FEHLER: nicht zuständig
End
Für alle Spezifikationen in der Entwurfsentscheidung
  Entferne die Spezifikation aus dem Datenelement
  Invalidiere den Peek des Datenelements
End
Lösche die Entwurfsentscheidung
FERTIG
```

Für die *undo*-Operation sind Direktzugriffe auf die Entwurfsentscheidung und auf

alle davon betroffenen Spezifikationen nötig. Für jede einzelne Spezifikation wird dann zusätzlich noch auf das Datenelement und den Peek zugegriffen.

setX: Siehe Abschnitt 8.6

```
Wenn keine globale Operation ausgeführt werden kann
  FEHLER: Kein Quorum
End
Wenn eine X-Sperren auf dem Datenelement existiert
  FEHLER: nicht möglich
End
Lese alle S-Sperren auf dem Datenelement
Lese den Peek des Datenelements
Berechne die Kombination aller S-Sperren mit dem Peek
Wenn die X-Sperre diese Kombination nicht impliziert
  FEHLER: nicht möglich
End
Wenn die X-Sperre nicht erfüllbar ist
  FEHLER: nicht erfüllbar
End
Setze die X-Sperre
FERTIG
```

Für die *setX*-Operation ist ein Direktzugriff auf das Datenelement mit weiteren Zugriffen auf alle Absichtssperren des Datenelements und dem Peek nötig. Zusätzlich ist ein Direktzugriff nötig, um die neue Absichtssperre zu setzen.

setS: Siehe Abschnitt 8.6

```
Wenn keine globale Operation ausgeführt werden kann
  FEHLER: Kein Quorum
End
Wenn der aktuelle Peek nicht gespeichert ist
  Berechne den aktuellen Peek
End
Wenn eine X-Sperre existiert
  Wenn die X-Sperre die S-Sperre nicht impliziert
    FEHLER: nicht möglich
  End
End
Lese alle S-Sperren auf dem Datenelement
Berechne die Kombination aller S-Sperren, der neuen S-Sperre
                                                  und dem Peek
Wenn diese Kombination nicht erfüllbar ist
  FEHLER: nicht möglich
End
Setze die S-Sperre
FERTIG
```

Die Zugriffspfade einer *setS*-Operation gehen genau wie bei *setX* über einen Direktzugriff auf das Datenelement zu den Absichtssperren und dem Peek. Auch hier wird die Absichtssperre mit einem Direktzugriff gesetzt.

releaseX, ***releaseS***: Siehe Abschnitt 8.6

```
Markiere die Absichtssperre als freizugeben
Wenn keine globale Operation ausgeführt werden kann
  FEHLER: Kein Quorum
End
Lese die Absichtssperre und alle damit verbundenen
                              Entwurfsentscheidungen
Für alle dieser Entwurfsentscheidungen
  Wenn die Entwurfsentscheidung nicht abgeschlossen ist
    FERTIG
  End
End
L = leere Liste von Entwurfsentscheidungen
```

```
Für alle Entwurfsentscheidungen, die durch die Absichtssperre
                                                    maskiert sind
  Wenn die Entwurfsentscheidung noch nicht globalisiert wurde
    Markiere die Entwurfsentscheidung als globalisiert
    Für die Entwurfsentscheidung zu L hinzu
  End
End
Gebe die Absichtssperre in einer globalen Operation frei und
                                          globalisiere die
                                  Entwurfsentscheidungen in L
FERTIG
```

Für die Freigabe einer Absichtssperre ist ein Direktzugriff auf die Absichtssperre nötig. Von dort wird auf alle mit der Absichtssperre verbundenen Entwurfsentscheidungen zugegriffen.

begin: Siehe Abschnitt 7.2

```
Generiere einen eindeutigen Bezeichner für die Entwurfsentsch.
Lege die Entwurfsentscheidung an
Markiere die Entwurfsentscheidung als aktiv
Gebe den Bezeichner zurück
FERTIG
```

Für eine *begin*-Operation ist nur ein Direktzugriff auf eine neue Entwurfsentscheidung nötig.

abort: Siehe Abschnitt 7.2

```
Lese die Entwurfsentscheidung
Wenn die Entwurfsentscheidung abgeschlossen ist
  FEHLER: schon abgeschlossen
End
Für alle Spezifikationen in der Entwurfsentscheidung
  Entferne die Spezifikation aus dem Datenelement
  Invalidiere den Peek des Datenelements
End
Für alle mit der Entwurfsentscheidung verbundenen
                                    Absichtssperren
  Löse die Verbindung
  Wenn die Absichtssperre als freizugeben markiert ist
    Rufe auf dieser Absichtssperre releaseX oder releaseS auf.
  End
End
Lösche die Entwurfsentscheidung
FERTIG
```

Die Zugriffspfade einer *abort*-Operation gehen direkt auf eine Entwurfsentscheidung und von dort über deren Spezifikationen auf die Datenelemente bis hin zum Peek der Datenelemente. Zusätzlich wird noch von der Entwurfsentscheidung auf die damit verbundenen Absichtssperren zugegriffen.

commit: Siehe Abschnitt 7.2

```
Markiere die Entwurfsentscheidung als abgeschlossen
Für alle mit der Entwurfsentscheidung verbundenen
                                    Absichtssperren
  Löse die Verbindung
  Wenn die Absichtssperre als freizugeben markiert ist
    Rufe auf dieser Absichtssperre releaseX oder releaseS auf.
  End
End
Lösche die Entwurfsentscheidung
FERTIG
```

Für die *commit*-Operation ist ein Direktzugriff auf die Entwurfsentscheidungen nötig. Zusätzlich wird noch von der Entwurfsentscheidung auf die damit verbundenen Absichtssperren zugegriffen.

12.1.3 Die Datenstrukturen

Nach der Diskussion der Operationen müssen fünf zentrale Entitäten verwaltet werden: Datenelemente, Peeks, Spezifikationen, Absichtssperren und Entwurfsentscheidungen. Hinzu kommen noch die Operationen selbst, die allerdings im Fall von lokalen Operationen keinerlei Speicherung bedürfen. Globale Operationen hingegen müssen auch nach deren Ausführung persistent gespeichert werden, um zeitweise abgekoppelten Replikaten diese nachträglich übermitteln zu können. Im Folgenden wird davon ausgegangen, daß globalen Operation mit allen assoziierten Daten vor der Verteilung an andere Replikate in einer eigenen, hier nicht behandelten Datenstruktur abgelegt werden. Die einzigen Zugriffe auf diese globalen Operationen mit den assoziierten Daten sind sequentielles Lesen sobald ein Quorum zur Verfügung steht. Danach werden die ausgeführten globalen Operationen in der Historie gespeichert und nur noch für die Ankopplung anderer veralteter Replikate ausgelesen (mit Direktzugriff und anschließendem sequentiellem Lesen bis zum Ende der Historie). Die Speicherung der globalen Operationen ist damit klar und für die folgende Diskussion nicht von Interesse.

Vier der zu verwaltenden Entitäten und die Zugriffspfade für die Operationen sind in Abbildung 12.1 aufgezeigt. Der Peek ist dabei nicht aufgeführt, da auf diesen immer nur über das Datenelement zugegriffen wird. Die Speicherung des Peeks sollte also auf jeden Fall in Verbindung mit dem Datenelement erfolgen.

Abbildung 12.1 kann zum Beispiel an der Operation *combine* erklärt werden. Diese führt einen Direktzugriff auf ein Datenelement durch, was durch den von außen kommenden Pfeil an das Datenelement ausgedrückt wird. Danach wird noch auf die zu dem Datenelement gehörenden Spezifikationen und Absichtssperren zugegriffen. Diese Zugriffe sind durch die Pfeile vom Datenelement zu den Spezifikationen und den Absichtssperren ausgedrückt. Alle anderen Pfeile und Beschriftungen entsprechen vergleichbaren Zugriffen der anderen Operationen.

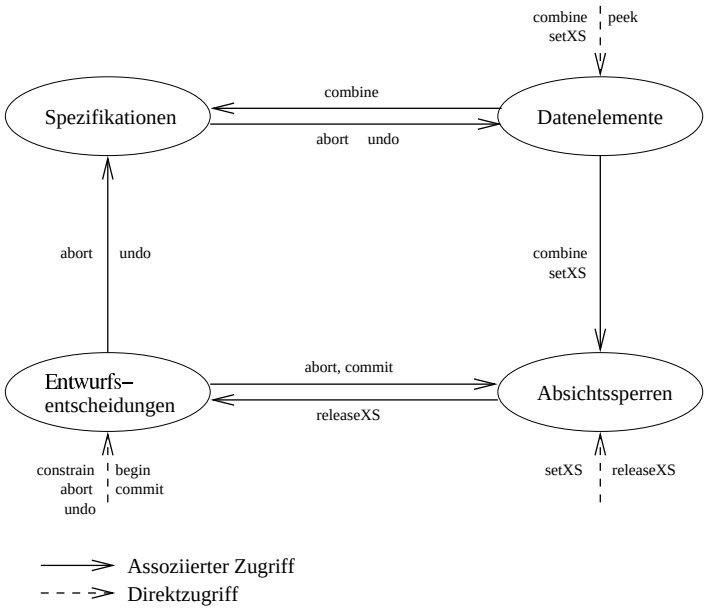


Abbildung 12.1: Zugriffspfade auf die Datenstrukturen

Zusätzliche Informationen für den Aufbau der Datenstrukturen bieten die Beziehungen zwischen den Entitäten und deren Kardinalitäten. Diese sind in Abbildung 12.2 dargestellt. Daraus geht hervor, daß Datenelemente und Entwurfsentscheidung keiner anderen Entität zugeordnet werden können und daher auch eigene unabhängige Datenstrukturen erhalten müssen. Jede Absichtssperre ist eindeutig einem Datenelement zugeordnet und sollte daher auch zusammen mit den Datenelementen gespeichert werden. Eine Spezifikation hingegen ist sowohl einem Datenelement als auch einer Entwurfsentscheidung zugeordnet und kann daher auch in beiden Strukturen verwaltet werden. Für die Speicherung der Spezifikationen an den Entwurfsentscheidungen sprechen nur die Zugriffspfade der Operationen *undo* und *abort*, die nicht zu oft ausgeführt werden sollten. Umgekehrt geht der Zugriffspfade der Operation *combine* von den Datenelementen zu den Spezifikationen. Da der weitaus größte Teil der Entwicklungsarbeit mit *combine*-Operationen durchgeführt wird, ist hier eine Speicherung der Spezifikationen mit den Datenelementen angezeigt. Dafür sprechen auch die Zugriffe auf den Peek eines Datenelements durch die Operationen *peek*, *setX* und *setS*, für die im Fall der Neuberechnung des Peeks ebenfalls Zugriffe von den Datenelementen auf die Spezifikationen durchgeführt werden.

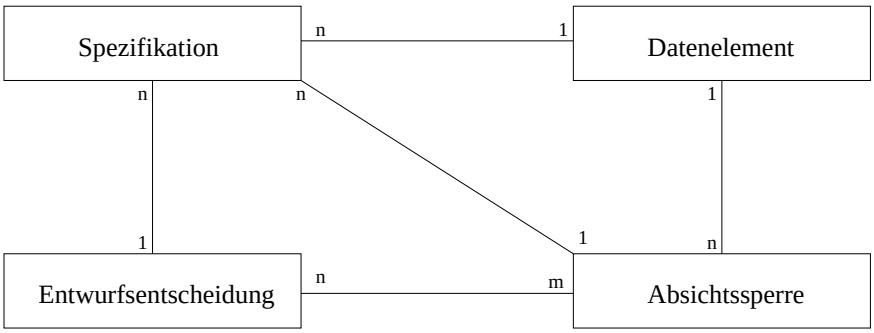


Abbildung 12.2: Kardinalitäten der Beziehungen zwischen den Datenstrukturen

Die Entwurfsentscheidungen werden also in einer eigenen Datenstruktur, der sogenannten *persistenten Entwurfsentscheidungstabelle* (*PET*), gespeichert. Diese *PET* ist äquivalent zur transienten Transaktionstabelle in herkömmlichen Datenbanken, allerdings müssen die Informationen in der *PET* auch nach Abschluß der Entwurfsentscheidung persistent bleiben, um die spätere Rücksetzung einer Entwurfsentscheidung durchführen zu können.

12.2 Architektur des Demonstrators

Da das entwickelte Modell für die Rechnerunterstützung der Produktentwicklung ein modifiziertes Datenbankmodell ist, muß der Demonstrator dieses widerspiegeln, in dem vor allem eine derartige Datenbank realisiert wird. Diese wird im Folgenden RV-Modell-DBMS oder *RV-DBMS* genannt. Da die im Produktentwicklungsprozeß verwendeten Werkzeuge sehr zahlreich sind und auch sehr spezielle Funktionalität haben, muß sich die Arbeit hier auf ein kleines Beispiel beschränken, mit dem anschaulich die Mächtigkeit des Revisionsmodells gezeigt wird.

Die geforderte Atomizität der Entwurfsentscheidungen führt sehr schnell zu der Idee, ein unterliegendes Standard-DBMS zu verwenden. Dabei wird zusätzlicher Aufwand für die Umsetzung der Besonderheiten des RV-Modells, wie zum Beispiel die Operationen *combine* und *peek*, die Entwurfsentscheidungen und die Absichtssperren nötig. Auch die Implementierung der *undo*-Operation, die sicherlich in dieser Form von keiner Datenbank unterstützt wird, bedeutet zusätzlichen Aufwand, so daß das DBMS praktisch nur noch als Data-Manager verwendet wird.

12.2.1 Unterliegendes Modell und Design

Der Demonstrator benutzt ein bestehendes DBMS als Basis für die Implementierung. Da Entwurfsentscheidungen im Gegensatz zu Transaktionen keinerlei Isolation benötigen, werden einzelne Zugriffe auf die Datenbasis in Transaktionen verpackt, die die Atomizität und Dauerhaftigkeit der einzelnen Operationen garantieren. Das DBMS wird damit als reiner Data Manager benutzt, auf den dann ein Constraint Manager aufgesetzt wurde. Auf diesem Constraint Manager aufbauend wurde der neu implementierte Scheduler und der Entwurfsentscheidungs-Manager aufgesetzt. Abbildung 12.3 zeigt dir Architektur für die Implementierung des Demonstrators ohne Replikation und Abkopplung.

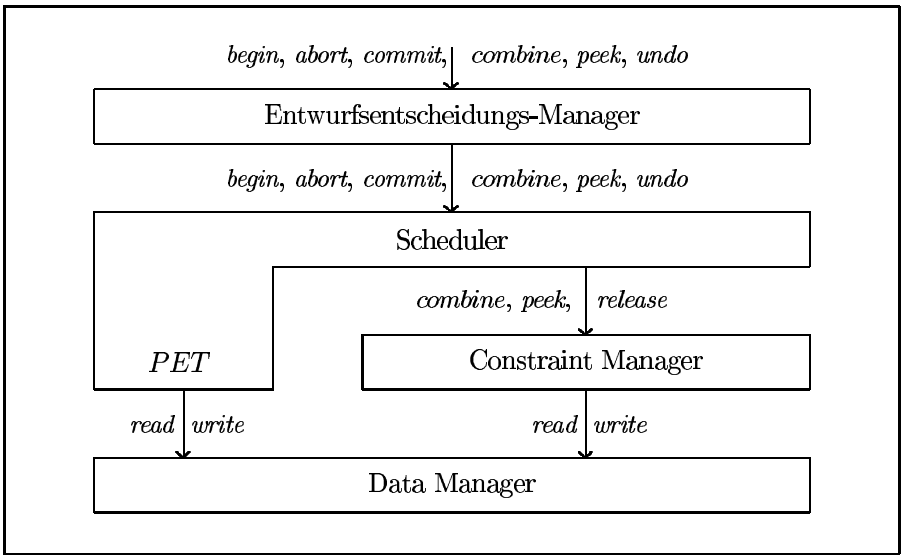


Abbildung 12.3: Architektur für das replizierte RV-Modell

Die Neuimplementierung des Schedulers hat zwei wichtige Vorteile: Die *PET* erledigt die komplette Verwaltung von Entwurfsentscheidungen, wodurch eine transiente Transaktionstabelle hinfällig wird. Zum Anderen kann der Scheduler in Termen von Constraint-Operationen arbeiten, die ja möglicherweise aus mehreren einzelnen Schreib- und Lese-Operationen bestehen. Wenn der Scheduler eines bestehenden DBMS eingesetzt wird, so werden also für eine *combine*-Operation mehrere Sperren gesetzt, wodurch natürlich auch ein zusätzlicher unnötiger Aufwand entsteht.

12.2.2 Architektur für die Abkopplung auf der Basis von Absichtssperren

Für die Abkopplung muß der Constraint-Manager zusätzlich die Absichtssperre verwalten und somit auch die Operationen *setX*, *setS*, *releaseX* und *releaseS* implementieren. Das lokale RV-DBMS bietet somit auch diese vier Operationen an. Aufbauend auf so einem erweiterten RV-DBMS liegt die Aufgabe bei der Abkopplung nur noch in der Verteilung der globalen Operationen. Hierfür wird ein Kommunikations Manager konzipiert (siehe Abbildung 12.4), der aus drei Schichten besteht und an jedem Replikat zu finden ist. Die Aufteilungsschicht übernimmt nur die Aufteilung der Operationen an den Kommunikationsmanager oder an das RV-DBMS.

Kommunikationsschicht
Die Kommunikationsschicht stellt für die anderen Schichten die Kommunikation zwischen den Replikaten zur Verfügung. Dabei werden die Eigenheiten der unterliegenden Kommunikationsmechanismen, wie zum Beispiel einer Middleware,

verborgen, so daß der Umstieg auf andere Kommunikationsprotokolle nur Änderungen in dieser Schicht nach sich zieht. Damit ist diese Schicht auch für die eindeutige Benennung und Auffindung von Replikaten zuständig und muß den überliegenden Schichten geeignete Methoden für deren Aufgaben zur Verfügung stellen.

Eine wichtige Aufgabe der Kommunikationsschicht ist die Prüfung der Verfügbarkeit der anderen Replikate. Insbesondere in WANs (Wide Area Networks) ist diese Entscheidung nicht immer einfach, vor allem kann sie oft nur über eine Anfrage mit einem Zeitlimit erledigt werden. Ist das Zeitlimit zu kurz gefaßt, so läuft er möglicherweise ab bevor das andere Replikat eine Chance zur Antwort hatte. Ist das Zeitlimit zu lange, so verlangsamt sich die Entscheidung linear mit dem Zeitlimit.

Quorumschicht
Aufgabe der Quorumschicht ist es, für eine zu verteilende Operation ein gültiges Quorum zu finden. An dieser Stelle ist zu bemerken, daß eine globale Operation durchaus in mehrere zu verteilende Operationen aufgeteilt wird, zum Beispiel in eine Anfrage, nach deren Genehmigung durch alle Replikate die Durchführung erfolgt (vergleiche auch das Two-Phase-Commit-Protokoll, [ISB96]). Falls das Replikat veraltet ist, so gehört zu der Aufgabe der Quorumschicht auch die Aktualisierung des Replikats, so daß dieses ein gültiges Quorum erlangen kann. Die Quorumschicht muß hierfür auch die nachzuziehenden globalen Operationen auf dem lokalen RV-DBMS ausführen.

Zu den Aufgaben der Quorumschicht gehört auch das Erkennen von abgekoppelten und damit nicht mehr erreichbaren Replikaten und die damit verbundene Anpassung des Quorums. Mit den Methoden der Kommunikationsschicht muß hierzu die Erreichbarkeit der Replikate über Netzverbindungen abgeklärt werden. In einigen Fällen ist es durchaus möglich, daß kein Quorum zustande kommen kann. In diesem Fall muß die Quorumschicht dieses erkennen und an die oberen Schichten weitergeben, so daß dem Benutzer der Fehlschlag der globalen Operation angezeigt und begründet werden kann.

Synchronisationsschicht
Die wesentliche Aufgabe der Synchronisationsschicht ist die Garantie der konsistenten Ausführung einer globalen Operation, ähnlich dem Two-Phase-Commit-Protokoll für verteilte Transaktionssysteme. Die erste Phase fragt dabei alle Replikate des Quorums, ob die Operation ausgeführt werden kann. Nur im Fall einer einstimmigen positiven Entscheidung darf die Operation ausgeführt werden.

12.3 Der Demonstrator

Im Rahmen dieser Arbeit wurde ein Demonstrator für das replizierte RV-Modell entwickelt. Ziel des Demonstrators war die Validierung der Arbeit und die Anschauung an einem einfachen Beispiel. Dabei wurde der Dienstgeber, also das RV-DBMS, mit voller Funktionalität implementiert. Beim Klient beschränkt sich der Demonstrator auf ein einfaches Beispiel, an dem sehr schön die Mächtigkeit und die Funktionsweise des Revisionsmodells gezeigt werden kann.

12.3.1 Implementierung

Ziel des Demonstrators war eine prototypische Implementierung zur einfachen Validierung des Konzepts. Dabei war immer klar, daß die Performanz hinter der Einfachheit und der Flexibilität zurücksteht. Für diese Zielsetzung war Java die beste Programmiersprache, insbesondere auch wegen der Plattformunabhängigkeit. Java RMI wurde auch

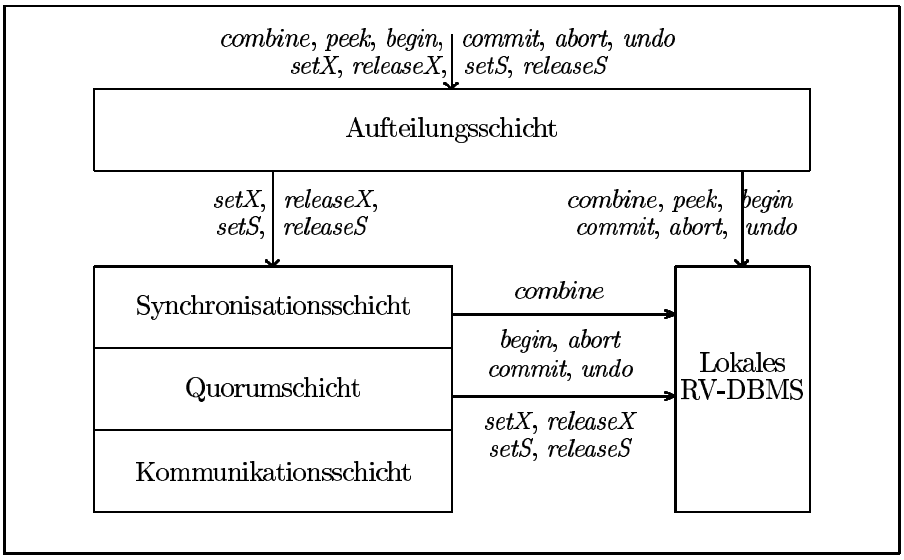


Abbildung 12.4: Architektur für das replizierte RV-Modell

aufgrund der kostenlosen Verfügbarkeit und der guten Anbindung an Java als Middleware verwendet. Der Data Manager wurde auf Basis von ObjectStore/PSE aufgebaut. Aufgrund der miserablen Zugriffszeiten von ObjectStore/PSE (zum Teil im Sekundenbereich) wurde zusätzlich ein einfacher Data Manager im Hauptspeicher implementiert, der somit natürlich keinerlei Persistenz liefert. Über einen eigenen Ablauffaden (Thread) werden die Daten aus dem Hauptspeicher periodisch auf die Platte geschrieben, um die Persistenz zu sichern. Selbstverständlich ist hierbei keine Wiederherstellbarkeit im Fehlerfall möglich. Es wird einfach auf das zuletzt geschriebene Hauptspeicherabbild aufgesetzt, mit allen dort aktiven Entwurfsentscheidungen, die aber beim Neustart automatisch zurückgesetzt werden.

Auf jedem Datenelement im Dienstgeber kann eine Menge von Spezifikationen gespeichert werden. Eine Spezifikation ist dabei ein beliebiges Objekt, welches die in Abbildung 12.5 vorgestellte Schnittstelle (Interface) Spezifikation implementiert. Da Absichtssperren nichts anderes als Spezifikationen mit anderen Bedeutungen sind, müssen auch Absichtssperren diese Schnittstelle implementieren. Die komplette Schnittstelle Spezifikation mit den Erweiterungen für die Absichtssperren ist im Anhang A abgedruckt.

Die meisten Methoden in der Schnittstelle Spezifikation sind trivial, da sie nur den Zustand einer Variable lesen oder ändern. Einzig die Berechnungsmethoden und die Methode *equals*() sind etwas interessanter. Die Methoden entsprechen genau denen aus Kapitel 10, wobei die Methode $isSatisfiable()$ gleichzusetzen ist mit $\neg equals(\mathcal{S}_{False}())$.

Die Verwirklichung des RV-Modells findet vor allem im Dienstgeber statt. Der Klient muß zwar statt scharfen Werten Spezifikationen in die Datenbank einbringen, aber ansonsten nicht großartig erweitert werden. Dieses Verhältnis wurde auch im Demonstrator reflektiert, in dem der Klient einfach gehalten wurde. Da die Schnittstelle zum Dienstgeber (siehe Anhang B) sehr generisch in Termen von Spezifikationen gehalten wurde, kann der Klient die komplette Funktionalität des RV-Modells ausnutzen.

Der Dienstgeber arbeitet mit mehreren Ablauffäden (Threads) und generiert Änderungsmeldungen (ChangeEvents) für jedes geänderte Datenelement. Klienten können sich somit, wie bei Java üblich, am Datenelement anmelden und erhalten dann die Änderungsmeldungen. Der Klient wurde mit Swing implementiert und darf daher nur einen einzelnen Ablauffaden (Thread) haben. Dies bedeutet für den Klienten allerdings

