BASE - ein begriffsbasiertes Analyseverfahren für die Software-Entwicklung

Dissertation
zur
Erlangung des Doktorgrades
der Naturwissenschaften
(Dr. rer. nat.)

dem

Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg

vorgelegt von

Stephan Düwel
aus Paderborn

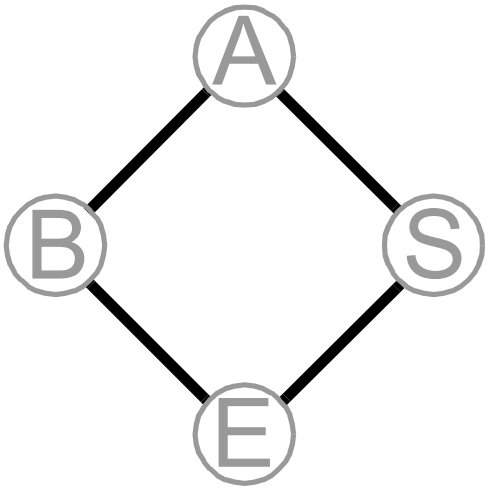
Marburg/Lahn 2000

Vom Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
als Dissertation am 05.07.2000 angenommen.

Erstgutachter: Prof. Dr. Wolfgang Hesse
Zweitgutachter: Prof. Dr. Heinz Peter Gumm

Tag der mündlichen Prüfung am 05.07.2000

BASE -
ein begriffsbasiertes Analyseverfahren für die Software-Entwicklung



Danksagung

Besonderen Dank schulde ich Wolfgang Hesse. Er gab den Anstoß zur Entwicklung von BASE. Seine jederzeit geduldige Betreuung und sein stetiges Interesse machten die Erstellung der vorliegenden Dissertation erst möglich. Nicht zuletzt steuerte er den Namen bei. Hiermit bedanke ich mich bei ihm für die Zusammenarbeit während meiner Zeit in Marburg.

Einen speziellen Dank möchte ich auch Heinz Peter Gumm aussprechen, der sich bereit erklärte, die vorliegende Arbeit zu begutachten.

Inna Schwab, Nguyen Huu An, Serdal Kaya, Jan Malcomes, Christian Näcker, Jörn Schimmelpfeng, Achim Sellmann, Ralf Wiehl und Thorsten Züchner danke ich für die geleisteten Vorarbeiten für das Werkzeug zu BASE.

Die Implementierung des Blockrelationen-Algorithmus entstand nach Hinweisen von Bernhard Ganter und Christian Lindig. Danke!

Dem Fehlerteufel spürten Jochen Hillebrand, Jörg Kesselmeier, Franz und Maria Düwel nach. Auf diesem Weg noch einmal vielen Dank für diese wertvolle Unterstützung.

Inhaltsverzeichnis

1 Einleitung

Im folgenden werden die Motivation, die grundlegenden Ideen und der Aufbau der vorliegenden Arbeit dargestellt. Die gesamte Einleitung ist als fortlaufender Text konzipiert. Überschriften sind mehr zur Orientierung als zur strikten Trennung der betreffenden Abschnitte eingefügt.

1.1 Objektorientierte Anwendungsentwicklung

Beginnend mit den Programmiersprachen Simula, Smalltalk, Eiffel und C++ in den 80'er Jahren hat das Paradigma der *Objektorientierung* in der Software-Entwicklung Anwendung gefunden. Von der *Programmierung* über den *Entwurf* bis hin zur *Analyse* ist es inzwischen als für den ganzen Entwicklungszyklus erfolgversprechend anerkannt. Zahlreiche Methoden zur Anwendungsentwicklung sind in den 90'er Jahren entwickelt, publiziert und verbreitet worden, die alle Projektphasen durch das objektorientierte Paradigma unterstützen (vergl. 2.4).

Zentrales Ergebnis der *objektorientierten Analyse* (*OOA*) ist ein *Klassenmodell*, das den *Untersuchungsbereich* beschreibt und bereits die Struktur für das spätere Software-System vorgibt. Die in der Analyse definierten *Klassen* werden während der weiteren Entwicklung in zunehmender Detaillierung ausgearbeitet. In der Implementierungsphase werden sie schließlich zu vollständig ausprogrammierten Klassen einer Programmiersprache. Für die Programmiersprachen-unabhängige Beschreibung der zu erstellenden Modelle bieten viele der erwähnten Methoden jeweils eigene Notationen an. In der *Unified Modeling Language* (*UML*, vergl. [BJR 99a]) bildet sich zur Zeit aber ein weit anerkannter Standard für eine solche Notation heraus.

1.2 Anwender-Partizipation

Oberstes Ziel eines Software-Entwicklungsprojekts ist es, den Auftraggeber zufriedenzustellen. Am Ende der Entwicklung soll also ein Software-System stehen, mit Hilfe dessen die gestellten *Anforderungen* erfüllt werden. Durch Fehler in der Analysephase kann das Projektziel erheblich verfehlt werden, weil eventuell am Bedarf des Auftraggebers vorbei entwickelt wird (Grady Booch: "Many projects fail for the simple reason that the developers fail to build the right thing: They either deliver a system that does not meet the expectations of its intended users, or they deliver a system that focuses on secondary functions at the expense of its primary use.", [L-W 00], hinterer Umschlagtext). Daß dies wirklich vorkommt, zeigen Untersuchungen. So heißt es in [W-O 92] zu einer Untersuchung von 46 Software-Projekten:
"Ein Projekt wurde zügig und ohne größere Probleme abgewickelt und deshalb zunächst auch als voller Erfolg gehandelt. Das entwickelte Softwaresystem wurde dem Auftraggeber termingerecht zur Verfügung gestellt und von diesem ohne größere Nachbesserungswünsche abgenommen, dann aber nie eingesetzt." ([W-O 92], S. 147).

Und die Standish Group konstatiert nach einer Befragung von 365 DV-Verantwortlichen verschiedener Branchen über 8380 Software-Systeme:
"On the success side, the average is only 16.2% for software projects that are completed on-time and on-budget. In the larger companies, the news is even worse: only 9% of their projects come in on-time and on-budget. And, even when these projects are completed, many are no more than a mere shadow of their original specification requirements. Projects completed by the largest American companies have only approximately 42% of the originally-proposed features and functions. Smaller companies do much better. A total of 78.4% of their software projects will get deployed with at least 74.2% of their original features and functions." ([Stan 95]).

Um die Anforderungen der Auftraggeber korrekt in die Entwicklung einzubringen, bindet man die Auftraggeber und die Anwender – also die *Fachexperten* – so früh wie möglich in die Entwicklung ein. Nur ein regelmäßiger Austausch zwischen ihnen und den Entwicklern kann sicherstellen, daß die Software wirklich nach den Anforderungen der Anwender und nicht nur nach dem – in der Regel lückenhaften und vielleicht sogar teilweise falschem – Verständnis der Entwickler vom Untersuchungsbereichs produziert wird und daß die Anwender angehalten werden, ihre Anforderungen präzise zu formulieren. Auch die von der Standish Group befragten DV-Verantwortlichen setzen die Nutzerbeteiligung und die klare Anforderungsspezifikation an die Spitze der Liste der wichtigen Erfolgsfaktoren für Software-

Projekte ("The three major reasons that a project will succeed are user involvement, executive management support, and a clear statement of requirements.", [Stan 95]). Aus der Untersuchung wurden folgende Ratschläge entwickelt:
"First, find the right user or users. Look for users up and down the organization. Second, involve the user (or users) early and often. Third, establish a quality relationship with the user(s) by keeping open lines of communication throughout the life of the project. Fourth, make it easy for them to be involved in the project. Last, but not least, talk to them and find out what they need. After all, the only reason the project exists in the first place is because someone needs to use the business application when it is finished." ([Stan 96]).

Um zu einem fruchtbaren Austausch zwischen den beiden Gruppen zu kommen, müssen die Entwickler in der Lage sein, ihre Modelle den Fachexperten so darzustellen, daß diese sie beurteilen und an ihren Anforderungen messen können. Es muß ein *Diskurs* zwischen Fachexperten und Entwickler stattfinden. Urs Andelfinger weist in [And 96] in diesem Zusammenhang auf das Problem der "pragmatischen Lücke" hin. Fachexperten und SW-Entwickler haben in der Regel verschiedene Vorstellungen vom Untersuchungsbereich. Ein gemeinsames Verständnis muß erst hergestellt werden. Außerdem sprechen sie nicht die gleiche Sprache. Damit die Entwickler ihr Modell mit den Anwendern diskutieren können, müssen sich beide Gruppen auf eine gemeinsame Sprache bzw. einen gemeinsamen Sprachgebrauch einigen. Die Anforderungen sind in der Anwendersprache formuliert. Deshalb müssen die Entwickler die Fachsprache des Anwendungsbereichs (zumindest teilweise) nachvollziehen.

1.3 Begriffe der Anwendersprache

Die aussagekräftigen Bestandteile der Anwendersprache sind ihre *Begriffe*, mit denen der Untersuchungsbereich beschrieben wird. Solche Begriffe können den Entwicklern völlig fremd oder aber auch in anderem Zusammenhang und damit in möglicherweise anderer Bedeutung vertraut sein. Um also überhaupt weitgehend mißverständnisfrei miteinander kommunizieren zu können, müssen Fachexperten und Entwickler erst einmal in einem Diskurs die wichtigen Begriffe des Untersuchungsbereichs untereinander klären ([Boo 91], S. 191: "By studying the problem’s requirements and/or by engaging in discussions with domain experts, the developer must learn the vocabulary of the problem domain."). Entsprechende Begriffsdefinitionen werden dann in einem Glossar oder einem *Datenlexikon* (data dictionary) festgehalten und stehen so allen Projektbeteiligten zur Verfügung. Entwicklungsarbeit ist ein Stück weit begriffliche Arbeit, gerade wenn es um das Auf-

finden der Kernelemente eines Anwendungsbereichs geht, die zu den tragenden Elementen eines Systemmodells werden sollen. Nach James Rumbaugh führt gerade der objektorientierte Ansatz dazu, daß die Entwickler sich mit den Begriffen des Anwendungsbereichs zu beschäftigen haben: "An object-oriented development approach encourages software developers to work and think in terms of the application domain through most of the software engineering life cycle. It is only when the inherent concepts of the application are identified, organized, and understood that the details of data structures and functions can be addressed effectively. Object-oriented development is a conceptual process [...]." ([RBP+ 91], S. 4).

James Martin und James Odell lassen sich in ihren Büchern erst einmal über “Begriffe” aus ([M-O 95], S. 25: "To learn the fundamentals of object orientation, we first need to understand what concepts are and how they are used. [...] An object is [...] an instance of a concept."). Da die zu Anfang der Entwicklung modellierten Klassen wegweisend für den ganzen Entwicklungszyklus sind[1], kommt ihrer Wahl eine besondere Bedeutung zu. Als Klasse zu modellieren sind ausgewählte Begriffe des Untersuchungsbereichs (Vergl. [RBP+ 91], S. 5: "The objects in the model should be application-domain concepts [...]." und [Str 92], S. 11: "Der Schlüssel zum guten Programm ist die Definition von Klassen, die jeweils ein einzelnes Konzept sehr „rein“ repräsentieren."). Martin und Odell schreiben dazu: "Object types are important, because they create conceptual building blocks for designing systems." ([M-O 92], S. 76) und "An object type is a concept." ([M-O 95], S. 34). Die Klassen bzw. die modellierten Begriffe sollten möglichst invariant gegenüber Veränderungen der Systemanforderungen eine Beschreibung des Untersuchungsbereichs ermöglichen.

1.4 Festlegen von Klassen

Viele Verfechter objektorientierter Modellierungsmethoden bringen als stärkstes Argument für die Objektorientierung die Feststellung, daß die Welt aus Objekten bestünde und so die Objektorientierung einfach das natürliche Paradigma zur Beschreibung von Systemen sei. Schließt man sich dieser Argumentation an, ist die Aufgabe, in der Analyse die richtigen Klassen zu wählen sehr einfach: Der Untersuchungsbereich besteht aus Objekten – man stelle fest, welchen Klassen diese Objekte angehören und schon ist man fertig. So schreibt z.B. Bertrand Meyer in [Mey 88] auf S. 51: “When software design is understood as operational modeling,

1. [Boo 91], S. 191: "Although their outside views may change in only minor ways, we find that the classes and objects we identify at this early stage of design usually carry through the entire design process."

object-oriented design is a natural approach: the world being modeled is made of objects – sensors, devices, airplanes, employees, paychecks, tax returned – and it is appropriate to organize the model around computer representations of these objects. This is why object-oriented designers usually do not spend their time in academic discussions of methods to find the objects: in the physical or abstract reality being modeled, the objects are just there for the picking. The software objects will simply reflect these external objects."

BASE (*Begriffsbasiertes Analyseverfahren für die Software-Entwicklung*) ist gerade ein Beitrag zu einer solchen "akademischen Diskussion". Die Sicht von Meyer versagt spätestens dann, wenn die zu erstellende Anwendung neuartig, sehr umfangreich oder komplex ist ([Boo 94], S. viii: "The first section examines the inherent complexity of software and the ways in which complexity manifests itself. We present the object model as a means of helping us manage this complexity. [...] Because the identification of meaningful classes and objects is the key task in object-oriented development, we spend considerable time studying the nature of classification."). Problematisch ist z.B. die Abgrenzung der Objekte untereinander und die Unterscheidung zwischen Objekten und Attributen, Vorgängen bzw. Beziehungen. Außerdem fällt die Entscheidung, ein Modellelement als Klasse zu betrachten, in verschiedenen Anwendungsbereichen unterschiedlich aus. So werden in einem System zur graphischen Repräsentation von Graphen Kanten als eigene Objekte behandelt, um ihnen graphische Attribute zuschreiben zu können. Bei der Speicherung der Struktur von Graphen, die auf eine graphische Repräsentation keine Rücksicht nimmt, sind dagegen die Kanten nur implizit über die als Attribut zu den Knoten gespeicherten Adjazenzlisten gegeben.

1.5 Eignung objektorientierter Analysemodelle

Selbst wer nicht vom Nutzen einer Diskussion über Wege zur Identifikation von Objekten und Klassen überzeugt ist, muß sich Gedanken über eine andere Diskussion machen – den oben angesprochenen Diskurs zwischen Fachexperten und Entwicklern während der Anwendungsentwicklung. Rumbaugh et al. schreiben in diesem Zusammenhang: "A good model can be understood and criticized by application experts who are not programmers." ([RBP+ 91], S. 5). Daß die objektorientierten Modelle der Entwickler eine adäquate Basis für die Kommunikation mit den Fachexperten bieten, ist keineswegs erwiesen. In [FKM+ 96] argumentieren die Autoren, daß zwar von den Vertretern objektorientierter Methoden behauptet wird, daß objektorientierte Analysemodelle durch die Anwender unmittelbar verstanden und bewertet werden können, doch Praxiserfahrungen diese These nicht uneinge-

schränkt stützen können. Durch die Nutzung abstrakter Modellierungselemente seien die entstehenden Modelle für Anwender weniger transparent als in der Theorie behauptet. So bietet z.B. UML eine eigene Notation für klassenbezogene Merkmale einer Klasse und empfiehlt die Hervorhebung von Konstruktoren einer Klasse. Dies sind zwei Modellaspekte, die nur von der objektorientierten Programmierung her zu verstehen sind und erst in den späten Phasen eine wichtige Rolle spielen. Von den Entwicklern werden diese Konzepte wie selbstverständlich in allen Modellen, die sie während der Entwicklung erstellen, gebraucht. Ist das entstehende Modell deshalb für die Fachexperten nicht vollkommen verständlich, können sie nur schwer entscheiden, ob aus dem Untersuchungsbereich die wichtigen Objekte zu Modellierung gewählt wurden. Den Entwicklern fehlt dagegen häufig die tiefgehende Kenntnis des Untersuchungsbereichs, um diese Frage von sich aus abschließend beurteilen zu können. Deshalb erscheint die Sichtweise von Bertrand Meyer (und anderen) zumindest sehr vereinfachend.

Für die Anwender nach wie vor leichter zugänglich erscheint das klassische Paradigma der Anwendungsentwicklung, bei dem Daten und Funktionen getrennt modelliert werden. Da eine Trennung der beiden Sichten zusammen mit der üblichen Vorgehensweise der funktionalen Zerlegung zu Problemen (aufgrund fehlender Modularität und unnötiger Datenabhängigkeiten) führt, werden die beiden Sichten in BASE nicht getrennt, sondern komplementär betrachtet. So kann das Analysemodell den Fachexperten aus beiden Sichten präsentiert werden und Datenabhängigkeiten werden gleichzeitig kontrolliert. In BASE kann auf diese Weise eine funktionale Zerlegung durchgeführt werden, ohne deren übliche Nachteile in Kauf nehmen zu müssen. Ein solches Vorgehen ist für die Fachexperten leichter zugänglich. Funktionale Zerlegung ist nicht nur eine "natürliche", intuitive Herangehensweise zur Lösung von Problemen, sondern wird von den Fachexperten auch zur Beschreibung von Geschäftsprozessen selber angewandt.

1.6 Das Vorgehen in BASE

Die *Anwendungsfall-getriebene* (*use case driven*) *Entwicklung* wurde im Bereich der objektorientierten Methoden zuerst von Ivar Jacobson (vergl. [JCJÖ 93]) empfohlen, ist aber in der Zwischenzeit von vielen anderen Autoren aufgenommen worden. Die Anwendungsfälle spiegeln die funktionalen Anforderungen an das System wider. Diese zu erfüllen, ist das wichtigste Ziel der Entwicklung. Deshalb sind die Anwendungsfälle des zu erstellenden Systems auch Ausgangspunkt in BASE.

Das betrachtete System wird ausgehend von den *Anwendungsfällen* funktional zer-

legt. Die dadurch entstehenden Datenabhängigkeiten, werden durch die verwendeten Datenstrukturen explizit gemacht und können so von den Entwicklern im Griff behalten werden. Die vorgestellte Methode bietet dabei einen Ansatz, die Granularität der Zerlegung zu kontrollieren. Untersucht wird dazu, ob in der Benutzung der Daten *Implikationen* der folgenden Form im aktuellen Modell vorliegen: "Alle Funktionen, die die Daten A_1, ... , A_n benutzen, betreffen auch die Daten B_1, ... , B_m." Von den Fachexperten kann beurteilt werden, ob solche Implikationen für die gesamte betrachtete Funktionalität Geltung haben, oder das untersuchte System noch nicht bedachte Funktionen enthält, die einer Implikation widersprechen. Im zweiten Fall ist eine solche Funktion noch in die Betrachtung einzuschließen, um das Systemmodell zu vervollständigen.

Aus den entstandenen Datenstrukturen können anwendungsfallübergreifend wichtige Datenelemente als mögliche *Klassenkandidaten* direkt abgelesen und weitere Vorschläge für *Klassen* und *Systemkomponenten* als Kombinationen von Datenelementen und Funktionen generiert werden. In bezug auf die zu erstellenden Klassen ergibt sich durch die zweite Möglichkeit ein *bottom-up-Verfahren*, in dem erst die Klassenbestandteile – Attribute und Operationen – und dann die Klasse betrachtet werden. Dies ist sinnvoll, solange die wichtigen (als Klasse zu modellierenden) Begriffe des Untersuchungsbereichs noch nicht klar sind.

Für ein top-down-Verfahren, das eine klarere Modellstruktur verspricht, fehlen in dieser frühen Phase noch Informationen ([Mey 88], S. 325: "[...] top-down development appears as an almost impossible recipe. Since the architecture is hierarchical, and built from the top, you are asked to make the most important decisions at the very beginning – when you have the least information at your disposal to avoid design errors! [...] Only a genius could do this."). Sind einmal im Konsens der beiden Gruppen die wichtigen Begriffe als Klassenkandidaten ausgemacht, können diese Klassen selber top-down entwickelt werden. Eine top-down-Entwicklung einer Klasse beginnt mit dem Begriff, den die Klasse implementieren soll. Sukzessive wird die Klassendefinition verfeinert, bis alle Attribute und Operationen der Klasse enthalten sind. Bei Anwendungen, deren grundlegende Klassen schon aus Erfahrung der Entwickler klar sind, kann BASE zur Überprüfung der vorgesehenen Modellierung eingesetzt werden.

Ziel des Vorgehens ist es, Konsens der Beteiligten über die wichtigen Begriffe des Untersuchungsbereichs und die Klassenkandidaten herzustellen.

1.6.1 Formale Begriffsanalyse

Die oben erwähnten Autoren Martin und Odell behandeln Begriffe als Einheiten von *Begriffsumfang* (*Extension*) und *Begriffsinhalt* (*Intension*). Der Umfang eines

Begriffs besteht aus allen Gegenständen, die unter diesen Begriff fallen, und der Inhalt aus allen gemeinsamen Merkmalen dieser Gegenstände. Dies ist die klassische Auffassung von “Begriff”.

Formale Begriffsanalyse (*FBA*, [G-W 96]) ist eine mathematische Theorie, die Begriffssysteme, so wie das Denken und Schließen in solchen Systemen mit dem beschriebenen Verständnis von Begriffen zu formalisieren versucht. Ausgangspunkt bei der Verwendung von FBA ist ein *formaler Kontext*, der durch eine einfache Inzidenzmatrix (Kreuztabelle) beschreiben wird. In ihm werden (formalen) *Gegenständen* (formale) *Merkmale* zugeschrieben. Daraus entsteht als mathematische Struktur ein *vollständiger Verband* aus Begriffen mit ihrer Ober-/Unterbegriffsordnung, der sog. *Begriffsverband*. Die Umfänge der formalen Begriffe bestehen aus formalen Gegenständen und die Inhalte aus formalen Merkmalen des zugrundeliegenden Kontexts. Endliche Verbände können durch sog. *Liniendiagramme* (Hasse-Diagramme) dargestellt werden. Erfahrungen der Darmstädter Arbeitsgruppe “Allgemeine Algebra und Diskrete Mathematik” um Rudolf Wille zeigen, daß bei Datenanalysen in vielen verschiedenen Fachgebieten solche Diagramme eine fruchtbare Basis für Diskussionen zwischen den Fachexperten und den analysierenden Mathematikern darstellten. Deshalb werden sie auch in BASE herangezogen, um den Dialog der Entwickler mit den Fachexperten zu unterstützen.

In der Entscheidung, was man als formale Gegenstände und Merkmale behandelt, ist man zunächst einmal frei. Die mathematische Theorie betrifft nur die Strukturaspekte des erstellten Modells, so daß man formale Kontexte mit einer beliebigen Semantik versehen kann, die als Zuordnung von Merkmalen zu Gegenständen interpretiert werden kann. Die Palette der Anwendungen reicht von typischen Klassifikationsaufgaben (z.B. Zuordnung von verschiedenen Eigenschaften zu Gesteinsarten zum Aufbau einer Wissensbasis, [Erd 98]) über Navigationsunterstützung in Dokumentensammlungen (z.B. für das WWW, [K-N 96]) bis zur Darstellung von Beziehungsgeflechten beliebiger Dinge (z.B. Abbildung der globalen Struktur der Benutzungsbeziehungen innerhalb einer Klassenbibliothek, [Vog 96], S. 115). Denkbar ist natürlich auch die gleichzeitige Betrachtung verschiedener formaler Kontexte, um verschiedene Aspekte eines Modells auszudrücken. Zu beachten ist, daß sich mit geänderter Semantik des Kontexts natürlich auch die Bedeutung der entstehenden formalen Begriffe ändert.

1.6.2 Formale Gegenstände und Merkmale in BASE

Eine "natürliche" Modellierung von Klassen mit Formaler Begriffsanalyse behandelt Objekte als formale Gegenstände und ihre Eigenschaften – etwa Attribute und Operationen – als formale Merkmale. Dies würde aber mit der Betrachtung der

einzelnen Objekte eine Analyse auf Instanzenebene verlangen. Deshalb wird in BASE ein anderer Ansatz verfolgt. Als (formale) Gegenstände werden hier *Dinge des Untersuchungsbereichs* und als (formale) Merkmale *Anwendungsfälle* und später *Funktionen* des zu entwickelnden Systems betrachtet. Die dabei entstehenden (formalen) Begriffe fassen in ihrem Umfang also Datenelemente und in ihrem Inhalt Funktionen zusammen. Ihr Umfang hat keine wirklich extensionale Bedeutung. Dieser Ansatz erlaubt aber eine Analyse auf der *Schemaebene* ([M-O 92], S. 75: "During Object Structure Analysis, the analysis team is more intent on identifying the types of objects than identifying the individual objects in a System.", [W-N 95], S. 161: "In object-oriented analysis we are not concerned with objects per se. On the contrary, our aim is to describe them only indirectly – through their behavior. Everything else is irrelevant and should be excluded from the analysis model. In an object-oriented context, we have one and only one way of describing object behavior, and that is through classes [...].("). Weiter sind in BASE die formalen Begriffe in der Regel nicht alle gewünschten Klassenkandidaten, aber das entstehende Liniendiagramm spiegelt die Datenabhängigkeiten der betrachteten Funktionen wider und kann zur Generierung von Vorschlägen für Klassenkandidaten genutzt werden.

Einen dem "natürlichen" verwandten Ansatz verfolgen Gregor Snelting und Frank Tip im Rahmen der Restrukturierung von Klassenbibliotheken (vergl. Kapitel 6). Sie untersuchen bestehenden Programm-Code. Die auftretenden Variablen, die zur Laufzeit die Objekte der Anwendung speichern, behandeln sie als formale Gegenstände, Attribute und Operationen der entsprechenden Klassen als formale Merkmale. Damit haben sie eine Möglichkeit, die erst zur Laufzeit angelegten Objekte durch die schon zur Compilierzeit vorliegenden Variablen mittelbar faßbar zu machen. Direkt stehen die Objekte zum Analysezeitpunkt jedoch nicht zur Verfügung, so entsprechen die Umfänge der in der Untersuchung von Snelting und Tip auftretenden formalen Begriffe auch nicht der extensionalen Bedeutung der untersuchten Klassen als Mengen der zugehörigen Objekte. In der Situation von BASE steht nicht einmal eine mittelbare Behandlung der Objekte durch Programmvariablen zur Verfügung.

1.6.3 Entwicklungsumgebung für BASE

In Kapitel 5 wird ein Analyse-Werkzeug für BASE vorgestellt. Der Benutzer eines solchen Systems gibt die Anwendungsfälle zusammen mit ihren Beschreibungen ein. Zu jedem Anwendungsfall wählt der Benutzer die relevanten Dinge aus der Beschreibung oder gibt sie ebenfalls frei ein. Aus diesen Angaben werden ein erstes Liniendiagramm und eine Basis von Implikationen innerhalb der formalen Gegenstände (Dinge) berechnet. Im Diagramm stehen anwendungsweit wichtige Din-

ge unten und bieten sich als erste Klassenkandidaten an.

Entscheidet sich der Anwender, einen Anwendungsfall funktional zu zerlegen, hat er jede Funktion in der Zerlegung ebenfalls zu beschreiben und mit Dingen zu indizieren. Dadurch entsteht ein neues Diagramm. Die Implikationen führen zu Nachfragen bei den Fachexperten. Eine Implikation beschreibt einen Zusammenhang zwischen Dingen, der im aktuellen Modell durch die Anwendungsfälle oder Funktionen gestiftet wird. Die Fachexperten können beurteilen, ob ein solcher Zusammenhang fachlich begründet ist. Andernfalls wird eine neue Funktion in der funktionalen Zerlegung betrachtet.

Eine neu betrachtete Funktion führt häufig auch zu neu zu betrachteten Dingen. So wird das Modell immer weiter angereichert.

Zu jedem Zeitpunkt kann der Benutzer des Systems Vorschläge für eine Systemzerlegung in Klassen und Komponenten generieren lassen.

Die Entwicklungsumgebung verbirgt die mathematischen Strukturen bis auf das Liniendiagramm völlig vor dem Benutzer, so daß die Anwendung der vorgestellten Methode keine Kenntnis einer neuen Modellierungssprache erfordert. Deshalb ist der Mehraufwand gegenüber herkömmlichen Methoden gering.

Damit bietet das Vorgehen eine Modellierung von Klassen, in dem es den Diskurs zwischen Fachexperten und Entwicklern und die Modellierung durch (gewohnte) funktionale Zerlegung unterstützt. Dies ist besonders wertvoll, wenn der Untersuchungsbereich den Entwicklern völlig fremd ist.

1.7 Aufbau der Arbeit

In Kapitel 2 werden die grundlegenden Konzepte der objektorientierten Modellierung sowie einige bekannte Entwicklungsmethoden zusammengestellt. Dies dient in seiner Breite vor allem der Vorbereitung des folgenden für den nicht mit den Ideen und Verfahren der objektorientierten Software-Entwicklung vertrauten Leser.
Als speziell interessante Punkte werden

- in 2.3.1 der Zusammenhang zwischen Klassen und dem hier verwandten Begriffsverständnis untersucht,
- in 2.5 die in BASE übernommene Anwendungsfall-getriebene Analyse nach Jacobson dargestellt,
- in 2.6 die Antworten der herkömmlichen Verfahren auf die Frage beleuchtet, wie Klassen in der Analyse identifiziert werden können.

Kapitel 3 befaßt sich mit der Formalen Begriffsanalyse (FBA). Nach einer Darstellung des zugrundeliegenden Begriffsverständnisses und der Einführung benötigter ordnungstheoretischer Grundlagen sind in Abschnitt 3.3 die Grundzüge der Formalen Begriffsanalyse zusammengefaßt. Sie sind grundlegend für das Verständnis der folgenden Kapitel. Die Abschnitte 3.4 und 3.5 bereiten einzelne Erweiterungen des grundsätzlichen Ansatzes von BASE vor.

Der Hauptteil der Arbeit besteht in den Kapiteln 4 und 5. Bei der jeweils ersten Bezugnahme auf eines der mathematischen Konzepte findet sich eine Referenz auf Kapitel 3 mit Angabe der betreffenden Seitenzahl. So ist die Beschäftigung mit dem Hauptteil der Arbeit auch ohne vorheriges Durcharbeiten von Kapitel 3 möglich.

Kapitel 4 erklärt das entwickelte Analyseverfahren anhand eines Beispiels. Dabei enthält Abschnitt 4.2 die Beschreibung des grundsätzlichen Ansatzes. Die Beantwortung der Frage nach Klassenkandidaten wird in 4.5 versucht und in 4.6 innerhalb der allgemeineren Fragestellung, wie das zu entwickelnde System in Bausteine zerlegt werden kann, wieder aufgenommen. Das Kapitel schließt mit einer vollständigen Vorgehensbeschreibung in 4.8. Die anderen Abschnitten untersuchen spezielle Punkte innerhalb von BASE. Der wichtigste darunter ist das Verfahren der funktionalen Zerlegung (Abschnitt 4.3).

Kapitel 5 gibt einen Eindruck von dem entwickelten Werkzeug zu BASE. Anhand der Darstellung soll klar werden, daß BASE ohne Kenntnis der mathematischen Grundlagen angewandt werden kann und deshalb keinen entscheidenden Mehraufwand gegenüber herkömmlichen Methoden verlangt. Der technische Aufbau des Werkzeugs ist auch kurz beschrieben.