```
public interface Spezifikation {

    // Das Datenelement
    public void setObject(String obj);
    public String getObject();

    // Die Zustaendigkeitsgruppe
    public int getZG();
    public void setZG(int zg);

    // Die Spezifikations-ID
    public String getOID();
    public void setOID(String oid);

    // Die Entwurfsentscheidung
    public String getEE();
    public void setEE(String ee);

    // Die Absichtssperre
    public Spezifikation getILock();
    public void setILock(Spezifikation b);

    // Initiierendes Replikat
    public void setReplica(ReplikatServerInfo rsi);
    public ReplikatServerInfo getReplica();

    // Berechnungsmethoden
    public Spezifikation combine(Spezifikation b);
    public boolean isSatisfiable();
    public boolean implies(Spezifikation b);

    // Gleichheit
    public boolean equals(Spezifikation b);
}
```

Abbildung 12.5: Das Interface Spezifikation

keine Einschränkung, zumal der Klient ja nur zu Demonstrationszwecken implementiert wurde.

12.3.1.1 Implementierte Erweiterungen

Im Demonstrator ist das automatische Setzen einer Absichtssperre im Dienstgeber implementiert. Trifft eine neue *combine*-Operation am Dienstgeber ein, so prüft dieser, ob eine ausreichende Absichtssperre für die neue Spezifikation vorhanden ist. Wenn dies nicht der Fall ist, dann wird automatisch eine Absichtssperre für das Datenelement gesetzt und zwar als X-Sperre mit genau der einzubringenden Spezifikation. Die X-Sperre wird sofort nach der *combine*-Operation automatisch freigegeben. Eine bessere Konfigurationsmöglichkeit des Systems im Hinblick auf Art und Dauer der Absichtssperre ist hier für ein reales System wünschenswert.

Ebenso wie für die Absichtssperren wurde die Automatisierung von Entwurfsentscheidungen eingeführt. Der Benutzer kann eine eigene Entwurfsentscheidung starten und in dieser seine Kommandos ausführen. Am Ende kann die Entwurfsentscheidung festgeschrieben oder abgebrochen werden. Wird eine Operation ohne aktive Entwurfsentscheidung ausgeführt, so startet der Dienstgeber automatisch eine Entwurfsentscheidung für genau diese eine Operation und beendet die Entwurfsentscheidung sofort nach Abschluß der Operation.

Aktive Entwurfsentscheidungen werden nicht, wie zuvor vorgeschlagen, im Fehlerfall für eine Wiederanbindung gespeichert. Der damit verbundene Mehraufwand zur Programmierung einer Liste aktiver Entwurfsentscheidungen und der Möglichkeit der Wiederanbindung an eine aktive Entwurfsentscheidung bringt für den Zweck des Demonstrators keine Vorteile. Damit werden aktive Entwurfsentscheidungen im Fehlerfall komplett zurückgesetzt.

12.3.1.2 Performanzoptimierung des Demonstrators

Im Hinblick auf eine gute Performanz wurden einige Optimierungen eingebaut, während an anderer Stelle noch viel Potential zur Verbesserung vorhanden ist. In der aktuellen Version wird zum Beispiel der aktuelle Peek der Datenelemente verwaltet und bei einer *release*-Operation als Teil einer *undo*-Operation nur invalidiert. Die Neuberechnung erfolgt erst beim nächsten Zugriff über *peek* oder *combine*. Dies hat den Vorteil, daß bei mehreren aufeinanderfolgenden *release*-Operationen auf einem Datenelement nicht jedesmal der Peek neu berechnet werden muß, sondern nur einmal nach der letzten *release*-Operation. Diese Vorgehensweise verteilt den Aufwand der Neuberechnung auch besser, da bei einer *undo*-Operation alle Spezifikationen einer Entwurfsentscheidung zurückgesetzt werden müssen. Dies würde die Neuberechnung der Peeks aller betroffenen Datenelemente nach sich ziehen, während für eine *combine*- oder *peek*-Operation höchstens ein Peek neu berechnet werden muß.

Insbesondere wenn viele Spezifikationen auf einem Datenelement gespeichert werden müssen, dann ist die Wahrscheinlichkeit hoch, daß gleiche Spezifikationen mehrfach vorhanden sind, zum Beispiel weil sich mehrere Entwickler den aktuellen Peek sichern wollten. In diesem Fall macht die einmalige Speicherung der Spezifikation mit mehreren „Eigentümern“ Sinn, damit die Spezifikation eben nur einmal in die Berechnung eingeht. Wenn ein Eigentümer die Spezifikation löscht, so wird er nur von der Eigentümerliste entfernt. Erst wenn diese Liste leer ist, wird die Spezifikation selbst gelöscht.

Eine andere Möglichkeit zur Steigerung der Performanz ist die Erkennung der relevanten Spezifikationen. Wird eine Spezifikation $\mathcal{S}()$ auf einem Datenelement durch eine andere impliziert, so muß die Spezifikation $\mathcal{S}()$ bei den Berechnungen nicht berücksichtigt werden. Im Fall von Intervallen wird der Peek durch maximal zwei Spezifikationen eindeutig definiert. Bei n-dimensionalen Quadern sind für die eindeutige Bestimmung des Peeks genau maximal 2^n Spezifikationen nötig, in vielen Fällen auch deutlich weniger.

Mit zunehmendem Fortschreiten in der Produktentwicklung müssen immer mehr Produkteigenschaften einem Datenelement zugeordnet werden. Hierfür benötigt ein Klient ein „neues“ bisher noch nicht benutztes Datenelement, welches der Klient in beliebiger Weise bearbeiten und einer neuen Produkteigenschaft zuweisen kann. Die Anforderungen an solche „neuen“ Datenelemente sind immer dieselben. Der Dienstgeber kann also eine Reihe von diesen Datenelementen im Voraus anlegen (prinzipiell werden im objektorientierten Fall nur die Identifikatoren mit wahren Absichtssperren belegt) und für die Klienten bereithalten. In der Implementierung sichert sich jedes Replikat sofort eine bestimmte Anzahl neuer Objekte mit X-Sperren auf der kompletten Domäne. Sobald die Klienten einen festgesetzten Teil dieser Objekte abgefragt haben, wird der Speicher mit der nächsten globalen Operation wieder aufgefüllt. Damit kann ein Klient direkt vom Dienstgeber ein neues Datenelement mit gesetzten Absichtssperren ohne globale Operationen bekommen. Die globalen Operationen werden sozusagen asynchron im Voraus und vor allem in Gruppen ausgeführt, so daß nicht für jedes einzelne Datenelement eine eigene globale Operation nötig wird. Nach der Freigabe dieser ersten Absichtssperre wird das Datenelement in den regulären Betrieb übernommen.

12.3.2 Die Anwendung

Im Gegensatz zum Dienstgeber, der auch bei der Performanzevaluierung eine zentrale Rolle spielt, dient der Klient nur der anschaulichen Darstellung der Funktionsweise des Revisionsmodells und des allgemeinen Demarkationsprotokolls. Zahlen als Ergebnisse sind dabei zwar interessant, aber eine viel höhere Anschaulichkeit wird durch grafische Ergebnisse erreicht, so daß die einzelnen Spezifikationen und deren Peek sofort zu erkennen sind. Da dreidimensionale Grafiken im Verhältnis zu einem deutlich erhöhten Implementierungsaufwand dieser Arbeit kaum Vorteile erbringen, wurde die Zahl der Dimensionen auf zwei beschränkt.

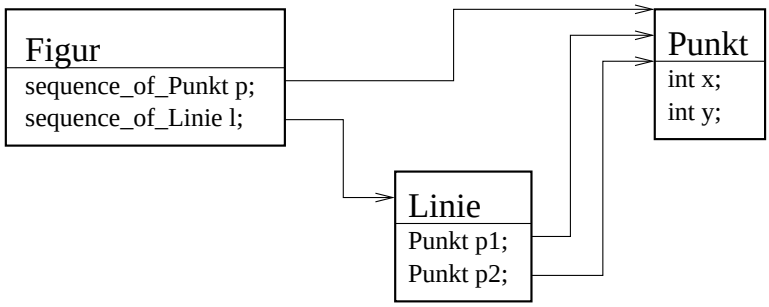


Abbildung 12.6: Das Schema für den Demonstrator

Das verwendete Modell (siehe Abbildung 12.6) bietet nur Figuren, Linien und Punkte als Domänen an. Figuren und Linien sind dabei scharfe Kollektionstypen, so daß die Freiräume nur in den Punkten auftreten. Eine Figur besteht aus einer Menge von Punkten und Linien. Jede Linie besteht aus genau zwei Punkten. Die Punkte selbst sind zweidimensionale Koordinaten mit einer maximalen Abweichung in beide Richtungen. Für das Beispiel wurden damit zweidimensionale Quader, also Rechtecke, als Spezifikationen gewählt, da diese sehr einfach verständlich und anwendbar sind.

Jeder Klient stellt genau eine Figur mit allen darin enthaltenen Linien und Punkten dar. Neue Punkte können einfach per Mausklick in die Figur eingebracht werden und mit anderen Punkten zu einer Linie verbunden werden.

Die Abbildungen 12.7 und 12.8 zeigen zwei Bildschirmabzüge aus der Entwicklung eines abstrakten Gebildes. In der ersten Abbildung sind die Freiräume noch deutlich größer und das fertige Produkt noch nicht so gut zu erkennen. In der zweiten Abbildung wird das Ziel etwas klarer und die Freiräume sind schon deutlich geringer geworden.

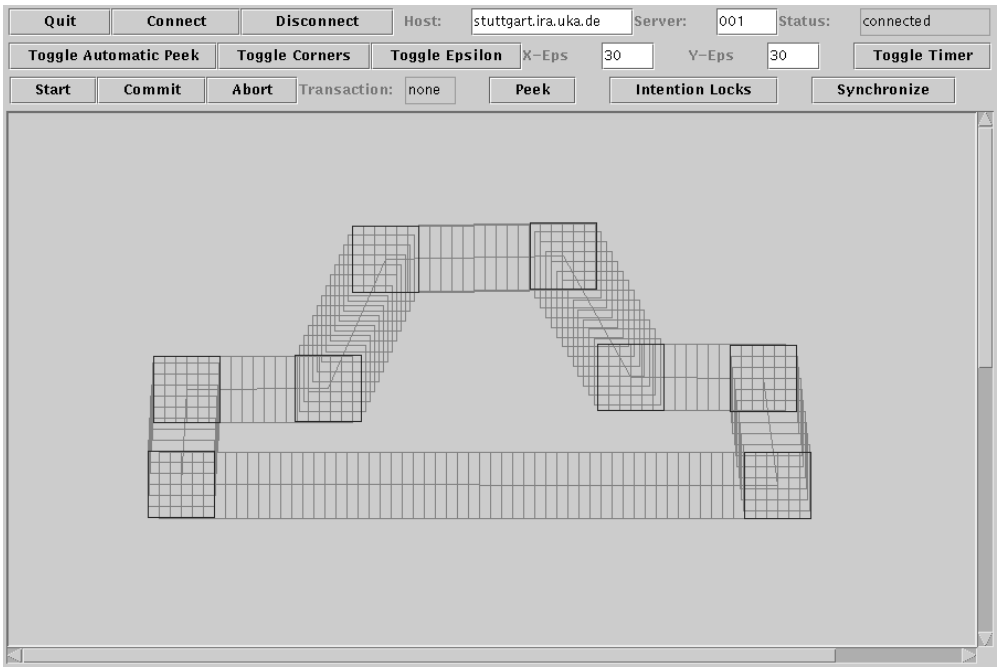


Abbildung 12.7: Die Figur in der frühen Entwicklungsphase

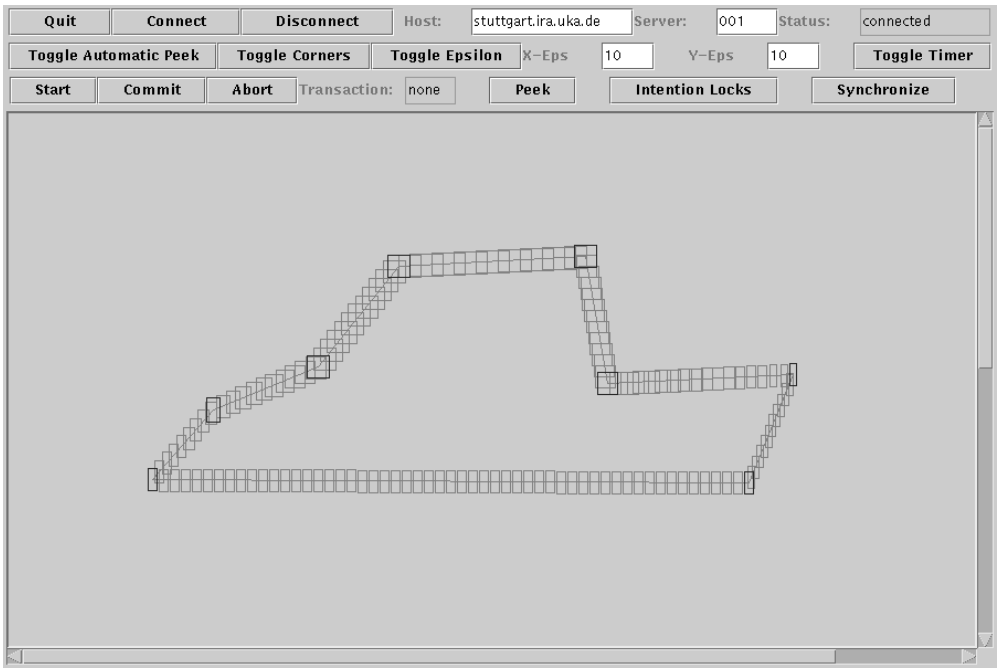


Abbildung 12.8: Die Figur kurz vor der Fertigstellung

12.3.3 Der Klient

Die Abbildungen 12.7 und 12.8 zeigen das Basisfenster des Klienten. In der obersten Zeile sind die Knöpfe zur Steuerung der Anbindung an einen Dienstgeber. In der zweiten Zeile können verschiedene Einstellungen gemacht werden, so zum Beispiel die Aktivierung einer automatischen *peek*-Operation nach einer durchgeführten Änderung oder die Festlegung des Epsilons für die beiden Dimensionen.

In der dritten Zeile findet sich die Entwurfsentscheidungssteuerung und die Knöpfe „Peek“, „Intention Locks“ und „Synchronize“, die im Folgenden kurz erläutert werden sollen:

Peek
Der Knopf „Peek“ führt eine einfach *peek*-Operation auf der Figur und den damit verbundenen Punkten und Linien durch.

Intention Locks
Durch den Knopf „Intention Locks“ öffnet sich ein neues Fenster, in dem sämtliche Datenelemente mit ihren Absichtssperren angezeigt werden. Der Entwickler kann in diesem Fenster Absichtssperren explizit setzen oder freigeben.

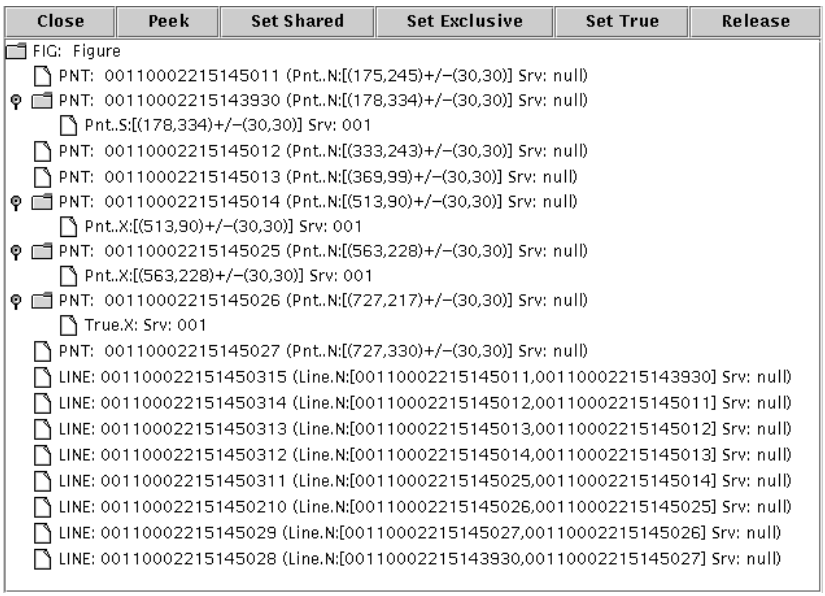


Abbildung 12.9: Das Absichtssperren-Fenster

Abbildung 12.9 zeigt das Absichtssperrenfenster für die im Demonstrator entwickelte Figur. Die Darstellung besteht aus einem Baum mit der darzustellenden Figur als Wurzel. Alle in der Figur enthaltenen Punkte und Linien sind direkte Kinder der Wurzel. Weitere Kinder können dann nur noch gesetzte Absichtssperren sein. Im Beispiel ist auf vier Punkten jeweils eine Absichtssperre gesetzt. Über die Steuerungsknöpfe können neue Absichtssperren gesetzt und alte freigegeben werden, sowohl für jeden Punkt einzeln, wie auch hierarchisch für die komplette Figur, wobei damit nur für alle Punkte, nicht jedoch für die mengenwertige und damit immer konfliktfreien Figuren und Linien, die geforderten Absichtssperren gesetzt werden.

Synchronize
Der Knopf „Synchronize“ führt eine scheinbar wirkungslose, weil leere globale Operation aus. Diese kann aber mehrere unterschiedliche Auswirkungen haben. Jede im Dienstgeber angekommene *undo*-Operation wird erst lokal ausgeführt und

dann mit der nächsten globalen Operation verteilt. Wenn gerade keine globale Operation für die Absichtssperren erforderlich ist, dann kann diese leere Operation zumindest den Austausch der *undo*-Operationen bewirken. Außerdem wird der Speicher an „neuen“ Datenelementen nur aufgefüllt, wenn gar keine Datenelemente mehr zur Verfügung stehen oder wenn eine globale Operation ausgeführt wird. Damit kann durch Synchronize auch der Speicher an neuen Datenelementen wieder aufgefüllt werden, wenn er den Mindestwert unterschritten hat. Falls ein Replikat abgekoppelt war, so verursacht die leere globale Operation natürlich auch die Wiedereingliederung des Replikats in ein neues Quorum und die damit verbundene Angleichung der zwischenzeitlich ausgeführten globalen Operation.

Kapitel 13

Evaluierung