In Kapitel 6 sind verschiedene Ansätze für die Software-Entwicklung zusammengestellt, die Formale Begriffsanalyse verwenden. Besonderes Augenmerk liegt dabei auf der jeweiligen Wahl von formalen Gegenständen und formalen Merkmalen.

Eine Darstellung des Beitrags von BASE zum Gebiet der objektorientierten Anwendungsentwicklung und möglicher Erweiterungen bzw. Kombinationen mit anderen Ansätzen beschließt die Arbeit in Kapitel 7.

Der kompakteren Darstellung wegen wurden einige Teile der mathematischen Grundlagen aus Kapitel 3 in den Anhang A ausgelagert. Im wesentlichen betrifft dies überlange Beweise. Es finden sich dort aber auch einige ergänzende Informationen, die für die Darstellung von BASE nicht direkt benötigt werden.

Der Anhang B enthält einen Teil über Algorithmen zur FBA. Zitiert werden die Algorithmen zur Berechnung der mathematischen Strukturen. Darüber hinaus wurde der üblicherweise genutzte Algorithmus zum automatischen Zeichnen von Linien-

diagrammen an die Bedürfnisse von BASE angepaßt.

Der Anhang C besteht aus der groben Anforderungsbeschreibung, auf deren Grundlage das in Kapitel 4 benutzte Beispiel modelliert wurde.

Für Interessierte sind in Anhang D die Schnittstellen der wichtigsten Klassen des in Kapitel 5 vorgestellten Werkzeugs angegeben.

2 Objektorientierte Anwendungs-entwicklung

Dieses Kapitel ist den Grundlagen der objektorientierten Anwendungsentwicklung – wie sie BASE zugrunde liegen – gewidmet.

Der erste Abschnitt 2.1 führt das verwendete Vokabular für das generelle Vorgehen bei der Software-Entwicklung ein. In 2.2 werden die grundlegenden Ideen der Objektorientierung darstellt. Diese entstammen der objektorientierten Programmierung (*OOP*). In 2.3 werden sie explizit in den Kontext der Systemanalyse gerückt. Abschnitt 2.4 stellt die am weitesten verbreiteten objektorientierten Methoden vor. Sie zeichnen sich einmal durch die von ihnen als Beschreibungsmittel vorgesehenen Modelle – d.h. ihre Notation – und zum anderen durch ein Vorgehensmodell, das die Anwendung der Modelle innerhalb der Entwicklung vorgibt, aus. Durchgehend wird als Notation die *UML* (Unified Modeling Language, [BJR 97]) in der Version 1.1 benutzt, während bei der Darstellung der Vorgehensaspekte die verschiedenen Methoden einzeln berücksichtigt werden, weil sich für diesen Bereich noch kein weitgehend akzeptierter Standard herausgebildet hat. Der Abschnitt 2.5 widmet sich dann ausführlicher der Methode der *Anwendungsfall-getriebenen Analyse*, wie sie von Jacobson eingeführt und für BASE übernommen wurde. In 2.6 wird untersucht, welche Vorgehensweisen innerhalb verschiedener bekannter Entwicklungsmethoden vorgesehen sind, um die zentralen Modellelemente – Objekte und Klassen – festzulegen. Diesen Analyseschritt zu unterstützen, ist das zentrale Anliegen von BASE.

2.1 Phasen der Anwendungsentwicklung

Bei der Entwicklung von Software-Systemen sind verschiedene Tätigkeiten durch-

zuführen, die man den Phasen *Analyse*, *Anforderungsdefinition* (oder "fachlicher Entwurf"), (technischer) *Entwurf*, *Implementierung*, *Integration*, *Betrieb & Wartung* zurechnen kann. Bei der *Analyse* verschafft man sich ein Bild vom *Untersuchungsbereich*, der durch Software unterstützt werden soll, die *Anforderungsdefinition* stellt heraus, was das Softwaresystem leisten muß, im *Entwurf* wird festgelegt, wie es das leisten kann und in der *Implementierung* wird der Entwurf in lauffähigen Programm-Code gegossen. Schließlich ist die neue Software noch beim Kunden zu installieren und ihr Betrieb unterstützend zu begleiten.

Der Terminus "Phase" rührt dabei von den traditionellen – und bis heute weit verbreiteten – *Vorgehensmodellen* (siehe auch 2.4), den *Wasserfall-Modellen* (vergl. [Boe 76]) her. Bei ihnen geht man davon aus, daß die Projekte nacheinander Phasen durchlaufen werden. Dabei findet sich in der Regel eine ähnliche Unterteilung wie oben. Häufig werden die Phasen Analyse und Anforderungsdefinition speziell in objektorientierten Methoden nicht getrennt. Modernere Vorgehensmodelle geben die strikt lineare Phasenabfolge auf und sehen eher zyklische Abläufe vor. Dadurch kann die Projektplanung besser situativ angepaßt werden. Eine solche Anpassung ist nach Untersuchungen von Software-Projekten gerade ein entscheidender Erfolgsfaktor (siehe [W-O 92]). Eine Zuordnung der Tätigkeiten zu den oben aufgezeigten oder ähnlichen Phasen ist aber auch in solchen Vorgehensmodellen oft zu finden. Einen guten Einstieg in und ersten Überblick über das Thema "Vorgehensmodelle" bietet [HMF 92].

Im weiteren werden die Phasenbezeichnungen von oben verwendet, um die Tätigkeiten während der Entwicklung und die dabei erzielten Ergebnisse einzuordnen. Ob die Phasen dabei rein sequentiell oder etwa zyklisch mehrfach durchlaufen werden, sei erst einmal dahingestellt. Der Abschnitt 2.4 geht auf diese Frage näher ein.

2.2 Prinzipien der Objektorientierung

Im Abschnitt 2.2.2 werden die wichtigsten Grundlagen der Objektorientierung erläutert. Die entsprechenden Zwischenüberschriften benennen diese Grundlagen. Zum Teil entstammen die betreffenden Ideen nicht originär der Objektorientierung, sondern wurden von ihr neu aufgenommen und interpretiert. Der erste Abschnitt 2.2.1 geht zur Motivation des Weiteren auf ein anderes Entwicklungsparadigma – das der funktionalen Zerlegung – ein. Diese Vorgehensweise wird bei der objektorientierten Systementwicklung in der Regel nicht praktiziert, weil sie in ihrer ursprünglichen Form zu wenig flexiblen Systemen führt, deren Struktur spätere Än-

derungen erschwert oder sogar innerhalb eines vertretbaren Aufwands unmöglich macht. BASE nimmt trotzdem die Idee der funktionalen Zerlegung auf, vermeidet jedoch die erwähnten Nachteile dieser Methode. Der abschließende Abschnitt 2.2.3 erläutert den wesentlichen Vorteil einer objektorientierten Vorgehensweise gegenüber den traditionellen Entwicklungsparadigmen.

2.2.1 Exkurs: Funktionale Zerlegung

Traditionell betrachtet man bei der Systementwicklung im wesentlichen zwei Sichten. Auf der einen Seite verwaltet ein System *Daten*. Für diese Datenverwaltung sind in der Entwicklung geeignete Beschreibungen der Daten – die *Datenstrukturen* – bereitzustellen. Auf der anderen Seite sind die *Funktionen* zu modellieren, die das System auf seinen Daten ausführt (vergl. z.B. SADT, [Ros 77]). Ein solches Verständnis von Software-Systemen wird unter der Überschrift "Algorithmen und Datenstrukturen"[1] angehenden Informatikern schon in der Grundausbildung eingeimpft.

Früher wurden dabei das Daten- und das Funktionsmodell weitgehend unabhängig voneinander erstellt. Das Paradigma war die *schrittweise Verfeinerung* ([Wir 83]). Dieses Vorgehen spiegelt eine natürliche und immer wiederkehrende Vorgehensweise des alltäglichen Lebens dar und bietet so ein naheliegendes Abstraktions- und Entwicklungskonzept. Man unterteile ein zu lösendes Problem sukzessive in Teilprobleme, bis man die entstandenen Teilprobleme direkt lösen kann und füge nach ihrer Lösung die Teillösungen zu einer Gesamtlösung zusammen. Wesentliches Problem bei der Software-Entwicklung ist die Erfüllung der funktionalen Anforderungen. Die entsprechenden übergreifenden Systemfunktionen zerlegt man in Teilfunktionen, bis man diese direkt verstehen und dann auch programmieren kann. Die Strukturierung des Systems geschieht so in funktional geprägter Sicht.

Die einzelnen Funktionen arbeiten aber in der Regel auf einem Daten-Pool zusammen, es bestehen also Datenabhängigkeiten zwischen ihnen. Diese werden jedoch durch die allein aus funktionalen Gesichtspunkten entwickelte Systemstruktur nicht widergespiegelt. Das ist problematisch. Ian Sommerville schreibt dazu: "Function-oriented design conceals the details of an algorithm in a function but system state information is not hidden. This can cause problems because a function can change the state in a way which other functions do not expect. Changes to a function and the way in which it uses the system state may cause unanticipated interactions with other functions." ([Som 92], S. 220). Auch Änderungen einer Datenstruktur sind problematisch, denn sie haben Auswirkungen auf viele verschie-

1. Nach [Wir 75] tragen eine Vielzahl von Lehrbüchern genau diesen Titel.

dene Funktionen, die möglicherweise in der Hierarchie der funktionalen Zerlegung weit von einander entfernt liegen und deren Funktionalität von der fachlichen Bedeutung nichts gemeinsam hat. Dadurch ist es schwer, alle von einer Änderung betroffenen Funktionen zu bestimmen. Insgesamt wird das System fehleranfällig und insbesondere änderungsfeindlich.

2.2.2 Prinzipien objektorientierter Modelle

Datenabstraktion

Um die oben geschilderten Nachteile zu vermeiden, modularisiert man Systeme nach dem Prinzip der *Datenabstraktion* (vergl. [L-Z 74]). Funktionen, die gleiche Daten benutzen, werden um diese Daten gruppiert. Eine solche Gruppierung nennt man *Modul*. Zugriffe von außerhalb des Moduls auf die Daten innerhalb werden nur über Funktionen des Moduls erlaubt. So sind die Daten gekapselt und Änderungen an ihnen beeinflussen (idealerweise) nur den eigenen Modul. Man unterteilt den Modul in einen *privaten* und einen *öffentlichen* Teil (die *Schnittstelle*). Nach dem Prinzip der Datenabstraktion bestehen die Modulschnittstellen nur aus Vereinbarungen von Funktionen des Moduls. Deren Implementierung und die von ihnen manipulierten Daten sind von außen unsichtbar.

Durch solche (klar definierten) Schnittstellen ist es möglich, einen Modul zu benutzen, ohne seine interne Organisation zu kennen. Insbesondere kann der Entwickler eines Moduls M_1 Funktionen des Moduls M_2 benutzen, ohne deren Implementierung zu kennen. Sieht er also sogenannte *stubs* vor, die die aufgerufenen Funktionen aus M_2 (etwa durch Rückgabe eines konstanten Werts oder Nutzung einer alten Version der entsprechenden Funktion) simulieren, kann er seinen Modul M_1 schon implementieren und testen, ohne auf die Fertigstellung der Funktionen aus M_2 warten zu müssen. So wird die parallele Arbeit mehrerer Entwickler möglich.

Um die Leistungen eines Moduls schon vor dessen Implementierung festzulegen und potentiellen anderen Entwicklern bekannt zu machen, beschreibt man sie in einer *Spezifikation*. Diese umfaßt eine Darstellung der *Syntax* und der *Semantik* der an der Schnittstelle angebotenen Funktionen. Das führt auf die *Abstrakten Datentypen*. Die Syntaxbeschreibung erfolgt dabei durch Angabe der Funktionsköpfe in einer Programmiersprache (oder einer ähnlichen Notation). Für die Darstellung der Semantik fehlt jedoch vielen Programmiersprachen das entsprechende Mittel, denn sie ist ja im Endeffekt durch den (vollständigen) Funktionsrumpf gegeben. Deshalb benutzt man eher mathematische Darstellungen. Zu unterscheiden sind die algebraische und operationale Spezifikation. Erstere beschreibt das Zusammenspiel der einzelnen Funktionen des abstrakten Datentyps. Operational bezieht man sich da-

gegen auf ein zugrundeliegendes mathematisches Modell, anhand dessen die Wirkungsweise jeder Funktion für sich (durch Angabe von Vor- und Nachbedingung oder nach dem Schema "Eingabe - Ausgabe - Funktionaler Zusammenhang zwischen beiden") angegeben wird.

Objekte und Klassen

Ein grundlegendes Prinzip der Objektorientierung ist gerade die Kapselung von Daten. *Objekte* speichern in ihren *Attributen* ihre eigenen Daten und führen Funktionen – genannt *Operationen* – auf diesen aus. Ein Objekt kann jedoch bei durchgehend praktizierter Datenkapselung die Attribute anderer Objekte nur via Operationen nutzen. Allerdings schleppt nicht jedes Objekt des selben Typs die Implementierung seiner Operationen mit sich herum, sondern überläßt dieses der entsprechenden Klasse. Eine *Klasse* beschreibt den Aufbau gleichartiger Objekte. In ihr werden die Attribute und Operationen definiert. Klassen fungieren also als *Module*. Sie werden aber auch als *Typ* ihrer Objekte behandelt ([Mey 88], S. 228: "Classes [...] are both modules and types."). Für den Entwurf eines objektorientierten Systems sind die Klassen die grundlegenden Systembausteine. Zur Laufzeit des Systems werden Objekte erstellt, die zusammenarbeiten, um die Systemfunktionalität bereitzustellen. Die zu einem bestimmten Zeitpunkt aktuelle Attributbelegung eines Objekts beschreibt dessen *Zustand*. Im Laufe der Zeit nimmt es verschiedene Zustände an, durchläuft seinen *Lebenszyklus*.

Ausgangspunkt der objektorientierten Ideen war die Programmierung (OOP) mit Sprachen wie *Simula* ([SIS 87]), *Smalltalk* ([G-R 83], *Eiffel* ([Mey 92]), *C++* ([Str 92]) und zuletzt *Java* ([A-G 96]. Durch die Fokussierung auf Klassen als grundlegende Software-Bausteine zwingen sie schon durch ihre Syntax zur gemeinsamen Betrachtung von Daten und Funktionen. Im Sinne der Schnittstelleneinschränkung auf Operationsdeklarationen wird die Datenabstraktion jedoch nicht von allen Programmiersprachen verlangt. Java verlangt aber z.B. eine solche Einschränkung in den Java-Interfaces (vergl. [A-G 96]). In Eiffel-Interfaces (vergl. [Mey 92]) kann man Attribute und parameterlose Funktionen (im Sinne von Operationen mit Rückgabewert) nicht unterscheiden. Deshalb kann die Entscheidung, ob man ein Attribut öffentlich macht oder nur eine öffentliche Zugriffsfunktion für es bereitstellt, getroffen werden, ohne eine Änderung der Klassenschnittstelle nötig zu machen.

Nachrichten

Objekte müssen zusammenarbeiten, um die Systemfunktionalität bereitstellen zu können, weil jedes Objekt für sich nur über seine eigenen Daten und eine eng begrenzte Funktionalität verfügt. Man stelle sich z.B. ein Textverarbeitungssystem

vor, in dem Paragraphen, Wörter und Zeichen als Objekte behandelt werden. Ein Paragraph kennt die Höhe und Breite seiner Zeilen, ein Zeichen kennt seine Höhe und Breite. Die Aufgabe, einen Paragraphen zu setzen, fällt dem Paragraphen selber zu. Er legt die Zeilenhöhe und -länge fest. Für das weitere Vorgehen ist er aber auf die Zusammenarbeit mit Wörtern und Zeichen angewiesen. Um festzustellen, ob das als nächstes zu setzende Wort noch in die Zeile paßt, muß er dessen Länge kennen, er schickt also diesem Wort eine Nachricht und fragt nach seiner Länge. Das Wort wiederum fragt seine Zeichen nach ihrer Länge und addiert die ermittelten Werte. Mit dem Ergebnis kann der Paragraph entscheiden, ob das Wort in die angefangene oder eine neue Zeile muß. Entsprechend setzt er die horizontale und vertikale Position. Diese übermittelt er dem Wort, das sie wiederum an seine Zeichen weitergibt. Jedes einzelne Zeichen stellt sich an der übergebenen Position dar. Genauso berechnet der Paragraph aus der Wortlänge die horizontale Position des nächstes Wortes in der selben Zeile.

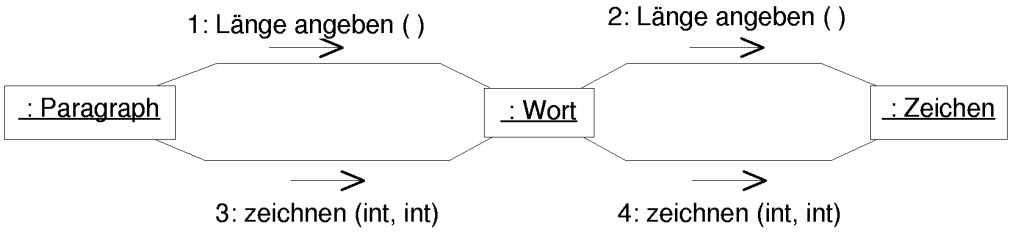


Abb. 2.1 Nachrichten zwischen Objekten eines Textverarbeitungssystems

In Anlehnung an die Terminologie für verteilte Systeme bezeichnet man das Objekt, das von einem anderen eine Leistung zu verlangt, als *Client-Objekt* und das Objekt, das die Leistung erbringt als *Server-Objekt*. Um von einem anderen Server-Objekt einen Dienst zu verlangen, schickt ihm das Client-Objekt eine *Nachricht*. Nach dem Prinzip der Datenkapselung bedeutet das, daß es beim Server-Objekt eine Operation aufruft.

Schon das kleine Beispiel zeigt, daß die Rollen von Client und Server nicht festgelegt sind. Zuerst geht eine Nachricht zur Längenermittlung vom Paragraphen zum Wort, so daß der Paragraph als Client und das Wort als Server auftreten. Dann aber schickt wiederum das Wort eine entsprechende Nachricht an seine Zeichen, so daß bei diesem Nachrichtenaustausch das Wort die Rolle des Client und das Zeichen die Rolle des Server übernehmen.

Um die Rollen von Client und Server herauszustellen, behandelt man im normalen Fall einer synchronen Nachrichtenverbindung, bei der der Client auf die Antwort des Servers wartet, die Wertrückgabe vom Server an den Client nicht als eigene Nachricht.

Abhängig von seinem Zustand kann ein Server-Objekt verschieden auf eingehende Nachrichten reagieren.

Vererbung

Neben der Änderungsfreundlichkeit steht die *Wiederverwendung* von Systembausteinen als Ziel der Modularisierung im Vordergrund. Teile von alten Systemen sollen in neuen wiederverwendet werden, damit man das Rad nicht immer neu erfinden muß. So kann man die Produktivität der Entwickler steigern. Speziell die Vertreter objektorientierter Methoden haben sich die Wiederverwendung auf die Fahnen geschrieben ([JCJÖ 93], S. 27: "Problems with reuse include finding, understanding and appropriateness of the thing to be reused. Object-orientation gives a completely new technique that strongly supports these issues.").

Klassen sind Kandidaten für wiederverwendbare Systembausteine. Da sie jedoch zum Zweck der Wiederverwendung oft modifiziert werden müssen, gibt es mit der *Vererbung* einen speziellen Mechanismus, schon bestehende Klassen an neue Anforderungen anzupassen. Von einer bestehenden *Oberklasse* wird eine neue *Unterklasse* dadurch abgeleitet, daß sie die Attribute und Operationen der Oberklasse übernimmt und zusätzlich eigene hinzufügen kann. So bleibt die alte Klasse als schon erprobter Systembaustein bestehen und in der neugebildeten existiert eine angepaßte Version.

Objekte der Unterklasse haben alle Attribute und Operationen der Oberklasse und können deshalb auch als Objekte der Oberklasse behandelt werden (was die Termini "Ober-" und "Unterklasse" erklärt). Auch typisierte objektorientierte Programmiersprachen erlauben es deshalb, Objekte der Unterklasse in einer Variable vom Typ der Oberklasse zu speichern. Dann sind aber für diese Objekte von außen auch nur die öffentlichen Attribute und Operationen der Oberklasse und nicht die in der Unterklasse zusätzlich definierten zugreifbar. In diesem Zusammenhang spricht man auch von dem *statischen* (Typ der Variable, die Oberklasse, direkt aus dem Programmcode zu bestimmen) und dem *dynamischen Typ* (wirklicher Objekttyp, die Unterklasse, erst zur Laufzeit bestimmbar) des betreffenden Objekts (vergl. [Mey 88], S. 226).

Vererbung hat genau wie das Klassenkonzept zwei verschiedene Aspekte. Betrachtet man Klassen als Module, die Attribute und Operationen zusammenfassen, so erweitert eine Unterklasse eine Oberklasse durch Hinzunahme neuer Attribute und Operationen. Dies betrifft den Wiederverwendungsaspekt der Vererbung. Betont man dagegen den Typaspekt der Klassen, ist eine Unterklasse eine Spezialisierung seiner Oberklassen und Vererbung bekommt einen taxonomischen Aspekt. ([Mey 88], S. 233: "So inheritance is specialization from the type viewpoint and extension from the module viewpoint.")

Häufig können in Unterklassen schon bestehende Operationen der Oberklassen einfacher oder effizienter implementiert werden, weil in der Unterklasse zusätzliche Attribute oder einschränkende Beziehungen zwischen den Attributen existieren. Deshalb erlaubt man, daß Unterklassen die Implementierung von Operationen ihrer Oberklassen *überschreiben*.

Polymorphie

Eine Nachricht kann an verschiedene Objekte (und auch an solche verschiedenen Typs) geschickt werden. Im besonderen ist jede Nachricht, die an ein Objekt einer Klasse *C* gehen kann, auch für Objekte aller Unterklassen von *C* sinnvoll.

Ein Objekt, das statisch vom Typ *C* ist, kann eigentlich (dynamisch) Exemplar einer Unterklasse von *C* sein. Sei *C'* diese Unterklasse. Ein anderes eine Nachricht sendendes Client-Objekt kennt nur den statischen Typ *C*. Das Wissen um den dynamischen Typ *C'* ist aber nötig, um die richtige Implementierung einer aufgerufenen Operation zu finden, denn Operationen von *C* können ja in *C'* überschrieben worden sein.

Die Eigenschaft, daß ein Client-Objekt darauf vertrauen kann, daß auf seine Nachricht hin immer die spezielle Implementierung der passenden Unterklasse zur Ausführung kommt, ohne selber den dynamischen Typ des Objekts kennen zu müssen, bezeichnet man als Polymorphie. Verschiedene Objekte können damit auf die gleiche Nachricht verschieden – ihrem Typ angepaßt – reagieren. Um dieses leisten zu können, benutzen objektorientierte Sprachen *spätes* oder *dynamisches Binden* – erst zur Laufzeit wird die Sprungadresse zu einem Operationsaufruf bestimmt. Manchmal wird die Zuweisung eines Unterklassenobjekts an eine Oberklassenvariable in einem objektorientierten Programm auch als polymorphe Zuweisung bezeichnet.

Zum Beispiel soll ein Grafik-Programm Zeichnungen, die aus beliebigen zweidimensionalen Figuren bestehen, bearbeiten. Dazu stehen u.a. die Klassen "Zeichnung", "Figur" und "Rechteck" zur Verfügung. "Rechteck" ist dann eine Unterklasse von "Figur". Eine interessante Fragestellung könnte die Ermittlung der Fläche aller in einer Zeichnung enthaltenen Figuren sein. Für beliebige Figuren könnte diese mit Hilfe einer Finite-Elemente-Approximation berechnet werden, während für Rechtecke nur die Seitenlängen multipliziert werden müssen. Die Klasse "Rechteck" überschreibt also die Operation "Fläche angeben" seiner Oberklasse "Figur". Eine Zeichnung pflegt eine Liste der in ihr enthaltenen Figuren. Um beliebige Figuren in dieser Liste speichern zu können, kommt als Variablentyp nur die Klasse "Figur" in Frage. Auch wenn eine Zeichnung Rechtecke enthält, werden diese in Variablen des Typs "Figur" gespeichert. Beim Anlegen der Liste findet in diesem Fall also eine polymorphe Zuweisung statt. Beim späteren Zugriff steht

dann aber zunächst nur die Information zur Verfügung, daß es sich um Figuren handelt.

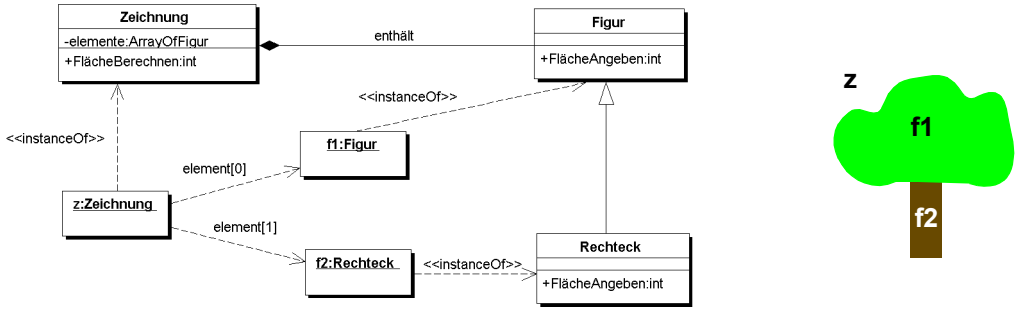


Abb. 2.2 Klassen und Objekte eines Grafik-Programms

Um die Gesamtfläche einer Zeichnung zu berechnen, müssen die Flächen der einzelnen Figuren bestimmt werden. Aus Effizienzgründen sollte für Rechtecke die einfachere Berechnungsvorschrift für ihre Oberfläche benutzt werden. Die Beispielzeichnung "z" schickt jetzt einfach an beide Figuren "element[0]" und "element[1]" die selbe Nachricht und ruft die Operation "FlächeAngeben" auf. Es wird dann automatisch die Implementierung des dynamischen Typs der jeweiligen Figur zur Ausführung gewählt, also für "f1" die komplexe und für das Rechteck "f2" die einfache Berechnungsvorschrift.

Zusammen mit der Vererbung ermöglicht das Prinzip der Polymorphie die Wiederverwendung einer bestehenden Klasse *C* mit allen ihren Eigenschaften, bei gleichzeitiger Möglichkeit, Erweiterungen durch eine neue Unterklasse *C'* von *C* einzubringen. Auch das Zusammenspiel von *C* mit anderen Klassen bleibt in *C'* erhalten. Klassen, die C benutzen, können ungeändert auch *C'* benutzen, wenn es für sie ausreichend ist, Objekte von *C'* als Objekte von *C* zu behandeln und sie die in *C'* zusätzlich definierten Attribute und Operationen nicht benötigen.

2.2.3 Das objektorientierte Entwicklungsparadigma

Im Bereich der Programmierung (*OOP*) hatte *Simula* ([SIS 87]) schon 1967 als erste Programmiersprache ein Klassenkonzept. *Smalltalk* ([G-R 83]), *Eiffel* ([Mey 92]) und *C++* ([Str 92]) waren die ersten weit verbreiteten Sprachen, die alle in 2.2 diskutierten Prinzipien umsetzen und immer noch aktuell sind. Mit dem steigenden Interesse an verteilten Systemen ist *Java* ([A-G 96]) zu den wichtigen Sprachen dazugekommen. Insbesondere die GUI-Programmierung ("Graphical User Interface") hat durch die Nutzung objektorientierter Sprachen enorm profitiert (wiederverwendbare Fensterklassen).

Programmiert man durchweg in Klassen – wie in Smalltalk, Eiffel und Java gar nicht anders möglich, enthalten die entstehenden Programme unter anderem auch Klassen als Repräsentanten von Dingen des Untersuchungsbereichs. Also ist die Idee, das objektorientierte Paradigma auch schon im Entwurf (*OOD*) und sogar schon in der Analyse (*OOA*) einzusetzen, nur naheliegend. Die Unterstützung der Software-Entwicklung durch ein einziges Paradigma über alle Entwicklungsphasen hinweg ist eine gemeinsame Idee der objektorientierten Modellierungsmethoden.

So vermeidet man Phasenbrüche. Coad/Yourdon schreiben dazu: "[...] moving from OOA to OOD is a progressive expansion of the model. [...] This expansion is in contrast to the radical movement from data flow diagrams to structure charts (or from data flow diagrams to an object-oriented representation). Such movement is abrupt and forever disjoint: designers get a hint from the analysis, and then go off to the “real” thing. Such approach fails to bring the requirements as a central issue into design. Moreover, traceability between the two is difficult and, in content, not very helpful." ([C-Y 90], S. 163). Die Autoren heben dabei auf die klassische funktionsorientierte Modellierung ab, bei der in der Analyse Datenflußdiagramme (vergl. [DeM 78]) und im Entwurf Strukturdiagramme (vergl. [Y-C 79]) weite Verbreitung gefunden haben. Wechselt man die Modellierungskonstrukte von Phase zu Phase, besteht die Gefahr, ohne Bezug zum Modell der vorhergehenden Phase neu zu entwickeln. Speziell problematisch ist die Verfolgbarkeit von Modelländerungen (besonders in Rückrichtung, wenn etwa im Entwurf festgestellt wird, daß am Analysemodell etwas geändert werden muß). Denn lassen sich die Modellierungskonstrukte der verschiedenen Phasen nicht in klar definierter Weise ineinander überführen, können Änderungen in einem Modell im anderen Modell nicht nachvollzogen werden.

2.3 Rollen der Klassen bei der Analyse und Modellierung

Da das Paradigma der Objektorientierung in der Programmierung (*OOP*) entstanden ist, sind die grundlegenden Prinzipien im vorigen Abschnitt teilweise technisch formuliert. Die Beschreibungen sind *lösungsorientiert* (Lösung ist das fertige Programm.) Bei der *objektorientierten Analyse* (*OOA*) behandelt man jedoch keine Klassen einer Programmiersprache, sondern solche, die man zur Beschreibung von Dingen des Untersuchungsbereichs benutzt. Statt technischer Kriterien stehen inhaltliche Aspekte im Vordergrund. Man nimmt eine *problemorientierte* Sichtweise ein.

2.3.1 Charakter von Klassen

Innerhalb von Programmiersprachen haben Klassen Typ- und Modulcharakter (siehe 2.2.2 auf Seite 25). Sie sind Typ ihrer Objekte und der Programmvariablen, die diese Objekte speichern. Der Modulaspekt betrifft die Gliederung von Attributen und Operationen. Dies hat weniger Bedeutung für den fachlichen Inhalt einer Klasse als für die spätere Implementierung, bei der durch die Modularisierung die Erstellung von redundantem Code innerhalb verschiedener Operationen vermindert wird. Unter fachlichen Gesichtspunkten tritt der Modulaspekt in den Hintergrund.

Bei einer top-down-Modellierung von Klassen steht zunächst der Typaspekt im Vordergrund. Nachdem die Semantik einer Klasse festgelegt ist, werden Attribute und Operationen zur genaueren Klassenbeschreibung modelliert. Aber auch wenn man dagegen bottom-up von bestehenden Attributen und Operationen ausgeht, werden diese nur zu solchen Klassen zusammengefaßt, die für den Untersuchungsbereich hilfreiche Abstraktionen darstellen. Zunächst einmal müssen die Klassen in ihrer fachlichen Bedeutung festgelegt sein, um überhaupt ein aussagekräftiges objektorientiertes Analysemodell aufstellen zu können. Der Modulaspekt ist also in der Analyse gegenüber dem Typaspekt sekundär.

In der Analyse dienen Klassen zur *Abstraktion*. Sie reduzieren die Betrachtung einer Menge einzelner Objekte auf deren Gemeinsamkeiten. Bei der Einteilung von Objekten in Klassen nimmt man eine Klassifikation vor ([Boo 94], S. 146: "The identification of classes and objects is the hardest part of object-oriented analysis and design. Our experience shows that identification involves both discovery and invention. [...] Ultimately, discovery and invention are both problems of classification [...].").

So bekommen die Klassen einen *extensionalen* und einen *intensionalen* Charakter, einerseits betrachtet man sie als Zusammenfassung ihrer Objekte, andererseits legen sie als Typ die gemeinsamen Eigenschaften der Objekte fest. ([Boo 94], S. 103: "A class is a set of objects that share a common structure and a common behavior."; S. 65: "For our purposes, we will use the terms type and class interchangeably.") Diese Dualität findet sich auch bei "Begriffen" (vergl. Kapitel 3) wieder. Deshalb ist die in 1.6.2 angedeutete Möglichkeit, formale Begriffe zu modellieren, die in ihrer Extension Objekte und in ihrer Intension Attribute und Operationen (als Typbeschreibung) zusammenfassen, "natürlich" genannt worden. (Die in BASE erstellten formalen Begriffe sind in diesem Sinne nicht natürlich. Wie in 1.6.2 aufgeführt ist eine "natürliche" Begriffsmodellierung für BASE nicht adäquat, weil die Analyse auf Instanzenebene – und nicht allein auf Typebene – durchgeführt werden müßte.)

Begriffe sind das zentrale Hilfsmittel, um Dinge des Untersuchungsbereichs zu klassifizieren. Klassen sollten die zentralen Begriffe implementieren, um den Untersuchungsbereich treffend beschreiben zu können. Diese Vorgabe findet sich bei vielen Autoren:

- James Rumbaugh: "The objects in the model should be application-domain concepts [...]." ([RBP+ 91], S. 5)
- James Martin und James Odell: "Object types are important, because they create conceptual building blocks for designing systems." ([M-O 92], S. 76) und "An object type is a concept." ([M-O 95], S. 34)
- Grady Booch: "Key abstractions are the classes and objects that form the vocabulary of the problem domain." ([Boo 94], S. 143)

2.3.2 Beziehungen zwischen Klassen

Die Klassen eines Untersuchungsbereichs sind nicht völlig unabhängig voneinander. Es bestehen vielfältige Beziehungen zwischen ihnen. Diese werden in objektorientierten Analysemodellen – wie auch in *E/R-Modellen* (vergl. [Che 76]) – als eigene Modellkonstrukte betrachtet.

Klassen kann man als um Operationen erweiterte *Entitätstypen* behandeln ([BPR 88], S. 416: "The notion of an object is synonymous with entity in the ER and LRDM methods.", Anm.: *LRDM = Logical Relational Design Methodology*, siehe [TYF 86]). Entsprechend könnte man Beziehungen auch wie im E/R-Modell betrachten. Im Zuge der *semantischen Datenmodellierung* (vergl. [A-H 87]) wurden zwei Beziehungsarten, die für die Strukturierung des Modells eine herausragende Rolle spielen (*Generalisierung* und *Aggregation*) durch eigene Modellkonstrukte hervorgehoben. Dies ist für objektorientierte Analysemodelle übernommen worden. Darüber hinaus wichtig sind Benutzungsbeziehungen, die mittels Nachrichtenaustausch zwischen den beteiligten Objekten hergestellt werden (siehe 2.2.2 auf Seite 25).

So betrachtet man in der objektorientierten Analyse die folgenden Beziehungen:

- *Generalisierung/Spezialisierung*
- *Aggregation*
- *Assoziation*
- *Botschaftsbeziehung* (zwischen Objekten)

Die in der Analyse modellierten Beziehungen müssen später im Entwurf auf Vererbung oder Benutzung abgebildet werden.

Generalisierung/Spezialisierung bezieht sich auf die fachliche bzw. inhaltliche Sicht von Klassen. Repräsentieren die in einer Spezialisierungsbeziehung stehen-

den Klassen Begriffe, so sollte eine entsprechende Ober-/Unterbegriff-Beziehung bestehen. *Vererbung* ist ein Mechanismus, Generalisierungsbeziehungen in der Software zu verwirklichen, doch können die beiden Beziehungsarten nicht gleichgesetzt werden. In der Analyse angesprochen wird vorwiegend der Typaspekt, nicht der Modulaspekt von Klassen (siehe 2.2.2 auf Seite 25). Also steht der taxonomische Aspekt und nicht der Wiederverwendungsaspekt der Vererbung im Vordergrund (siehe 2.2.2 auf Seite 27).

Aggregation bezeichnet eine Teil-/Ganzes-Beziehung. Die Notwendigkeit dieses Modellkonstrukts war und ist Gegenstand anhaltender kontroverser Diskussionen. Man könnte Aggregation auch als Spezialfall der Assoziation (s.u.) behandeln. In den meisten Methoden und auch in UML ([BJR 99a] ist sie aus den übrigen Assoziationen herausgehoben. Kontrovers sind auch die Auffassungen, wie stark Klassen durch eine Aggregation gebunden werden, ob etwa ein Teil eines Ganzen ohne dieses Ganze als Objekt bestehen kann oder z.B. beim Löschen des umfassenden Objekts mit gelöscht wird. In der UML wird deshalb sogar noch die *Komposition* als starke Aggregation betrachtet.

Nicht alle inhaltlichen Zusammenhänge zwischen Klassen sind von der Form "ist ein" (Spezialisierung) oder "ist Teil von" (Aggregation), sondern es treten auch Beziehungen mit anderer Semantik auf, diese heißen *Assoziationen*.

Benutzungsbeziehungen, die zur Laufzeit auf Objektebene hergestellt werden, müssen nicht unbedingt durch eine auf Klassenebene interessante fachliche Beziehung gestiftet sein. Weil sie aber dennoch bei der Analyse und im Entwurf eine Rolle spielen, sind sie in der Liste der Klassenbeziehungen aufgeführt. Peter Coad und Edward Yourdon ([C-Y 90]) zum Beispiel dokumentieren sie auf Klassenebene. *Botschaftsbeziehungen* stellen Kanäle dar, auf denen Nachrichten (vergl. 2.2.2 auf Seite 25) zwischen Objekten ausgetauscht und so Benutzungsbeziehungen zwischen den entsprechenden Klassen gestiftet werden.

2.4 Objektorientierte Analyse- und Modellierungsmethoden

Seit Anfang der 90’er Jahre sind viele verschiedene Methoden zur objektorientierten Anwendungsentwicklung veröffentlicht worden. Die bekanntesten Methoden sind:

- *OOAD* von Grady Booch [Boo 94] (seit 1982)
- *OOSE* von Ivar Jacobcon et al. [JCJÖ 93] & [J-C 95] (seit 1985)
- *OOSA* von Sally Shlaer & Stephen J. Mellor [S-M 88], [S-M 92] & [S-M 93]

- *OSA* von David W. Embley & Barry D. Kurtz [EKW 92] (seit 1988)
- *OOA/D* von Peter Coad & Edward Yourdon [C-Y 90] & [C-Y 91] (seit 1989)
- *BON* von Kim Waldén & Jean-Marc Nerson [W-N 95] (seit 1989)
- *RDD* von Rebecca Wirfs-Brock et al. [WWW 90]
- *OMT* von James Rumbaugh et al. [RBP+ 91], [Rum 95] & [Rum 96]
- *Fusion* von Derek Coleman et al. [CAB+ 94] (seit 1992)
- *OOA&D* von James Martin & James Odell [M-O 92]

Diese Methoden sollen hier nicht eingehend verglichen werden, denn das ist schon zu Genüge unternommen worden (vergl. [SSB 91], [C-F 92], [M-P 92], [Ste 93], [ZGHK 97]). Nur die wichtigsten Modellierungsaspekte (die den Methoden gemeinsam sind) werden dargestellt. Allein die Frage, welches Verfahren die Autoren vorschlagen, um in der Analyse Klassenkandidaten zu finden, wird tiefgehender untersucht, weil dies die zentrale Fragestellung dieser Arbeit ist.

2.4.1 Methode = Modellierungskonzepte + Notation + Vorgehensmodell

Alle Methoden unterstützen gewisse *Modellierungskonzepte*. Die grundlegenden (Klasse, Objekt, Nachricht, Beziehungen zwischen Klassen und Objekten) werden von allen (mit kleinen Nuancen) geteilt. Sie wurden schon in 2.2 und 2.3 dargestellt. Aufbauend auf diesen grundlegenden Konzepten werden innerhalb eines Software-Entwicklungsprojekts *Modelle* erstellt. Die in einer Methode betrachteten Modelltypen bilden weitergehende Modellierungskonzepte. Modelle werden in einer (methodenspezifischen) *Notation* dargestellt. Deshalb sind Notation und Modellierungskonzepte eng miteinander verwoben. Zusätzlich enthalten die Methoden ein *Vorgehensmodell*, das die Vorgehensweise bei der Erstellung der Modelle beschreibt. Die verschiedenen Autoren legen dabei verschiedene Schwerpunkte. Einige betrachten nur die Modellierungskonzepte als primär, andere legen auch großen Wert auf ihre spezifische Notation oder ihr eigenes Vorgehensmodell. Eine vollständige Methode muß alle drei Aspekte berücksichtigen.

Zunächst sollen die verschiedenen involvierten Modellierungskonzepte beleuchtet werden. Mit der *Unified Modeling Language* (UML, [BJR 99a]) bildet sich für diesen Aspekt und die Notation von Modellen gegenwärtig ein Standard aus.[1]

1. Das ursprüngliche Ansinnen der Autoren der UML war, eine "Unified Method" (Projekttitel bis zur Version 0.9) zu erstellen, die Modellierungskonzepte, Notation und Vorgehensmodell umfaßt. Ab der Version 1.0 wurde das Projekt unter dem Titel "Unified Modeling Language" weitergeführt und die Anstrengungen auf die Standardisierung der Konzepte und der Notation konzentriert. Die Intention, eine komplette Methode inklusive Vorgehensmodell als Standard zu etablieren, erschien zunächst wohl doch zu ambitioniert. Inzwischen gibt es auch Publikationen zu einem "Unified Process" (z.B. [Krc 99], [BJR 99b], [Jac 99]).

Hier soll die Notation der UML in der Version 1.1 (nach [BJR 97]) verwendet werden. Auf die verschiedenen Modellierungskonzepte und Notationen der oben aufgelisteten Methoden wird nicht vertiefend eingegangen. Die UML bietet einen Kompromiß der Modelltypen und Notationen von Grady Booch, James Rumbaugh und Ivar Jacobson, zusätzlich garniert mit einigen Zutaten aus anderen Quellen. Dieser Kompromiß ist im wesentlichen durch Einbeziehung aller wichtigen Modelltypen der drei Amigos – wie sich die Autoren der UML selber nennen – entstanden. (Die Modelltypen der UML und ihre Herkunft werden noch einmal in 3.3 als Beispiel für die Formale Begriffsanalyse betrachtet.)

Wie bei den meisten anderen Autoren auch werden Modelle in erster Linie in Diagrammform präsentiert. Die Leitidee für die Notation ist "Software through pictures", wie ein weitverbreitetes Werkzeug für die Rumbaugh-Methode OMT und UML (vergl. [Aon 99]) heißt. Die wichtigsten Diagrammarten werden in 2.4.2 vorgestellt. Die Diagramme für die späten Phasen der Entwicklung werden hier vollkommen ausgespart, weil sie in diesem Zusammenhang von geringem Interesse sind.

Im Hinblick auf die Vorgehensmodelle wird in 2.4.3 stärker differenziert, weil es für diesen Aspekt keine allgemein akzeptierte Vorlage gibt ([Hes 97], S. 123: "At the moment it seems to be rather likely that no standard will evolve and the competing approaches will coexist at least for the next decade.").

2.4.2 Modelltypen

Unterschieden werden von allen Autoren bis auf Coad/Yourdon Modelltypen für die *statischen* und für die *dynamischen Aspekte* des Systems.

Statikmodelle

Im Kern des Interesses steht das *Klassenmodell*, das die Systemklassen und ihre Beziehungen beschreibt. Es ist das Rückgrat der Systembeschreibung. Wiedergegeben wird es in einem Klassendiagramm.

Solche *Klassendiagramme* geben die statische Systemstruktur wieder. Das vorliegende Beispiel in Abbildung 2.3 wurde (in leicht abgeänderter Weise) in einem Projekt innerhalb einer Lehrveranstaltung erstellt. Thema war die Modellierung eines Bibliothekssystems für die Fachbereichsbibliothek.

Klassen sind als Kästen eingezeichnet, in denen zunächst der Klassenname, dann die Attribute und schließlich die Operationen angegeben sind. *Generalisierungsbeziehungen* werden durch Pfeile von der Unterklasse zur Oberklasse notiert. Verbindungen durch einfache Linien kennzeichnen *Assoziationen*. Handelt es sich dabei um eine *Aggregation*, so ist dies durch eine Raute am Linienende der umfassenden

Klasse deutlich gemacht.

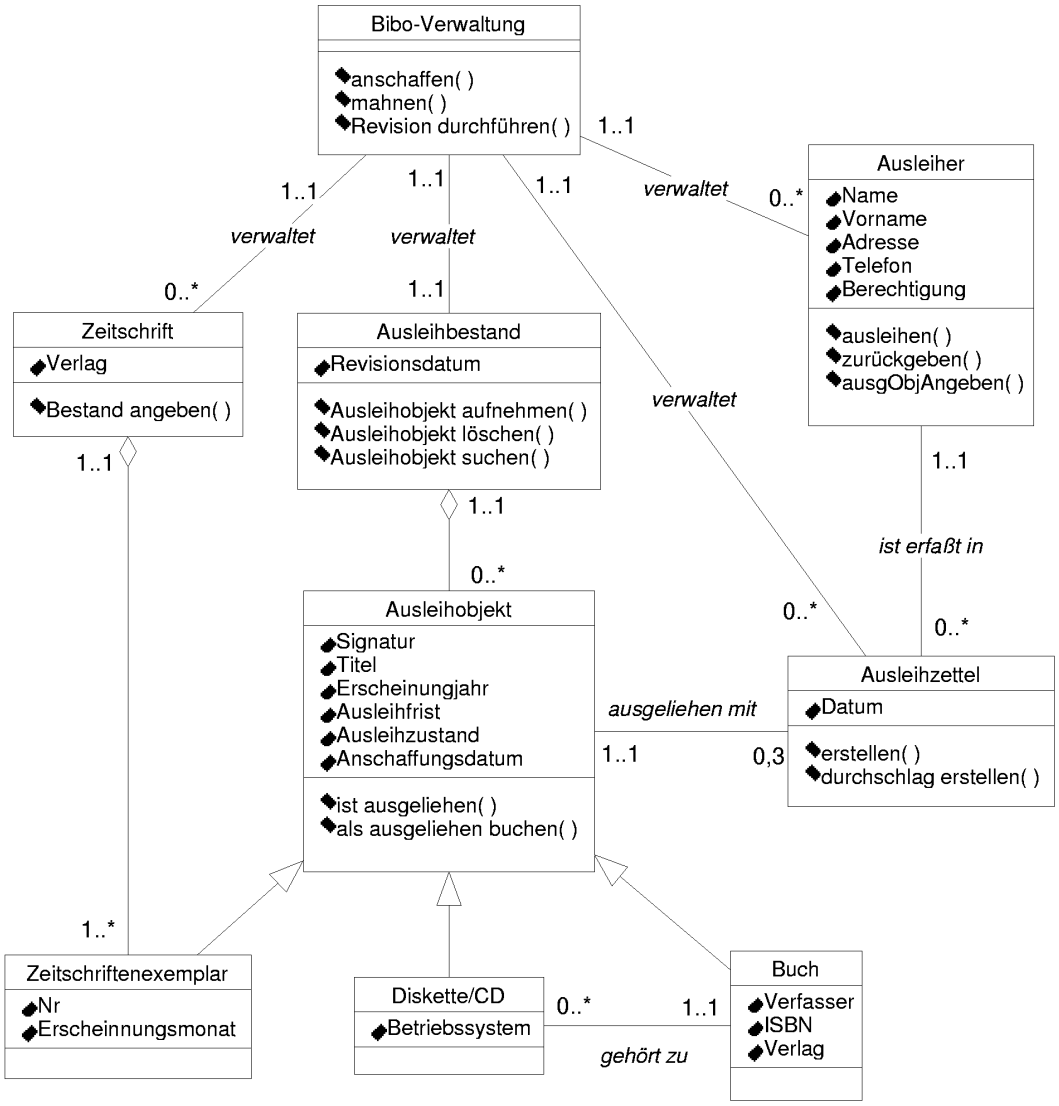


Abb. 2.3 Klassenmodell für ein Bibliothekssystem

In der Analysephase wird ein erstes problemorientiertes Klassenmodell erstellt, das den Untersuchungsbereich beschreibt. In der weiteren Entwicklung werden die Klassen dieses Modells in Richtung Implementierung detailliert und das Modell wird dabei durch lösungsorientierte Klassen, die aus Implementierungsgründen nötig oder sinnvoll sind, ergänzt.

In realen Anwendungen betrachtet man viel mehr Klassen als in dem oben angegebenen Beispiel-Klassendiagramm ([JCJÖ 93], S. 195: "For a medium-sized project, typically between 30 and 100 will be specified."). Die entsprechenden Modelle sind dann in ihrer Gesamtheit nur noch schwer zu überblicken. Deshalb be-

trachtet man oberhalb der Klassen *Systemkomponenten* als Strukturierungsebene und definiert *Sichten* für bestimmte Aspekte eines Klassenmodells, damit die entsprechenden Klassendiagramme übersichtlich bleiben. Komponenten fassen zusammengehörige Klassen zusammen. Sie sind untereinander disjunkt. Um ihre Abhängigkeit voneinander gering zu halten, versucht man die Klassen so zu gruppieren, daß sie innerhalb einer Komponente miteinander in Beziehung stehen, aber wenige Klassenbeziehungen über Komponentengrenzen hinweg gehen. Für jede Komponente entwickelt man ein eigenes Klassendiagramm. Mit Sichten kann man dann noch Filter bilden, die Klassen aus Diagrammen ausblenden (z.B. Hilfsklassen, die im Entwurf zugefügt wurden).

Besondere Bedeutung gewinnt eine *Komponentenzerlegung* des Systems im Hinblick auf Wiederverwendung. Anstatt nur einzelne Klassen wiederzuverwenden, sollen fachlich zusammengehörige Klassen als ganze Komponenten in neuen Projekten Anwendung finden. "Component Ware" ist ein großes Schlagwort ([Szy 98], S. xii: "[...] component technology is expected to be THE cornerstone of software in the years to come."). Ziel ist es, von der kompletten Neuentwicklung von Systemen in allen ihren Einzelteilen immer weiter weg zu kommen, sondern fertige Komponenten, die man entweder schon früher selber entwickelt hat, oder die man bei einem Komponentenanbieter einkauft, nach einer eventuellen Anpassung zusammenzustecken. Daraus resultieren Anforderungen an Komponenten ([Szy 98], S. "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.").

Eine Betonung der Wiederverwendung von Komponenten trifft nicht nur auf die späten Entwicklungsphasen zu. Schon Jacobson schreibt dazu: "Reusability is of course applicable during coding, since it can influence productivity significantly. [...] What can give even higher productivity is reuse in other development phases. Other parts of the construction phase may benefit when reusing entire designs in several systems. Additionally, reuse should also be viewed as natural during analysis and testing." ([JCJÖ 93], S. 26). Wirklich etabliert wurden diese Ideen innerhalb des Entwurfs von Erich Gamma durch seine *design patterns* (Entwurfsmuster) in [GHJV 95]. Entwurfmuster drücken in der Regel eine spezifische Art der Beziehungen und der Zusammenarbeit mehrerer Klassen aus, die immer wieder in Systementwürfen eingesetzt werden kann. Für ein spezifisches System ist die fachliche Ausprägung dieser Klassen noch festzulegen. Aber sie bleiben eine Einheit, die man zu einer Komponente machen kann. Ist die Nutzung von Entwurfsmustern erst standardisiert, reicht es dann, gegenüber anderen Systemteilen und auch anderen Entwicklern Bezug auf diese Komponente zu nehmen, ohne auf ihre einzelnen Bestandteile abheben zu müssen. So entstehen Standardkriterien, Komponenten zu

bilden. Quasi fortsetzend sind auch *analysis patterns* (Analysemuster) betrachtet worden ([Fow 97]). Der Komponentenbegriff bekommt also auch in den frühen Phasen der objektorientierten Entwicklung immer mehr Bedeutung.

Die völlige Unabhängigkeit von anderen Systemkomponenten braucht im hier diskutierten Zusammenhang nicht zu gelten, wenn die betrachteten Komponenten nicht als eigene Produkte entwickelt und vertrieben bzw. mehrfach eingesetzt werden sollen. Fachlich zusammengehörige Klassen werden zum gemeinsamen Entwurf zusammengefaßt (vergl. [Hes 98]). Dieses Zusammenfassen kann hierarchisch bis zum Gesamtsystem fortgesetzt werden. Die Forderungen nach klaren Schnittstellen bleibt erhalten, weil nur so die Wechselwirkung einer Komponente mit anderen Systemteilen modelliert werden kann, ohne die Interna der Komponente offen zu legen.

Die Abgrenzung des Komponentenbegriffs ist hier insbesondere deshalb vorgenommen, weil der Sprachgebrauch im Hinblick auf Komponentenzerlegungen sehr unterschiedlich ist. In UML und Java heißen Komponenten "packages", in BON und Eiffel "clusters", bei Booch "class categories", bei Coad/Yourdon "subjects", in Fusion "components", bei Embley/Kurtz "high-level components", bei Jacobson und Wirfs-Brock "subsystems"[1] und bei Rumbaugh "modules" oder "subsystems". Festzuhalten bleibt, daß nahezu alle Methoden Komponenten zur Zusammenfassung von Klassen vorsehen.

Dynamikmodelle

Wie die Objekte zur Laufzeit miteinander zusammenarbeiten, ist im Klassenmodell noch nicht beschrieben. Diese Zusammenarbeit ist nötig, um die Systemfunktionalität bereitzustellen. Deshalb ist es wichtig, sich schon in der Analyse ein Bild von dieser Zusammenarbeit zu machen. Für jede Systemfunktion – häufig beschrieben als *Anwendungsfall* (*use case*, siehe 2.5) – ist aufzuzeigen, wie sie durch zusammenarbeitende Objekte bereitgestellt werden kann ([JCJÖ 93], S. 200: "The objects should provide the complete functionality of the use cases."). Nur so kann abgeschätzt werden, ob die funktionalen Anforderungen durch ein auf dem Analysemodell aufbauendes System erfüllt werden können.

Aspekte der Zusammenarbeit von Objekten werden in *Interaktionsdiagrammen* beschrieben. UML sieht dafür zwei konkurrierende Diagrammarten vor: Das *Sequenzdiagramm* (Jacobsons "interaction diagram") betont den Ablaufaspekt, das *Kollaborationsdiagramm* (Boochs "object diagram") stellt dagegen die beteiligten Objekte und die durch die gegenseitige Benutzung entstehende Struktur in den Vordergrund. Häufige Anwendung und auch verbreitete Werkzeugunterstützung

1. Jacobson betrachtet "components" in oben dargestelltem eingeschränkten Sinn.

finden die Sequenzdiagramme (objectiF 3.2 von microTOOL ([Mic 99]) und Innovator 6.1 von MID ([Mid 99]) unterstützen z.B. nur Sequenzdiagramme.).

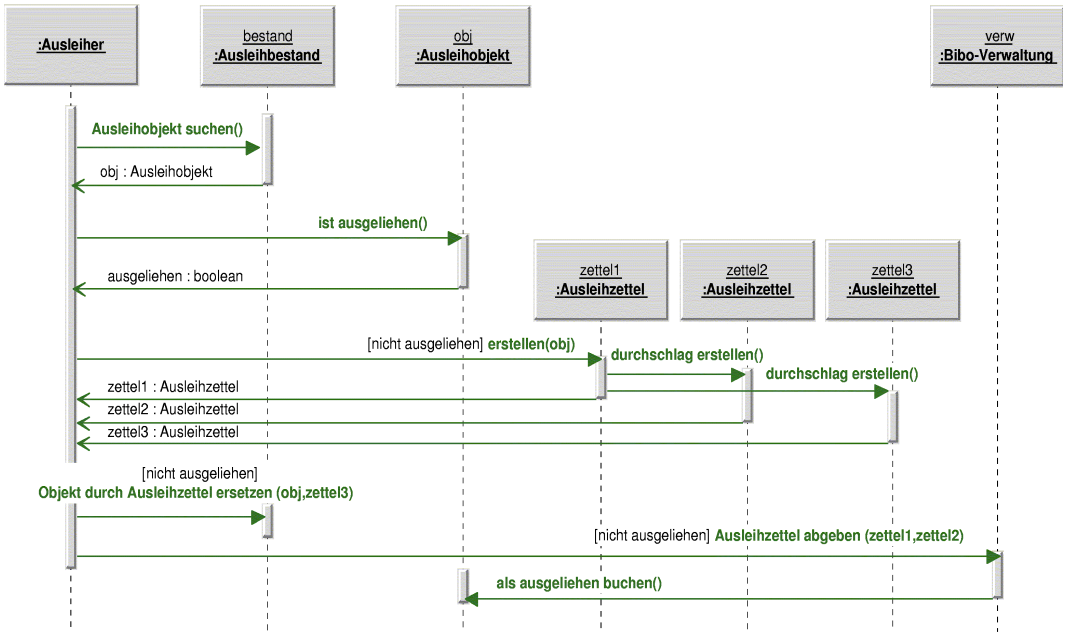


Abb. 2.4 Sequenzdiagramm für die Ausleihe bei einer Bibliothek

Ein Beispiel für ein Sequenzdiagramm ist in Abbildung 2.4 zu sehen. Es ist wie das Klassendiagramm in Abbildung 2.3 auf Seite 36 in Anlehnung an das Studentenpraktikum zur Erstellung eines Bibliothekssystems entstanden und gibt den Ablauf eines Ausleihvorgangs wieder. Kästen symbolisieren *Objekte*. Der betreffende Objektname ist vor und der Name der zugehörigen Klasse nach dem Doppelpunkt angegeben. Die Zeitachse verläuft vertikal. Gestrichelte senkrechte Linien sind die "*Lebenslinien*" der Objekte. Wird ein Objekt erst erstellt – wie hier der Ausleihzettel –, beginnt seine Lebenslinie weiter unten im Diagramm. In diesem Diagramm existieren alle Objekte auch noch nach dem dargestellten Ablauf. Das Löschen von Objekten wird ansonsten durch eine vorzeitig mit einem Kreuz endende Lebenslinie gekennzeichnet. Balken auf den Lebenslinien geben an, welche Objekte die Kontrolle bei der Ausführung haben. Zwischen den Lebenslinien sind die *Nachrichten* zwischen den Objekten an Pfeilen angetragen. In eckigen Klammern können Vorbedingungen für das Versenden einer Nachricht angegeben werden.

Durch die Objekte bzw. ihre Klassen und Nachrichten (die Operationsaufrufe bedeuten) sind Querverbindungen zum Klassenmodell gegeben.

Objekte können in Abhängigkeit von ihrem Zustand verschieden auf Nachrichten reagieren. Also ist es wichtig, auch ihre Zustände und möglichen Zustandsänderungen zu beschreiben. Bei RDD ist dieser Aspekt außer Acht gelassen, alle ande-

ren Methoden arbeiten mit *Zustandsdiagrammen* oder erwähnen sie zumindest (*state charts* von David Harel ([Har 87]), die auf endlichen Automaten ([Kle 56]) aufbauen, Abbildung 2.5).

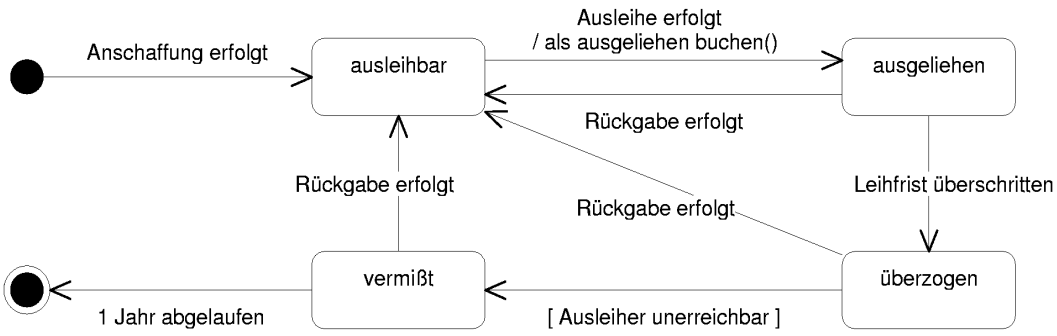


Abb. 2.5 Zustandsdiagramm der Klasse *Ausleihobjekt*

Die Kästen in solche Zustandsdiagrammen sind die *Zustände* eines Objekts. Start- und Endzustand sind durch Punkte (Endzustand mit einem Ring) angegeben. Sie einzubeziehen macht es möglich, das Erstellen und Löschen des Objekts auch zu erfassen. *Zustandsübergänge* sind durch Pfeile beschrieben. Sie können durch *Ereignisse* oder das Eintreten einer *Bedingung* (in eckigen Klammern angegeben) angeregt werden. Solche Ereignisse können global sein und im Diagramm einfach benannt werden oder im Eintreffen einer Nachricht beim betrachteten Objekt bestehen. Zustände können auch hierarchisch aufgebaut sein, das heißt, daß ein Zustand selber wieder durch ein Zustandsdiagramm gegeben ist. Die dort dokumentierten Unterzustände und Zustandsübergänge werden durch den umgebenden Zustand verborgen.

In 2.2.2 auf Seite 25 ist dargelegt, daß der Zustand eines Objekts durch seine Attributbelegung gegeben ist. In den Zustandsmodellen wird er aber einfach benannt. Das erlaubt eine Beschreibung auf höherer Abstraktionsebene – sogar, wenn noch nicht alle Attribute einer Klasse festgelegt sind. Später müssen diese Zustände mit Hilfe der Attribute beschrieben werden können. Dies stiftet einen Zusammenhang zu der Klassendarstellung im Klassenmodell.

Ereignisse als das Eintreffen einer Nachricht führen zu einem Operationsaufruf. Die Nachrichten sind in den Interaktionsdiagrammen dargestellt, zu denen so eine Querverbindung besteht. So geht in Abbildung 2.4 das Ausleihobjekt in Folge der letzten Nachricht im Ablauf des Sequenzdiagramms in den Zustand "ausgeliehen" über. Genauso besteht über die betreffenden Operationen eine weitere Verbindung zum Klassendiagramm.