Das RV-Modell bietet einen sehr einfachen und intuitiven Konsistenzbegriff, der für den Entwurfsprozeß alle nötigen Garantien bietet und in der Praxis gut durchgesetzt werden kann. Die erlaubte Nebenläufigkeit ist sehr hoch, da keine Sperren gesetzt werden. Nur wenn eine Aktion einen Konflikt auslöst, muß der Entwickler einen externen Kontakt mit dem Konfliktgegner suchen, um den Konflikt aufzulösen. Die Nebenläufigkeit wird dabei aber durch das RV-Modell nicht eingeschränkt. Im Fall eines Konflikts wird dem Entwickler der Konfliktgegner genannt und - je nach Implementierung eines Klienten - auch die Interaktion mit diesem initiiert. Wie im Folgenden zu sehen ist, kann auch die Performanz des Systems als zufriedenstellend betrachtet werden.

Die Evaluierung des RV-Modells muß isoliert ohne Vergleichsmöglichkeit betrieben werden. Da das RV-Modell für den interaktiven Einsatz konzipiert wurde, ist vor allem die Antwortzeit (response time) für einzelne Anfragen gegen den Dienstgeber interessant. Die Frage des Durchsatzes spielt hier eine wesentlich kleinere Rolle, da die Anwendung des RV-Modells tatsächlich nur interaktiv Sinn macht. Die zu treffenden Entscheidungen können praktisch nicht vorprogrammiert werden und dann komplett alleine ablaufen. Die einzubringenden Constraints sind nicht einfache Werte, die aus vorangegangenen Lesezugriffen berechnet werden können. Da schon die gelesenen Datenelemente Spezifikationen enthalten, von denen der Anwender den für ihn relevanten Teil extrahieren und wieder einbringen muß, können auch die Ergebnisse einer Entwurfsentscheidung nur von Menschen festgelegt werden. Besonders deutlich wird diese Besonderheit des RV-Modells bei einer geplanten Abkopplung, da vor der Abkopplung ein ungefähres „Wissen“ über die während der Abkopplung benötigten Datenelemente und möglicherweise sogar über die geplanten Modifikationen dieser Datenelemente vorhanden sein sollte. Im Gegensatz zu der kurzfristigen Vorausladung von Absichtssperren, die in Abschnitt 9.3 vorgeschlagen wurde, bedarf das Setzen der Absichtssperren vor einer längerfristigen Abkopplung Kontextinformationen über den Stand der Entwicklung und die geplanten Arbeiten. Diese Informationen haben nur die Entwickler, so daß das automatische Bestimmen einer Menge von Datenelementen, auf denen Absichtssperren gesetzt werden müssen, hier sicherlich nicht zum Erfolg führt.

Die folgende Diskussion befaßt sich also mit der Antwortzeit für die einzelnen Aktionen, die der Benutzer ausführen kann. Dabei müssen zuerst die Ziele gesetzt werden, so daß die Ergebnisse der Evaluierung mit diesen Zielen abgeglichen werden können. Die theoretische Diskussion des asymptotischen Aufwands für die einzelnen Operationen wird gefolgt von einigen praktischen Messungen, die dann mit den zu theoretisch erwartenden Ergebnissen verglichen werden. Im Fazit der Evaluierung wird dann das

Erreichen der Ziele überprüft.

13.1 Anforderungen an das Laufzeitverhalten

Der weitaus größte Teil der interaktiv ausgeführten Operationen soll sofort erledigt werden, also mit Wartezeiten, die für den Benutzer praktisch nicht wahrnehmbar sind. Dies ist der Fall, wenn die Antwortzeiten unter etwa 100ms liegen. Nur für wenige aufwendige Operationen sind größere Antwortzeiten zu tolerieren. Dabei muß für den Benutzer allerdings erkennbar sein, daß die Wartezeit gerechtfertigt ist und das System auch wirklich arbeitet. Unter Umständen muß dann auch ein Fortschrittsbalken (progress bar) eingeführt werden, um das Fortschreiten der Arbeiten dem Benutzer aufzuzeigen.

Im Revisionsmodell ist die Wiederankopplung die einzige Operation, die auf den ersten Blick so aufwendig ist, daß Wartezeiten akzeptiert werden können. Hinzu kommt, daß nach einer Abkopplung der Entwickler an seinen Arbeitsplatz zurück kommt und dann möglicherweise auch einige administrative Gänge erledigen muß, während derer die Wiederankopplung durchgeführt werden kann. Damit sind Wartezeiten im hohen Sekundenbereich sicherlich akzeptabel, sollten aber die Minutengrenze nur für wirklich aufwendige Wiederankopplungen (zum Beispiel nach einer sehr langen Abkopplungsphase) übersteigen.

Die Anforderungen bewegen sich also im Bereich bis maximal 100ms für alle Operationen bis auf die Wiederankopplung, für die durchaus etwas mehr Zeit investiert werden darf.

13.2 Theoretischer Aufwand der Operationen

Für die folgende Diskussion des theoretischen asymptotischen Aufwands der einzelnen Operationen wird vorausgesetzt, daß sowohl der Peek aller Spezifikationen, wie auch die Kombination aller S-Sperren (der SPeek) gespeichert wird. Kommt eine neue Spezifikation oder S-Sperre hinzu, so muß diese nur mit dem (S)Peek verknüpft werden. Nur wenn eine Spezifikation oder S-Sperre entfernt wird, dann muß der (S)Peek neu berechnet werden. Dies wird allerdings nicht sofort, sondern erst bei Bedarf erledigt. Die folgenden Überlegungen orientieren sich an dem Pseudo-Code für die einzelnen Operationen, wie er in Abschnitt 12.1.2 vorgestellt wurde.

begin
: Der Start einer Entwurfsentscheidung ist sicherlich konstant in der Laufzeit. Einzig die Erzeugung eines eindeutigen Identifikators und das Anlegen eines Eintrages in der Entwurfsentscheidungstabelle sind hier durchzuführen. Es ist allerdings auf eine skalierbare Implementierung der Tabelle zu achten, da die *PET* im Lauf des Entwicklungsprozesses ständig wächst.

commit
: Bei der Implementierung des RV-Modells bietet es sich an, sämtliche Änderungen sofort in der Datenbasis durchzuführen, schon allein um Konflikte zu erkennen. Außerdem können dadurch der Abbruch einer aktiven Entwurfsentscheidung und die Rücksetzung einer abgeschlossenen Entwurfsentscheidung fast gleich behandelt werden. Unter dieser Voraussetzung sind die Aktionen zum Festschreiben einer Entwurfsentscheidung minimal. Nur der Eintrag in der *PET* muß entsprechend geändert werden.

abort* und *undo
: Der Abbruch beziehungsweise das Rücksetzen einer Entwurfsentscheidung gestaltet sich schon etwas aufwendiger. Prinzipiell müssen alle im Rahmen der Entwurfsentscheidung eingebrachten Spezifikationen zurückgesetzt werden und die

Informationen über die Entwurfsentscheidung müssen aus der *PET* gelöscht werden. Das Zurücksetzen einer eingebrachten Spezifikation bedeutet die Entfernung der Spezifikation vom Datenelement und die Invalidierung des Peeks. Letzteres ist sicherlich konstant im Aufwand. Auch die Entfernung der Spezifikation ist bei geeigneter Implementierung der Spezifikationsmenge für ein Datenelement (zum Beispiel als Hash-Tabelle) konstant. Der Aufwand für die *abort*-Operation ist also linear in der Anzahl der eingebrachten Spezifikationen. Für die *undo*-Operation muß auch noch die entsprechende Information in eine Liste eingetragen werden, die bei der nächsten globalen Operation abgearbeitet wird. Auch dieser Eintrag ist in konstanter Zeit zu erledigen.

computePeek
Nachdem eine Spezifikation durch *undo* oder *abort* gelöscht wird, muß der Peek invalidiert werden. Beim nächsten Zugriff auf das Datenelement mit *peek* oder *combine* muß dieser Peek neu berechnet werden. Wie in Abschnitt 11.4 festgestellt, ist die Berechnung der Konjunktion von n Rechtecken in linearem Aufwand möglich.

computeSPeek
Zum Setzen der Absichtssperren und für die *combine*-Operation wird der Peek aller S-Sperren (SPeek) benötigt. Äquivalent zum Peek aller Spezifikationen kann dieser zwar bei Bedarf neu berechnet werden, die vorgestellte Implementierung speichert den SPeek aber explizit ab und invalidiert ihn, wenn eine S-Sperre freigegeben wurde. Die Neuberechnung des SPeek ist - äquivalent zur Operation *computePeek* - im Aufwand linear in der Anzahl der S-Sperren.

combine
Alle drei Abfragen nach lokaler Widerspruchsfreiheit, Maskierung durch S-Sperren und Maskierung durch X-Sperre sind im Aufwand konstant. Nur wenn der aktuelle Peek oder der SPeek auf dem Datenelement zuvor invalidiert wurde, dann muß die Neuberechnung des (S)Peeks (*computePeek* oder *computeSPeek*) mit linearem Aufwand vor der *combine*-Operation ausgeführt werden.

peek
Die Operation *peek* muß nur den aktuellen Peek zurückgeben und ist damit sicherlich konstant im Aufwand. Einzig die möglicherweise auszuführende Neuberechnung des Peeks (*computePeek*) kann dabei einen linearen Aufwand verursachen.

globalOperation
Die Absichtssperroperationen müssen alle in globalen Operationen ausgeführt werden. Der Aufwand für eine solche globale Operation ist zwar je nach Implementierung durchaus aufwendig, aber doch konstant. Insbesondere die Eigenschaften der Netzverbindung (Latenzzeit und Bandbreite) kann globale Operationen stark verlangsamen. Einzig die Anzahl der Replikate im Quorum kann den Aufwand für die globale Operation beeinflussen, da das initiierende Replikat die globale Operation an alle Replikate des Quorums schicken muß. Der Aufwand hierfür sollte maximal linear mit der Größe des Quorums sein, bei einer intelligenten nebenläufigen Programmierung besteht der Aufwand aus einem konstanten Teil der Netzverbindung und einem linearen Anteil der Kommunikationssteuerung. Bis aber der lineare Anteil, der ja aus reinen lokalen Berechnungen besteht, den konstanten Anteil der Netzverbindung überholt, muß die Anzahl der Replikate schon enorm hoch sein. Der asymptotische Aufwand ist also linear, in der Praxis kann aber von einem konstanten Aufwand ausgegangen werden.

Ein anderer Faktor ist die Größe der globalen Operation. Beim Setzen von Absichtssperren ist diese konstant. Bei der Freigabe wird allerdings zusätzlich die Information über alle im Rahmen der Absichtssperre eingebrachten Entwurfsentscheidungen übermittelt. Die Größe der globalen Operation ist also ein Produkt aus der Zahl der Entwurfsentscheidungen pro Absichtssperre mit der Zahl der

Spezifikationen pro Entwurfsentscheidung. Insbesondere wenn die Absichtssperren länger gesetzt sind, kann diese Zahl recht groß werden.

Wenn das letzte verwendete Quorum nicht mehr gültig ist, so muß für eine globale Operation ein neues Quorum gefunden werden. Wenn noch genügend Replikate vom letzten Quorum übrig sind, dann ist diese Reorganisation sehr einfach möglich. Der Aufwand hierfür teilt sich wieder in den konstanten Teil für die Kommunikation und den linearen Anteil für die Kommunikationssteuerung. Wenn kein Quorum mehr gebildet werden kann, dann ist die globale Operation momentan nicht ausführbar.

Wenn das Replikat für eine oder mehrere globale Operationen nicht mehr Teil des Quorums war und somit veraltet ist, dann muß eine Wiederankopplung (*connect*) erfolgen, die separat behandelt wird.

setX
Der reine Berechnungsteil der Operation *setX* muß mehrere Bedingungen prüfen. Wenn eine X-Sperre bereits existiert, dann ist die Operation abzulehnen. Ansonsten muß die neue X-Sperre mit dem SPeek und möglicherweise mit dem Peek kompatibel sein. Abgesehen von den Neuberechnungen der Peeks ist der Aufwand für die Operation *setX* damit konstant.

setS
Eine potentielle neue S-Sperre muß mit dem SPeek, dem Peek der Spezifikationen und einer möglicherweise gesetzten X-Sperre kompatibel sein. Abgesehen von der Berechnung der Peeks sind die Kompatibilitätsprüfungen konstant.

releaseX