2.4.3 Vorgehensmodelle

Eine eingehende Untersuchung der Vorgehensmodelle der verschiedenen Methoden findet sich in [Hes 97]. An dieser Stelle soll nur kurz beleuchtet werden, wie die in 2.4.2 beschriebenen Modelle angewandt werden, um die Entwicklung voranzutreiben. Vereinfachend werden dabei die verschiedenen Ausprägungen der oben diskutierten Modelle nicht unterschieden. Wo es wichtig ist, wird auf Unterschiede hingewiesen, doch generell reicht es, davon auszugehen, daß alle Methoden Modelle wie die obigen enthalten. In 2.6 wird dann eingehender verglichen, was die verschiedenen Autoren vorschlagen, um überhaupt erst einmal zu Klassen für ein Klassenmodell zu kommen.

In **Boochs** Vorstellung beginnt die Entwicklung mit einer Konzeptphase, in der die wichtigsten Anforderungen festgelegt werden. Als Technik empfiehlt er dafür Prototyping (vergl. [BKKZ 92]). In der darauf folgenden Analyse sollen zyklisch

- Objekte und Klassen gefunden,
- die Semantik der Objekte und Klassen festgelegt,
- die Beziehungen zwischen Klassen und Objekten identifiziert und
- die Schnittstellen der Klassen spezifiziert

werden. Insbesondere gesteht Booch dem Durchspielen von *Szenarien* (Arbeitsabläufe mit dem zu erstellenden System, siehe 2.5) eine große Rolle zu. Mit Hilfe dieser Technik werden im zweiten Schritt den Klassen *Verantwortlichkeiten* für Aufgaben – für die danach Operationen definiert werden – und damit verbundene Attribute zugeordnet. Die Zusammenarbeit von Objekten in einem Szenario beschreibt man durch Interaktionsdiagramme. Klassen und Interaktionsdiagramme werden also verzahnt entwickelt. Zum einen bieten die Objekte der Klassen und ihre Operationen die Basis für die Interaktionsdiagramme, zum anderen führt das Durchspielen der Szenarien anhand der Interaktionsdiagramme zur Entdeckung von neuen Objekten und Verantwortlichkeiten. Lebenszyklen für einzelne Klassen werden ebenfalls festgelegt. So ist durch diesen Schritt das Grundgerüst des gesamten Modells hergestellt. Im dritten Schritt werden Generalisierungs-, Aggregations- und Assoziationsbeziehungen zwischen den Klassen modelliert und so das Klassenmodell weiter verfeinert. Die Spezifikation der Klassenschnittstellen soll schon in einer Programmiersprache erfolgen. In der Entwurfs- und Implementierungsphase wird das nun schon bestehende Systemmodell im Hinblick auf seine Implementierung weiter ergänzt, etwa durch Hinzunahme implementierungsspezifischer Klassen.

Dem grundsätzlichen Ansatz von **Jacobson** für die frühen Phasen ist hier ein eigener Abschnitt (2.5) gewidmet, weil er den Ausgangspunkt für BASE bildet. In einer Anforderungsphase werden die Anwendungsfälle (Typen von Szenarien) zu-

sammengestellt und jeweils beschrieben und möglichst durch einen Oberflächen-Prototyp lebendig gemacht. Aus diesen Beschreibungen werden Objekte und Klassen abgeleitet, die ein direktes Gegenstück im Untersuchungsbereich aufweisen. So bekommt man ein erstes Klassenmodell. Dabei sollten Klassennamen, fachlich bedingte Attribute und Assoziationen zwischen den Klassen betrachtet und festgelegt werden. In der anschließenden Analysephase wird das in den Anwendungsfällen ausgedrückte Systemverhalten auf die Verantwortlichkeiten der einzelnen Klassen verteilt. Hier kommen die Interaktionsdiagramme ins Spiel. Im späteren Entwurf wird das Klassenmodell an die Entwicklungsumgebung – im wesentlichen die Programmiersprache – angepaßt. Dabei betrachtet Jacobson auch Zustandsmodelle.

Shlaer/Mellor sehen als ersten Entwicklungsschritt eine Zerlegung in Entwicklungsebenen ("domains"): die Anwendungsebene, einige Service-Ebenen (darunter die Benutzeroberfläche), die Architekturebene und einige Implementierungsebenen. Entwicklungsebenen können weiter in Komponenten zerfallen. In der Analyse wird dann das Klassenmodell aufgebaut, wonach die Lebenszyklen der Objekte modelliert werden. Zu den Zuständen gehören Aktionen, die bei Eintreten des Zustands ausgeführt werden. Sie werden im nächsten Schritt durch Datenflußdiagramme beschrieben. Aus dem so entstandenen Analysemodell der Anwendungsebene leitet man die Anforderungen an die Service-Ebenen ab und führt für diese Analysen nach dem selben Schema durch. In späteren Phasen werden die Analysemodelle weiter bis hin zur Implementierung umgesetzt. In diesem Zusammenhang betonen die Autoren die Möglichkeit der Code-Generierung besonders.

Embley/Kurtz legen sich in ihrer Methode OSA (Object-Oriented Systems Analysis) auf keine Vorgehensweise fest ([EKW 92], S. 14: "We make no strict rules about which modeling activity should take place before another. [...] Analysts may develop models top-down by decomposition or bottom-up by composition."). Für sie ist ein Objekt eine Person, ein Ort oder ein Ding. Normalerweise werden in der Analyse gleich Klassen von logisch zusammengehörigen Objekten betrachtet, denen auch gleichzeitig synonyme Namen zugewiesen werden können. In einem Beispiel beginnen sie mit einem Interaktionsdiagramm für die Objekte des Untersuchungsbereichs, entwickeln dann ein Klassenmodell und beschreiben anschließend ausgesuchte Klassen mit Hilfe von Zustandsdiagrammen genauer. OSA-Modelle werden später zu Systemspezifikation und -entwurf ausgebaut, auf denen aufbauend schließlich die Implementierung erfolgt.

Nach **Coad/Yourdon** sind in der Analyse folgende Tätigkeiten auszuführen:

- Objekte und Klassen identifizieren
- Generalisierungs- und Aggregationsbeziehungen ("structure") festlegen

- das entstandene Modell in Komponenten ("subjects") zerlegen
- Attribute der Klassen und Assoziationen modellieren
- Operationen ("services") und Botschaftsbeziehungen definieren

So entwickelt man nach und nach das Klassenmodell, das am Ende auch die Objektinteraktionen auf Klassenebene wiedergibt. Zustandsmodellierung ist kein fester Bestandteil der Methode, sondern eine Option innerhalb des Entwurfs. Insgesamt ist das Entwurfsmodell eine Erweiterung des Analysemodells, die in erster Linie in der Hinzunahme neuer Attribute und Operationen besteht. Im Idealfall kann der Entwurf dann direkt in die Implementierung mit einem objektorientierten Datenbanksystem umgesetzt werden.

In **BON** erfolgt die Analyse in den Schritten:

- Festlegen der Systemgrenzen
- Zusammenstellen von Klassenkandidaten
- Auswählen von Klassen und gruppieren in Komponenten
- Definieren der Klassen
- Beschreiben des Systemverhaltens
- Festlegen von Attributen und Operationen

Im ersten Schritt beschäftigen sich Waldén/Nerson mit Subsystemen, Benutzermetaphern und Anwendungsfällen, wonach die eigentliche objektorientierte Entwicklung mit der Auflistung von Klassenkandidaten beginnt. Wenn dann Klassen daraus ausgewählt wurden, werden Vererbungs- und Benutzungsbeziehungen zwischen ihnen und eine Komponentenaufteilung auch gleich betrachtet. So erhält man schon ein Klassenmodell und Interaktionsbeschreibungen. Nach einer informalen Beschreibung der Rollen und der Leistungen der Klassen wird die Zusammenarbeit der Objekte innerhalb der Anwendungsfälle beschrieben. Schließlich werden die Klassenschnittstellen formal mit Attributen und Operationen festgelegt. Im folgenden Entwurf wird das entstandene Modell verfeinert und komplettiert. Das resultierende Entwurfsmodell verfeinert man danach weiter, bis es ein ausführbares Programm ergibt.

Nach **Wirfs-Brock** sollte man in einer Explorationsphase Klassen aufspüren, denen man zunächst Attribute und eine Zweckbeschreibung zuordnet. Dabei gruppiere man Klassen mit übereinstimmenden Attributen, um Kandidaten für Oberklassen herauszustellen. Aus dem Zweck einer Klasse leite man ihre Verantwortlichkeiten ab. Die Zusammenarbeit verschiedener Klassen, um solche Verantwortlichkeiten erfüllen zu können, werden als nächstes untersucht. In der darauffolgenden Analysephase ist die Vererbungshierarchie der Klassen festzulegen. Für jede Klasse sind die Verpflichtungen festzuhalten, die aus der Benutzung durch andere Klassen entstehen. Orientiert an den Benutzungsbeziehungen wird eine Komponenten-

aufteilung vorgenommen. Operationen werden modelliert, um die aufgezeichneten Verpflichtungen einer Klasse erfüllen zu können. Werkzeuge bei diesen Tätigkeiten sind das Klassenmodell und Karteikarten, auf denen alle Angaben zu Klassen und Komponenten verzeichnet sind. Das Durchspielen von Szenarien ermöglicht die Überprüfung des Modells und bietet Anhaltspunkte für seine Verfeinerung. Die anschließende Implementierung des entstandenen Entwurfs ist dadurch vereinfacht, daß eine Modularisierung schon vorgegeben ist.

Nach der Festlegung der Anforderungen startet **OMT** mit der Analysephase, in der ein Klassenmodell, ein Dynamikmodell (Interaktions- und Zustandsdiagramme) und ein Funktionsmodell (Datenflußdiagramme mit Objekten, Interaktionsdiagramme) erstellt werden. Nach der Identifikation von Klassen werden erst Assoziationen, dann Attribute und Vererbungsbeziehungen modelliert. Dabei sollten Attribute, die selber wieder Klassen als Typen haben, besser als Assoziationen dargestellt werden. Darauf folgt optional eine Komponentenaufteilung. Anhand von Szenarien wird eine Benutzeroberfläche entwickelt. Für jedes Szenario wird in einem Interaktionsdiagramm die Zusammenarbeit der beteiligten Objekte entwikkelt. Dabei betrachtet Rumbaugh statt Nachrichten Ereignisse, auf die ein Objekt reagiert. Anhand solcher Ereignisse entwickelt man Zustandsmodelle für die einzelnen Klassen. Das Funktionsmodell wird top-down – d.h. ausgehend von den Funktionen des Gesamtsystems – erstellt. Nach vielfacher Kritik wurden in die ursprünglich allein vorgesehenen Datenflußdiagramme Objekte integriert und auch Interaktionsdiagramme vorgesehen. Operationen der Klassen entstehen aus der Betrachtung der drei Modelle. Dabei werden in der Analyse erst nur Operationen ins Klassenmodell aufgenommen, die dem Funktionsmodell zu entnehmen sind. Die vollständige Modellierung der Operationen geschieht erst im Entwurf, der sich in System- und Objekt-Entwurf aufteilt. In ersterem wird die Komponentenzerlegung des Systems (eventuell neu) festgelegt, werden Aspekte der Nebenläufigkeit, Verteilung und Datenhaltung betrachtet, also die Systemarchitektur geschaffen. Nach dem Objekt-Entwurf ist dann schließlich jede Klasse zur direkten Implementierung vorbereitet. Diese kann dann fast standardmäßig durchgeführt werden, wobei der wesentliche Punkt die Implementierung der Operationen ist.

In der Methode **Fusion** startet die Analyse mit einer Liste von Klassenkandidaten, aus denen Klassen ausgewählt und ihre Attribute, Generalisierungs- und Aggregationsbeziehungen modelliert werden. In einem zweiten Schritt wird dann die Systemschnittstelle nach außen beschrieben, indem für Szenarien die nötige Kommunikation des Systems mit außenstehenden Agenten und die inbegriffene Abfolge von Ereignissen in Interaktionsdiagrammen dargelegt wird. Die dabei neu gewonnenen Einsichten führen zu einer Verfeinerung des Klassenmodells. Eine Generalisierung der Szenarien und darauf aufbauende Beschreibung der Systemfunktionen

schließen die Analyse ab. Im Entwurf ist dann für jede Systemfunktion in einem Objekt-Interaktionsdiagramm darzulegen, wie die Objekte Nachrichten austauschen, um diese Funktion insgesamt erbringen zu können. Hieraus wird schrittweise das komplette Klassenmodell abgeleitet. Zunächst untersucht man die Sichtbarkeit der Objekte untereinander, entwickelt dann die Klassenschnittstellen und legt schließlich die Vererbungsbeziehungen fest. Der entstandene Entwurf kann dann meist direkt in standardisierter Weise implementiert werden.

Martin/Odell betrachten eine lineare Abfolge von Analyse, Entwurf und Implementierung. Ein eigenes Vorgehensmodell stellen sie aber nicht vor, sondern betrachten nur die verschiedenen Modellelemente. Sie unterscheiden Modelle für Systemstruktur und -verhalten. In der Analyse werden sie getrennt, im Entwurf verwoben behandelt. Das Klassenmodell beschreibt die Systemstruktur. Um es in der Analyse zu entwickeln, sind die Klassen zu identifizieren, Assoziations-, Generalisierungs- und Aggregationsbeziehungen zu modellieren. Eine Beschreibung des Systemverhaltens ergibt sich durch Zustands- und Interaktionsdiagramme. Im Entwurf werden dann Attribute und Operationen der Klassen und eine Vererbungshierarchie festgelegt. Eine Umsetzung in eine objektorientierte Programmiersprache kann dann in direkter Weise geschehen.

Gemeinsam ist allen Methoden, daß das Klassenmodell das zentrale Modellierungskonzept ist. In der Analyse wird ein erstes Modell entwickelt, im Entwurf das selbe verfeinert. Dabei heben die meisten Autoren das Zusammenspiel von Statik- und Dynamikmodell hervor. Viele ziehen Anwendungsfälle oder Szenarien in Betracht, um Anforderungen zu entwickeln bzw. Modelle zu überprüfen. Die Umsetzung des Entwurfs in der Implementierungsphase wird von den meisten Autoren nur am Rande betrachtet, die zentrale Aufgabe ist Analyse und Entwurf.

Einige Methoden betrachten schon in den frühen Phasen eine Komponentenzerlegung. Wolfgang Hesse geht noch weiter, er orientiert sein Vorgehensmodell EOS (Evolutionäre objektorientierte Software-Entwicklung, vergl. [H-W 94] und [Hes 96]) vollkommen an den Software-Bausteinen – Komponenten und Klassen. Dazu bietet EOS eine explizite Unterstützung von Entwicklungszyklen (auf System-, Komponenten- und Klassenebene). Für jeden Baustein werden die Phasen Analyse, Entwurf, Implementierung und operationeller Einsatz durchlaufen. Die entstehenden Zyklen können auch mehrfach und auch unvollständig in den Gesamtentwicklungszyklus eingebaut werden. Die meisten oben zitierten Methoden sprechen zwar ein iteratives oder inkrementelles Vorgehen an, sehen aber Iterationen nicht explizit vor. Die Baustein-Orientierung in EOS fördert insbesondere die Wiederverwendung. Komponenten erhalten ein besonders starkes Gewicht.

2.5 Anwendungsfall-getriebene Analyse

Software-Systeme werden für Kunden entwickelt, um bestimmte Aufgaben zu übernehmen. Diese Aufgaben teilen sich auf in typische *Anwendungsfälle* (*use cases*). Deshalb bieten die Anwendungsfälle einen guten Ansatzpunkt bei der Systemanalyse. Sie beschreiben die Funktionalität eines Systems von der Warte der außenstehenden Benutzer aus.

Anwendungsfälle wurden zuerst von Ivar Jacobson innerhalb der objektorientierten Analyse populär gemacht, später aber von vielen anderen Autoren aufgegriffen. Schon in [J-C 95] weisen Jacobson und Christerson auf Querzüge zu Rumbaugh ([Rum 94]), Booch, Wirfs-Brock und Rubin/Goldberg ([R-G 92]) hin und schildern Reaktionen dieser Autoren. In 2.4.3 wurde schon hervorgehoben, daß viele Methoden die Untersuchungen von Szenarien vorsehen. Dabei sind Szenarien Ausprägungen von Anwendungsfällen. Anwendungsfälle werden auch von der UML ([BJR 99a]) unterstützt und sind Leitfaden für den "Unified Process" ([Jac 99]).

Sie werden jeweils von einem "Aktor" angestoßen, dieser kann außerhalb des Systems stehen oder auch ein anderer Anwendungsfall innerhalb des Systems sein. Die Beziehungen zwischen Aktoren und Anwendungsfällen werden in einem Anwendungsfalldiagramm dargestellt (Abbildung 2.6).

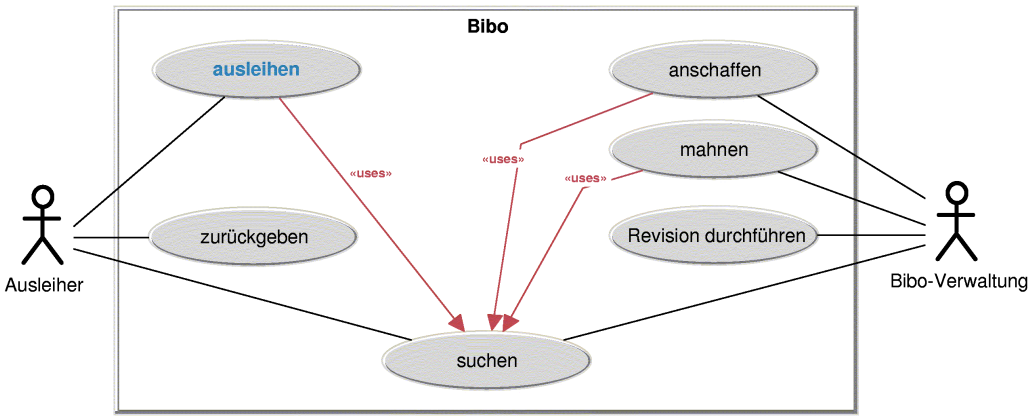


Abb. 2.6 Anwendungsfalldiagramm für eine Bibliothek

Das Rechteck im Diagramm symbolisiert die Systemgrenze, links und rechts davon stehen die Aktoren, die von außerhalb des Systems mit diesem in Interaktion treten. Im Beispiel sind das der Ausleiher und die Bibliotheksverwaltung. In den Ovalen sind die Anwendungsfälle angegeben. Durch einfache Linien ist gekennzeichnet, von welchem Aktor sie angestoßen werden. Anwendungsfälle können untereinander in "*uses*"- oder "*extends*"-Beziehung stehen. Das erste meint, daß ein

Anwendungsfall die Leistungen eines anderen nutzt, indem er eine Teilaufgabe an ihn delegiert. Im Beispiel wird innerhalb mehrerer anderer Anwendungsfälle nach Ausleihobjekten gesucht. "*extends*"-Beziehungen sind in diesem Beispiel nicht enthalten. Sie liegen vor, wenn ein Anwendungsfall als Spezialisierung eines anderen betrachtet wird.

Hat man die Anwendungsfälle in Prosa beschrieben, versucht man als nächstes plausibel zu machen, daß die Objekte des Systems (die der Intuition des Entwicklers und seiner Beschäftigung mit den Anwendungsfällen entsprungen oder wie in 2.6 beschrieben identifiziert worden sind) die Anwendungsfälle ausführen können. Dazu beschreibt man sie üblicherweise in Sequenzdiagrammen (vergl. 2.4.2). Ein solches Diagramm ist für das Bibliotheksbeispiel in Abbildung 2.4 auf Seite 39 angegeben.

Als erstes Objekt ist links der Aktor zu sehen, der den Anwendungsfall anstößt, rechts davon stehen oben die Objekte, die Operationen innerhalb des Anwendungsfalls ausführen.

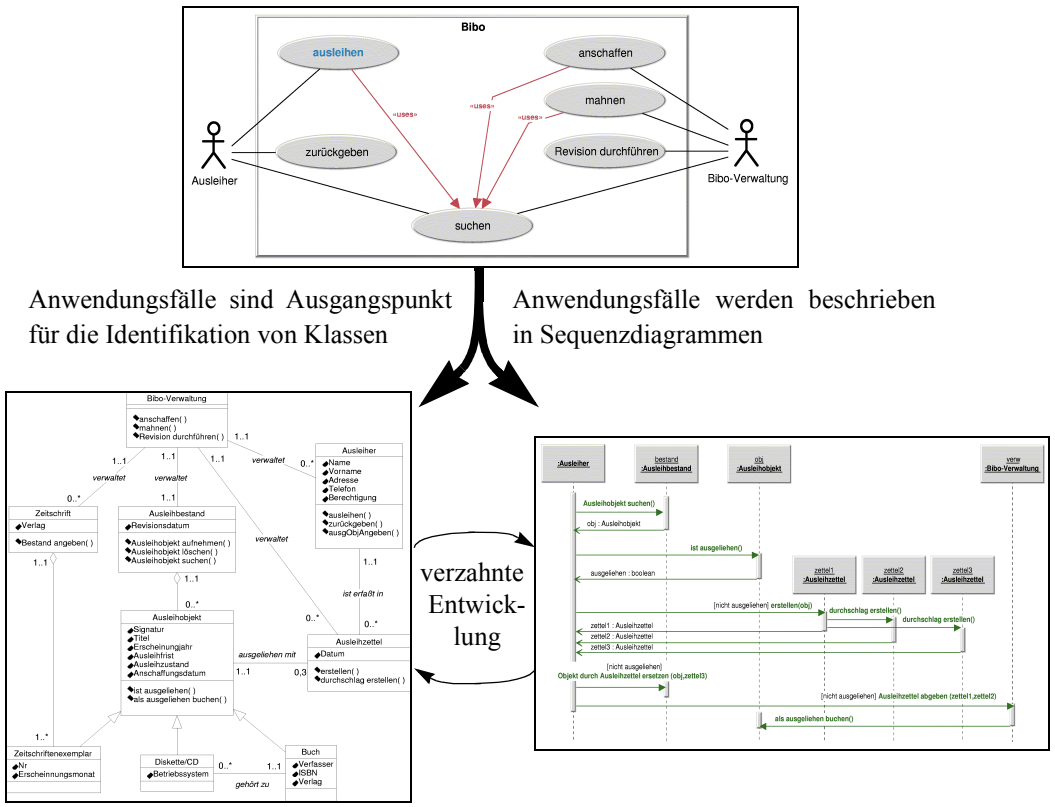


Abb. 2.7 Anwendungsfall-getriebene Analyse

Primäres Ziel der ersten Phase der Anwendungsfall-getriebenen Systemanalyse ist es, ein Klassenmodell und für jeden Anwendungsfall ein Sequenzdiagramm zu er-

stellen. Diese beiden zusammen erklären, wie die Objekte des Systems die Anwendungsfälle ausführen können. Das Klassenmodell und die Sequenzdiagramme werden verzahnt entwickelt, sie müssen in den vorkommenden Klassen, Operationen und Attributen übereinstimmen. Man braucht die Klassen, um die Sequenzdiagramme erstellen zu können. Andersherum führt die Beschäftigung mit den Details der Anwendungsfälle mittels der Sequenzdiagramme zur Entdeckung neuer Operationen und auch Klassen.

Abbildung 2.7 stellt das Vorgehen in der ersten Phase der Anwendungsfall-getriebenen Systemanalyse dar. Dabei wurde die inkrementelle Verfeinerung des Anwendungsfallmodells (wie Jacobson sie vorsieht) der Einfachheit halber nicht berücksichtigt. Die durch die dick eingezeichneten Pfeile angedeuteten Tätigkeiten zu unterstützen, ist das Anliegen von BASE. Der nächste Abschnitt beleuchtet zunächst einmal, welche Ideen für die Identifikation von Klassen in der Entwicklung schon durch die herkömmlichen Methoden vorliegen.

2.6 Identifikation von Objekten und Klassen

In 2.4.3 wurde aufgezeigt, daß das in der Analyse hergestellte Klassenmodell die Basis für den ganzen Software-Entwicklungszyklus bildet. Die in ihm enthaltenen Klassen leben – eventuell modifiziert – bis in die Implementierung fort. Also kommt ihrer Wahl eine besondere Bedeutung zu. Im folgenden werden Vorgehensweisen zu ihrer Ermittlung zusammengestellt und verglichen. Wenn nicht angegeben sind die zugehörigen Literaturquellen der Aufstellung im Abschnitt 2.4 auf Seite 33 zu entnehmen.

2.6.1 Vorgehen der einzelnen Methoden

Shlaer/Mellor bezeichnen die Identifikation von Objekten als ziemlich leichte ("pretty easy") Aufgabe. Man beschäftige sich mit den Dingen des Untersuchungsbereichs. Diese sind entweder

- materielle Dinge,
- Rollen von Personen oder Organisationen,
- auftretende Ereignisse,
- Interaktionen oder
- Aufzeichnungen über gewisse Dinge.

Alle diese Dinge geben Anlaß zu Klassenkandidaten. Zu jedem sind zum einen Kriterien anzugeben, die es erlauben, Dinge als klassenzugehörig zu bezeichnen,

zum anderen Ausschlußkriterien. Weiter sollten der Zusammenhang mit anderen Klassenkandidaten und vorhandene Hintergrundinformationen der Fachexperten festgehalten werden. Danach sollten passende Namen gesucht werden. Die Autoren weisen in diesem Zusammenhang auf die Probleme durch Homonyme und Synonyme und die verschiedenen Fachsprachen hin. Jeder Klassenkandidat ist dann daraufhin zu überprüfen, ob

- die in ihm zusammengefaßten Dinge gleiche Eigenschaften haben und die Formulierung einheitlicher Operationen erlauben,
- er durch Attribute beschrieben werden kann und nicht selber nur ein Attribut ist,
- die Charakterisierung der Klassenzugehörigkeit keine alternativen Fälle auflistet oder
- die Charakterisierung der Klassenzugehörigkeit nicht nur in einer Aufzählung der Objekte besteht.

Kandidaten, die diese Kriterien nicht erfüllen, sind von der Betrachtung auszuschließen und man muß nach besseren Abstraktionen suchen.

Coad/Yourdon sehen die Identifikation von Klassen als Abstraktion von dem ein- oder mehrmaligen Auftreten eines Dings im Untersuchungsbereich. Die Suche nach ihnen führt zu einer Beschäftigung mit dem Problembereich und so zu einem Verständnis und einer Abgrenzung desselben. Eine intensive Betrachtung des Untersuchungsbereichs schließt die Untersuchung vieler Quellen neben den Anforderungsdokumenten – andere schriftliche Quellen, Abbildungen und Gespräche mit Anwendern – ein. Beachtung sollten Substantive finden, ohne daß sie automatisch zu Klassen werden würden. Zu suchen sind (vergl. Shlaer-Mellor):

- Generalisierungs- und Aggregationsstrukturen,
- andere Systeme,
- Systemschnittstellen,
- aufzuzeichnende Ereignisse,
- Rollen von Personen,
- Orte,
- organisatorische Einheiten der betrachteten Organisation.

Diese ergeben Klassenkandidaten. Zu überprüfen sind die Kandidaten, ob

- Daten über sie zu speichern sind,
- sie Dienste für das System ausführen,
- sie durch mehr als ein Attribut gekennzeichnet werden,
- alle zugehörigen Objekte gemeinsame Attribute tragen,
- alle zugehörigen Objekte gemeinsame Dienste zur Verfügung stellen,

- sie der Beschreibung der essentiellen Systemanforderungen dienen und nicht erst aus reinen Implementierungsentscheidungen nötig sind.

Auszuschließen sind dagegen Kandidaten,

- über die keine Informationen gehalten werden müssen,
- die nur unbenötigte Dienste anbieten,
- von denen nur ein Objekt existiert, es sei denn dieses allein bündelt Operationen, wobei aber auch fachlich begründete Ausnahmen zugelassen werden,
- die aus anderen Modellelementen abgeleitet bzw. berechnet werden können.

Diese Ausschlußkriterien werden häufig in der Diskussion mit den Anwendern über das erste Klassenmodell festgestellt.

In der Methode **Fusion** stellen Klassen Begriffe des Untersuchungsbereichs dar. Man bekommt Klassenkandidaten aus der Betrachtung der Substantive in der Anforderungsbeschreibung. Wirklich interessant sind aber nur die, die für das Verständnis des Untersuchungsbereichs wichtige Dinge betreffen. Die Kandidatenliste von Fusion sieht so aus:

- materielle Dinge,
- Personen und Organisationen,
- Abstraktionen (von den Autoren nicht weiter präzisiert).

Nach einer Betrachtung der Beziehungen zwischen den Kandidaten (Kommunikations-, Assoziations-, Aggregations- und durch Aktionen gestiftete Beziehungen) wird die Kandidatenliste überprüft, ob

- die Kandidaten relevant oder für eine Problembeschreibung nebensächlich sind,
- sich Kandidaten überschneiden,
- die gewählten Namen zutreffend sind.

Dabei sollten bei möglichen Alternativen immer die allgemeineren Klassen gewählt werden.

Nach **Rumbaugh** sind Objekte materielle Dinge oder Begriffe. Man sollte sich bewußt sein, daß nicht alle nötigen Klassen in Anforderungsdokumenten zu finden sind, sondern die in ihnen zu identifizierenden nur einen Ausgangspunkt bilden. Bei der Erstellung eines ersten Klassenmodells werden weitere Klassen ergänzt. Zum ersten Ausgangspunkt gelangt man, indem man Substantive der Anforderungsbeschreibungen betrachtet. Dies sind Klassenkandidaten. Auszusondern sind dann:

- redundante Klassenkandidaten, die man durch Synonyme erhält,
- für das angestrebte System irrelevante Klassenkandidaten,
- Klassenkandidaten, die keine spezifische, abgrenzende Beschreibung erlauben,

- Kandidaten mit einfachen Datentypen, die man besser als Attribute einer anderen Klasse modelliert, wenn nicht eine zusätzliche veränderbare Eigenschaft von ihnen wichtig ist,
- Klassenkandidaten, die Operationen ausdrücken und nicht selber innerhalb von Systemfunktionen manipuliert werden,
- Namen, die Rollen und nicht die wirkliche Natur von Objekten bezeichnen,
- Implementierungskonstrukte.

Bei diesem ersten Ansatz zerbreche man sich noch nicht den Kopf über Vererbungsbeziehungen. Generalisierende Klassen können noch später eingeführt werden. Nach der Erstellung des kompletten Klassenmodells gibt es folgende Anzeichen von fehlenden Klassen:

- asymmetrische Modellierung von gleichartigen Assoziationen oder parallelen Spezialisierungsbeziehungen (→ Einführen analoger Klassen),
- unvereinbare Attribute oder Operationen in einer Klasse (→ Aufsplitten der Klasse),
- Schwierigkeiten, sauber zu generalisieren (→ Aufsplitten von Klassen),
- eine Operation, die keiner Klasse passend zugeordnet werden kann (→ Einfügen einer neuen Klasse),
- Assoziationen gleichen Namens und gleicher Bedeutung von einer Klasse zu mehreren anderen (→ Einfügen einer neuen Oberklasse),
- Rollen innerhalb von Assoziationen, die die Semantik der betreffenden Klassen schon allein beschreiben (→ Umwandeln der Assoziation in eine Klasse).

Hat dagegen eine Klasse keine Attribute, Operationen und Assoziationen, ist es fraglich, ob sie überhaupt gebraucht wird.