Die Freigabe einer X-Sperre darf nur erfolgen, wenn alle Entwurfsentscheidungen, die im Rahmen der X-Sperre Spezifikationen eingebracht haben, beendet wurden. Der Aufwand für diese Prüfung ist linear in der Zahl der Entwurfsentscheidungen pro X-Sperre.

releaseS
Die Freigabe einer S-Sperre darf nur erfolgen, wenn alle Entwurfsentscheidungen, die im Rahmen der S-Sperre Spezifikationen eingebracht haben, beendet wurden. Der Aufwand für diese Prüfung ist linear in der Zahl der Entwurfsentscheidungen pro S-Sperre.

disconnect
Die Abkopplung eines Replikats vom Netz muß und kann nicht immer explizit erfolgen. Wenn die Abkopplung aber geplant ist, dann werden üblicherweise vor der Abkopplung eine Reihe von Absichtssperren gesetzt. Danach ist es sinnvoll, den am letzten Quorum beteiligten Replikaten noch eine Nachricht über die geplante Abkopplung zu schicken, so daß der nächste Zugriff auf das Quorum keinen Fehler und die damit verbundene Neuorganisation des Quorums verursacht.

Der Aufwand für die Benachrichtigung des Quorums ist im Gegensatz zu globalen Operationen zwar nur eine Meldung an die beteiligten Replikate, der asymptotischen Aufwand entspricht trotzdem dem einer globalen Operation (*globalOperation*).

connect
Die Wiederankopplung eines Replikates an das Netz bedarf zuerst der Identifikation eines Replikates, welches zumindest selbst glaubt, den aktuellsten Stand zu haben. Das ankoppelnde Replikat besorgt sich nun von diesem aktuelleren Replikat alle zwischenzeitlich ausgeführten globalen Operationen und paßt sich selbst dem aktuellen Stand an. Der Aufwand hierfür ist natürlich linear mit der Zahl der nachzuziehenden globalen Operationen, allerdings ist auch der Aufwand für die einzelnen globalen Operationen (siehe unter *globalOperation*) zu berücksichtigen.

Nach einer längeren Abkopplung kann dies einen größeren Zeitaufwand bedeuten, der allerdings nicht vermeidbar ist.

Die Auffindung eines Replikats mit dem aktuellen Stand ist hierbei nicht immer ganz einfach. Möglicherweise muß eine ganze Kette von Replikaten verfolgt werden. Im schlimmsten Fall muß ein zentraler Bootstrapping-Server angefragt werden und alle existierenden Replikate müssen nach dem Zeitstempel ihrer letzten globalen Operation befragt werden. Der asymptotischen Aufwand hierfür ist also linear mit der Anzahl aller existierenden Replikate.

Sobald das Replikat auf dem neuesten Stand ist, kann es ein anderes Replikat aus dem Quorum auffordern, das aktualisierte Replikat ins Quorum einzubringen. Diese Integration ist eine globale Operation (*globalOperation*) und hat daher genau den Aufwand, der zuvor für globale Operationen ermittelt wurde.

13.3 Praktische Meßergebnisse

Die Laufzeitmessungen wurden zum größten Teil auf dem Linux-Notebook des Autors und dem institutseigenen Alpha-Cluster durchgeführt. Da zum Zeitpunkt der Fertigstellung der Arbeit Rechner mit bis zu 2 GHz verfügbar waren, waren die verwendeten Systeme mit 450 MHz (PC) beziehungsweise 533 MHz (Alphas) deutlich veraltet. Eine signifikante Verbesserung der gemessenen Laufzeiten kann also sicherlich allein durch schnellere Hardware erreicht werden. Einzig die Tests zur Wiederankopplung eines Replikats wurden auf einer SUN Ultra-60 Workstation durchgeführt, da die gemessenen Zeiten auf dem Linux-Notebook definitiv zu schlecht waren. Hierfür war nicht nur die langsame Hardware, sondern auch die schlechte Netzunterstützung (per ftp zum Teil deutlich unter 100KB/sec) verantwortlich.

Die praktischen Tests wurden in zwei Richtungen durchgeführt. Für die prinzipielle Anwendbarkeit des Demonstrators wurden bei mehreren Demonstrationen die Antwortzeiten im Klienten gemessen. Abgesehen von Absichtssperroperationen lagen diese Zeiten alle im Bereich von 20 bis 60 Millisekunden (bei Verwendung des hauptspeicherbasierten Data Managers). Einen großen Teil davon stellt die Kommunikation zwischen Klient und Dienstgeber dar. Bei der ursprünglichen Implementierung mit ObjectStore/PSE schluckten die Zugriffe auf den Data Manager zum Teil bis zu einer Sekunde und das System war damit kaum mehr anwendbar. Deutliche Unterschiede in den Ausführungszeiten der einzelnen *combine*-, *peek*- und *undo*-Operationen sind dabei nicht zu erkennen.

Darüber hinaus wurden einzelne Operationen im Hinblick auf ihre Skalierbarkeit geprüft. Da bei dem verwendeten Modell der Rechtecke als Spezifikationen der aktuelle Peek durch maximal vier Spezifikationen spezifiziert wird, macht ein Datenelement mit einhundert oder sogar noch mehr Spezifikationen in diesem Szenario wenig Sinn. Trotzdem wurden die Tests zur Skalierbarkeit des Systems mit Blick auf die Anzahl der Spezifikationen pro Datenelement durchgeführt. Dabei war festzustellen, daß selbst der hauptspeicherbasierte Datamanager bei großen Datenmengen ein echtes Performanzproblem darstellt. Die asynchrone Speicherung in Dateien verursachte regelmäßige Ausreißer mit enormen Peeks. Zum Teil lagen die Zeiten über eine Sekunde höher als die normalen Werte ohne zusätzliche Belastungen im System. Um die reine Zeit für die Durchführung der Operation im Hauptspeicher einschließlich des Netzzugriffs zu messen, wurde der schreibende Ablauffaden (Thread) im Datamanager daher für die Messungen deaktiviert.

Die Tests werden noch weiter unterschieden in den angekoppelten Betrieb, zu dem aber auch globale Operationen und damit die Quorumsbildung gehört. Dieser angekoppelte Betrieb wurde komplett auf einer Maschine lokal durchgeführt, allerdings mit getrenntem Dienstgeber und Klient. Die Netzwerkverbindung ging somit über das lokale loopback-Interface, was durchaus auch im Produktentwicklungsprozeß realistisch

ist (ein Replikat pro Entwickler). Bei dem Test des abgekoppelten Betriebs wurden die Replikate dann über mehrere Rechner verteilt, so daß die Abstimmung über das lokale Netzwerk erfolgte. Der genaue Versuchsaufbau ist mit der Vorstellung der Meßergebnisse beschrieben.

13.3.1 Angekoppelter Betrieb

Als Test wurde eine bestimmte Anzahl von Spezifikationen auf ein Datenelement aufgebracht und anschließend drei Operation ausgeführt. Eine neue Spezifikation wurde mit Hilfe der *combine*-Operation eingebracht. Dieselbe Spezifikation wurde direkt danach durch Ausführung der *undo*-Operation wieder entfernt. Eine anschließende *peek*-Operation verursacht die Neuberechnung des Peeks. Die für die jeweiligen Operationen gemessenen Zeiten sind in den Abbildungen 13.1, 13.2 und 13.3 in Millisekunden über der Anzahl der Spezifikationen (in tausend) aufgetragen.

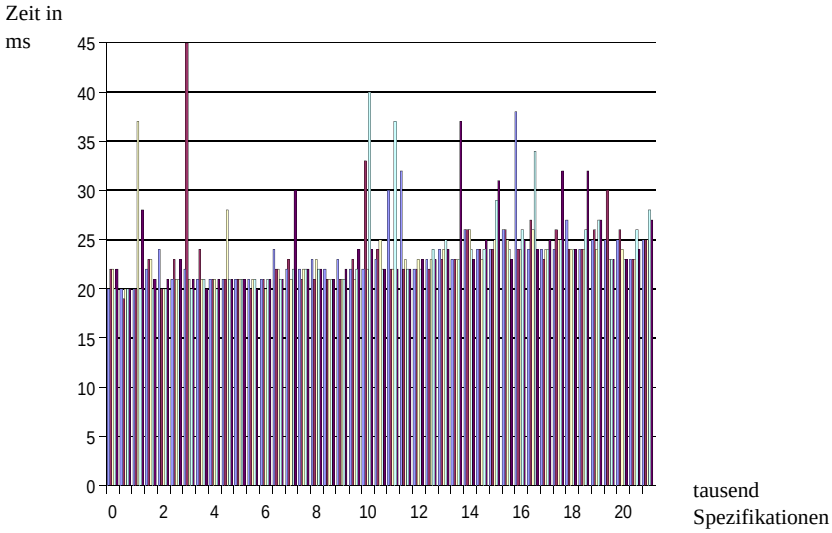


Abbildung 13.1: Einfügen einer neuen Spezifikation (*combine*)

Abgesehen von einigen wenigen Ausreißern im Bereich bis maximal 30 Millisekunden, liegen die Antwortzeiten für das Einfügen einer neuen Spezifikation (Abbildung 13.1) bei 20 bis 25 Millisekunden. Wie auch nach der theoretischen Diskussion zu erwarten war, ist die Antwortzeit für eine *combine*-Operation nicht abhängig von der Zahl der Spezifikationen auf dem Datenelement.

Auch das Zurücksetzen einer Entwurfsentscheidung ist unabhängig von der Anzahl der Spezifikationen auf dem Datenelement, da ja nur die Spezifikation entfernt wird. Die gemessenen Werte (Abbildung 13.2) liegen - von einigen kleineren Ausreißern abgesehen - bei 15ms und darunter. Damit ist das Entfernen einer Spezifikation in den absoluten Zahlen wie zu erwarten schneller als das Einbringen einer Spezifikation.

Die Neuberechnung des Peeks geschieht erst bei der folgenden *peek*-Operation. Der Aufwand für diese *peek*-Operation (Abbildung 13.3) sollte aufgrund der Neuberechnung des Peeks linear mit der Anzahl der Spezifikationen pro Datenelement sein. Da aber in der Grafik kein linearer Anteil zu erkennen ist, scheint dieser im Vergleich zu dem konstanten Aufwand, insbesondere für die Netzverbindung, so gering zu sein, daß selbst für 20000 Spezifikationen kein Anstieg der Antwortzeit verzeichnet werden kann.

Die gemessenen Werte deuten auf hervorragende Skalierbarkeit des Systems im Hinblick auf die Anzahl der Spezifikationen pro Datenelement hin. Natürlich ist bei den verwendeten Spezifikationsarten die Kombination der Spezifikationen extrem einfach

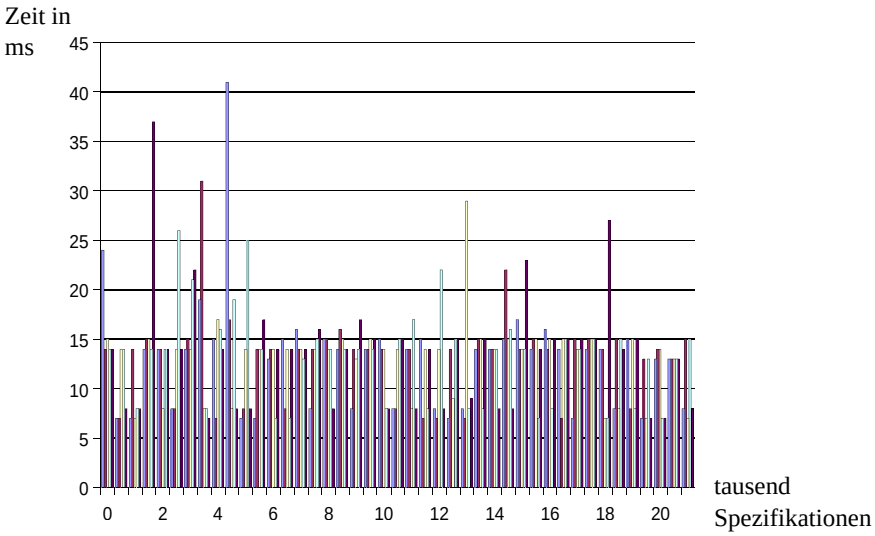


Abbildung 13.2: Löschen einer existierenden Spezifikation (*undo*)

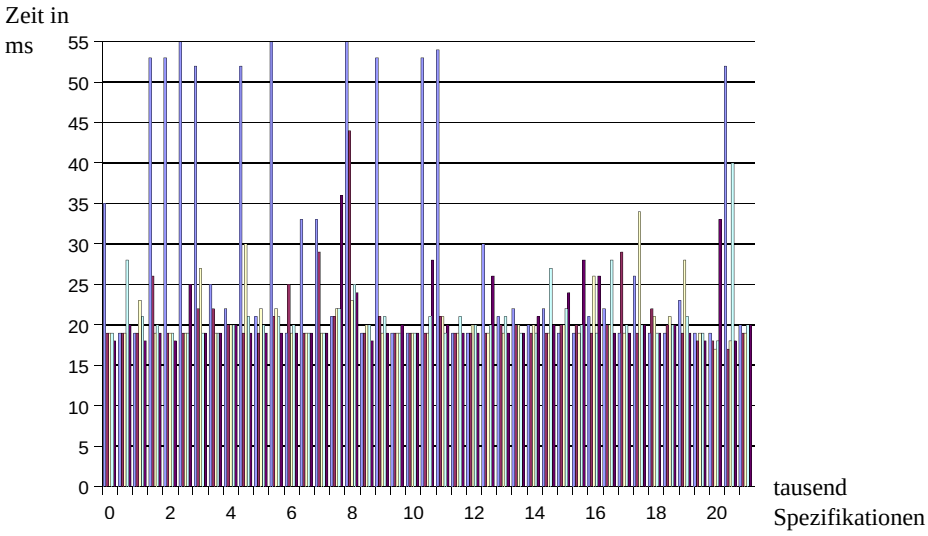


Abbildung 13.3: Auslesen des aktuellen Peeks (*peek*)

und effizient zu berechnen. Werden hier kompliziertere Spezifikationen eingesetzt, so können die Berechnungen entsprechend aufwendiger werden, bei der eingesetzten iterativen Berechnung des Peeks ist der asymptotische Aufwand allerdings trotzdem nur linear in der Zahl der Spezifikationen.

13.3.1.1 Performanz der globalen Operationen

Eine weitere wichtige Frage der Performanz sind die Absichtssperroperationen, da hier immer ein Quorum benötigt wird und damit Kommunikation mit allen am Quorum beteiligten Replikaten nötig ist. Dies ist allerdings kein eigenes Merkmal des RV-Modells, sondern ein Problem der Synchronisation von Replikaten. Trotzdem wurden Leistungsmessungen für unterschiedliche Quorumgrößen (siehe Tabelle 13.4) durchgeführt, die aber aufgrund der beschränkten Anzahl der verfügbaren Rechner nicht wirklich repräsentativ sein kann. Die Tabelle zeigt die in einem lokalen Netz (LAN) durchschnittlich benötigte Zeit für eine globale Operation in Millisekunden über der Anzahl der beteiligten Replikate. Dabei liegt jedes Replikat auf einem eigenen Rechner und sämtliche Kommunikation zwischen den Replikaten geht somit über das lokale Netzwerk. Die Zahl der Replikate eines Datenelements sollte aber bei der Replikation selbst gering sein. Auf jeden Fall sollte die Zahl der Replikate die Zahl der beteiligten Entwickler nicht übersteigen (jeder hat seine eigene Kopie). Die Absichtssperroperationen hängen natürlich auch von der Anzahl der schon gesetzten Absichtssperren ab. Da aber pro Datenelement nur wenig Absichtssperren zu erwarten sind und das Setzen von mehreren hundert oder sogar tausend Absichtssperren auf einem Rechteck ziemlich sinnlos ist, wurde in diesem Bereich keine Leistungsmessung im Hinblick auf die Skalierbarkeit durchgeführt.

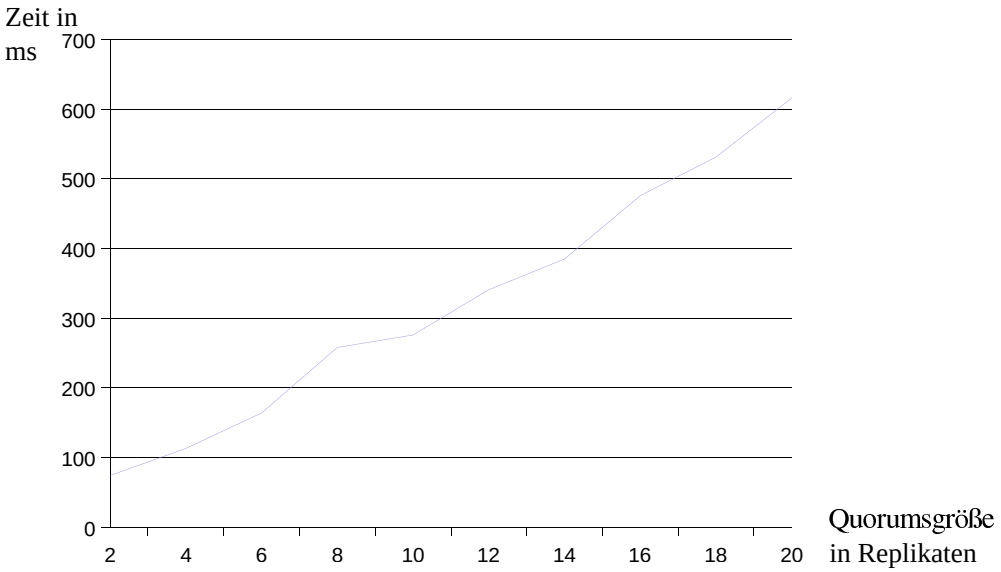


Abbildung 13.4: Skalierbarkeit von Absichtssperroperationen

Es gilt allerdings noch zu betonen, daß der Aufwand für globale Operationen auch stark davon abhängt, ob Replikate des letzten Quorums abgekoppelt und damit nicht mehr erreichbar sind und ob einzelne am neuen Quorum zu beteiligende Replikate länger abgekoppelt waren und erst die zwischenzeitlich ausgeführten Operationen ausgetauscht werden müssen. Selbst wenn alle Replikate den neuesten Stand haben, kann der Ausfall eines Replikats eine Auszeit auf dem Netz und die anschließende Verminderung des Quorums um genau dieses Replikat verursachen. Zum Bootstrapping muß sogar ein zumindest logisch zentraler Replikatserver angefragt werden, der dann alle existierende Replikate zurückgibt. Im allerersten gültigen Quorum müssen dann mehr als die

Hälfte der existierenden Replikate vertreten sein. Diese Anfrage ist aber nur für das erste Quorum überhaupt nötig und für Dienstnehmer und Replikate, die keine anderen Replikate mehr kennen und damit einen völlig neuen Einstieg in das System suchen. Wenn die Menge der Replikate einigermaßen stabil ist (zum Beispiel ein Replikat pro Entwickler), dann wird der Replikatserver nur in ganz wenigen Spezialfällen angefragt werden müssen.