Wirfs-Brock schlägt vor, die Anforderungsbeschreibung nach Substantiven zu durchsuchen. Diese klassifiziere man in offensichtliche Klassen, Kandidaten, die offensichtlich keine sinnvollen Klassen abgeben, und Kandidaten, für die eine Entscheidung nicht ohne weiteres getroffen werden kann. Klassenkandidaten sollten eine Abstraktion von Dingen des Untersuchungsbereichs bilden und es sollte leichtfallen, ihre Bedeutung und ihren Zweck klar zu formulieren. Insbesondere ist folgendes zu beachten:

- Materielle Dinge ergeben Klassenkandidaten.
- Bei Synonymen ist dasjenige zu wählen, welches für die Systemteile außerhalb des betrachteten Klassenkandidatens das aussagekräftigste ist.
- Ergibt sich durch den Zusatz eines Adjektivs ein geändertes Verhalten, so erhält man eine neue Klasse, ansonsten nicht.
- Bei Sätzen im Passiv ist auch das implizite Subjekt zu betrachten.

- Kategorien von Klassen sollten als eigene Klassen modelliert werden, ohne auf Generalisierungsbeziehungen abzuheben.
- Systemschnittstellen sollten so detailliert wie möglich modelliert werden.
- Attribute werden selber nicht zu Klassen, sondern ihre Typen.

Ist dabei das Verständnis verschiedener Systemkomponenten verschieden ausgeprägt, werden auch die entsprechenden Modellteile verschieden detailliert beschrieben. Dies ist zulässig, weil eine Komponente eine Modularisierungseinheit ist, die aus der Sicht der anderen Komponenten als Black Box behandelt werden soll. Deshalb kann die detaillierte Modellierung ihrer Interna zurückgestellt werden, wenn nur ihre Schnittstelle klar beschrieben ist.

Jeder Klassenkandidat wird zusammen mit einer Beschreibung seiner Bedeutung auf einer Karteikarte (vergl. [B-C 89]) festgehalten. Aus der Anforderungsspezifikation liest man auch Attribute ab. Gemeinsame Attribute verschiedener Klassen, mit denen gemeinsames Verhalten verbunden ist, führen zur Modellierung von Oberklassen. Man erstelle so viel Oberklassen wie möglich. Das System erfüllt verschiedene Verantwortlichkeiten – Information zu speichern oder Aktionen auszuführen. Diese erhält man aus den Verben der Anforderungsbeschreibung und den Klassenkandidaten. Insbesondere sollte man Szenarien durchspielen, die das Systemverhalten herausstellen. Die identifizierten Verantwortlichkeiten müssen dann den Klassen zugeordnet werden, so daß

- sie gleichmäßig verteilt sind,
- möglichst allgemein gehalten sind,
- Verhalten und zugehörige Information zusammen in einer Klasse modelliert sind,
- Information zu einer bestimmten Sache nur an einer Stelle zusammen gehalten wird,
- komplexe Verantwortlichkeiten so auf logisch zusammenhängende Klassen verteilt werden, daß einzelne Teile der am besten passenden Klasse zugeordnet sind.

Gruppierungen von Verantwortlichkeiten, sie sich keiner Klasse zuordnen lassen, können in neuen Klassen gekapselt werden. Bei Zweifeln während der Zuordnung sollten Szenarien durchgespielt werden, um aus mehren Möglichkeiten die beste auszuwählen. Klassen ohne zugeordnete Verantwortlichkeiten sollten noch einmal unter die Lupe genommen werden.

Eine rigide – und gleichzeitig angemessene – Vorgehensweise für die Identifikation von Klassen anzugeben, halten **Waldén/Nerson** für unmöglich. In ihrem (von Eiffel übernommenen) Verständnis sind Klassen Implementierungen von abstrakten Datentypen. Was einen solchen Datentyp ausmacht, ist sein Verhalten als Reak-

tion auf Nachrichten (und nicht seine interne Struktur). Die Autoren beziehen sich in diesem Zusammenhang auf RDD von Wirfs-Brock. Besonderen Wert legen sie auf aussagekräftige Namen für Klassen und Operationen. In der Analyse sollten Klassennamen aus dem Fachvokabular der Anwender verwendet werden. Die mit den Klassen verbundenen Begriffe und Operationen müssen für die Anwender sinnvoll sein. Sich allein auf materielle Dinge zu konzentrieren, ist nicht ausreichend. Sie bieten nicht automatisch eine stabile Basis für das Klassenmodell und sind vielleicht nur Ausprägungen tieferliegender, abstrakterer Begriffe, die bessere Klassenkandidaten bieten. Klassen können auch Verhalten, das ohne zugehörige Daten bereit gestellt werden kann, kapseln. Denn im Verständnis der abstrakten Datentypen wird von objektinternen Daten völlig abstrahiert wird, indem nur das nach außen sichtbare Verhalten beschrieben wird. Speziell ist das der Fall, wenn das modellierte Verhalten von mehreren Klassen geteilt wird, es komplex und schwierig einer bestehenden Klasse zu zuordnen ist oder es so speziell ist, daß es zwar einer Klasse zugeordnet werden kann, aber nur von einigen benutzenden Klassen wirklich benötigt wird. Eine Klasse sollte nicht unvereinbare Abstraktionen miteinander vermischen. In der Entwicklung von Klassen sollten immer wieder die Rollen von Anbieter und Nachfrager von Leistungen vom Entwickler eingenommen werden, um das gewünschte Verhalten zu modellieren. Im Sinne der Wiederverwendung sollte dabei immer eine etwas allgemeinere Lösung angestrebt werden, als für den Spezialfall gerade nötig ist. Hat eine Klasse sehr wenig Attribute und Operationen, so kann das ein Zeichen von schwacher Abstraktion sein.

Jacobson startet mit der Beschreibung von Anwendungsfällen (siehe 2.5). Bei Schwierigkeiten, diese zu erstellen, sieht er eine Unterstützung durch ein erstes Klassenmodell vor, das aber nicht Basis der Software-Entwicklung werden soll. Es enthält nur Klassen, die in direkter Weise Dinge des Untersuchungsbereichs wiedergeben. Um ein solches anzulegen, verweist er auf die heuristischen Vorgehensweisen, wie sie von Coad/Yourdon oder Booch (in [Boo 91]) beschrieben sind. Die Klassen dieses ersten Modells bieten für ihn das Vokabular für die Diskussion zwischen Entwicklern und Anwendern und die Beschreibung der Anwendungsfälle. Die bevorzugte Reihenfolge ist aber, Klassen aus den Anwendungsfällen abzuleiten. Ist also das Anforderungsmodell mit Anwendungsfallbeschreibungen und Entwurf der Benutzerschnittstelle erstellt, geht es darum, das eigentliche Klassenmodell zu entwickeln. Jacobson unterscheidet drei Arten von Objekten:

- Schnittstellenobjekte
- Entitätsobjekte
- Kontrollobjekte

Schnittstellenobjekte übernehmen die Kommunikation des Systems mit seinen Benutzern und anderen externen Systemen. Man findet sie entweder

- direkt im Entwurf der Benutzerschnittstelle,
- ausgehend von den Aktoren, die mit dem System in Interaktion treten, oder
- in dem Teil der Anwendungsfallbeschreibungen, der die schnittstellengebundene Funktionalität betrifft.

Funktionalität, die direkt an die Schnittstelle gebunden ist, sollte Schnittstellenklassen zugeteilt werden, damit spätere Änderungen der Schnittstelle lokal bleiben.

Entitätsobjekte modellieren Information, die das System lange – auch über Ausführung eines Anwendungsfalls hinaus – hält. Sie entspringen zum einen dem ersten Klassenmodell, zum anderen aber auch den Anwendungsfällen. In der Regel modellieren sie Begriffe des Untersuchungsbereichs. Information, die immer nur in Zusammenhang mit anderer Information angesprochen wird, sollte meistens als Attribut und nicht als eigene Klasse modelliert werden. Funktionalität, die den bisherigen Klassen nicht angemessen zugeordnet werden kann, wird in neuen *Kontrollklassen* gebündelt. In einem ersten Ansatz teilt man jedem Anwendungsfall eine solche Kontrollklasse zu. Wenn jedoch die gesamte im Anwendungsfall angesprochene Funktionalität auf Schnittstellen- und Entitätsklassen aufgeteilt ist, kann diese Klasse sofort wieder entfallen. Andererseits kann die von der Kontrollklasse angebotene Funktionalität sehr komplex sein. Dann wird sie nach dem Bezug zu verschiedenen Aktoren bzw. logischen Gesichtspunkten auf mehrere Klassen aufgeteilt. Beschrieben werden alle Klassen durch ihre Rollen und Verantwortlichkeiten. Zu beachten ist, daß jeweils nur ein Anwendungsfall betrachtet wird. Spielt eine Klasse in mehreren Anwendungsfällen eine Rolle, ist ihre Beschreibung den neu hinzukommenden Verantwortlichkeiten anzupassen. Treten dabei Widersprüche auf, muß die Klasse aufgespalten werden.

In der **OBA** (Object Behavior Analysis) von Kenneth S. Rubin und Adele Goldberg ([R-G 92]) ist – wie schon der Name sagt und ähnlich wie bei Jacobson – das Systemverhalten der Ausgangspunkt der Analyse. Dieses Verhalten wird in Szenarien beschrieben, die auf der Basis von Interviews mit Anwendern und Fachexperten entstehen. In Skripten werden die einzelnen Schritte innerhalb der Bearbeitung einer Aufgabe in ihrer Ausführungsreihenfolge festgehalten. Zusätzlich ist für jeden Schritt definiert, welche Entität ihn initiiert und welche Entität ihn ausführt. Zu diesen Entitäten werden nötige Attribute abgeleitet (Darstellung in einem semantischen Netz, vergl. [Rei 91]). Alle Entitäten, die Leistungen innerhalb der Szenarien anbieten, sind Objektkandidaten. Darunter sind die, die selber keine Leistungen nachfragen, Kandidaten für Datenobjekte. Häufig tauchen Zusammenfassungen von Objekten auf, diese müssen nicht unbedingt zu Objekten im Analysemodell führen. Die besten Klassenkandidaten bieten Leistungen an und fragen selber Leistungen von anderen Objekten nach. Es ist bei diesen aber darauf zu achten, daß ih-

nen nicht zu viele verschiedene Rollen (durch die angebotenen Leistungen) zugeschrieben werden. Alle Informationen zu den Klassen werden auf Karteikarten festgehalten. Besonderes Gewicht legen die Autoren auf die Verfolgbarkeit der Entwurfsentscheidungen.

Booch diskutiert in seinen Büchern die "klassischen" Ansätze zur objektorientierten Modellierung wie die von Shlaer/Mellor oder Coad/Yourdon, verhaltensorientierte Ansätze wie die von Wirfs-Brock oder Rubin/Goldberg, Anwendungsfall-getriebene Analyse wie bei Jacobson, grammatikalische Analyse nach Abbott ([Abb 83]) wie in Fusion, bei Rumbaugh oder Wirfs-Brock, weiter Ansätze, die zunächst strukturierte Methoden anwenden und anhand von Datenflußdiagrammen eine funktionale Zerlegung vornehmen und den Ansatz der Bereichsanalyse. Schon in den Ausführungen zu den Vorstellungen von Waldén/Nerson tauchte das Schlagwort "Wiederverwendung" auf. Um diese als wichtigen Produktivitätsfaktor auf breiter Basis möglich zu machen, werden Ansätze zur Bereichsanalyse verfolgt. Idee ist es, nicht allein das gerade zu entwickelnde System zu betrachten, sondern den ganzen Anwendungsbereich, in dem es plaziert ist. Klassenkandidaten aus einer solchen weiter gesteckten Analyse haben größere Chancen, in weiteren Projekten auch wirklich wiederverwendet werden zu können.
Nach der Diskussion der verschiedenen Ansätze empfiehlt Booch, zunächst

- materielle Dinge,
- Rollen von Dingen und
- auftretende Ereignisse

als Klassenkandidaten ins Auge zu fassen. Dann sollten verhaltensorientierte Methoden angewandt werden, um Abstraktionen, die sich direkt aus den function points ([Alb 79]) des Systems ergeben, zu betrachten. Function points sind primäre Systemfunktionen, denen zur Aufwandsschätzung nach Erfahrungswerten Gewichte zugeordnet werden, deren Summe eine Kennzahl für die Systemkomplexität ergibt. Sie drücken das nach außen sichtbare Systemverhalten aus. Die ermittelten function points sind nach funktionalen Gesichtspunkten zu gliedern. Für jede Gruppe von zusammengehörigen functions points ist – wie bei Jacobson oder in der OBA beschrieben – ein Szenario zu betrachten. Mit den Techniken der Anwendungsfall-getriebenen Analyse erhält man so Klassenkandidaten. Als Werkzeug können Karteikarten dienen, um den Erkenntnisprozeß zu unterstützen.

Einen Ansatz mit gänzlich anderem Schwerpunkt ist die von **Ortner/Schienmann** propagierte normsprachliche Anwendungsentwicklung ([Schi 97], [Ort 97]). Sie konzentriert sich darauf, Informationen über den Untersuchungsbereich in einer normierten Sprache darzustellen und dabei insbesondere Fachbegriffe normsprachlich zu rekonstruieren. Dazu werden zunächst umgangssprachliche Aussagen über

den Untersuchungsbereich gesammelt. Die enthaltenen Fachbegriffe werden durch exemplarische Einführung, Pädikatorenregeln auf Basis schon geklärter Begriffe und explizite Definitionen zwischen Entwicklern und Anwendern vereinbart. Danach werden die Aussagen in eine Normsprache überführt, die nur die Nutzung eines (im wesentlichen im vorherigen Schritt) festgelegten Wortschatzes und die Einhaltung fester Satzbaupläne erlaubt, um die Mehrdeutigkeiten umgangssprachlicher Formulierungen zu reduzieren. Die so entstandene Beschreibung des Untersuchungsbereichs wird dann in ein objektorientiertes Modell überführt, dessen Klassen aus den rekonstruierten Fachbegriffen herrühren.

2.6.2 Diskussion der Ansätze

Im wesentlichen werden also folgende Unterstützungen zur Identifikation von Objekten und Klassen vorgeschlagen:

- Listen von Klassenkandidaten

  Verschiedene Arten von Dingen, die gute Klassenkandidaten abgeben, werden aufgelistet und durch Beispiele illustriert.

  Favoriten sind dabei materielle Dinge und Rollen von Personen oder Organisationen. Das macht auch Sinn, denn häufig ist eine wichtige Aufgabe des entstehenden Systems die Verwaltung und Manipulation von Daten über materielle Dinge. Verantwortlich sind dafür Stellen der Organisation, die dadurch gewisse Rollen übernehmen. Im zu entwickelnden System geschieht das durch Klassen. Booch bemerkt dazu: "Many of the tangible things and roles that we encounter early in the life cycle will carry through all the way to implementation, because they are so fundamental to our conceptual model of the problem." ([Boo 94], S. 237).

  Daneben übernimmt Booch noch Ereignisse aus den Kandidatenlisten anderer Autoren. Coad/Yourdon heben dabei darauf ab, daß solche Ereignisse durch das System aufzuzeichnen sind, Booch sieht dagegen ihre Bedeutung darin, daß es ein Objekt geben muß, das auf das Ereignis reagiert.

  Weitere Vorschläge für Klassenkandidaten sind dagegen im allgemeinen Zusammenhang nicht so leicht nachzuvollziehen, wie zum Beispiel die Erwähnung von Orten bei Coad/Yourdon. Andere sind so wenig spezifisch (wie zum Beispiel die "Abstraktionen" der Fusion-Methode), daß sie in der konkreten Entwicklungssituation nicht hilfreich erscheinen. Wenn andererseits die Bedeutung der angegebenen Klassenkandidaten nur durch Beispiele zu erklären versucht wird, ist ihr Nutzen in Modellierungssituationen, die anders als die genutzten Beispiele liegen, wegen fehlender Übertragbarkeit gering.

- Positiv-Listen für Objekteigenschaften

  Um die gewonnenen Klassenkandidaten nach ihrer Eignung als Klassen beurteilen zu können, werden Eigenschaften aufgelistet, die sie erfüllen sollten.

  Die meisten genannten Kriterien zielen dabei auf die logische Zusammengehörigkeit einerseits der Objekte, andererseits der Attribute und Operationen einer Klasse ab. Um dies beurteilen zu können, ist aber eine tiefergehende Analyse der Struktur der betreffenden Klassenkandidaten nötig. Das heißt, daß man den Kandidaten als Klasse modelliert, um zu entscheiden, ob er einen guten Klassenkandidaten abgibt. Von einer solchen einmal durchgeführten Modellierung wird man sich aber nur schwer wieder lösen können.

  Andere Kriterien überprüfen die Relevanz für das zu entwickelnde System im Hinblick auf Speicherung von Daten und Ausführung von Operationen. Solche Fragen bieten einen guten Ausgangspunkt für die weitere Analyse.

  Rein syntaktische Kriterien wie "mindestens zwei Attribute" erscheinen wenig hilfreich, weil sie keinerlei Bezug zum Untersuchungsbereich aufweisen.

- Negativ-Listen für Objekteigenschaften

  Solche Listen dienen dazu, Klassenkandidaten von der weiteren Betrachtung auszuschließen.

  Neben den Umkehrungen der Positiv-Kriterien wird auf die Abgrenzung der verschiedenen Klassenkandidaten untereinander abgehoben. Rumbaugh führt sogar Punkte auf, die erst nach einer weitgehenden Erstellung des Klassenmodells zu beurteilen sind. Für den ersten Schritt der Kandidatenauswahl erscheinen solche Ratschläge nur sehr eingeschränkt brauchbar.

- Grammatikalische Analyse

  Der Tenor der entsprechenden Vorschläge lautet: "Substantive in den Anforderungsbeschreibungen ergeben Klassenkandidaten, Verben ihre Operationen."

  Vorteil einer solchen Methode ist ihre leichte Anwendbarkeit. Sie führt in der Regel aber zu vielen wertlosen Kandidaten. Schwierigkeiten ergeben sich dadurch, daß grammatikalische und semantische Kategorien nicht übereinstimmen. So ist es etwa in vielen Sprachen möglich und üblich, Verben als Substantive zu benutzen. Eine intensive Beschäftigung mit diesem Ansatz führt dazu, daß die Aufmerksamkeit der Entwickler nur auf die Anforderungsdokumente ausgerichtet ist. So werden die Anwender nicht in erforderlichem Maße in den Entwicklungsprozeß einbezogen.

- Anwendungsfallbasierte Analyse

  In Szenarien wird die Systemfunktionalität beschrieben. Durch die Modellierung der einzelnen Schritte innerhalb eines Szenarios und die Aufteilung der entsprechenden Verantwortlichkeiten stößt man auf die Objekte und Klassen.

Da die Erfüllung der funktionalen Anforderungen an das System das zentrale Entwicklungsziel darstellt, bietet dies einen guten Ausgangspunkt. Insbesondere ist später ein Test der implementierten Funktionalität gegen die spezifizierten Szenarien möglich. Dabei besteht das elegantere Vorgehen darin, statt konkreter Szenarien Anwendungsfälle als ihre Typen zu betrachten, wie Jacobson es vorsieht.

Werden die Anwendungsfälle der Reihe nach einzeln betrachtet, müssen eventuell schon herausgearbeitete Klassenbeschreibungen ergänzt werden, wodurch es in ihnen zu Widersprüchen kommen kann. BASE bietet die Möglichkeit, die Gesamtheit der Anwendungsfälle zu untersuchen und Abhängigkeiten schon früh festzustellen.

- Bereichsanalyse

  Dieser Ansatz scheint – besonders im Hinblick auf Wiederverwendung von Klassen – vielversprechend. Problematisch ist aber seine praktische Umsetzung, weshalb Guillermo Arango ([Ara 89]) zwischen "pure domain analysis" und "practical domain analysis" unterscheidet. Er weist daraufhin, daß für einen wirksamen Einsatz von Wiederverwendung oft implizit vorhandene Information explizit bereitgestellt werden muß. Die Leitfrage seines Ansatzes zur praktisch durchführbaren Bereichsanalyse ist, wie ein Modell des Untersuchungsbereichs inkrementell so erstellt werden kann, bis gewisse Vorgaben über die Modellmächtigkeit in bezug auf vorgesehene Wiederverwendung erreicht werden. In seinem Ansatz sind also Fälle der Wiederverwendung vorherzusehen und es ist zu beschreiben, in welchem Maße ihre Modellierung schon aus dem bereits vorliegenden Modell abgeleitet werden können soll. Ein solcher Ansatz muß inkrementell angelegt sein, denn nur so kann man sich in ein unbekanntes Gebiet einarbeiten. Dennoch bleibt die Frage nach dem passenden Ansatzpunkt, die gerade ohne tiefgehende Kenntnis nicht zuverlässig beantwortet werden kann. Außerdem besteht die Gefahr, das wesentliche Ziel des Bereitstellens der benötigten Funktionalität aus den Augen zu verlieren, weil die Struktur des Anwendungsbereichs erst noch erarbeitet wird. Es ist zu befürchten, daß viel Aufwand zu wenig zielgerichtet betrieben wird. Überhaupt ist der enorme Zeitbedarf für die Bereichsanalyse – nach Arango sind Fälle der Wiederverwendung vorauszuplanen und überprüfbare Kriterien für das Maß der möglichen Wiederverwendung zu definieren – für viele Projekte ein k.o.-Kriterium, weil sie unter großem Zeitdruck durchgeführt werden. Für den Auftraggeber ist in den meisten Fällen das Argument, daß man wiederverwertbare Bausteine für spätere Projekte erhält, nicht von so großem Gewicht wie ein schneller Projektabschluß. Laut Booch ist eine dermaßen tiefgehende Analyse des Anwendungsbereichs auch meistens nicht nötig: "For highly complex systems, domain analysis may in-

volve a formal process using the resources of multiple domain experts and developers over a period of many months. In practice, such a formal analysis is rarely necessary. Often, all it takes to clear up a design problem is a brief meeting between a domain expert and a developer. It is truly amazing to see what a little bit of domain knowledge can do to assist a developer in making intelligent design decisions." ([Boo 94], S. 158).

- Strukturierte Analyse

  Die Idee, mit strukturierten Techniken die Systemanalyse anzugehen und später erst auf objektorientierte Techniken umzuschalten, wird zwar von Booch angesprochen, aber dann sofort abgelehnt. Durch ein solches Vorgehen verliert man gerade die Durchgängigkeit der Ausdrucksmittel und des Entwicklungsparadigmas mit den in 2.2.3 diskutierten Konsequenzen. Außerdem sind speziell in strukturierten Techniken erfahrene Entwickler versucht, auch im Entwurf den strukturierten Vorgehensweisen zu folgen und mit einer funktionalen Zerlegung zu enden. Diese Probleme hat auch Rumbaugh zur Kenntnis nehmen müssen. Die in OMT ursprünglich vorgesehenen Datenflußdiagramme wurden durch andere ersetzt, die eine deutlichere Verbindung zum Klassenmodell aufweisen. In [Rum 95] schreibt er: "This approach to the functional model is a major departure from our book “Object-Oriented Modeling and Design”, which proposed a more conventional use of data flow diagrams that was both misunderstood and poorly accepted." ([Rum 95], S. 10).

- Normsprachlicher Ansatz

  Problematisch erscheint an diesem Ansatz die Akzeptanz der Normsprache durch die Anwender. Sie ist zwar "angelehnt an die Gebrauchssprache" ([Schi 97], S. 15), laut Ortner sogar "in ihrem Charakter natürlich wirkend" ([Ort 95], S. 149), aber dennoch sagt Schienmann selber, daß "für die Kommunikation mit dem Anwender [...] die Gebrauchssprache des jeweiligen Anwendungsbereichs in der Funktion als Erläuterungssprache [...] unverzichtbar" ist ([Schi 97], S. 75). Außenstehende werden auch Mühe haben, in der Normsprache "natürliche" Sprache wiederzuerkennen, sie lehnt sich eher an die Prädikatenlogik an. Für die verwandten Symbole werden jedoch Paraphrasierungen vorgesehen. Der Ausbau der Möglichkeiten der Übersetzung von Normsprache in Gebrauchssprache wird von Schienmann denn auch als Forschungsziel bezeichnet. Zusätzlich betrachtet er noch Diagrammsprachen (wie in 2.4.2 dargestellt) und Spezifikationssprachen innerhalb der Systementwicklung. Mit der Normsprache ist eine zusätzliche Ausdrucksform in die Entwicklung eingeflossen. Durch sie wird es ermöglicht, Modelle aus natürlichsprachlichen Anforderungsbeschreibungen zu erzeugen, ohne daß die Anwender in diesen Prozeß einbezogen werden. Damit können sie ihn auch nicht nachvollziehen. Ortner

schreibt dazu: "Die Sprache sollte so aufgebaut sein, daß es dem Anwender nicht schwerfällt, die Einschränkungen in Grammatik und Lexikon zu akzeptieren, während ihm das Ziel, eine dadurch bessere Formalisierbarkeit der Ergebnisse in den nachfolgenden Entwicklungsphasen zu erreichen, verborgen bleiben kann." ([Ort 95], S. 149). Und weiter: "Eine Konstruktionssprache (Normsprache), die wir dabei als “Zwischensprache” (Konsolidierungssprache) im Übergang von fachsprachlichen Aussagen (ermittelt mit Benutzern) zu diagrammsprachlichen Darstellungen der Ergebnisse (ohne Beteiligung der Benutzer von Entwicklern erstellt) einsetzen werden, [...]" ([Ort 97], S. 11). Das ist insbesondere im Hinblick auf inkrementelle und evolutionäre Entwicklungsstrategien ein Problem, weil die Entwicklung der erstellten Modelle im Zuge der Betrachtung zusätzlicher Anforderungen von den Anwendern nicht verfolgt werden kann. Welche der rekonstruierten Begriffe als Klassen modelliert werden, erläutert Schienmann leider nicht weiter.

- Begriffliche Ansätze

  Klassen sollen Begriffe des Untersuchungsbereichs implementieren.

  Die Autoren Booch, Rumbaugh, Martin/Odell und die Methode Fusion stellen in ihren Büchern diese Rolle von Begriffen bei der Klassenmodellierung zwar heraus, doch geben sie keine konkreten Hinweise, wie diese Vorstellung das Finden von Klassenkandidaten leiten kann. Ortner und Schienmann beschäftigen sich zentral mit den Fachbegriffen des Untersuchungsbereichs, doch ist ihr Vorgehen wenig zielgerichtet (siehe auch oben).

Eine rigide und immer erfolgversprechende Vorgehensweise für die Identifikation von Objekten und Klassen vorzuschreiben, erscheint unmöglich. Grady Booch schreibt in diesem Zusammenhang: "The amateur software engineer is always in search of magic, some sensational method or tool whose application promises to render software development trivial. It is the mark of the professional software engineer to know that no such panacea exists." ([Boo 92], S. 229)

Ziel sollte es aber sein, den Entwicklern Leitfäden für die Auseinandersetzung mit dem Untersuchungsbereich in der Kommunikation mit den Fachexperten an die Hand zu geben, mit Hilfe derer er sich den Untersuchungsbereich erschließen kann. Klassenkandidaten, die aus einer Datensicht ins Auge springen, bieten eine gute Ausgangsposition. Zur Orientierung an den funktionalen Systemanforderungen muß aber eine Betrachtung der Systemfunktionalität geleistet werden. Anwendungsfälle sind hier ein guter Anfang. Um häufiges Nachbessern von modellierten Klassen zu vermeiden, sollten sie in ihrer Gesamtheit behandelt werden können. Nach diesen Ideen ist BASE aufgebaut.

3 Formale Begriffsanalyse

Formale Begriffsanalyse ([G-W 96]) formalisiert die Behandlung von Begriffen und das Schließen in begrifflichen Systemen auf mathematischer Grundlage. Sie wird in BASE als Werkzeug zur Identifikation von Klassenkandidaten eingesetzt. Abschnitt 3.1 stellt das zugrunde liegende Verständnis von Begriffen (als Abstraktionen der realen Welt) dar und gibt einen kurzen Hinweis auf die geschichtlichen Quellen dieses Begriffsverständnisses. Wesentliche mathematische Strukturen innerhalb der Formalen Begriffsanalyse sind vollständige Verbände. Deshalb werden in 3.2 die nötigen ordnungstheoretischen Grundlagen von der Definition von Ordnungen bis zu Verbänden dargestellt. Darauf aufbauend werden in 3.3 die wichtigsten Konzepte der Formalen Begriffsanalyse – formale Kontexte, Begriffsverbände und deren Liniendiagramme – eingeführt. Um einen schnellen Einblick zu bekommen, reicht es, allein diesen Abschnitt anhand seiner Definitionen nachzuvollziehen. Dadurch sind alle Grundlagen gegeben, um die Grundidee des Ansatzes von BASE verfolgen zu können. 3.4 und 3.5 behandeln spezielle weiterführende Aspekte der Formalen Begriffsanalyse, die für Verfeinerungen innerhalb von BASE von Bedeutung sind.

In die Darstellung sind auch Beweise von Aussagen einbezogen, um die mathematischen Sachverhalte in sich geschlossen darzustellen. Außerdem sind in der benutzten Literatur (insbesondere in [G-W 96]) viele Beweise nur in sehr knapper Form dargestellt, so daß eine ausführliche Ausarbeitung ratsam schien, um die enthaltenen Gedankengänge und Argumentationslinien klarer herauszubringen. Definitionen, Aussagen und Beweise, die aus Lehrbüchern übernommen wurden, sind durch entsprechende Referenzen gekennzeichnet. Stehen diese am Anfang von Beweisen, beziehen sie sich auch auf die zugehörigen Aussagen. Referenzen der Form "Vergl. ..." bezeichnen Stellen, an denen nicht unerhebliche Änderungen gegenüber den zitierten Büchern vorgenommen wurden. Dies ist insbesondere bei der Behandlung von Implikationen der Fall, weil die Formulierung von Implikationen gegenüber der zitierten Literatur verändert wurde. Teile, die keinen Literaturver-

weis enthalten, sind höchsten implizit in der benutzten Literatur enthalten und sind deshalb neu entwickelt worden. Im wesentlichen ist die gesamte Darstellung an [G-W 96] angelehnt.

3.1 Begriffsverständnis

In der DIN 2330 werden "Begriffe" wie folgt eingeführt:

"Jeder Mensch lebt in einer Umwelt von Gegenständen, die einmalig, d.h. zeitgebunden sind, und deshalb “individuelle Gegenstände” genannt werden. [...] Um die unüberschaubare Menge der individuellen Gegenstände, die den Menschen begegnen, handhaben zu können, ist es notwendig, die Gesamtheit der Gegenstände für Wahrnehmung und Verhalten zu strukturieren und zu ordnen. Die gedankliche Zusammenfassung von individuellen Gegenständen zu gedachten “allgemeinen Gegenständen” führt zu Denkeinheiten, die als Begriffe bezeichnet werden. Die Zusammenfassung vollzieht sich auf der Basis von Merkmalen. Unter Merkmalen versteht man die Eigenschaften eines einzelnen individuellen Gegenstandes oder diejenigen Eigenschaften von individuellen Gegenständen, die zur Analyse der entsprechenden Begriffe herangezogen werden. [...] In Begriffen werden somit die gemeinsamen Merkmale einer Menge von individuellen Gegenständen, die die Menschen an diesen feststellen und zur gedanklichen Ordnung benutzen, zusammengefaßt. Die Gesamtheit der Merkmale, die eine gedankliche Zusammenfassung von individuellen Gegenständen und die gegenseitige Abgrenzung der Begriffe ermöglichen, ist der Begriffsinhalt (Intension). [...] Unter dem Umfang eines Begriffes (Extension) versteht man die Gesamtheit aller individuellen Gegenstände, die sämtliche Merkmale dieses Begriffs haben. [...] Hierarchische Beziehungen stellen ein Über- und Unterordnungsverhältnis zwischen Begriffen her. Hinsichtlich des Begriffsumfanges bedeutet dies, daß alle Gegenstände, die unter einen engeren Begriff (untergeordneten Begriff) fallen, auch unter den weiteren Begriff (übergeordneten Begriff) fallen, dieser aber noch zusätzliche Gegenstände umfaßt." ([DIN 79], 3.1-3.4, 4.3)

Begriffe haben danach dualen Charakter: Sie besitzen eine extensionale und eine intensionale Beschreibung. Eine grundlegende Beziehung zwischen ihnen ist die Ober-/Unterbegriffbeziehung, die oben extensional erklärt ist. Genau dieses Begriffsverständnis wird von der Formalen Begriffsanalyse mathematisch formalisiert.

Die systematische Beschäftigung mit der Abstraktion von konkreten Dingen der Welt – und so der Bildung von Begriffen – findet sich schon bei Plato und Aristote-

les[1], ohne daß sie ein Wort für Begriffe gehabt hätten[2]. Die Abstraktion geschieht dabei durch die Zusammenfassung der gemeinsamen Merkmale von diversen konkreten Dingen.[3] Eine begriffliche Aufarbeitung ist Grundlage jedes Wissens.[4]

Schon im Mittelalter finden sich dann in der Lehre der *proprietates terminorum* ("Eigenschaften der Begriffe", vergl. [Duf 89]) Ideen, die der Unterscheidung zwischen Begriffsumfang und -inhalt nahekommen.[5] Explizit formuliert ist diese Unterscheidung zwischen Begriffsumfang ("étendue") und -inhalt ("compréhension") zuerst in der Logik von Port-Royal (vergl. [Ris 70], S. 70).
Denn dort heißt es:
"Obgleich alle existierenden Dinge einzelne sind, haben alle Menschen mittels der Abstraktionen, die eben erklärt wurden, verschiedene Arten von Ideen. Die einen nämlich stellen uns nur ein einziges Ding dar, wie die Idee, die jeder von sich selbst hat, und die anderen Ideen stellen, und zwar in gleicher Weise, mehrere Dinge dar, wie jemand ein Dreieck sich vorstellt, ohne dabei auf etwas anderes zu achten als darauf, daß es eine Figur mit drei Linien und drei Winkeln ist; dann kann ihm die Idee, die er sich davon gebildet hat, dazu dienen, alle anderen Dreiecke sich vorzustellen. Die Ideen, die nur ein einziges Ding repräsentieren, heißen einzelne oder individuelle, und das, was sie darstellen, Individuen; die Ideen, die mehrere Dinge repräsentieren, heißen universelle, gemeinsamen oder allgemeine. [...] Wenn wir hier aber von allgemeinen Wörtern sprechen, so verstehen wir die eindeutigen darunter, die mit universellen und allgemeinen Ideen verbunden sind. Nun gibt es aber in diesen universellen Ideen zwei Dinge, deren Unterscheidung sehr wichtig ist, nämlich den Inhalt und den Umfang. Ich nenne Inhalt der Idee die Attribute, die die Idee in sich schließt, und die man von ihr nicht entfernen kann, ohne die Idee zu zerstören, so wie der Inhalt der Idee des Dreiecks Ausdehnung, Gestalt, drei Linien, drei Winkel und die Gleichheit dieser drei Winkel mit zwei rechten usw. in sich schließt. Ich nenne Umfang der Idee die Subjekte, denen diese Idee zukommt, die man auch die einem allgemeinen Wort Untergeordneten nennt,

1. [Hal 71], S. 780: "Der philosophische Gebrauch der Äquivalente des Terminus Begriff setzt mit dem sokratisch-platonischen Philosophieren ein [...].", [Wil 95], S. 4: "Das Denken in Begriffen spielte jedoch für die griechische Philosophie von Anfang an eine zentrale Rolle."
2. [Wei 88], S. 2: "Plato has no explicit theory of concepts; nor he has a word for concept in the sense that, say, Sextus Empiricus or Abelard has both."
3. [Hal 71], S. 780: "Die kaum überschätzbare Leistung des sokratischen Denkens bestand auch darin, die Frage nach dem, was später (allgemeiner) Begriff genannt wurde, d.h. nach den gemeinsamen Merkmalen (Eigenschaften) von Dingen, Ereignissen und Handlungen, explizit und methodisch zum ersten Mal gestellt zu haben."
4. [Hal 71], S. 781: "Ein weiteres Thema bildet der Gedanke, daß nur das was begrifflich als λόγος erfaßt wird, Gegenstand des Wissens sein kann."
5. [Hal 71], S. 782: "Die in der Theorie der propritates terminorum entwickelte Unterscheidung zwischen der Signifikation und der Supposition eines Terminus kann nach den meisten Texten als Unterscheidung zwischen Intension und Extension gedeutet werden."

wobei dieses im Hinblick auf sie das übergeordnete heißt, wie sich die Idee des Dreiecks überhaupt auf alle verschiedenen Arten von Dreiecken erstreckt." ([A-N 94], S. 46, 47)

3.2 Ordnungstheoretische Grundlagen

Zunächst einmal soll geklärt werden, was im folgenden unter einer Ordnung verstanden wird. Eine Ordnung ist eine binäre Relation auf einer Menge. Da später auch andere binäre Relationen (innerhalb formaler Kontexte) eine große Rolle spielen, wird zunächst allgemein eingeführt, was unter einer binären Relation zu verstehen ist.

Definition 1 (*Binäre Relation*)

Eine *binäre Relation* R zwischen zwei Mengen X und Y ist eine Teilmenge des kartesischen Produkts $X \times Y$.
Statt $(x, y) \in R$ schreibt man auch $x\,R\,y$.
Ist $X = Y$, so heißt R binäre Relation auf X.
Durch die Vorschrift $y R^{-1} x :\Leftrightarrow x R y$ für $y \in Y$ und $x \in X$ erhält man die zu R *inverse Relation* R^{-1} zwischen Y und X (auch *konverse Relation* genannt).
([G-W 96], 0.1, Definition 1, S. 1)

3.2.1 Ordnungen

Das zentrale Konzept dieses Kapitels ist das der Ordnung.

Definition 2 (*Ordnung*)

Eine binäre Relation R auf einer Menge X heißt *reflexiv*, wenn

$$x R x \qquad \forall x \in X$$

Eine binäre Relation R auf einer Menge X heißt *antisymmetrisch*, wenn

$$x R y \quad \text{und} \quad y R x \quad \Rightarrow \quad x = y \qquad \forall x, y \in X$$

Eine binäre Relation R auf einer Menge X heißt *transitiv*, wenn

$$x R y \quad \text{und} \quad y R z \quad \Rightarrow \quad x R z \qquad \forall x, y, z \in X$$

Eine reflexive, antisymmetrische und transitive Relation auf einer Menge X heißt eine *Ordnung* auf X. (X, R) heißt dann *geordnete Menge*.
([G-W 96], 0.1, Definition 2, S. 1)

Beispiel 1 (*Ordnungen*)

a) $\leq$ auf der Menge der reellen Zahlen $\mathbb{R}$ ist eine Ordnung, mit der man täglich umgeht – zumindest mit ihren Einschränkungen auf Teilmengen, z.B. $\leq$ auf $\{0, \ldots, 9\}$.

b) Genauso geläufig ist die alphabetische Ordnung $\leq_\alpha$ auf $\{a, \ldots, z\}$.
Weil eine Relation als Teilmenge eines kartesischen Produkts erklärt ist, kann man durch Mengenoperationen neue Relationen erklären.
Zum Beispiel ist $\leq \cup \leq_\alpha$ eine Ordnung auf $\{0, \ldots, 9\} \cup \{a, \ldots, z\}$, denn die Reflexivität überträgt sich direkt von den Ausgangsordnungen, Antisymmetrie und Transitivität ergeben sich ebenfalls aus ihnen, weil eine Ziffer und ein Buchstabe nie in Relation $\leq \cup \leq_\alpha$ stehen.
Diese Ordnung ist nicht so "schön" wie die ersten, weil man mit ihrer Hilfe nicht alle betrachteten Elemente vergleichen kann.

c) Solche Effekte hat man aber auch bei "natürlich gegebenen" Ordnungen. Eine große Rolle spielt im folgenden die Inklusionsordnung $\subseteq$ auf der Potenzmenge $\mathfrak{P}(X)$ einer Menge X, in der nicht alle Teilmengen verglichen werden können, wenn X mehr als ein Element enthält.

Bemerkung 1 (*Schreibweise* $\leq$)

Meistens benutzt man für eine Ordnung auf einer Menge X das Zeichen $\leq$.
In diesem Fall bezeichnet man $\leq^{-1}$ mit $\geq$ und schreibt weiter für $x, y \in X$

$$x < y \quad \text{statt} \quad x \leq y \quad \text{und} \quad x \neq y$$

Die Beobachtung, daß nicht alle Elemente einer geordneten Menge paarweise miteinander vergleichbar sein müssen, führt zu der nächsten Definition.

Definition 3 (*Vergleichbar, Kette, lineare Ordnung*)

Ist $(X, \leq)$ eine geordnete Menge, so heißen $x, y \in X$ *vergleichbar*, wenn $x \leq y$ oder $y \leq x$ gilt, sonst *unvergleichbar*.
Sind in $Y \subseteq X$ je zwei Elemente vergleichbar, so heißt Y eine *Kette* in X und $(Y, \leq)$ *linear geordnet*.
([Bir 67], I.1, S. 2)

Insbesondere ist hier die Definition von Ordnungen so ausführlich ausgeführt, weil sie in der mathematischen Literatur nicht einheitlich angegeben wird. Die hier verwendete Definition schließt sich der Formulierung in [G-W 96] an. Häufig wird die hier definierte Ordnung als "partielle Ordnung" (vergl. [Bir 67]) oder Halbordnung (vergl. [Ern 82]) bezeichnet. Im Englischen bekommen dann (partiell) geordnete Mengen den schönen Namen "poset" ("partially ordered set"). Lineare Ordnungen werden auch als total bezeichnet (vergl. [Bir 67]).

Bei der Vereinbarung der Schreibweise ≤ wurde auch schon die inverse Relation ≥ angesprochen. Diese ist mit ≤ ebenfalls eine Ordnung, wie der folgende Satz zeigt.

Satz 1 (≥)

Ist $(X, \leq)$ eine geordnete Menge, so ist auch ≥ eine Ordnung auf X.

Beweis: ([Bir 67], I, Theorem 2, S. 3)

Reflexivität, Antisymmetrie und Transitivität von ≥ ergeben sich durch Umstellen der Bedingungen direkt aus den entsprechenden Eigenschaften von ≤.

So ergibt sich eine Dualität in ordnungstheoretischen Überlegungen, die später in verbandstheoretischen Überlegungen noch weiter ausgeführt wird.

Definition 4 (*Duale Ordnung, duale ordnungstheoretische Aussage*)

Ist $(X, \leq)$ eine geordnete Menge, so heißt ≥ die zu ≤ *duale Ordnung*.
$(X,\leq)^d := (X,\geq)$ heißt *dual geordnete Menge*.
Zu einer *ordnungstheoretischen Aussage* A über $(X, \leq)$, die außer Variablen, Konstanten und logischen Operatoren nur das Zeichen ≤ enthält, erhält man die *duale Aussage* A^d, indem man in A alle Vorkommen von ≤ durch ≥ ersetzt.
([Bir 67], I.2, S. 3; [Grä 98], I.1.1, S. 3)

Bemerkung 2 (*Dualitätsprinzip für geordnete Mengen*)

Eine ordnungstheoretische Aussage A über eine geordnete Menge $(X, \leq)$ gilt genau dann, wenn A^d in $(X, \leq)^d$ gilt.
([Grä 98], I.1.1, S. 3)

Mit dieser Beobachtung kann die Betrachtung von dualen Fällen in ordnungstheoretischen Beweisen auf einen Fall gekürzt werden, weil sich der duale mit demselben Beweis über die duale Ordnung ergibt. Genauso ergibt sich eine Vereinfachung in der Formulierung von ordnungstheoretischen Definitionen.

Häufig interessant ist, alle Elemente kennzeichnen zu können, die größer oder kleiner als ein vorgegebenes Element sind bzw. zwischen zwei vorgegebenen Elementen liegen.

Definition 5 (*Hauptideal, Hauptfilter, Intervall*)

Seien $(X, \leq)$ eine geordnete Menge und $x, y \in X$.
Dann heißt $(x] := \{z \in X \mid z \leq x\}$ das *Hauptideal* von x.
Dual heißt $[y) := \{z \in X \mid z \geq y\}$ der *Hauptfilter* von y.
Ist $x \leq y$, so ist $[x, y] := \{z \in X \mid x \leq z \leq y\}$ ein *Intervall* in X.
([G-W 96], 0.1, Definition 5, S. 3)

3.2.2 Verbände

Häufig sucht man zu einer Menge von mehreren Elementen ein Element, das größer oder kleiner als alle Elemente der vorgegebenen Menge ist. Solche Elemente werden in der folgenden Definition gekennzeichnet.

Definition 6 (*Untere/obere Schranke, Infimum/Supremum*)

Ist $(X, \leq)$ eine geordnete Menge und $Y \subseteq X$, so heißt ein $x \in X$ mit $x \leq y \quad \forall y \in Y$ eine *untere Schranke* von Y.

Dual erklärt man *obere Schranken*.

Gibt es eine untere Schranke $x_0 \in X$ von Y, so daß für alle unteren Schranken $x \in X$ von Y die Ungleichung $x \leq x_0$ gilt, so nennt man x_0 *Infimum* von Y und schreibt $x_0 = \wedge Y$ oder $x_0 = inf\ Y$.

Dual definiert man das *Supremum* $\vee Y$ (*sup Y*) als kleinste obere Schranke von Y.

Man schreibt auch $\bigwedge\limits_{y \in Y} y := \wedge Y$ und $\bigvee\limits_{y \in Y} y = \vee Y$.

Ist $Y = \{y, z\}$, so schreibt man $y \wedge z$ für das Infimum und $y \vee z$ für das Supremum der beiden Elemente.

([Bir 67], I.4, S. 6)

Aufgrund der Antisymmetrie von $\leq$ sind $\wedge Y$ und $\vee Y$ eindeutig, wenn sie existieren.

Die Zeichen $\wedge$ und $\vee$ sind in dieser Arbeit für die Kennzeichnung von Infima bzw. Suprema zweier Elemente reserviert. Das logische "Und" und das logische "Oder" in Aussagen werden durchgehend als Worte geschrieben.

Hilfssatz 1 ($\leq$ *dargestellt durch* $\wedge$ *oder* $\vee$)

In einer geordneten Menge $(X, \leq)$ gilt für alle $x, y \in X$:

$$x \leq y \Leftrightarrow x \wedge y = x \Leftrightarrow x \vee y = y$$

([Bir 67], I.5, Lemma 1, S. 5)

Beweis: Ist $x \leq y$, so ist x untere Schranke von $\{x, y\}$.

Andererseits sind alle unteren Schranken von $\{x, y\}$ kleiner oder gleich x.

Also ist $x \wedge y = x$.

In der anderen Richtung folgt $x \leq y$ direkt aus $x \wedge y = x$.

Für das Supremum dual.

Im folgenden Hilfssatz sind die wichtigsten Rechenregeln für Infimum und Supremum zusammengefaßt.

Hilfssatz 2 (*Rechenregeln für Infimum und Supremum*)

Sei $(X, \leq)$ eine geordnete Menge, T eine beliebige Indexmenge und $Y_t \subseteq X$ für alle $t \in T$.

Wenn die betreffenden Infima und Suprema existieren, gilt folgendes:

a) $\bigwedge_{t \in T} x = x$, $\bigvee_{t \in T} x = x$ (Idempotenz)

b) $x \wedge y = y \wedge x, x \vee y = y \vee x$ (Kommutativität)

c) $\bigwedge_{t \in T}(\bigwedge Y_t) = \bigwedge\left(\bigcup_{t \in T} Y_t\right)$, $\bigvee_{t \in T}(\bigvee Y_t) = \bigvee\left(\bigcup_{t \in T} Y_t\right)$ (Assoziativität)

d) $x \wedge (x \vee y) = x, x \vee (x \wedge y) = x$ (Absorption)

Beweis: ([Bir 67], I.5, Lemma 1, S. 8)

a) ist klar denn, $x \leq x$ und für jede andere untere Schranke u und obere s gilt $u \leq x$ bzw. $x \leq s$.

b) ist direkt durch die Definition von $x \wedge y$ bzw. $x \vee y$ gegeben.

c) Es ist $\bigwedge_{t \in T}(\bigwedge Y_t) \leq \bigwedge Y_t \leq y \quad \forall t \in T, y \in Y_t$.

D.h. $\bigwedge_{t \in T}(\bigwedge Y_t) \leq \bigwedge\left(\bigcup_{t \in T} Y_t\right)$

Andererseits ist $\bigwedge\left(\bigcup_{t \in T} Y_t\right) \leq y \quad \forall y \in Y_t, t \in T$.

Also $\bigwedge\left(\bigcup_{t \in T} Y_t\right) \leq \bigwedge Y_t \quad \forall t \in T$ und damit $\bigwedge\left(\bigcup_{t \in T} Y_t\right) \leq \bigwedge_{t \in T}(\bigwedge Y_t)$.

Für Suprema dual.

d) ergibt sich aus Hilfssatz 1.

Nicht in allen Ordnungen sind alle Elemente vergleichbar – wie am Beispiel der Inklusionsordnung auf Potenzmengen (Beispiel 1.c) auf Seite 65) gesehen. Eine schwächere Eigenschaft ist die mittelbare Vergleichbarkeit über ein gemeinsames Infimum und Supremum. Dies führt zur Definition von Verbänden.

Definition 7 (*Verband, vollständiger Verband*)

Eine geordnete Menge $(V, \leq)$ heißt *Verband*, wenn zu je zwei Elementen $x, y \in V$ das Infimum $x \wedge y$ und das Supremum $x \vee y$ existieren.

Ein Verband $(V,\leq)$ heißt *vollständig*, wenn zu jeder beliebigen Teilmenge $X \subseteq V$ das Infimum $\bigwedge X$ und Supremum $\bigvee X$ existieren.

Werden mehrere Verbände gleichzeitig betrachtet, finden auch die Schreibweisen $x \wedge_V y, x \vee_V y, \bigwedge_V X, \bigvee_V X$ Anwendung.

Sind dagegen mehrere Ordnungen auf ein und derselben Menge involviert, so wird die Bezeichnung der jeweiligen Ordnung als Index benutzt.

([Bir 67], I.4, S. 6)

Die Ordnungsrelation eines Verbands $(V, \leq)$ kann nach Hilfssatz 1 mit Hilfe von Infima oder Suprema beschrieben werden. Damit ist die Struktur von $(V, \leq)$ allein durch die Abbildungen $\wedge, \vee : V^2 \to V$ gegeben und $(V, \leq)$ ist eine (universelle) Algebra $(V,\wedge,\vee)$.

Folgerung 1 (*Ketten als Verbände*)

Jede linear geordnete Menge $(X, \leq)$ ist ein Verband.

Beweis: Je zwei Elemente $x, y \in X$ sind vergleichbar und es ist $x \wedge y$ das kleinere von beiden und $x \vee y$ das größere.

Bemerkung 3 (*Null-/Einselement, endliche Verbände*)

In der Definition eines vollständigen Verbands $(V, \leq)$ ist insbesondere die Existenz von $\mathbf{0}_V := \wedge V$ und $\mathbf{1}_V := \vee V$ verlangt.
Damit ist ein vollständiger Verband nicht leer.
Jeder endliche nicht leere Verband ist vollständig, denn per Induktion bekommt man mit Hilfssatz 2.c) Infima und Suprema beliebiger endlicher Mengen in einem Verband.
([Bir 67], I.4, S. 6)

Beispiel 2 (*Die Ordnungen aus Beispiel 1*)

a) $(\mathbb{R}, \leq)$ ist linear geordnet, also nach Folgerung 1 ein Verband, aber kein vollständiger Verband, denn es gibt kein kleinstes und kein größtes Element, also kein $x \in \mathbb{R}$ mit $x = \wedge\mathbb{R}$ oder $x = \vee\mathbb{R}$.

b) $(\{0, \ldots, 9\}, \leq)$ und $(\{a, \ldots, z\}, \leq_\alpha)$ sind dagegen als nichtleere endliche Verbände vollständig.

c) $(\{0, \ldots, 9\} \cup \{a, \ldots, z\}, \leq \cup \leq_\alpha)$ ist kein Verband, weil es etwa kein $x \in \{0, \ldots, 9\} \cup \{a, \ldots, z\}$ mit $x = 0 \wedge a$ gibt.

d) $(\mathfrak{P}(X), \subseteq)$ ist für jede Menge M ein vollständiger Verband, in dem Infima durch Schnittmengen und Suprema durch Vereinigungen gegeben sind, denn sind T eine Indexmenge und $Y_t \subseteq X \;\; \forall t \in T$, so ist für alle $t \in T$ dann $\bigcap_{s \in T} Y_s \subseteq Y_t$, also $\bigcap_{s \in T} Y_s \subseteq \bigwedge_{s \in T} Y_s$ und andersherum $\bigwedge_{s \in T} Y_s \subseteq Y_t$, also $\bigwedge_{s \in T} Y_s \subseteq \bigcap_{s \in T} Y_s$.
Dual sieht man $\bigvee_{t \in T} Y_t = \bigcup_{t \in T} Y_t$.

Die Definition von Verbänden ist selbstdual in dem Sinne, daß die duale Ordnung auch wieder einen Verband ergibt.

Satz 2 (*Dualitätsprinzip für Verbände*)
Ist $(V, \leq)$ ein (vollständiger) Verband, so auch $(V, \leq)^d$.
([Bir 67], I.4, S. 6)

Beweis: Sei $(V, \leq)$ ein vollständiger Verband und $X \subseteq V$.
Setze $inf_{\geq} X := \bigvee_{\leq} X$.
Dann gilt $inf_{\geq} X \geq x$ für alle $x \in X$, d.h. $inf_{\geq} X$ ist bezüglich $\geq$ eine untere Schranke von N.
Für alle unteren $\geq$-Schranken y von N gilt $x \geq \bigvee_{\leq} X = inf_{\geq} X$.
Also ist $inf_{\geq} X$ wirklich bezüglich $\geq$ das Infimum von N.
Dual erledigt man das Supremum.
Die Betrachtung nicht-vollständiger Verbände erhält man durch die Beschränkung auf zweielementige Mengen X.

Satz 2 zeigt den Zusammenhang zwischen $(V, \leq)$ und $(V, \leq)^d$. Durch das Umdrehen von Ordnung und Infima↔Suprema erhält man die duale Sicht.

Definition 8 (*Duale verbandstheoretische Aussage*)
Zu einer *verbandstheoretischen Aussage A* über einen Verband $(V, \leq)$, die außer Variablen, Konstanten und logischen Operatoren nur die Zeichen $\leq$, $\wedge$, $\vee$, $\bigwedge$, $\bigvee$, $\mathbf{0}_V$, $\mathbf{1}_V$ enthält, erhält man die *duale Aussage* A^d, indem man in A alle Vorkommen dieser Zeichen durch die dualen Zeichen $\geq$, $\vee$, $\wedge$, $\bigvee$, $\bigwedge$, $\mathbf{1}_V$, $\mathbf{0}_V$ ersetzt.
([G-W 96], 0.2, S. 6)

Bemerkung 4 (*Gültigkeit von dualen Aussagen*)
Eine verbandstheoretische Aussage A über einen Verband $(V, \leq)$ gilt genau dann, wenn A^d in $(V, \leq)^d$ gilt.

3.2.3 Liniendiagramme

Endliche geordnete Mengen kann man in Liniendiagrammen graphisch darstellen. Um diese einzuführen zu können, dient die folgende Definition.

Definition 9 (*Unterer/oberer Nachbar*)
Ist $(X, \leq)$ eine geordnete Menge und $x \in X$, so heißt ein Element $y \in X$ mit $y < x$ ein *unterer Nachbar* von x, wenn für jedes $z \in X$ mit $y \leq z \leq x$ schon $z = y$ oder $z = x$ ist.
Dann ist x ein *oberer Nachbar* von y und man schreibt $y \prec x$.
([Bir 67], I.3, S. 4)

Satz 3 (*Begründung für Liniendiagrammdarstellung*)

Ist $(X, \leq)$ eine endliche geordnete Menge, so ist $\leq$ in folgender Weise schon durch die Relation $\prec$ bestimmt:

$$y \leq x$$
$$\Leftrightarrow$$
$$\exists\, k \in \mathbb{N}_0, x_0, \ldots, x_k \in X : x_0 = y, x_k = x, \forall i \in \{0, \ldots, k-1\} : x_i \prec x_{i+1}$$

Beweis: ([Grä 98], I.2, Lemma 1, S. 12)

Die Rückrichtung ergibt sich per Induktion aus der Transitivität von $\leq$, weil auch $y = x_0 \leq \ldots \leq x_k = y$ gilt.

Jede nichtleere Kette in X ist als Teilmenge von X endlich und enthält somit nach Folgerung 1 und Bemerkung 3 ein kleinstes und ein größtes Element.

Sei $\mathfrak{K}$ die Menge aller Ketten in X, deren kleinstes Element y und deren größtes Element x ist.

$\{y, x\}$ ist eine solche Kette und $\mathfrak{K}$ ist damit nicht leer.

Wähle aus $\mathfrak{K}$ eine Kette $x_0 < \ldots < x_k$ mit maximaler Elementanzahl. (Möglich, weil $\mathfrak{K}$ als Teilmenge von $\mathfrak{P}(X)$ endlich ist und alle Elemente von $\mathfrak{K}$ nach oben ebenfalls endliche Mengen sind.)

Dann gilt $x_i \prec x_{i+1} \quad \forall i \in \{0, \ldots, k-1\}$, denn sonst gäbe es ein $j \in \{0, \ldots, k-1\}$ und ein $z \in X$ mit $x_j < z < x_{j+1}$ und $\{x_0, \ldots, x_k, z\} \in \mathfrak{K}$ hätte eine größere Elementanzahl als $\{x_0, \ldots, x_k\}$.

$(X,\prec)$ kann man als gerichteten Graphen auffassen. Nach Satz 3 ist die Ordnungsstruktur von $(X,\leq)$ durch diesen Graphen vollständig bestimmt.

Liniendiagramme stellen diesen gerichteten Graphen dar. Anders als bei der Darstellung allgemeiner gerichteter Graphen wird aber die Kantenrichtung nicht durch Pfeilspitzen, sondern durch die vertikale Orientierung von Start- und Zielknoten der Kanten angegeben.

Definition 10 (*Liniendiagramm*)

Eine endliche geordnete Menge $(X, \leq)$ wird durch ein *Liniendiagramm* (*Hasse-Diagramm*) dadurch dargestellt, daß

- alle Elemente von X so als Knoten gezeichnet werden, daß im Fall $y < x$ der Knoten für y unterhalb des Knotens für x liegt,
- Knoten für $x, y \in X$ mit $y \prec x$ durch eine (von x nach y absteigende) Kante verbunden werden.

([Bir 67], I.3, S. 4)

Selbstverständlich ist die Darstellung durch ein Liniendiagramm nicht eindeutig. Solange die vertikale Anordnung der durch Linien verbundenen Punkte (sie gibt die Ordnung wieder) erhalten bleibt, können die Punkte beliebig verschoben wer-

den. Es ist auch nicht ohne weiteres klar, was ein gutes Liniendiagramm ausmacht. Eine Minimierung der Überschneidungen von Linien erhöht normalerweise die Übersichtlichkeit. In einigen Fällen kann es aber auch günstig sein, Überschneidungen in Kauf zu nehmen, um schon bekannte Teilstrukturen herauszustellen.

Beispiel 3 (*Liniendiagramme*)

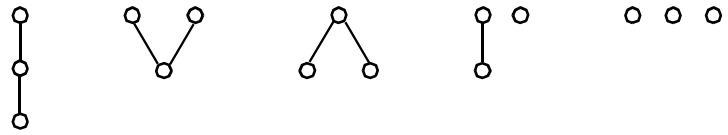


Abb. 3.1 Geordnete Mengen mit 3 Elementen

In Abbildung 3.1 ist nur die erste (linear geordnete) Menge ein Verband.
Der Verband $(\mathfrak{P}(\{1, 2, 3, 4\}), \subseteq)$ kann durch nebenstehendes Liniendiagramm dargestellt werden.

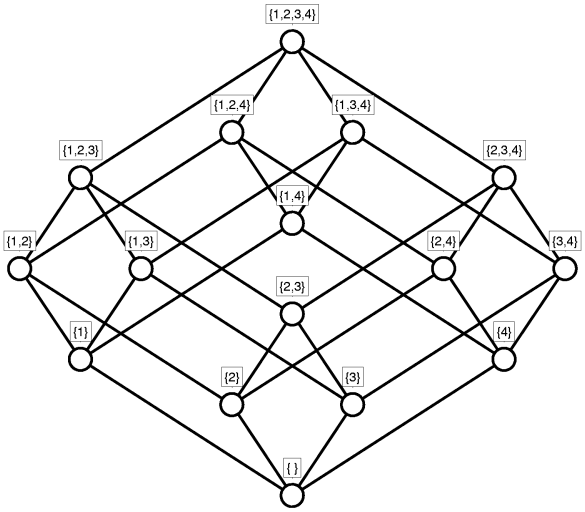


Abb. 3.2 Liniendiagramm von $(\mathfrak{P}(\{1,2,3,4\},\subseteq)$

Schon in den einfachen Beispielen in Abbildung 3.1 fallen die Knoten ins Auge, an denen das Liniendiagramm verzweigt. Im Liniendiagramm eines Verbands ist oft die höchste Verzweigung die des globalen Supremums $\mathbf{1}_V$ zu seinen unteren Nachbarn und die tiefste die vom globalen Infimum $\mathbf{0}_V$ zu seinen oberen Nachbarn. Entsprechend werden die folgenden Sprechweisen vereinbart.

Definition 11 ($\wedge$-*irreduzibel*, $\vee$-*irreduzibel*)

Ist $(V, \leq)$ ein vollständiger Verband, so betrachte für $v \in V$:

$$v^* := \wedge\{x \in V \mid x > v\} \text{ und}$$
$$v_* := \vee\{x \in V \mid x < v\}$$

v heißt $\wedge$-*irreduzibel* oder *infimum-irreduzibel*, wenn $v^* > v$, sonst $\wedge$-*reduzibel* oder *infimum-reduzibel*.

v heißt $\vee$-*irreduzibel* oder *supremum-irreduzibel*, wenn $v_* < v$, sonst $\vee$-*reduzibel* oder *supremum-reduzibel*.