13.3.2 Abgekoppelter Betrieb

Um sinnvolle Untersuchungen der Abkopplung durchzuführen, sind mindestens drei Replikate nötig, da sonst nach der Abkopplung keinerlei globale Operationen mehr durchgeführt werden können. Eine Ausdehnung auf mehr als drei Replikate bringt aber nichts, da in diesem Abschnitt vor allem die Menge der ausgetauschten Daten interessant ist. Diese ist aber nur abhängig von der Arbeit, die auf einem Replikat durchgeführt wurde und nicht von der Anzahl der Replikate. Letztere spielt nur bei der Durchführung von globalen Operationen im angekoppelten Betrieb eine Rolle und wurde schon im vorherigen Abschnitt diskutiert.

Der Versuchsaufbau für die Abkopplung ist in Abbildung 13.5 dargestellt. Die drei Alphas und das Linux-Notebook hängen angekoppelt am Netz. Während eines der drei Replikate für einige Zeit abkoppelt ist, werden auf den zwei angekoppelten Replikaten globale Operationen ausgeführt und Entwurfsentscheidungen durchgeführt. Da Absichtssperren nur über ein Quorum gesetzt oder freigegeben werden können, ist dies nur auf den angekoppelten Replikaten möglich. Auf dem abgekoppelten Replikat können aber im Rahmen der dort gesetzten Absichtssperren Entwurfsentscheidungen eingebracht werden. Da hier nur die ausgetauschten Datenmengen interessieren, wurde im Versuch nur auf den angekoppelten Replikaten gearbeitet. Gemessen wurde dann die Zeit, die für den Austausch der zwischenzeitlich durchgeführten Arbeiten bei der Wiederankopplung nötig war.

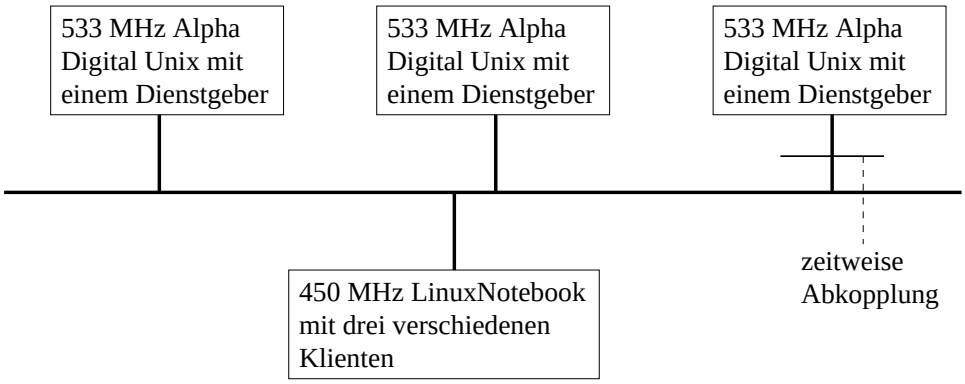


Abbildung 13.5: Der Versuchsaufbau für die Abkopplung

Die lokalen Arbeiten auf den Replikaten bleiben im Vergleich zum angekoppelten Betrieb unverändert. Die Abkopplung braucht, abgesehen von den angekoppelt zu setzenden Absichtssperren, keine besonderen Zusatzaktionen. Nur die Wiederankopplung und der damit verbundene Austausch der zwischenzeitlich durchgeführten globalen Operationen mit allen assoziierten Daten ist bei der Abkopplung ein zusätzlicher Aufwand. Hierzu wurden Messungen durchgeführt, die die benötigte Zeit für die Wiederankopplung und die Ausführung einer globalen Operation, im Rahmen derer dann die Wiederankopplung durchgeführt wird, in Abhängigkeit von der Zahl der auszutauschenden, also während der Abkopplung ausgeführten, globalen Operationen darstellt. Ein enorm wichtiger Parameter in dieser Messung ist die Größe der globalen Operation, das heißt,

die Anzahl der mit der Freigabe einer Absichtssperre auszutauschenden Entwurfsentscheidungen, beziehungsweise deren Spezifikationen. Die Messungen wurden also für verschiedene Spezifikationszahlen pro Absichtssperre durchgeführt und in Abbildung 13.6 aufgetragen. Die Zeit für die Wiederankopplung in Sekunden ist dabei über den Spezifikationen (X-Achse) und den Absichtssperren (Y-Achse) aufgetragen. Anhand des Diagramms ist zu erkennen, da die Dauer der Wiederankopplung nur von der gesamten Anzahl der abgekoppelt eingebrachten Spezifikationen abhängt. Es spielt also keine Rolle, ob viele Absichtssperren mit wenigen Spezifikationen oder wenige Absichtssperren mit vielen Spezifikationen zu übertragen sind.

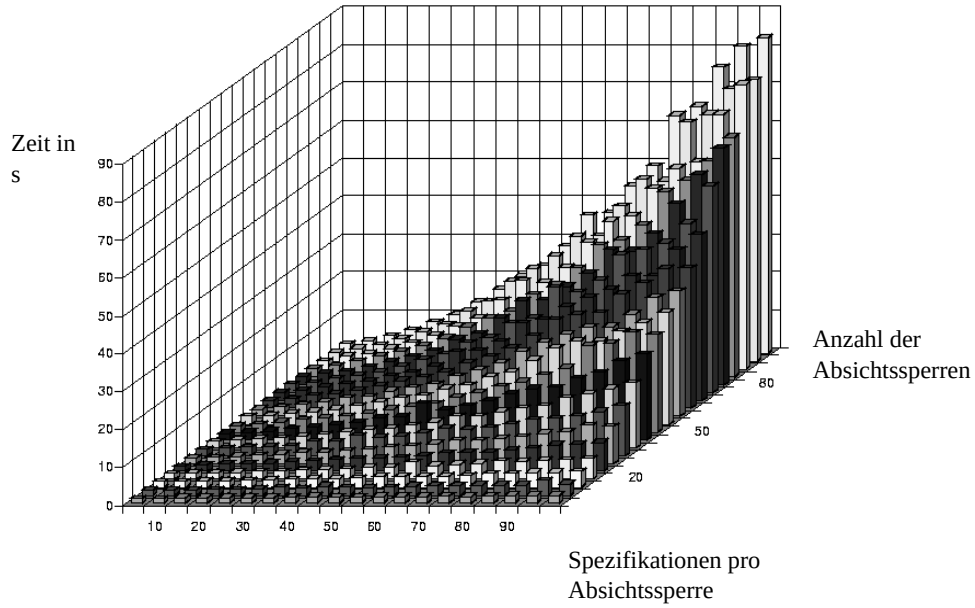


Abbildung 13.6: Dauer der Wiederankopplung (3D)

Aufgrund der etwas realistischeren Situation von vielen Absichtssperren mit wenigen Spezifikationen wurde dieser Fall näher untersucht. Es wurde von fünf eingebrachten Spezifikationen pro Absichtssperre ausgegangen, die entsprechend in Entwurfsentscheidungen verpackt sind, und die Dauer der Wiederankopplung über der Anzahl der zu übertragenden Absichtssperroperationen in Hunderter-Schritten aufgetragen (siehe Abbildung 13.7). Dabei sind unter der Kurve drei Bereiche zu erkennen. Der mit „Exe“ markierte Bereich stellt die für die lokale Nachführung der Arbeiten nötige Zeit dar, während „RMI“ die für den Datenaustausch benötigte Zeit darstellt. Der Bereich „All“ stellt den sonstigen Zeitbedarf dar, der aber sehr gering und vor allem unabhängig von der Datengröße ist.

Die Tests wurden für ein Maximum von 1850 Absichtssperren, also 9250 Spezifikationen, durchgeführt. In der Praxis ist die Anzahl der auszutauschenden Spezifikationen stark abhängig von der Dauer der Abkopplung, die nicht begrenzt ist. Es können also durchaus bei langer Abkopplungsdauer auch mehr Spezifikationen übertragen werden. Wenn allerdings über einen Abkopplungszeitraum von 40 Stunden, also eine Arbeitswoche, pro Minute drei Spezifikationen eingebracht werden, dann entstehen dabei 7200 Spezifikationen, also deutlich weniger als das Maximum im Test.

Bei dem getesteten Maximum von 1850 Absichtssperren beziehungsweise 9250 Spezifikationen dauert die Wiederankopplung 160 Sekunden, also etwas mehr als zweieinhalb Minuten. Der größte Teil der Zeit ist die Datenübertragung mit der dafür notwendigen Serialisierung in Java RMI. Ohne diese Datenübertragung läge die Zeit für die nicht optimierte Wiederankopplung gerade noch bei einer Minute.

Für die Langsamkeit von Java RMI sprechen nicht nur die schlechten hier gemessenen

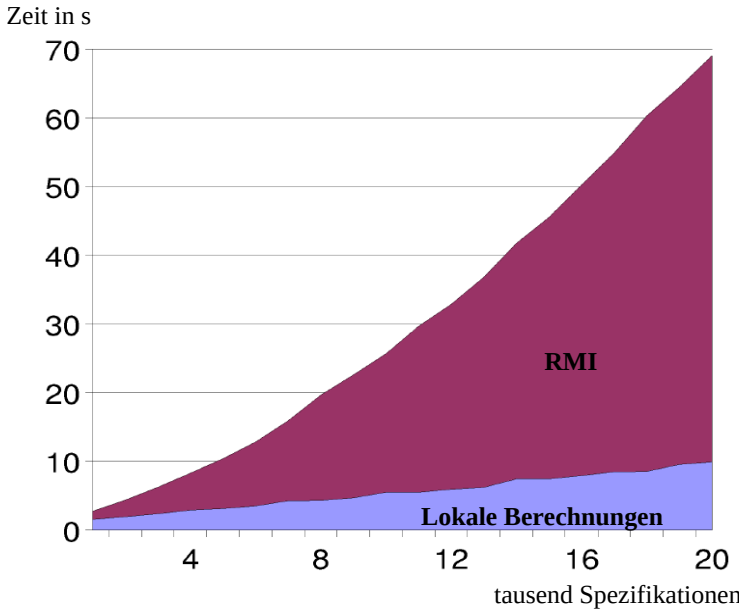


Abbildung 13.7: Dauer der Wiederankopplung (2D)

Werte. Auch in der Forschung haben sich verschiedene Gruppen mit RMI beschäftigt, aus denen zum Beispiel die Implementierung von Java-Party hervorgegangen ist, ein Tool zur Unterstützung von parallelen Java-Programmen. Teil von Java-Party ist eine alternative RMI-Implementierung (KaRMI), die eine durchschnittliche Leistungssteigerung um 45 Prozent erreicht, im Maximum sogar bis zu 71 Prozent ([NPH99]). Auch durch Komprimierung, intelligentere Speicherung oder kompletter Austausch der Middleware kann die Zeit noch weiterer reduziert werden.

Insgesamt ist auch die Kurve der reinen Nachführung der Operationen nicht ganz linear. Nach einer theoretischen Evaluierung müsste die Dauer der Wiederankopplung linear mit der Anzahl der Absichtssperroperationen steigen. Die hier festgestellte Nicht-Linearität könnte zum Beispiel von der Java-Speicherverwaltung herrühren und kann unabhängig von der Ursache problemlos durch eine inkrementelle Wiederankopplung in konstanten Schritten beseitigt werden.

13.3.3 Fazit der Evaluierung

Die gemessenen Antwortzeiten für Anfragen gegen den Dienstgeber[1] sind durchweg im Bereich einiger zehn bis maximal hundert Millisekunden und genügen damit durchaus den genannten Anforderungen für interaktive Systeme. Insbesondere unter der Voraussetzung, daß eine professionelle Implementierung eine deutliche Leistungssteigerung im Vergleich zum Prototyp erbringt, ist das RV-Modell sehr wohl in der Praxis einsetzbar.

Die Skalierbarkeit im Hinblick auf die Anzahl der Datenelemente hängt im eindimensionalen Modell sicherlich nur von dem unterliegenden Data Manager ab und stellt damit zumindest kein größeres Problem als in anderen DBMS dar. Einzige die Anzahl der Spezifikationen auf einem Datenelement könnte in einem langen Produktentwicklungsprozeß zu aufwendigen Berechnungen bei den Operationen führen. Wenn aber immer ein aktueller Peek gespeichert wird, dann begrenzen sich die Berechnungen in den meisten Fällen auf einfache Operationen. Nur die Neuberechnung des Peeks bedeutet einen linearen Aufwand, der aber durch die Markierung der für den Peek relevanten Spezifikationen minimiert werden kann. Außerdem ist die Neuberechnung des Peeks nur nach

[1] Die Arbeit im Klienten selbst ist um ein vielfaches schneller. Der größte Teil der Zeit wird durch Netzverbindungen und Datenbankzugriffe aufgezehrt.

einer *undo*-Operation nötig, die aber einen Rückschritt in der Entwicklung darstellt, der wenn irgendwie möglich vermieden werden soll. Die oft an der Benutzeroberfläche angebotenen „UNDO"-Möglichkeiten zur Rücksetzung einzelner Aktionen sind dabei völlig unabhängig von der *undo*-Operation im RV-Modell, da über den Aufruf zum UNDO an der Benutzeroberfläche keine kompletten Entwurfsentscheidungen, sondern nur einzelne Teilschritte rückgängig gemacht werden. Durch eine intelligente Speicherung der vorherigen Peeks (Before Images) während einer Entwurfsentscheidung kann diese Art von UNDO-Operationen unabhängig von der Anzahl der Spezifikationen effizient implementiert werden.

Globale Operationen liegen schon bei wenigen Replikaten über den Anforderungen für die interaktiven Operationen. Bei 20 Replikaten liegt die gemessene Zeit sogar bei 500ms, also bei einer halben Sekunde. Hier sind intelligentere Implementierungen über einen Austausch der Middleware bis hin zum Broadcast nötig, um die Werte akzeptabel zu machen. Ist die Zahl der beteiligten Replikate allerdings sehr hoch oder sind die Netzverbindungen sehr langsam, dann können hier nicht immer die gewünschten Werte erreicht werden. Allerdings ist diese Art der Synchronisation von Replikaten auch in anderen Protokollen (zum Beispiel Two-Phase-Commit) nötig. Die globalen Operationen sind damit keine Eigenschaft des RV-Modells, sondern eine Eigenschaft der Replikation und müssen damit hingenommen werden.

Die Wiederankopplung nach einer Netzpartitionierung scheint das einzige massive Performanzproblem zu sein. Diverse Optimierungen können die absoluten Zeiten, insbesondere die von Java/RMI verursachte lange Übertragungsdauer, sicherlich deutlich vermindern ([NPH99]). Nach einer längeren Abkopplung muß aber trotzdem mit einer Wiederankopplungszeit von Minuten gerechnet werden. Nach den gestellten Anforderungen aus Abschnitt 13.1 werden hier die Anforderungen nur knapp erfüllt. Mit einer professionellen Implementierung kann der Wert sicherlich auf ein deutlich niedrigeres und damit annehmbares Niveau gedrückt werden.

Ein weiterer Optimierungsansatz könnte versuchen, die Wiederankopplung asynchron im Hintergrund auszuführen und dem Entwickler gleichzeitig neues Arbeiten zu erlauben. In diesem Fall würde die aufwendige Wiederankopplung zwar eine höhere Systemlast bedeuten, sich aber nicht als direkte Wartezeit für den Entwickler äußern.

Um den Dienstgeber für eine reale Produktentwicklungsunterstützung durch das RV-Modell einzusetzen, ist nur noch die Verbesserung der Performanz und Skalierbarkeit wünschenswert. Insbesondere bedeutet dies einen Umstieg von ObjectStore PSE auf einen alternativen Data Manager, da gerade der Data Manager im Demonstrator das größte Hindernis in den Punkten Skalierbarkeit und Performanz darstellt. Eine komplette Neuimplementierung in einer systemnäheren Umgebung (zum Beispiel in C++) würde die Performanz auch deutlich steigern. Mit dem Wechsel der Programmiersprache verbunden ist zwangsweise auch die Wahl einer alternativen Middleware, da Java RMI nur mit Java zusammenarbeitet. Trotzdem bietet auch der prototypische Demonstrator in allen Punkten akzeptable Performanz und kann daher gut angewendet werden.

Kapitel 14

Zusammenfassung und Ausblick

14.1 Zusammenfassung