([G-W 96], 0.2, Definition 11, S. 7)

Betrachtet man zu einem Element v eines vollständigen Verbands $(V, \leq)$ alle echt größeren Elemente und bildet das Infimum von diesen, so kommt man auf v selber, wenn v $\bigwedge$-reduzibel ist. v läßt sich dann als Infimum anderer Elemente darstellen oder ist das globale Supremum $\mathbf{1}_V$. Ist dagegen v $\bigwedge$-irreduzibel, so ergibt die beschriebene Infimumbildung ein Element $v^* \neq v$ und dieses v^* ist der einzige obere Nachbar von v. Dual ist w_* der einzige untere Nachbar für ein $\bigvee$-irreduzibles Element w. Im Liniendiagramm endlicher Verbände treten also die Verzweigungen nach oben genau an den $\bigwedge$-reduziblen und nach unten genau an den $\bigvee$-reduziblen Elementen auf. (Ausnahme $\mathbf{1}_V$ bzw. $\mathbf{0}_V$: $\mathbf{1}_V$ ist immer $\bigwedge$-reduzibel, hat aber natürlich keine Verzweigung nach oben und dual $\mathbf{0}_V$ ist immer $\bigvee$-reduzibel.)

Bezieht man "Irreduzibilität" auf allgemeine Verbände, so ist eine andere Formulierung naheliegend. Garret Birkhoff bezeichnet ein Element v eines beliebigen Verbands $(V, \leq)$ als $\wedge$-irreduzibel (mit kleinem $\wedge$), wenn für alle $x, y \in V$ gilt:

$$x \wedge y = v \implies x = v \text{ oder } y = v$$

Dual ist dann v als $\vee$-irreduzibel (mit kleinem $\vee$) erklärt, wenn für alle $x, y \in V$ gilt: $x \vee y = v \implies x = v$ oder $y = v$

([Bir 67], III.3, S 58)

Die in Definition 10 beschriebene Eigenschaft ist stärker, denn ist v $\bigwedge$-irreduzibel und sind $x, y > v$, so ist $x \wedge y \geq \bigwedge\{x \in V \mid x > v\} = v^* > v$, also v $\wedge$-irreduzibel.

Betrachtet man als Beispiel eines vollständigen Verbands das reelle Intervall [0,1] mit der natürlichen Ordnung $\leq$, so sieht man, daß $\bigwedge$-irreduzibel echt stärker ist als $\wedge$-irreduzibel, denn alle Elemente von [0,1] sind $\wedge$-irreduzibel wegen der linearen Ordnung, aber $\bigwedge$-reduzibel (dual genauso für die Suprema).

Da im folgenden im wesentlichen vollständige Verbände betrachtet werden, sollen Birkhoffs Konzepte von Irreduzibilität nicht weiter betrachtet werden.

Zudem sind später alle betrachteten Verbände endlich, so daß dann die Definitionen zusammenfallen.

Definition 12 (*Atom, Coatom*)

Ist $(V, \leq)$ ein vollständiger Verband, so heißen die oberen Nachbarn von $\mathbf{0}_V$ Atome und dual die unteren Nachbarn von $\mathbf{1}_V$ Coatome.

([G-W 96], 0.2, S. 7)

Am Beispiel von ([0,1],≤) sieht man, daß nicht jeder vollständige Verband Atome und Coatome enthält. Wenn das aber der Fall ist, so sind Atome $\bigvee$-irreduzibel und Coatome $\bigwedge$-irreduzibel.

Bemerkung 5 (*Duales Liniendiagramm*)

Aus einem Liniendiagramm für $(X, \leq)$ erhält man ein Liniendiagramm für die dual geordnete Menge $(X, \leq)^d$, indem man es einfach auf den Kopf stellt.

3.2.4 Homomorphismen

Wie schon durch das Weglassen von Elementbezeichnern in Abbildung 3.1 angedeutet, ist man aus ordnungstheroetischer Sicht eher an der durch die Ordnung auf einer Menge gegebene Struktur als an den einzelnen Mengenelementen interessiert. Die folgende Definition stellt die Hilfsmittel zum Vergleich von Ordnungsstrukturen bereit.

Definition 13 (*Ordnungserhaltend, Ordnungseinbettung, Ord.-Isomorphismus*)
Eine Abbildung $\varphi : X \to Y$ zwischen zwei geordneten Mengen $(X, \leq_X)$ und $(Y, \leq_Y)$ heißt *ordnungserhaltend*, *isoton* oder *Ordnungshomomorphismus*, wenn für alle $x, z \in X$ mit $x \leq_X z$ auch $\varphi(x) \leq_Y \varphi(z)$ gilt.
Gilt zusätzlich noch die Umkehrung $\varphi(x) \leq_Y \varphi(z) \Rightarrow x \leq_X z$,so ist φ eine *Ordnungseinbettung* (φ ist dann wegen der Antisymmetrie von $\leq_X$ injektiv).
Eine bijektive Ordnungseinbettung heißt *(Ordnungs-)Isomorphismus*. Man schreibt dann $X \cong Y$.
([Bir 67], I.2, S. 2)

Bis auf Isomorphie sind in Abbildung 3.1 alle dreielementigen geordneten Mengen angegeben, d.h. alle möglichen Ordnungsstrukturen.

Strukturvergleiche zwischen Verbänden sollten auch über Infima und Suprema zu unternehmen sein. Deshalb sind die folgenden "Morphismen" interessant.

Definition 14 (*Verbandshomomorphismen*)
V und W seien zwei Verbände und $\varphi : V \to W$ eine Abbildung.

a) φ heißt *infimum-erhaltend* oder $\wedge$-*Morphismus* (*supremum-erhaltend* / $\vee$-*Morphismus*), wenn für alle $x, y \in V$ gilt:
$$\varphi(x \wedge_V y) = \varphi(x) \wedge_W \varphi(y) \qquad (\varphi(x \vee_V y) = \varphi(x) \vee_W \varphi(y))$$

b) Sind V und W vollständige Verbände und gilt $\varphi(\bigwedge_V X) = \bigwedge_W \varphi(X)$ bzw. $\varphi(\bigvee_V X) = \bigvee_W \varphi(X)$ für alle $X \subseteq V$, so heißt φ *vollständiger* $\wedge$-*Morphismus* respektive *vollständiger* $\vee$-*Morphismus*.

c) φ heißt *Verbandshomomorphismus*, wenn φ infimum-erhaltend und supremum-erhaltend ist.

d) Sind V und W vollständige Verbände und ist φ sowohl vollständiger $\wedge$-Morphismus als auch vollständiger $\vee$-Morphismus, so heißt φ *vollständiger Verbandshomomorphismus* oder *Vollhomomorphismus*.

([Bir 67], II.3, S. 24; [Ern 82], 6, S. 112)

Wenn in den folgenden Abschnitten vollständige Verbände betrachtet werden, stehen drei Homomorphismen-Konzepte zur Auswahl, um Abbildungen strukturer-

haltende Eigenschaften zuzusprechen: Ordnungshomomorphismus, Verbandshomomorphismus und vollständiger Verbandshomomorphismus. Daß diese drei wirklich auseinanderfallen, zeigen die folgenden Beispiele.

Beispiel 4 (*Verschiedene Homomorphismen zwischen Verbänden*)

a) Folgende zwei Liniendiagramme beschreiben vollständige Verbände:

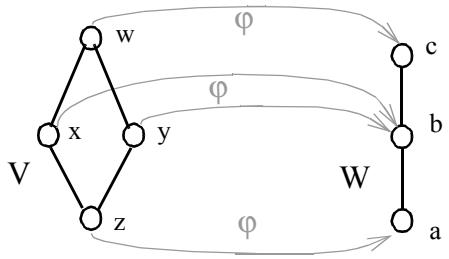


Abb. 3.3 Zwei Verbände und eine ordnungserhaltende Funktion

Die durch Pfeile angedeutete Abbildung φ ist ein Ordnungshomomorphismus, aber nicht infimum-erhaltend, also kein Verbandshomomorphismus, denn $\varphi(x \wedge y) = \varphi(z) = a \neq b = b \wedge b = \varphi(x) \wedge \varphi(y)$.

b) $\overline{\mathbb{R}_+} := \{x \in \mathbb{R} \mid x \geq 0\} \cup \{\infty\}$ ist mit der natürlichen Ordnung $\leq$ ein vollständiger Verband.
Betrachte weiter W aus a) und:

$$\varphi : \overline{\mathbb{R}_+} \to W, \varphi(x) = \begin{cases} a \text{ für } x \in \mathbb{R} \\ b \text{ für } x = \infty \end{cases}$$

Dann ist φ ein Verbandshomomorphismus, aber kein vollständiger Verbandshomomorphismus, denn $\varphi(\vee\mathbb{R}) = \varphi(\infty) = b \neq a = \vee\varphi(\mathbb{R})$.

Interessiert man sich jedoch nur für Isomorphismen, fallen alle Homomorphismen-Konzepte zusammen. Deshalb sind in Definition 14 "Verbandsisomorphismen" gar nicht erst gesondert definiert.

Satz 4 (*Verbandsisomorphismus ≅ Ordnungsisomorphismus*)

V und W seien zwei Verbände und $\varphi : V \to W$ eine Abbildung.

a) φ ist genau dann ein bijektiver Verbandshomomorphismus, wenn φ ein Ordnungsisomorphismus ist.

b) Sind V und W vollständige Verbände, so ist zusätzlich zu den beiden Eigenschaften aus a) äquivalent, daß φ ein bijektiver vollständiger Verbandshomomorphismus ist.

([Bir 67], II.3, Lemma 2, S. 24; [Ern 82], 6.5 Satz, S. 116)

Beweis: ([Grä 98], I.3, S. 20; vergl. [Ern 82], 6.4 Proposition, S. 116)
Siehe Anhang A, Seite 254.

3.2.5 Galois-Verbindungen

Der wesentliche Zusammenhang zwischen Extensionen und Intensionen eines Begriffssystems läßt sich in den hier formalisierten Situationen durch eine Galois-Verbindung beschreiben. Dieses Konzept wurde von Oystein Ore ([Ore 44]) eingeführt. Galois-Verbindungen treten in vielen verschiedenen Zusammenhängen auf, insbesondere beschreibt der Hauptsatz der Galois-Theorie eine solche Verbindung zwischen den Unterkörpern einer galoisschen Körpererweiterung und der zugehörigen Galois-Gruppe (vergl. [Kun 91], 10.4, S. 119). Diesem Beispiel verdanken sie ihren Namen.

Definition 15 (*Galois-Verbindung*)
Sind $(X, \leq_X)$ und $(Y, \leq_Y)$ geordnete Mengen, $\varphi : X \to Y$ und $\psi : Y \to X$ Abbildungen mit

a) $x_1 \leq_X x_2 \Rightarrow \varphi(x_2) \leq_Y \varphi(x_1)$ für alle $x_1, x_2 \in X$ und

b) $y_1 \leq_Y y_2 \Rightarrow \psi(y_2) \leq_X \psi(y_1)$ für alle $y_1, y_2 \in Y$ und

c) $x \leq_X \psi(\varphi(x))$ für alle $x \in X$ und $y \leq_Y \varphi(\psi(y))$ für alle $y \in Y$,

dann heißt das Paar (φ, ψ) *Galois-Verbindung* zwischen $(X,\leq_X)$ und $(Y,\leq_Y)$ und die Abbildungen φ und ψ heißen *dual adjungiert*.
([Ore 44], 2., S. 495)

Besonders hervorstechend ist die folgende Eigenschaft von Galois-Verbindungen.

Hilfssatz 3 (*Abgeschlossenheit dual adjungierter Abbildungen*)
Ist (φ, ψ) eine Galois-Verbindung zwischen zwei geordneten Mengen $(X, \leq_X)$ und $(Y, \leq_Y)$, so gilt

$$\varphi\psi\varphi = \varphi \text{ und } \psi\varphi\psi = \psi.$$

Beweis: ([Bir 67], V.7, S. 123)
Für $x \in X$ ist nach Definition 15.c) $\varphi(x) \leq_Y \varphi(\psi(\varphi(x)))$.
Andererseits folgt aus $x \leq_X \psi(\varphi(x))$ (Definition 15.c)
nach Definition 15.a) $\varphi(\psi(\varphi(x))) \leq_Y \varphi(x)$.
Das bedeutet $\varphi(\psi(\varphi(x))) = \varphi(x)$, also insgesamt $\varphi\psi\varphi = \varphi$.
$\psi\varphi\psi = \psi$ folgt analog.

3.2.6 Hüllensysteme

Die Inklusionsordnung auf Mengenfamilien spielt im folgenden eine herausragende Rolle. Das in Beispiel 2.d) auf Seite 69 angegebene Ergebnis, daß Potenzmengen mittels der Inklusionsordnung geordnet einen vollständigen Verband bilden, kann auf allgemeinere Mengen übertragen werden.

Definition 16 (*Hüllensystem*)

Sei X eine beliebige Menge.

Ein gegen beliebige Durchschnitte abgeschlossenes System $\mathcal{U} \subseteq \mathfrak{P}(X)$ heißt ein *Hüllensystem* auf X, d.h. für jede Teilmenge $\mathcal{X} \subseteq \mathcal{U}$ ist $\bigcap \mathcal{X} \in \mathcal{U}$.

([G-W 96], 0.3, Definition 14, S. 8)

Insbesondere gilt für ein Hüllensystem $\mathcal{U}$ auf einer Menge X, daß $X = \bigcap \varnothing \in \mathcal{U}$.

Eng verbunden mit den Hüllensystemen sind die Hüllenoperatoren. Sie spielen bei vielen Algorithmen innerhalb der Formalen Begriffsanalyse eine tragende Rolle (siehe Anhang B).

Definition 17 (*Hüllenoperator*)

Sei X eine beliebige Menge.

Eine Abbildung $\varphi : \mathfrak{P}(X) \to \mathfrak{P}(X)$ heißt *Hüllenoperator* auf X, wenn gilt:

a) $V \subseteq W \Rightarrow \varphi(V) \subseteq \varphi(W)$ für alle $V, W \subseteq X$ (Monotonie)

b) $V \subseteq \varphi(V)$ für alle $V \subseteq X$ (Extensität)

c) $\varphi(\varphi(V)) = \varphi(V)$ für alle $V \subseteq X$ (Idempotenz)

Die Mengen $\varphi(V)$ mit $V \subseteq X$ heißen dann Hüllen von φ.

([G-W 96], 0.3, Definition 14, S. 8)

Im wesentlichen beschreiben Hüllensysteme und Hüllenoperatoren die gleichen Sachverhalte. Insbesondere bilden die Hüllen eines Hüllenoperators ein Hüllensystem. Sogar ist jedes Hüllensystem die Menge der Hüllen eines Hüllenoperators und umgekehrt kann ein Hüllenoperator allein über das Hüllensystem seiner Hüllen definiert werden. Deshalb nennt man die Elemente eines Hüllensystems auch Hüllen.

Hilfssatz 4 (*Hüllensystem* $\leftrightarrow$ *Hüllenoperator*)

Sei X eine beliebige Menge.

Ist $\mathcal{U} \subseteq \mathfrak{P}(X)$ ein Hüllensystem auf X, so definiert

$\varphi_{\mathcal{U}}(V) := \bigcap \{A \in \mathcal{U} \mid V \subseteq A\}$ für $V \subseteq X$ einen Hüllenoperator auf X.

Ist umgekehrt $\varphi : \mathfrak{P}(X) \to \mathfrak{P}(X)$ ein Hüllenoperator auf X, so ist das System seiner Hüllen $\mathcal{U}_{\varphi} := \{\varphi(Z) \mid Z \subseteq X\}$ ein Hüllensystem auf X.

Weiter ist $\varphi_{\mathcal{U}_{\varphi}} = \varphi$ und $\mathcal{U}_{\varphi_{\mathcal{U}}} = \mathcal{U}$.

Beweis: ([G-W 96], 0.3, Satz 1, S. 9)

Siehe Anhang A, Seite 255.

An der Definition von $\varphi_{\mathcal{U}}$ kann man ablesen, daß zu einer beliebigen Menge $V \subseteq X$, die Menge $\varphi_{\mathcal{U}}(V)$ die bzgl. Inklusion kleinste Hülle aus $\mathcal{U}$ ist, in der V enthalten ist.

Wie schon oben angedeutet, bildet ein Hüllensystem mit der Inklusionsordnung einen vollständigen Verband. Verbände dieser Art spielen in der Formalen Begriffsanalyse eine zentrale Rolle. Bis auf Isomorphie sind sogar alle vollständigen Verbände von dieser Form.

Satz 5 (*Hüllensysteme* $\leftrightarrow$ *vollständige Verbände*)

Ist X eine Menge und $\mathcal{U} \subseteq \mathfrak{P}(X)$ ein Hüllensystem auf X, so ist $(\mathcal{U}, \subseteq)$ ein vollständiger Verband mit $\bigwedge \mathfrak{X} = \bigcap \mathfrak{X}$ und $\bigvee \mathfrak{X} = \varphi_{\mathcal{U}}(\bigcup \mathfrak{X})$ für alle $\mathfrak{X} \subseteq \mathcal{U}$. Andersherum ist jeder vollständige Verband isomorph zum Verband der Hüllen eines Hüllensystems.

Beweis: ([G-W 96], 0.3, Hilfssatz 3, S. 9)

Sei X eine Menge und $\mathcal{U} \subseteq \mathfrak{P}(X)$ ein Hüllensystem auf X.
Wie in Beispiel 2.d) auf Seite 69 schon gesehen, gilt $\bigwedge \mathfrak{X} = \bigcap \mathfrak{X}$ in $(\mathcal{U}, \subseteq)$.
Nach Hilfssatz 4 ist $\varphi_{\mathcal{U}}(\bigcup \mathfrak{X}) \in \mathcal{U}_{\varphi_{\mathcal{U}}} = \mathcal{U}$.
Wegen der Extensität von $\varphi_{\mathcal{U}}$ gilt für alle $V \in \mathfrak{X}$, daß $V \subseteq \bigcup \mathfrak{X} \subseteq \varphi_{\mathcal{U}}(\bigcup \mathfrak{X})$.
D.h. $\varphi_{\mathcal{U}}(\bigcup \mathfrak{X})$ ist in $(\mathcal{U}, \subseteq)$ eine obere Schranke von $\mathfrak{X}$.
Für jede obere Schranke S von $\mathfrak{X}$ in $(\mathcal{U}, \subseteq)$ gilt $\bigcup \mathfrak{X} \subseteq S$.
Wegen $S \in \mathcal{U} = \mathcal{U}_{\varphi_{\mathcal{U}}}$ existiert ein $U \in \mathcal{U}$ mit $S = \varphi_{\mathcal{U}}(U)$.
Nach Hilfssatz 4 gilt damit auch
$\varphi_{\mathcal{U}}(\bigcup \mathfrak{X}) \subseteq \varphi_{\mathcal{U}}(S) = \varphi_{\mathcal{U}}(\varphi_{\mathcal{U}}(U)) = \varphi_{\mathcal{U}}(U) = S$.
Also ist $\varphi_{\mathcal{U}}(\bigcup \mathfrak{X}) = \bigvee \mathfrak{X}$.
Insgesamt ist $(\mathcal{U}, \subseteq)$ ein vollständiger Verband.
Sei nun $(V, \leq)$ ein vollständiger Verband.
Betrachte $\mathfrak{V} := \{(x] \mid x \in V\}$.
Teilmengen von $\mathfrak{V}$ haben die Form $\{(x] \mid x \in U\}$ mit einem $U \subseteq V$ und es gilt $\bigcap\{(x] \mid x \in U\} = (\bigwedge U]$ (auch für den Fall $U = \varnothing$).
Damit ist $\mathfrak{V}$ ein Hüllensystem.
Die Abbildung $\psi : V \to \mathfrak{V}, x \mapsto (x]$ ist nach Konstruktion von $\mathfrak{V}$ ein Ordnungsisomorphismus mit der Inversen $\psi^{-1} : \mathfrak{V} \to V, W \mapsto \bigvee W$.

3.3 Kontexte und Begriffsverbände

Ausgangspunkt für die Formale Begriffsanalyse sind eine Menge von (formalen) *Gegenständen*, eine Menge von (formalen) *Merkmalen* und eine *Inzidenzrelation*, die wiedergibt, welche Merkmale auf welche Gegenstände zutreffen. Diese drei Bestandteile werden in einem *formalen Kontext* zusammengefaßt. Aus diesem wird ein *Begriffsverband* berechnet. Ein formaler Begriff dieses Verbands faßt in seiner Extension genau die Gegenstände zusammen, die im zugrundeliegenden

Kontext alle formalen Merkmale seiner Intension aufweisen. Die Ordnung auf diesen Begriffen ist die Ober-/Unterbegriffsordnung.

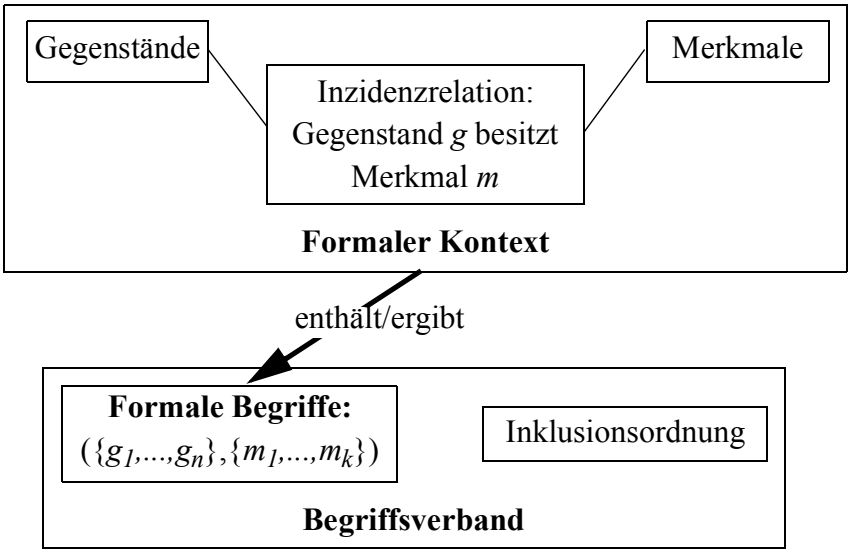


Abb. 3.4 Strukturen der Formalen Begriffsanalyse

Es folgt nun eine mathematische Formalisierung hierfür.

Die mathematische Theorie der Formalen Begriffsanalyse wurde seit Ende der 70'er Jahre vornehmlich von der Arbeitsgruppe "Allgemeine Algebra und Diskrete Mathematik" um Rudolf Wille an der Technischen Universität Darmstadt als Instrument zur Datenanalyse entwickelt (siehe [G-W 96]). Die grundlegende Konstruktion eines vollständigen Verbands aus einer beliebigen binären Relation, die in der Formalen Begriffsanalyse von einem formalen Kontext zu einem Begriffsverband führt, findet sich schon bei Garret Birkhoff ([Bir 67]). Auch die Interpretation von Gegenständen und Merkmalen behandelt er am Rande in Verbindung mit der Booleschen Logik ([Bir 40], VIII, S. 122 ff.).

3.3.1 Formale Kontexte

Grundlage einer formalen Begriffsanalyse ist ein *formaler Kontext*. Er umfaßt die betrachteten Gegenstände und Merkmale und die Gegenstand-Merkmal-Zuordnung. Dabei ist die Bezeichnung "Kontext" im herkömmlichen Sinne zu verstehen. Der formale Kontext gibt den Rahmen an, in dem die weiter folgende Analyse gültig ist. Verläßt man diesen Rahmen, verliert sie eventuell ihre Gültigkeit. Der wichtigste Schritt bei der Formalen Begriffsanalyse ist demnach die Auswahl eines passenden formalen Kontexts.

Definition 18 (*Formaler Kontext*)

Ein *formaler Kontext* (G, M, I) besteht aus zwei Mengen G und M und einer binären Relation I zwischen G und M.
Die Elemente von G heißen *(formale) Gegenstände*, die von M *(formale) Merkmale* und I heißt *Inzidenzrelation*.
$g\ I\ m$ mit $g \in G$ und $m \in M$ wird gelesen als "Der Gegenstand g trägt das Merkmal m.".
(G, M, I) heißt endlich, wenn G und M endlich sind.
([G-W 96], I.1.1, Definition 18, S. 17)

Endliche formale Kontexte lassen sich durch Kreuztabellen darstellen, indem die Tabellenzeilen mit den formalen Gegenständen und die Tabellenspalten mit den formalen Merkmalen benannt werden und die Zeile zu einem formalen Gegenstand g in der Spalte zu einem formalen Merkmal m ein Kreuz erhält, wenn m auf g zutrifft.

Beispiel 5 (*Formaler Kontext als Kreuztabelle*)

In diesem Beispiel werden schon in 2.4.2 angesprochene Sachverhalte als formaler Kontext betrachtet. In nebenstehender Tabelle sind einige bekannte Autoren von OOA-Methoden aufgeführt und mit der Unified Modeling Language in Bezug auf die jeweils verwandten Beschreibungstechniken verglichen.

	Use case diagram	Class diagram	Sequence diagram	Collaboration diagram	Statechart diagram	Activity diagram	Component diagram	Deployment diagram	Timing diagram	Data flow diagram	Object diagram	State transition graph	Fence diagram	Domain chart
UML	X	X	X	X	X	X	X	X						
Booch		X		X	X		X	X	X					
Coad/Yourdon		X			X					X				
Jacobson	X	X	X		X						X	X		
Martin/Odell		X		X	X	X							X	
Rumbaugh		X	X	X	X					X				
Shlaer/Mellor		X		X	X		X			X				X

Abb. 3.5 Formaler Kontext zu OO-Methoden und Beschreibungstechniken

Dies ergibt einen formalen Kontext. Formale Gegenstände sind *UML* und die Autoren, formale Merkmale die angebotenen Beschreibungstechniken. Die durch Kreuze in der Tabelle angegebene Inzidenzrelation beschreibt, welche Technik in welcher Methode vorgesehen ist.

Die Struktur eines formalen Kontexts ist sehr einfach. Die Tabellendarstellung tritt bei vielen Problemen (insbesondere Klassifikationsaufgaben) in natürlicher Weise auf.

3.3.2 Formale Begriffe

Um jetzt zu den formalen Begriffen des Kontext zu kommen, betrachtet man für jeden formalen Gegenstand die auf ihn zutreffenden formalen Merkmale und zu jedem formalen Merkmal alle formalen Gegenstände, die es tragen.

Definition 19 (*Gemeinsame Merkmale/Gegenstände*)

Ist (G, M, I) ein formaler Kontext und $A \subseteq G$ eine Menge von Gegenständen, so ist $A' := \{m \in M \mid g\ I\ m \ \ \forall g \in A\}$
die Menge der gemeinsamen formalen Merkmale der formalen Gegenstände in A und dual für $B \subseteq M$ $B' := \{g \in G \mid g\ I\ m \ \ \forall m \in B\}$
die Menge der formalen Gegenstände, die alle formalen Merkmale aus B tragen.
Für den Fall, daß mehrere formale Kontexte gleichzeitig betrachtet werden, wird auch A^I bzw. B^I statt A' und B' mit der jeweiligen Inzidenzrelation I geschrieben.
In dem Sonderfall, daß G und M die gleiche Menge oder die Elemente von G und M nur schwer zu unterscheiden sind, schreibt man auch $A^{\uparrow}$ und $B^{\downarrow}$ statt A' und B'.
([Bir 67], V.7, S. 122; [G-W 96], 1.1, Definition 18, S. 17)

Die Eigenschaften der Abbildungen, die sich durch den Übergang zu gemeinsamen formalen Merkmalen bzw. Gegenständen ergeben, lassen sich im wesentlichen dadurch charakterisieren, daß diese beiden Abbildungen eine Galois-Verbindung zwischen den Potenzmengen der formalen Gegenstände und der formalen Merkmale bilden.

Satz 6 (*Galois-Verbindung durch ′*)

Ist (G, M, I) ein formaler Kontext, so bilden die Abbildungen
$\mathfrak{P}(G) \to \mathfrak{P}(M), A \mapsto A'$ und $\mathfrak{P}(M) \to \mathfrak{P}(G), B \mapsto B'$ eine Galois-Verbindung zwischen $(\mathfrak{P}(G), \subseteq)$ und $(\mathfrak{P}(M), \subseteq)$.
([Bir 67], V.7, S. 122)

Beweis: ([G-W 96], 1.1, Hilfssatz 10, S. 18)

Zeige: $A \subseteq C \Rightarrow C' \subseteq A'$ für alle $A, C \subseteq G$.
Ist $m \in C'$, so gilt $g\ I\ m$ für alle $g \in C$, also auch für alle $g \in A$.
D.h. $m \in A'$ und insgesamt $C' \subseteq A'$.
Analog gilt $B \subseteq D \Rightarrow D' \subseteq B'$ für alle $B, D \subseteq M$.
Zeige: $A \subseteq A''$ für alle $A \subseteq G$.
Ist $g \in A$, so gilt $g\ I\ m$ für alle $m \in A'$, d.h. $g \in A''$.
Damit folgt $A \subseteq A''$.
Analog gilt $B \subseteq B''$ für alle $B \subseteq M$.

Im folgenden sind die wichtigsten Regeln im Umgang mit den ′-Abbildungen noch einmal zusammengefaßt.

Folgerung 2 (*Rechenregeln für ′*)

Ist (G, M, I) ein formaler Kontext, so gilt:

a) $\left(\bigcup_{t \in T} A_t\right)' = \bigcap_{t \in T} A_t' \quad \forall A_t \subseteq G, t \in T$ und

$\left(\bigcup_{t \in T} B_t\right)' = \bigcap_{t \in T} B_t' \quad \forall B_t \subseteq M, t \in T$ mit einer beliebigen Indexmenge T

b) $A \subseteq C \Rightarrow C' \subseteq A' \quad \forall A, C \subseteq G$ und $B \subseteq D \Rightarrow D' \subseteq B' \quad \forall B, D \subseteq M$

c) $A \subseteq A'' \quad \forall A \subseteq G$ und $B \subseteq B'' \quad \forall B \subseteq M$

d) $A''' = A' \quad \forall A \subseteq G$ und $B''' = B' \quad \forall B \subseteq M$

e) $A \subseteq B' \Leftrightarrow B \subseteq A' \Leftrightarrow A \times B \subseteq I \quad \forall A \subseteq G, B \subseteq M$

Beweis: ([G-W 96], 1.1, Hilfssatz 10, S. 18 und Hilfssatz 11, S. 19)

a) Es gilt:

$$\left(\bigcup_{t \in T} A_t\right)' = \left\{ m \in M \mid g\ I\ m \quad \forall g \in \bigcup_{t \in T} A_t \right\}$$

$$= \{ m \in M \mid \forall t \in T : g\ I\ m \quad \forall g \in A_t \}$$

$$= \bigcap_{t \in T} \{ m \in M \mid g\ I\ m \quad \forall g \in A_t \} = \bigcap_{t \in T} A_t'$$

Diese Gleichung ist auch im Fall $T = \varnothing$ richtig, dann ist

$$\left(\bigcup_{t \in T} A_t\right)' = \varnothing' = M$$

Der Fall einer Vereinigung von Merkmalsmengen ergibt sich analog.

b) und c) ergeben sich direkt aus Satz 6.

d) folgt mit Hilfssatz 3 auf Seite 76 aus Satz 6.

e) Seien $A \subseteq G, B \subseteq M$.