Das Ziel dieser Arbeit war die Entwicklung einer Datenbankunterstützung für den arbeitsteiligen Produktentwicklungsprozeß am Beispiel des SFB 346. Dafür wurde die Produktentwicklung analysiert und ein Modell des Entwicklungsprozesses erarbeitet. Anhand dieses Modells wurde schnell klar, daß herkömmliche Datenbanken den Produktentwicklungsprozeß nur sehr unzureichend unterstützen. Bei der Entwicklung der Datenbankunterstützung wurde in drei Schritten vorgegangen.

1. Durch die Betrachtung eines einzelnen Entwicklers wurde den prinzipiellen Eigenschaften der Produktentwicklung Rechnung getragen. Dies ist die kontinuierliche Verfeinerung der Produktbeschreibung vom leeren Blatt Papier hin zum gesuchten Produkt. Die Verfeinerung wird durch ständiges Aufprägen von neuen Spezifikationen erreicht, die die möglichen Produkte immer weiter einengen. Die Datenkonsistenz reduziert sich dabei auf die Widerspruchsfreiheit der Spezifikationen.

2. In der Praxis arbeiten in der Regel viele Entwickler gleichzeitig an dem gesuchten Produkt. Dabei entsteht vor allem das Problem, daß Spezifikationen von verschiedenen Entwicklern eingebracht werden und nur sehr schwer auseinandergehalten werden können. Hierfür wurden Entwurfsentscheidungen zur Gruppierung von semantisch zusammengehörigen Spezifikationen eingeführt, mit denen ein zuständiger Ansprechpartner für den Konfliktfall verknüpft ist. Da kaskadierende Rücksetzungen wegen der geforderten Beständigkeit nicht erlaubt sind, wurden die Entwurfsentscheidungen autonom gemacht, in dem jede Entwurfsentscheidung ihre eigenen Voraussetzungen selbst wieder einbringt. Trotzdem wird der für die Entwurfsentscheidung zuständige Ansprechpartner benachrichtigt, sobald Voraussetzungen ungültig werden.

3. Ein weiteres Problem ergibt sich durch die Abkopplung einiger Benutzer, die autonom ohne Netzverbindung mit anderen Entwicklern arbeiten wollen. Hierbei muß verhindert werden, daß unabhängig voneinander Spezifikationen eingebracht werden, die sich bei einer späteren Zusammenführung widersprechen. Das Demarkationsprotokoll wurde hierfür als Basis genommen und erweitert. Dabei können

auf einem Replikat durch Absichtssperren gewisse Freiräume für die weiter Entwicklung reserviert werden, im Rahmen derer Entwickler neue Spezifikationen einbringen können. Dabei wurde bewiesen, daß auf mehreren Replikaten gleichzeitig gearbeitet werden kann und trotzdem die Widerspruchsfreiheit nicht gefährdet ist.

Damit wurde in der Arbeit ein intuitives Modell für die Datenbankunterstützung des arbeitsteiligen Produktentwicklungsprozesses in teilweise entkoppelten Umgebungen vorgestellt. Das RV-Modell beruht in erster Linie auf der Verfeinerung von Spezifikationen mit der Widerspruchsfreiheit und Beständigkeit als Konsistenzbedingung. Für die Abkopplung wurden die auf das Demarkationsprotokoll aufbauenden Absichtssperren zur Sicherung der Widerspruchsfreiheit eingeführt. Dieses Modell wirft allerdings ein paar weitere Fragen und Erweiterungen auf, die im Rest der Arbeit behandelt wurden:

Allgemeine Erweiterungen
Das Revisionsmodell verwaltet nur widerspruchsfreie Produktbeschreibungen und ist damit immer konsistent. In einigen Fällen kann es aber dennoch wünschenswert sein, kurzzeitig Inkonsistenz in der Produktentwicklung zu haben, die später aufgelöst wird. Eine derartige Erweiterung wurde hier diskutiert.

Darüber hinaus wurde für den angekoppelten Betrieb das automatische Setzen und Freigeben von Absichtssperren betrachtet. Damit kann den Entwicklern die Arbeit im angekoppelten Betrieb deutlich vereinfacht werden. Hierbei gibt es einige Freiheitsgrade, zum Beispiel wann eine automatisch gesetzte Absichtssperre wieder freigegeben werden soll.

Abstraktes Modell
In der Anforderungsanalyse wird die zu verwaltende Produktbeschreibung als Konjunktion von Spezifikationen betrachtet. Darauf baut auch das entwickelte Revisionsmodell auf. Das abstrakte Modell definiert einen abstrakten Datentyp Spezifikation mit den fundamentalen Methoden *combine*, *peek* und *implies* und wendet diesen auf das Revisionsmodell an. Damit wird das abstrakte Modell unabhängig von der Konjunktion und kann auch andere Verknüpfungen betrachten, die die Anforderungen an die Methoden erfüllen.

Spezifikationen
Die Beschreibung der Anforderungen an das fertige Produkt werden durch Spezifikationen realisiert. Eine Spezifikation beschreibt eine für den Entwickler akzeptable Menge von Produkten. Unterschiedlichste Formen von Spezifikationen können im Zusammenhang mit dem RV-Modell benutzt werden. In der Arbeit wurden Kugeln, Quader und Fuzzy-Mengen als Spezifikationsformen ausführlich diskutiert und die Besonderheiten des RV-Modells im Zusammenhang mit diesen Spezifikationsformen beschrieben.

Demonstrator und Evaluierung
Im Rahmen der Arbeit wurde ein prototypischer Demonstrator entwickelt, der das abstrakte RV-Modell mit den Absichtssperren für die Abkopplung implementiert. Anhand dieses Demonstrators konnten wichtige Leistungsdaten gewonnen werden, die die Alltagstauglichkeit des RV-Modells und damit auch der spezielleren RV-Modelle belegen. Theoretische Überlegungen speziell im Hinblick auf die Implementierung zeigen auch, daß die meisten Operationen mit konstantem zeitlichen Aufwand zu erledigen sind. Nur im Fall eines Rückschritts in der Entwicklung wird der Aufwand linear in der Anzahl der Spezifikationen auf einem Datenelement. In der Praxis ist dieser lineare Anteil allerdings um Größenordnungen kleiner als der konstante Teil, zum Beispiel für den Netzzugriff, so daß nach den praktischen Messungen auch der Aufwand für die Rückschritte in der Entwicklung konstant ist.

Einzig die Wiederankopplung eines Replikats und der damit verbundene Austausch der zwischenzeitlich durchgeführten Arbeiten verursacht hohe Laufzeiten.

Ein großer Teil davon kann durch verbesserte Implementierungen mit einer schnelleren Middleware kompensiert werden. Dennoch wird die Wiederankopplung eines Replikats immer eine aufwendige Operation bleiben.

14.2 Ausblick

Das in der Arbeit entwickelte RV-Modell wurde speziell für die Anforderungen der Produktentwicklung konzipiert und für die Arbeitsteiligkeit und die Abkopplung erweitert. Mit dem entwickelten Demonstrator konnte das Konzept überprüft werden und eine gute Performanz an der Benutzerschnittstelle nachgewiesen werden. Trotzdem wirft auch diese Arbeit einige weiterführende Fragen auf, deren Bearbeitung im Zusammenhang mit dem RV-Modell von Interesse wäre.

Mögliche Szenarien für das RV-Modell:
Das Szenario der maschinenbaulichen Produktentwicklung und die damit verbundenen besonderen Anforderungen an ein DBMS waren der Grund für die Entwicklung des RV-Modells. Damit wird das Modell sicherlich der zugrundegelegten Produktentwicklung gerecht und kann diese sinnvoll unterstützen.

Interessant ist aber auch die Untersuchung des RV-Modells im Hinblick auf andere Szenarien. Hier stellt sich natürlich sofort die Frage, wie das RV-Modell für Entwicklungsprozesses aus anderen Bereichen anwendbar ist. Darüber hinaus lassen sich aber möglicherweise auch andere Bereiche finden, in denen der konvergierende Charakter des hier vorgestellten Modells hilfreich ist. Da die Grundidee des RV-Modells die Ausarbeitung einer gesuchten Lösung (im vorliegenden Fall das zu fertigende Produkt) ist, sind gerade solche Lösungsfindungsprozesses interessante Kandidaten als Anwendung für das RV-Modell. Dies könnte zum Beispiel auch in der Kriminalistik sein, da auch dort Lösungen (zum Beispiel ein Täter) gesucht wird, der von Zeugen beschrieben wird. Die unterschiedlichen Zeugenaussagen (zum Beispiel über die Größe des Täters) müssen dann so kombiniert werden, daß eine möglichst genaue Beschreibung des Täters resultiert.

Evaluierung in einem kommerziellen System:
Da das RV-Modell bisher nur in einem prototypischen Demonstrator implementiert wurde und Vergleiche mit den traditionellen transaktionsbasierten Systemen wenig Sinn machen, wäre eine kommerzielle und optimierte Implementierung des RV-Modells sehr interessant. Die Evaluierung eines marktfähigen Systems, insbesondere aber die reale Anwendung in einem größeren Umfeld zur Produktentwicklung, können erst die wahren Vorteile und Probleme des RV-Modells im Vergleich zu den etablierten Produkten aufzeigen.

Asynchrone Wiederankopplung:
Einige Ansätze zur Optimierung der sehr aufwendigen Wiederankopplung fallen eher in den technischen und implementierungsorientierten Bereich. Die in Abschnitt 13.3.3 angedachte asynchrone Wiederankopplung bedarf allerdings einiger konzeptioneller Überlegungen. So müssen zum Beispiel während der Wiederankopplung neue Spezifikationen eingebracht werden können. Noch etwas aufwendiger aber umso interessanter ist das Setzen von neuen Absichtssperren während der Wiederankopplung. An dieser Stelle müssen die Auswirkungen auf die globale Historie betrachtet werden und die Verletzung der Konsistenz ausgeschlossen werden.

Mehrdimensionales Modell:
Die komplette Arbeit hat sich auf eindimensionale Spezifikationen beschränkt, also solche, die nur eine Produkteigenschaft benutzen. Eine Verallgemeinerung

des Revisionsmodells auf mehrdimensionale Spezifikationen wäre für die Entwickler eine weitaus mächtigere Unterstützung. Damit könnten Zusammenhänge zwischen den Produkteigenschaften über Spezifikationen beschrieben und vom System überprüft werden. Leider führt gerade diese Mächtigkeit zu dem Problem der Erfüllbarkeit eines Constraint-Systems und ist damit NP-vollständig.

Weitere Untersuchungen im Hinblick auf ein effizientes mehrdimensionales Modell sind hier von großem Interesse. Sicherlich muß die Mächtigkeit des mehrdimensionalen Modells an irgendeiner Stelle reduziert werden, zum Beispiel bei der Mächtigkeit des Constraint-Systems. Ein mehrdimensionales Modell mit einer effizienten Implementierung wäre, falls ein solches realisierbar ist, der hier vorgestellten Lösung deutlich überlegen.

Integration von Alt-Systemen In der Praxis ist es leider kaum denkbar, daß sämtliche existierende Werkzeuge im Produktentwicklungsprozeß durch neue, auf dem RV-Modell basierende, ersetzt werden können. Ein Miteinander von etablierten Werkzeugen auf der Basis des Ersetzungsmodells und neuen Werkzeugen auf der Basis des RV-Modells ist in der Praxis unumgänglich. Die Herausforderung hierbei ist die Integration aller Werkzeuge zu einer durchgängigen Unterstützung des Entwurfsprozesses.

Einfache Lösungen, wie zum Beispiel die Konvertierung von Daten, sind zwar wünschenswert, aufgrund der unterschiedlichen Modelle allerdings kaum sinnvoll machbar. Auch die Verbindung der beiden Zugriffsparadigmen zu einem Modell scheint nicht machbar zu sein.

Einen interessanten Ansatz können Trigger bilden, die Bedingungen zwischen Daten aus den unterschiedlichen Modellen darstellen. Im einfachsten Fall kann somit ein scharfer Wert genau einer Spezifikationsmenge zugeordnet werden, so daß der scharfe Wert immer alle Spezifikationen erfüllen muß. Damit lassen sich Daten aus den unterschiedlichen Modellen auf einfache Art miteinander verbinden. Speziell dieser Trigger-Ansatz scheint sehr vielversprechend zu sein und sollte daher in weiteren Arbeiten untersucht werden.

Anhang A

Das Interface Spezifikation

Der vom realisierten Dienstgeber akzeptierte Typ Spezifikation wurde über das Java Interface Spezifikation realisiert.

```
public interface Spezifikation {
}
```

Hinter einer Spezifikation können sich sowohl normale Spezifikationen auf einem Datenelement, wie auch Absichtssperren verbergen. Die Art der Spezifikation (ob S-Sperre, X-Sperre oder normale Spezifikation) wird über den Typ mit den folgenden Methoden festgelegt und verwaltet.

```
public static final int NOLOCK = 0;
public static final int S_LOCK = 1;
public static final int X_LOCK = 2;

// Spezifikationstyp
public void setType(int type);
public int getType();
```

Die folgenden acht Methoden gelten für alle Typen von Spezifikationen und kapseln nur den Zugriff auf die entsprechenden Variablen.

```
// Das Datenelement
public void setObject(String obj);
public String getObject();

// Die Zustaendigkeitsgruppe
public int getZG();
public void setZG(int zg);

// Die Spezifikations-ID (zum Vergleich)
public String getOID();
public void setOID(String oid);

// Initiierendes Replikat
public void setReplica(ReplikatServerInfo rsi);
public ReplikatServerInfo getReplica();
```

Da der Demonstrator auf dem abstrakten Modell aus Kapitel 10 beruht, wurden die Methoden `combine`, `implies` und `equals` implementiert. `isSatisfiable` ist ein negierter Vergleich mit der falschen Spezifikationen ($\mathcal{S}_{False}()$) und `needsIntentionLock` gibt zurück, ob eine Menge von Spezifikationen dieser Art auf einem Datenelement einen Konflikt verursachen kann. Im verwendeten Datenmodell gibt zum Beispiel eine Figur immer falsch zurück, während ein Punkt mit wahr antwortet.

```
// Berechnungsmethoden
public Spezifikation combine(Spezifikation b);
public boolean equals(Spezifikation b);
public boolean implies(Spezifikation b);
public boolean isSatisfiable();
public boolean needsIntentionLock();
```

Für normale Spezifikationen, die keine Absichtssperren darstellen, muß noch die Entwurfsentscheidung und die entsprechende Absichtssperre gespeichert werden. Die folgenden Methoden kapseln den Zugriff auf diese beiden Attribute.

```
// Die Absichtssperren und Entwurfsentsch. (nur NOLOCK)
public String getEE();
public void setEE(String ee);
public Spezifikation getILock();
public void setILock(Spezifikation l);
```

Mit jeder Absichtssperre müssen die im Rahmen der Absichtssperre eingebrachten Spezifikationen und deren Entwurfsentscheidung verwaltet werden. Der Zugriff auf diese Daten wird über die folgenden Methoden möglich gemacht.

```
// Alle Spezifikationen und Entwurfsentsch. (nicht NOLOCK)
public String[] allEEs();
public void setEEs(String ees[]);
public void addEE(String ee);
public void delEE(String ee);
public Spezifikation[] allSpezs();
public void addSpez(Spezifikation s);
public void delSpez(Spezifikation s);
```

Wenn eine Absichtssperre freigegeben wird, so können durchaus noch Entwurfsentscheidungen im Rahmen der Sperre aktiv sein. Da eine Absichtssperre aber erst nach Beendigung aller Entwurfsentscheidungen, die im Rahmen der Absichtssperre Spezifikationen eingebracht haben, freigegeben werden darf, wird die Absichtssperre erst entsprechend markiert, damit nicht weitere Entwurfsentscheidungen diese Absichtssperre benutzen. Sobald die letzte dieser Entwurfsentscheidungen beendet wird, kann auch die Absichtssperre freigegeben werden.

```
// Soll gelöscht werden (nicht NOLOCK))
public void setToBeDeleted(boolean value);
public boolean getToBeDeleted();
```

Die restlichen Methoden dienen dem Kopieren einer Spezifikation (für ObjectStore/PSE) und dem einfachen textuellen Darstellen.

```
// Sonstiges
public Spezifikation createDuplicate();
public String toString();
```

Anhang B

Die Dienstgeber-Schnittstelle

In diesem Teil des Anhangs ist die Schnittstelle des Dienstgebers für das RV-Modell beschrieben. Die Schnittstelle ist ein einfaches Java Interface, welches vom Dienstgeber implementiert wird.

```
public interface RVServer extends Remote {
}
```

In der Schnittstelle sind einige Werte für die Methoden vordefiniert. Die `PREP_`-Werte können von der Methode `gopPrep` zur Vorbereitung einer globalen Operation zurückgegeben werden. Die `VOTE_`-Werte werden von der Operation `voteForLastGlobalOperation` zur Abstimmung über eine unterbrochene globale Operation zurückgegeben.

```
public static final int PREP_RETURN_OK  = 0; (In Ordnung)
public static final int PREP_RETURN_ILL = 1; (Nicht erlaubt)
public static final int PREP_RETURN_OLD = 2; (Veraltetes Replikat)
public static final int PREP_RETURN_BAD = 3; (Falsche Operation)
public static final int PREP_RETURN_ERR = 4; (Allgemeiner Fehler)
public static final int PREP_RETURN_CFO = 5; (Andere Operation muß
                                               beendet werden)

public static final int VOTE_RETURN_ERR = 0; (Allgemeiner Fehler)
public static final int VOTE_RETURN_OLD = 1; (Veraltete Operation)
public static final int VOTE_RETURN_YES = 2; (Ja)
public static final int VOTE_RETURN_EXE = 3; (Schon ausgeführt)
public static final int VOTE_RETURN_NO  = 4; (Nein)
```

Die Methode `getFreeObject` gibt den Identifikator eines neuen Objektes zurück, welches initial mit einer wahren X-Sperre ($\mathcal{S}_{True}()$) belegt ist. Die Methoden `addRVListener` und `removeRVListener` erlauben dem Klienten, sich als Listener auf einem Objekt einzutragen. Ändert sich dieses Objekt, so werden alle daran angemeldeten Listeners benachrichtigt.

```
public String getFreeObject()
```

```
    throws RemoteException;
public void addRVListener(String object, RVListener c)
    throws RemoteException, RVException;
public void removeRVListener(String object, RVListener c)
    throws RemoteException, RVException;
```

Die folgenden vier Methoden dienen der Entwurfsentscheidungssteuerung und bedürfen keiner weiteren Erklärung.

```
public String begin(int zg)
    throws RemoteException, RVException;
public void commit(String eeid)
    throws RemoteException, RVException;
public void abort(String eeid)
    throws RemoteException, RVException;
public void undo(String eeid, int zg)
    throws RemoteException, RVException;
```

Die Methoden `constrain` und `peek` sind die Hauptzugriffsmethoden auf die Datenelemente. `peekAll` gibt zur besseren Veranschaulichung im Demonstrator alle Spezifikationen auf einem Datenelement zurück. `getIntentionLocks` gibt alle auf dem Datenelement vorhandenen Absichtssperren (die ja nichts anderes als Spezifikationen sind) zurück.

```
public Spezifikation peek(String object)
    throws RemoteException, RVException;
public Spezifikation[] peekAll(String object)
    throws RemoteException, RVException;
public Spezifikation[] getIntentionLocks(String object)
    throws RemoteException, RVException;
public Spezifikation[] constrain(String eeid, Spezifikation s)
    throws RemoteException, RVException;
```

Die folgenden fünf Methoden dienen der Ausführung von globalen Operationen. Für den Klient ist dabei nur die Methode `execute` zur Ausführung einer globalen Operation interessant. Die anderen Methoden dienen der Kommunikation zwischen den Dienstgebern über ein Drei-Phasen-Protokoll. Die erste Phase wird durch `gopPrep` ausgeführt. `gopInfo` und `gopExec` stellen die zweite und dritte Phase dar, während `voteForLastGlobalOperation` der Beendigung einer unterbrochenen globalen Operation dient.

```
public GlobalOperationResult execute(GlobalOperation g)
    throws RemoteException, RVException;
public int voteForLastGlobalOperation(GlobalOperation g,
                                      Quorum q)
    throws RemoteException, RVException;
public int gopPrep(GlobalOperation g, Quorum q)
    throws RemoteException, RVException;
public void gopInfo(GlobalOperation g, boolean info)
    throws RemoteException, RVException;
public GlobalOperationResult gopExec(GlobalOperation g)
    throws RemoteException, RVException;
```

Die Methode `getEE` liefert zu einer gegebenen Spezifikation die Entwurfsentscheidung, in der die Spezifikation eingebracht wurde. Die Methode `getEEs` liefert alle Entwurfsentscheidungen, die auf einem Datenelement Spezifikationen eingebracht haben. Mit dieser

Methode kann der Klient alle an einem Datenelement beteiligten Zugehörigkeitsgruppen auflisten. `getSpecs` liefert alle Spezifikationen, die in einer Entwurfsentscheidung eingebracht wurden, während `active` zurückgibt, ob eine Entwurfsentscheidung aus der übergebenen Liste noch aktiv ist.

```
public String getEE(Spezifikation s)
    throws RemoteException, RVException;
public String[] getEEs(String object)
    throws RemoteException, RVException;
public Spezifikation[] getSpecs(String eeid)
    throws RemoteException, RVException;
public boolean active(String eeid[])
    throws RemoteException, RVException;
```

Die Schnittstelle wird durch drei Abfragemethoden nach dem letzten Quorum, der Historie und den zu globalisierenden Entwurfsentscheidungen komplettiert. Auch diese drei Methoden sind nur für die Kommunikation zwischen den Dienstgebern gedacht und daher für den Klient uninteressant.

```
public EEList getEEList()
    throws RemoteException, RVException;
public Quorum getLastQuorum()
    throws RemoteException, RVException;
public GlobalOperation[] getHistory(GlobalOpId g)
    throws RemoteException, RVException;
```

Anhang C

Symbolverzeichnis

C	Constraints
D	Eine Entwurfsentscheidung
E	Entwickler
F_h	Haltekraft des Robotergreifers
F_k	Kolbenkraft am Robotergreifer
L	Die Liest-von-Halbordnung
L'	Kann-lesen-von-Halbordnung
P	Ein Punkt im mehrdimensionalen Raum
Δ	Überschneidung zweier Kugeln im EBI-Verfahren
Ω	Die Basismenge einer Fuzzy-Menge
α	Schwellenwert einer Fuzzy-Menge
$Sat()$	Erfüllbarkeit einer Spezifikation
ϵ	Radius für Kugeln
$\mathbb{V}_m$	Die Menge der m-dimensionalen Vektoren
$\overline{B_\epsilon(x)}$	Eine Kugel um x mit Radius ϵ
$\lhd$	kann lesen von
μ	Eine Fuzzy-Menge
$\prec$	liest von
d	Anzahl der Domänen einer Spezifikation
d	Kolbendurchmesser des Robotergreifers
e	Anzahl der Entwickler
h	Abbildung von Datentypen auf Vektoren
p	Anzahl der Produkteigenschaften
p	Öldurck am Robotergreifer
q	Dimension eines Teilentwurfsraums
t	Ein Zeitpunkt in der Produktentwicklung
t_0	Der Beginn der Produktentwicklung
t_e	Das Ende der Produktentwicklung
AERV	Arbeitsteilige Eindimensionale Widerspruchsfreie Freiräume
AMRV	Arbeitsteilige Mehrdimensionale Widerspruchsfreie Freiräume
AM	Anforderungsmodellierung
ARV	Arbeitsteilige Widerspruchsfreie Freiräume

CSCW	Computer Supported Cooperative Work
EBI	Einfache Berechnung durch Iteration
FM	Funktionsmodellierung
GDT	Die General Design Theory
GEB	Genaue Ergebnisberechnung
LAN	Local Area Network
PAU	Polytop-Approximation der Umgebungen
PM	Prinzipmodellierung
PPM	Das Produkt- und Produktionsmodell
SFB 346	Der Sonderforschungsbereich 346
SQL	System Query Language
WAN	Wide Area Network
$Komb(\mathbb{S})$	Die Kombination (*combine*) aller Spezifikationen aus $\mathbb{S}$
PET	Persistente Entwurfsentscheidungstabelle
R	Ein Replikat
$Var(\mathcal{S}())$	Die von $\mathcal{S}()$ benutzten Variablen
X, Y	Eine Produkteigenschaft oder ein Datenelement
ZG	Eine Z-Gruppe
$\delta(x, y)$	Die euklidische Distanz zwischen zwei Vektoren
$\mathcal{S}_{False}()$	Die unerfüllbare Spezifikation (falsch)
$\mathcal{S}_{True}()$	Die erfüllte Spezifikation (wahr)
$\mathbb{B}$	Menge aller Kugeln
$\mathbb{C}$	Konfliktmenge
$\mathbb{D}$	Eine Domäne
$\mathbb{E}$	Kanten im Entwicklungsgraph
$\mathbb{G}$	Menge der Produkteigenschaften
$\mathbb{K}$	Knoten im Entwicklungsgraph
$\mathbb{P}$	Erfüllungsmenge oder Freiraum einer eindimensionalen Spezifikation
$\mathbb{R}$	Ergebnisse einer Entwurfsentscheidung
$\mathbb{S}$	Menge von Spezifikationen
$\mathbb{U}$	Das Universum
$\mathbb{V}$	Voraussetzungen einer Entwurfsentscheidung
$\wedge_{real}$	Die reale Schnittmengenbildung
x, y	Werte von Produkteigenschaften oder Datenelementen
$\mathcal{S}()$	Eine Spezifikation
p	Dimension des Entwurfsraums

Literaturverzeichnis

[AO97] ANTONSSON, E. K. und K. N. OTTO: *Improving Engineering Design with Fuzzy Sets.* In: DUBOIS, D., H. PRADE und R. R. YAGER (Herausgeber): *Fuzzy Information Engineering: A Guided Tour of Applications*, Seiten 633–654. 1997.

[AS99] ANTONSSON, E. K. und H.-J. SEBASTIAN: *Fuzzy Sets in Engineering Design.* In: ZIMMERMANN, H.-J. (Herausgeber): *Practical Applications of Fuzzy Technologies*, Kapitel 2, Seiten 57–117. Kluwer Academic Publishers, 1999.

[Bie99] BIEWER, BENNO: *Fuzzy-Methoden.* Springer Verlag (ISBN 3-540-61943-7), 1999.

[BK95] BOSC, P. und J. KACPRZYK: *Fuzziness in Database Management Systems.* Physica Verlag Heidelberg (ISBN 3-7908-0858-X), 1995.

[BM86] BRY, F. und R. MANTHEY: *Checking Consistency of Database Constraints: a Logical Basis.* In: *Proceedings of the 12th International Conference on Very Large Data-Bases (VLDB)*, Seiten 13–20, August 1986.

[BM98] BRAHA, D. und O. MAIMON: *A Mathematical Theory of Design: Foundations, Algorithms and Applications.* Kluwer Academic Publishers (ISBN 0-7923-5079-0), 1998.

[BMGM94] BARBARA-MILLA, D. und H. GARCIA-MOLINA: *The Demarcation Protocol: A Technique for Maintaining Constraints in Distributed Database Systems.* In: *VLDB Journal 3(3)*, Seiten 325–353, 1994.

[Bos96] BOSS, B.: *Fuzzy-Techniken in objektorientierten Datenbanksystemen zur Unterstützung von Entwurfsprozessen.* Doktorarbeit, Universität Karlsruhe, DISDBIS Vol. 4 Infix Verlag, St. Augustin, Deutschland (ISBN 3-89601-404-8), 1996.

[Che98] CHEN, G.: *Fuzzy Logic in Data Modeling. Semantics, Constraints and Database Design.* Kluwer Academic Publishers (ISBN 0-7923-8253-6), 1998.

[DB92] DOURISH, P. und V. BELOTTI: *Awareness and Coordination in Shared Work Spaces.* In: *Proceedings of th ACM Conference on Computer-Supported Cooperative Work (CSCW), Toronto, Canada*, November 1992.

[EGLT76] ESWARAN, K. P., J. N. GRAY, R. A. LORIE und I. L. TRAIGER: *The Notions of Consistency and Predicate Locks in a Database System.* Communications of the ACM, 19(11):624–633, November 1976.

[ERY+95] EVERSHEIM, W., A. ROGGATZ, R. E. YOUNG, G. PERRONE und R. E. GIACHETTI: *Precision Convergence in a Simultaneous Engineering Problem Using Fuzzy Constraint Technology.* In: *Third European Congress on Fuzzy and Intelligent Technologies*, Band 2, Seiten 1254–1256, 1995.

[ERZD95] EVERSHEIM, W., A. ROGGATZ, H.-J. ZIMMERMANN und T. DERICHS: *Kurze Produktentwicklunsgszeiten durch Nutzung unsicherer Informationen.* In: *Informationstechnik und Technische Informatik (37):5*, Seiten 47–53. R. Oldenbourg Verlag, 1995.

[FA97] FRÜHWIRTH, T. und S. ABDENNADHER: *Constraint-Programmierung.* Springer Verlag (ISBN 3-540-60670-X), 1997.

[GMB85] GARCIA-MOLINA, H. und D. BARBARA: *How to assign Votes in a Distributed System.* In: *Journal of the ACM*, Band 32 der Reihe *4*, Seiten 841–860, 1985.

[GR93] GRAY, J. N. und A. REUTER: *Transaction Processing.* Morgan Kaufman (ISBN 1-55860-190-2), 1993.

[HKLP98] HILLEBRAND, G., P. KRAKOWSKI, P. C. LOCKEMANN und D. POSSELT: *Integration-Based Cooperation in Concurrent Engineering.* In: *Proceedings of the 2nd International Enterprise Distributed Object Computing Workshop (EDOC '98), San Diego, CA*, {ggh, krakowski, lockeman, posselt}@ira.uka.de, November 1998.

[HV00] HORVÁTH, I. und J. S. M. VERGEEST: *Engineering Design Research: Anno 2000.* In: *Proceedings of the International Design Conference - Design 2000, Dubrovnik*, 2000.

[ISB96] XOPEN (THE OPEN GROUP), http://www.opengroup.org: *Distributed Transaction Processing*, Januar 1996. Six Volume Set.

[KGK93] KRUSE, R., J. GEBHARDT und F. KLAWONN: *Fuzzy-Systeme.* Leitfäden und Monographien der Informatik. B. G. Teubner, Stuttgart, 1993.

[KLP00] KUPER, G., L. LIBKIN und J. PAREDAENS (Herausgeber): *Constraint Databases.* Springer Verlag (ISBN 3-540-66151-4), 2000.

[Kot96] KOTTMANN, D.: *Replikation in vernetzten Systemen mit mobilen Teilnehmern.* Doktorarbeit, Universität Karlsruhe, DISDBIS Vol. 20 Infix Verlag, St. Augustin, Deutschland (ISBN 3-89601-420-X), 1996.

[KY95] KLIR, G. J. und B. YUAN: *Fuzzy Sets and Fuzzy Logic.* Prentice Hall (ISBN 0-13-101171-5), 1995.

[Lan97] LANG, H.-P.: *Entwicklung einer Fuzzy-Komponente zur Verwaltung und Verarbeitung unscharfer Informationen in objektorientierten Systemen.* Diplomarbeit, University of Karlsruhe, Februar 1997.

[NPH99] NESTER, C., M. PHILIPPSEN und B. HAUMACHER: *A More Efficient RMI for Java.* In: *Proceedings of the ACM 1999 Java Grande Conference, San Francisco, USA.*, Seiten 152–159, 1999.

[Obj95] OBJECT DESIGN, http://www.odi.com/: *ObjectStore C++ API User Guide, Rel. 4*, Juni 1995.

[Rei95] REICH, Y.: *A critical review of General Design Theory.* In: *Research in Engineering Design*, Seiten 1–18, 1995.

[Rev98] REVESZ, P. Z.: *Constraint Databases: A Survey.* In: LIBKIN, L. und B. THALHEIM (Herausgeber): *Semantics in Databases*, Seiten 209–246. Springer LNCS 1358, 1998.

[Sar93] SARASWAT, V.: *Concurrent Constraint Programming.* MIT Press (ISBN 0-262-19297-7), 1993.

[SML95] STURM, R., J. A. MÜLLE und P. C. LOCKEMANN: *Temporized and Localized Rule Sets.* In: *In Proceedings of the 2nd International Workshop on Rules in Database Systems*, Band 985 der Reihe *Lecture Notes in Computer Science*, Seiten 131–146. Springer, 1995.

[Stu97] STURM, R.: *Dynamische Regelmengen zur Beschreibung von Entwurfsspielräumen.* Doktorarbeit, Universität Karlsruhe, VDI-Verlag (ISBN: 3-18-349510-4), 1997.

[Tom98] TOMIYAMA, T.: *General Design Theory and its Extensions and Applications.* In: GRABOWSKI, H., S. RUDE und G. GREIN (Herausgeber): *Universal Design Theory.* Shaker Verlag, Aachen, Germany, 1998.

[Wie86] WIENDAHL, H.-P.: *Betriebsorganisation für Ingenieure.* Hanser-Studienbücher (ISBN 3-446-14565-6), 1986.

[Wit95] WITTE, RENE: *Wissensrevision in Fuzzy Entwurfsdatenbanken: Entwurf und Realisierung.* Diplomarbeit, Universität Karlsruhe, 1995.

[Wit02a] WITTE, RENE: *Architektur von Fuzzy-Informationssystemen: zur Repräsentation und Verarbeitung unscharfer Daten.* Doktorarbeit, Karlsruhe, Univ., Fak. für Informatik, Diss., http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=2002/informatik/3, 2002.

[Wit02b] WITTE, RENE: *Fuzzy Belief Revision.* In: *9th International Workshop on Non-Monotonic Reasoning NMR*, April 2002.

[Yos81] YOSHIKAWA, H.: *General design theory and a CAD system.* In: SATA, T. und E. A. WARMAN (Herausgeber): *Man-Machine Communication in CAD/CAM, Proceedings of the IFIP Working Group 5.2 Working Conference*, Seiten 35–38. North-Holland, Amsterdam, 1981.

[Zad65] ZADEH, L. A.: *Fuzzy Sets.* Information and Control, 8:338–353, 1965.

Stichwortverzeichnis