Alle drei Aussagen bedeuten $g\ I\ m \quad \forall g \in A, m \in B$.

Formale Begriffe lassen sich mit Hilfe der ′-Abbildungen wie folgt definieren.

Definition 20 (*Formaler Begriff*)

Ein *formaler Begriff* eines formalen Kontexts (G, M, I) ist ein Paar (A, B) mit $A \subseteq G, B \subseteq M$ und $A' = B$ und $B' = A$.

A heißt dann der *Umfang* (die *Extension*) und B der *Inhalt* (die *Intension*) von (A, B).

$\mathfrak{B}(G, M, I)$ sei die Menge aller formalen Begriffe des Kontexts (G, M, I).

([G-W 96], 1.1, Definition 20, S. 18)

Formale Begriffe sind also dadurch gekennzeichnet, daß ihr Inhalt genau aus den formalen Merkmalen besteht, die auf alle formalen Gegenstände des Umfangs zutreffen, und umgekehrt ihr Umfang gerade die formalen Gegenstände enthält, die alle formalen Merkmale aus dem Inhalt tragen. Ein formaler Begriff ist nach Definition 20 schon durch die Angabe seines Umfangs oder seines Inhalts vollständig bestimmt. Im folgenden Hilfssatz kann so schon eine Darstellung aller formalen Begriffe eines formalen Kontexts formuliert werden. Umfang und Inhalt sind nämlich Bilder unter den entsprechenden $'$-Abbildungen.

Hilfssatz 5 (*Darstellung von formalen Begriffen*)
Ist (G, M, I) ein formaler Kontext, so gilt

$$\mathfrak{B}(G, M, I) = \{(A'', A') \mid A \subseteq G\} = \{(B', B'') \mid B \subseteq M\}$$

([G-W 96], 1.1, S. 19)

Beweis: Nach Folgerung 2.d) ist $A''' = A' \quad \forall A \subseteq G$.
Daraus folgt $\{(A'', A') \mid A \subseteq G\} \subseteq \mathfrak{B}(G, M, I)$.
Für $(A, B) \in \mathfrak{B}(G, M, I)$ ist nach Definition 20 $B = A'$ und
$A = B' = A''$, was auch die umgekehrte Inklusion liefert.
$\mathfrak{B}(G, M, I) = \{(B', B'') \mid B \subseteq M\}$ folgt analog.

Folgerung 3 (*Hüllensystem der Begriffsumfänge/-inhalte*)
Sei (G, M, I) ein formaler Kontext.
Dann bilden die Begriffsumfänge von (G, M, I) ein Hüllensystem und die Abbildung $'' : \mathfrak{P}(G) \to \mathfrak{P}(G)$ ist der zugehörige Hüllenoperator.
Genauso ist die Menge aller Begriffsinhalte ein Hüllensystem mit dem Hüllenoperator $'' : \mathfrak{P}(M) \to \mathfrak{P}(M)$.
([G-W 96], 1.1, S. 19)

Beweis: Daß die Abbildungen Hüllenoperatoren sind, folgt aus Folgerung 2:
b) ergibt zweimal angewandt die Monotonie, in c) steht die Extensität und aus d) folgt die Idempotenz.
Ihre Bilder sind nach Hilfssatz 5 gerade die Begriffsumfänge bzw. -inhalte.

Demnach ist zu $A \subseteq G$ A'' der kleinste Begriffsumfang, der A enthält, und dual B'' zu $B \subseteq M$ der kleinste Begriffsinhalt, der B umfaßt.

3.3.3 Begriffsverbände

Begriffe dienen zur Klassifikation. Dementsprechend ist die Ober-/Unterbegriffsrelation zwischen ihnen eine besonders wichtige Beziehung. Ein Begriff heißt Oberbegriff eines anderen, wenn er allgemeiner ist, also mehr Gegenstände umfaßt.

Definition 21 (*Ober-/Unterbegriffsordnung*)

Sind (A, B) und (C, D) formale Begriffe eines formalen Kontexts (G, M, I), so heißt (C, D) ein *Oberbegriff* von (A, B), geschrieben $(A, B) \leq (C, D)$, wenn $A \subseteq C$.
(A, B) heißt dann *Unterbegriff* von (C, D).
Die Bedingung $D \subseteq B$ ist äquivalent zu $A \subseteq C$.
([G-W 96], 1.1, Definition 21, S. 20)

Die Ober-/Unterbegriffsrelation wird hier als Ordnung bezeichnet und durch das Zeichen ≤ notiert. Dies ist gerechtfertigt, weil sie ist durch die Inklusionsordnung auf der Potenzmenge der Gegenstände $\mathfrak{P}(G)$ definiert ist und formale Begriffe schon allein durch ihren Umfang festgelegt sind. Deshalb übertragen sich die Ordnungseigenschaften.

Die Äquivalenz der Bedingung $D \subseteq B$ erhält man mit $B = A', D = C'$ und $A = B', C = D'$ aus Folgerung 2.b). Auch die Ober-/Unterbegriffsordnung hat also dualen – extensionalen und intensionalen – Charakter.

Mit dieser Ordnung bilden die formalen Begriffe eines formalen Kontexts einen vollständigen Verband, wie der nächste Satz zeigt.

Satz 7 (*Hauptsatz über Begriffsverbände*)

Ist (G, M, I) ein formaler Kontext, so ist $\underline{\mathfrak{B}}(G, M, I) := (\mathfrak{B}(G, M, I), \leq)$ mit der Ober-/Unterbegriffsordnung aus Definition 21 ein vollständiger Verband mit:

$$\bigwedge_{t \in T}(A_t, B_t) = \left(\bigcap_{t \in T} A_t, \left(\bigcup_{t \in T} B_t\right)''\right)$$

$$\bigvee_{t \in T}(A_t, B_t) = \left(\left(\bigcup_{t \in T} A_t\right)'', \bigcap_{t \in T} B_t\right)$$

für alle Indexmengen T und $(A_t, B_t) \in \mathfrak{B}(G, M, I) \quad \forall t \in T$.

Beweis: ([G-W 96], 1.1, Satz 3, S. 20)

Nach Folgerung 2.a) ist $\left(\bigcup_{t \in T} B_t\right)' = \bigcap_{t \in T} B_t' = \bigcap_{t \in T} A_t$.

Also ist $\left(\bigcap_{t \in T} A_t, \left(\bigcup_{t \in T} B_t\right)''\right)$ nach Hilfssatz 5 ein formaler Begriff.

Daß er gerade das Infimum der Begriffe (A_t, B_t) bildet, ist nach der Definition der Ober-/Unterbegriffsordnung und Beispiel 2.d) auf Seite 69 klar.

Für das Supremum betrachtet man dual $\left(\bigcup_{t \in T} A_t\right)'$.

Definition 22 (*Begriffsverband*)

$\underline{\mathfrak{B}}(G, M, I)$ aus Satz 7 heißt *Begriffsverband* des Kontexts (G, M, I).
([G-W 96], 1.1, Definition 21, S. 20)

Vor allen in der französischsprachigen Literatur ist nach Marc Barbut und Bernard Monjardet ([B-M 70]) auch die Bezeichnung *Galois-Verband* (Franz. *treillis de Galois*) üblich.

Um solche Begriffsverbände, die aus einem formalen Kontext entstanden sind, wird es im folgenden gehen. Beliebige vollständige Verbände können aber auch als Begriffsverbände behandelt werden. In [G-W 96] ist der folgende Satz als Teil des Hauptsatzes über Begriffsverbände formuliert.

Satz 8 (*Vollständige Verbände als Begriffsverbände*)

Ein vollständiger Verband $(V, \leq)$ ist isomorph zu $\underline{\mathfrak{B}}(V, V, \leq)$.

([G-W 96], 1.1, Satz 3, S. 20)

Beweis: Definiere

$\varphi : V \to \underline{\mathfrak{B}}(V, V, \leq), \varphi(x) = (\{y \in V \mid y \leq x\}, \{y \in V \mid x \leq y\})$.

φ ist wohldefiniert, denn in $\underline{\mathfrak{B}}(V, V, \leq)$ gilt:

$\{y \in V \mid y \leq x\}^{\uparrow} = \{z \in V \mid y \leq z \ \ \forall y \in V : y \leq x\} = \{z \in V \mid x \leq z\}$

$\{y \in V \mid x \leq y\}^{\downarrow} = \{z \in V \mid z \leq y \ \ \forall y \in V : x \leq y\} = \{z \in V \mid z \leq x\}$

Also sind die Bilder unter φ wirklich formale Begriffe aus $\underline{\mathfrak{B}}(V, V, \leq)$.

Weiter ist φ ordnungserhaltend, denn aus $x \leq z$ in V folgt

$\{y \in V \mid y \leq x\} \subseteq \{y \in V \mid y \leq z\}$, d.h. $\varphi(x) \leq \varphi(z)$ in $\underline{\mathfrak{B}}(V, V, \leq)$.

Definiere weiter $\psi : \underline{\mathfrak{B}}(V, V, \leq) \to V, \psi((A, B)) = \bigvee_V A$.

ψ ist ebenfalls ordnungserhaltend, für $A \subseteq C$ in V gilt $\bigvee_V A \leq \bigvee_V C$.

Zeige: $\psi = \varphi^{-1}$.

Für $x \in V$ gilt: $\psi(\varphi(x)) = \psi((\{y \in V \mid y \leq x\}, \{y \in V \mid x \leq y\})) = x$.

Für $(A, B) \in \underline{\mathfrak{B}}(V, V, \leq)$ ist

$B = A^{\uparrow} = \{z \in V \mid y \leq z \ \ \forall y \in A\} = \{z \in V \mid \bigvee_V A \leq z\}$ und

$A = B^{\downarrow} = \{z \in V \mid \bigvee_V A \leq z\}^{\downarrow} = \{z \in V \mid z \leq \bigvee_V A\}$.

Also ist

$$\varphi(\psi((A, B))) = \varphi\left(\bigvee_V A\right)$$

$$= (\{z \in V \mid z \leq \bigvee_V A\}, \{z \in V \mid \bigvee_V A \leq z\}) = (A, B)$$

Damit ist φ ein Ordnungsisomorphismus zwischen $(V,\leq)$ und $\underline{\mathfrak{B}}(V,V,\leq)$.

Mit (G, M, I) ist auch (M, G, I^{-1}) ein formaler Kontext. Dieser Kontext führt gerade zum dualen Begriffsverband.

Satz 9 (*Dualer Begriffsverband*)

Ist (G, M, I) ein formaler Kontext, so gilt $\underline{\mathfrak{B}}(M, G, I^{-1}) \cong \underline{\mathfrak{B}}(G, M, I)^d$ und $\varphi : \underline{\mathfrak{B}}(M, G, I^{-1}) \to \underline{\mathfrak{B}}(G, M, I)^d, \varphi((B, A)) = (A, B)$ ist ein Isomorphismus.

([G-W 96], 1.1, S. 22)

Beweis: Es gilt für $A \subseteq G$ und $B \subseteq M$:

$$(B, A) \in \mathfrak{B}(M, G, I^{-1}) \quad \Leftrightarrow \quad B^{I^{-1}} = A \quad \text{und} \quad A^{I^{-1}} = B$$
$$\Leftrightarrow \{g \in G \mid m\, I^{-1}\, g \ \forall m \in B\} = A \quad \text{und} \quad \{m \in M \mid m I^{-1} g \ \forall g \in A\} = B$$
$$\Leftrightarrow \{g \in G \mid g\, I\, m \ \forall m \in B\} = A \quad \text{und} \quad \{m \in M \mid g I m \ \forall g \in A\} = B$$
$$\Leftrightarrow B^I = A \quad \text{und} \quad A^I = B \quad \Leftrightarrow \quad (A, B) \in \mathfrak{B}(G, M, I)$$

Also ist φ wohldefiniert und bijektiv.
Weiter gilt für $(A, B), (C, D) \in \mathfrak{B}(G, M, I)$

$$(B, A) \leq_{\underline{\mathfrak{B}}(M, G, I^{-1})} (D, C) \Leftrightarrow B \subseteq D$$
$$\Leftrightarrow (C, D) \leq_{\underline{\mathfrak{B}}(G, M, I)} (A, B) \Leftrightarrow (A, B) \leq_{\underline{\mathfrak{B}}(G, M, I)^d} (C, D)$$
$$\Leftrightarrow \varphi((B, A)) \leq_{\underline{\mathfrak{B}}(G, M, I)^d} \varphi((D, C))$$

Also ist φ ein Ordnungsisomorphismus und $\underline{\mathfrak{B}}(M, G, I^{-1}) \cong \underline{\mathfrak{B}}(G, M, I)^d$.

Damit kann man die Rollen von formalen Gegenständen und formalen Merkmalen vertauschen. Nach einer solchen Vertauschung betrachtet man lediglich den dualen Verband. Ein zugehöriges Liniendiagramm stellt sich dabei einfach auf den Kopf.

3.3.4 Liniendiagramme von Begriffsverbänden

Die spezielle Struktur der Begriffsverbände erlaubt eine reduzierte Beschriftung der Knoten in einem Liniendiagramm. Ausgehend von Definition 10 auf Seite 71 sind für alle Knoten die vollständige Extension und Intension der entsprechenden formalen Begriffe anzugeben. Das wird schnell sehr unübersichtlich, wie schon ein kleines Beispiel zu Satz 8 zeigt.

Beispiel 6 (*Liniendiagramme zu* $(V,\leq)$ *und* $\underline{\mathfrak{B}}(V,V,\leq)$)

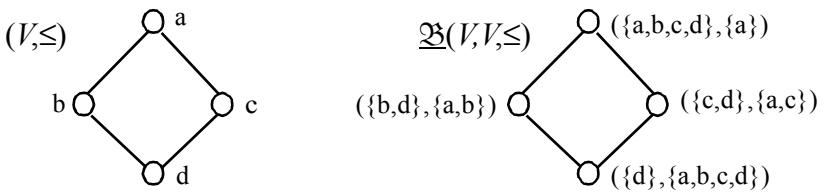


Abb. 3.6 Ein endlicher Verband als Begriffsverband

$(V, \leq)$ wird als Beispiel eines endlichen Verbands dem Begriffsverband $\underline{\mathfrak{B}}(V, V, \leq)$ gegenüber gestellt. Satz 8 besagt, daß diese beiden Verbände isomorph sind, was man im Beispiel gut an den Liniendiagrammen erkennen kann.

In einem Liniendiagramm eines Begriffsverbands reicht es aber auch aus, formale

Gegenstände nur an dem untersten Knoten, zu dessen formalen Begriff sie gehören, anzugeben und dual formale Merkmale nur am obersten Knoten, dessen formalem Begriff sie zukommen, zu notieren. Diese reduzierte Beschriftung wird nun vorbereitet.

Definition 23 (*Gegenstands- und Merkmalsbegriff*)
Sind (G, M, I) ein formaler Kontext und $g \in G$ und $m \in M$, so schreibt man $g' := \{g\}', g'' := \{g\}''$ und $m' := \{m\}', m'' := \{m\}''$.
$\gamma g := (g'', g')$ heißt dann der *Gegenstandsbegriff* zu g und
$\mu m := (m', m'')$ der *Merkmalsbegriff* zu m.
([G-W 96], 1.1, Definition 22, S. 23)

Nach Hilfssatz 5 auf Seite 83 sind γg und μm formale Begriffe des Kontexts (G, M, I).

Folgende Charakterisierung von Gegenstands- und Merkmalsbegriffen liefert die Begründung für die angesprochene reduzierte Schreibweise in Liniendiagrammen von Begriffsverbänden.

Folgerung 4 (*Charakterisierung von Gegenstands-/Merkmalsbegriffen*)
Sind (G, M, I) ein formaler Kontext und $g \in G$ und $m \in M$, so gilt

$$\gamma g = \bigwedge\{(A, B) \in \underline{\mathfrak{B}}(G, M, I) \mid g \in A\}$$

$$\mu m = \bigvee\{(A, B) \in \underline{\mathfrak{B}}(G, M, I) \mid m \in B\}$$

Beweis: Ist $(A, B) \in \underline{\mathfrak{B}}(G, M, I)$ mit $g \in A$, so gilt nach Folgerung 3 auf Seite 83 $g'' \subseteq A'' = A$, also $\gamma g = (g'', g') \leq (A, B)$.
Außerdem ist $g \in g''$, also $\bigwedge\{(A, B) \in \underline{\mathfrak{B}}(G, M, I) \mid g \in A\} \leq \gamma g$.
Damit folgt die Gleichheit.
Für μm dual.

γg ist für einen formalen Gegenstand g also der kleinste Begriff, der g in seinem Umfang führt, und dual μm der größte Begriff, der m in seinem Inhalt enthält. Bei der Beschriftung des Liniendiagramms trägt man g demnach nur am zu γg gehörenden Knoten an, alle mit aufsteigenden Linienzügen erreichbaren formalen Begriffe enthalten ebenfalls g in ihren Umfängen. Dual notiert man m nur am zu μm gehörenden Knoten an, alle mit absteigenden Linienzügen erreichbaren formalen Begriffe enthalten auch m in ihren Inhalten. Man nennt dies eine *reduzierte Beschriftung* im Gegensatz zu der in Beispiel 6 benutzten vollständigen Notation. Jeder formale Begriff ist als Infimum von Merkmalsbegriffen oder als Supremum von Gegenstandsbegriffen darstellbar.

Folgerung 5 (*Infima von Merkmals- / Suprema von Gegenstandsbegriffen*)
Sind (G, M, I) ein formaler Kontext und $A \subseteq G$, $B \subseteq M$, so gilt
$(A'', A') = \bigvee_{g \in A} \gamma g$ und $(B', B'') = \bigwedge_{m \in B} \mu m$.

Beweis: Nach Folgerung 2.a) auf Seite 82 ist $A' = \left(\bigcup_{g \in A} \{g\} \right)' = \bigcap_{g \in A} g'$.
$\bigcap_{g \in A} g'$ ist nach Satz 7 auf Seite 84 gerade der Begriffsinhalt von $\bigvee_{g \in A} \gamma g$.
Also ist $(A'', A') = \bigvee_{g \in A} \gamma g$.
Für (B', B'') dual.

So reicht bei Liniendiagrammen von Begriffsverbänden die reduzierte Beschriftung aus. Bei der reduzierten Beschriftung werden die Diagrammknoten der Atome des Begriffsverbands immer mit formalen Gegenständen und die Knoten zu den Coatomen immer mit formalen Merkmalen beschriftet.

Folgerung 6 (*Beschriftung für Atome und Coatome*)
Ist (G, M, I) ein formaler Kontext, so sind alle Atome von $\underline{\mathfrak{B}}(G, M, I)$ Gegenstandsbegriffe und alle Coatome Merkmalsbegriffe.

Beweis: Sei (A, B) ein Atom von $\underline{\mathfrak{B}}(G, M, I)$.
Nach Definition 21 auf Seite 84 und Folgerung 5 ist
$\mathbf{0}_{\underline{\mathfrak{B}}(G, M, I)} = (M', M'') = \bigvee_{g \in M'} \gamma g$.
Mit Folgerung 5 und Hilfssatz 2.c) auf Seite 68 folgt also

$$(A, B) = \bigvee_{g \in A} \gamma g = \bigvee_{g \in M'} \gamma g \vee \bigvee_{g \in A \setminus M'} \gamma g = \mathbf{0}_{\underline{\mathfrak{B}}(G, M, I)} \vee \bigvee_{g \in A \setminus M'} \gamma g$$
$$= \bigvee_{g \in A \setminus M'} \gamma g$$

Wegen $(A, B) > \mathbf{0}_{\underline{\mathfrak{B}}(G, M, I)}$ ist $A \setminus M' \neq \emptyset$.
Für $g \in A \setminus M'$ gilt $\mathbf{0}_{\underline{\mathfrak{B}}(G, M, I)} < \gamma g \leq (A, B)$, also $\gamma g = (A, B)$.
Im Fall der Coatome führt die duale Argumentation zur Behauptung.

Das folgende Beispiel 7 zeigt die reduzierte Liniendiagrammbeschriftung.

Die Knoten des Liniendiagramms stellen die Begriffe des Begriffsverbands dar. Die Ober-/Unterbegriffsordnung ist dergestalt wiedergegeben, daß vom Knoten eines Unterbegriffs zu den Knoten seiner Oberbegriffe aufsteigende Linienzüge führen. Gegenstände sind an dem untersten Knoten notiert, unter dessen Begriff sie fallen, und Merkmale an dem obersten Knoten, dessen Begriff sie kennzeichnen. Dabei werden Gegenstände etwas unterhalb und Merkmale etwas oberhalb der Knoten angegeben. Zur besseren Kennzeichnung sind hier die Gegenstände außerdem fett gedruckt.

Beispiel 7 (*Liniendiagramm mit reduzierter Beschriftung*)

Angegeben ist ein Liniendiagramm zum Kontext aus Beispiel 5 auf Seite 80.

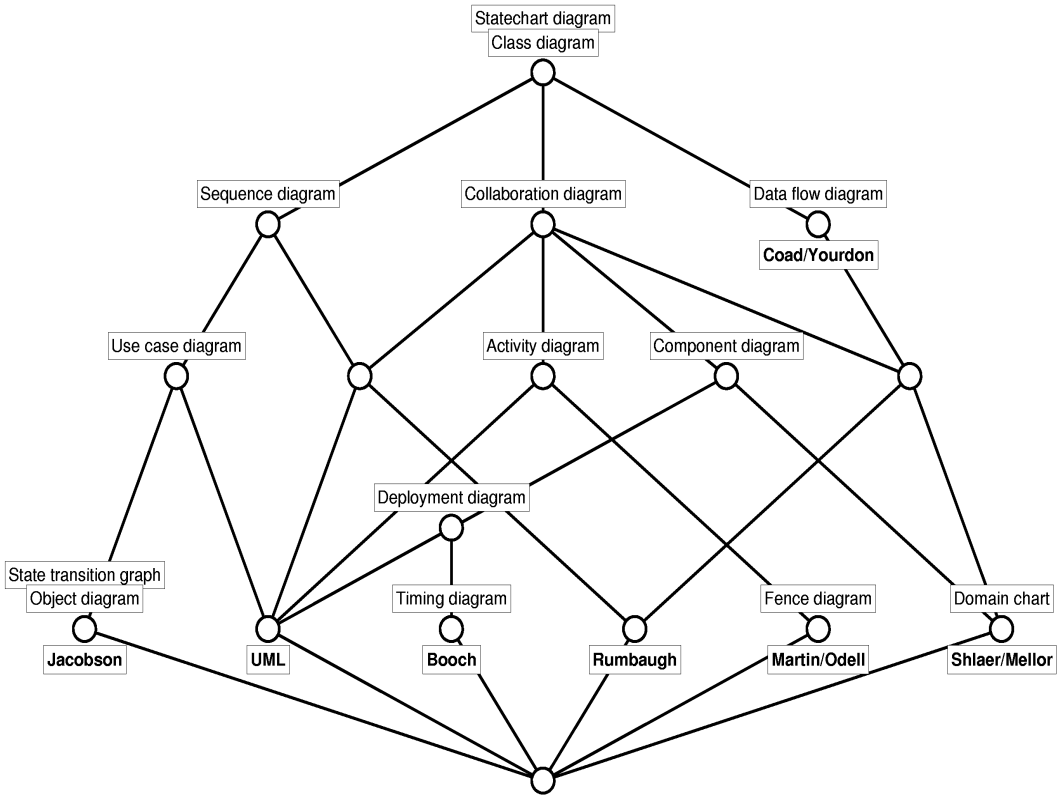


Abb. 3.7 Liniendiagramm zu Beispiel 5

Das Liniendiagramm enthält die volle Information des formalen Kontexts.

Folgerung 7 (*Formaler Kontext* $\leftrightarrow$ *Liniendiagramm*)

Sind (G, M, I) ein formaler Kontext und $g \in G$ und $m \in M$, so gilt

$$g\,I\,m \;\Leftrightarrow\; \gamma g \leq \mu m \quad .$$

Beweis: Gilt $g\,I\,m$, so ist $g \in m'$.

Nach Folgerung 4 ist also $\gamma g \leq (m', m'') = \mu m$.

Ist dagegen $\gamma g \leq \mu m$, so ist nach Folgerung 2.c) auf Seite 82 $g \in g'' \subseteq m'$, d.h. $g\,I\,m$.

Sucht man zu einem Gegenstand die zutreffenden Merkmale, verfolgt man vom Knoten, an dem der Gegenstand notiert ist, alle aufsteigende Linienzüge. Die an ihnen auftauchenden Merkmale sind die auf den Gegenstand zutreffenden. Genau umgekehrt findet man durch die absteigenden Linienzüge zu einem Merkmal alle Gegenstände, die es tragen.

So steht in Beispiel 7 der mit *Use case diagram* beschriftete Knoten für den forma-

len Begriff ({*UML, Jacobson*},{*Use case diagram, Sequence diagram, Class diagram, Statechart diagram*}). Dieser ist z.B. ein Oberbegriff von ({*UML*},{*Use case diagram, Sequence diagram, Class diagram, Statechart diagram, Activity diagram, Collaboration diagram, Deployment diagram, Component diagram*}) (mit *UML* bezeichneter Knoten), d.h. $\mu(\textit{Use case diagram}) \geq \gamma(\textit{UML})$.

Liniendiagramm und formaler Kontext sind also Repräsentationen derselben Sachverhalte. Aber die Liniendiagrammstruktur erhellt einige Dinge deutlicher als der formale Kontext. Die exponierte Stellung von *Statechart diagram* und *Class diagram* am Supremum des ganzen Verbands macht auf einen Blick deutlich, daß alle betrachteten Methoden Klassendiagramme zur Modellierung statischer und Zustandsdiagramme zur Beschreibung dynamischer Aspekte benutzen. Weiter suggerieren die Coatome eine Gegenüberstellung der Methoden, die zur Ablaufbeschreibung Sequenzdiagramme, Kollaborationsdiagramme oder Datenflußdiagramme benutzen. Die direkte Zuordnung der Methode von Coad/Yourdon zu den Datenflußdiagrammen zeigt an, daß diese Autoren (als einzige) keine der anderen beiden objektbezogenen Darstellungsmittel zur Interaktionsmodellierung vorsehen (da alle unteren Nachbarn des betreffenden Knotens auch untere Nachbarn eines anderen Coatoms sind.

Ebenfalls leicht abzulesen ist, daß bis auf drei Ausnahmen die UML alle Modellierungskonstrukte enthält, die einer ihrer drei Autoren propagiert. In der Tat sind nur Diagrammtypen, die in den einzelnen Methoden lediglich als Zusatz zu den eigentlich primären Modellierungskonstrukten betrachtet wurden oder die sich in der Anwendung der jeweiligen Methode als inadäquat erwiesen hatten, der Notationsvereinheitlichung zum Opfer gefallen. So verzichtete Jacobson auf seine Objektdiagramme, die eigentlich frühe Klassendiagramme waren (UML erlaubt sogar auch die Darstellung von Objekten in Klassendiagrammen.), und auf die *state transition graphs*, eine Erweiterung der Zustandsdiagramme. Rumbaugh war schon vor der Entwicklung der UML immer mehr auf Distanz zu den Datenflußdiagrammen gegangen (siehe 2.4.3 auf Seite 44). Die *timing diagrams* von Booch ([Boo 91]) fanden schon in [Boo 94] keine Erwähnung mehr. Bis auf diese kleineren Abweichungen ist UML eine Vereinigung der von den drei Autoren auch schon vorher vertretenen Modellierungskonstrukte ergänzt um die von Martin/Odell übernommenen Aktivitätsdiagramme. Dieser Sachverhalt ist dem Diagramm zu entnehmen, indem man vom Gegenstandsbegriff zu *UML* aus die aufsteigenden Linienzüge betrachtet. Alle auftretenden formalen Merkmale – hier Beschreibungsmittel – bis auf *Activity diagram* sind über absteigende Linienzüge auch einem der drei formalen Gegenstände – Autoren – *Jacobson*, *Booch* oder *Rumbaugh* zugeordnet.

Weiter sieht man auch den spezifischen Einfluß der einzelnen Autoren. Alle drei Paare von *UML* mit einem der drei Autoren kommt als Begriffsumfang vor. {*UML,*

Jacobson} ist Umfang des mit *Use case diagram* beschrifteten Begriffs. Jacobsons spezifischer Beitrag zur UML ist eben gerade die Anwendungsfallbetrachtung. Analog ist durch das Liniendiagramm dokumentiert, daß Booch die Einsatz- und Komponentendiagramme eingebracht hat. Denn das Supremum der Gegenstandsbegriffe zu *UML* und *Booch* ist der Merkmalsbegriff zu *Deployment diagram* und weiter ist auch *Component diagram* im Inhalt dieses Begriffs enthalten, ohne einem der beiden anderen Autoren der UML zugeordnet zu sein. Das von Rumbaugh übernommene Spezifikum ist die gleichzeitige Betrachtung von Sequenz- und Kollaborationsdiagrammen. Inhalt des Begriffs mit Umfang {*UML, Rumbaugh*} ist dementsprechend {*Sequence diagram, Collaboration diagram*}.

Diese Ausführungen zeigen, wie das Liniendiagramm des Begriffsverbands interpretiert werden kann und so Anlaß zu weiterführenden Überlegungen gibt, indem Zusammenhänge in ihrer Struktur dargestellt werden und so die Untersuchung von inhaltlichen Fragestellungen unterstützt wird.

3.4 Implikationen

Mit dem formalen Kontext und dem Liniendiagramm des entsprechenden Begriffsverbands stehen bereits zwei verschiedene Repräsentationen des untersuchten Sachverhalts zur Verfügung. Mit der Betrachtung von Implikationen kommt noch eine dritte interessante Darstellung hinzu.

Zunächst können Implikationen aber unabhängig von einem formalen Kontext für beliebige Mengen erklärt werden. Bernhard Ganter und Rudolf Wille führen in [G-W 96] Implikationen auch erst allgemein ein, betrachten aber dann nur Merkmalsimplikationen eines formalen Kontexts. Die im folgenden weitgehend zitierten Vincent Duquenne und J.L. Guigues, die mit [G-D 86] die grundlegende Arbeit für die Behandlung von Implikationen eines formalen Kontexts veröffentlicht haben, betrachten von vorn herein nur Merkmalsimplikationen. Dies bedeutet aber keine Einschränkung, weil sich auf einer beliebigen Menge definierte Implikationen als Merkmalsimplikationen eines geeigneten Kontexts behandeln lassen (siehe Bemerkung 7 auf Seite 105).

Innerhalb von BASE werden nicht nur Implikationen zwischen formalen Merkmalen, sondern insbesondere Implikationen zwischen formalen Gegenständen untersucht. Diese ergeben sich aber vollkommen dual. Die sonst übliche Einschränkung auf die Betrachtung von Merkmalsimplikationen rührt wohl der in der Umgangssprache üblichen Gebrauch von "Begriffen", "Gegenständen" und "Merkmalen" her. Bernhard Ganter schreibt in [Gan 00]: "Während die Hierarchie von Begriffen auf der Seite der Gegenstände eher mittels der Teilmengenbeziehung formuliert

wird (“alle Säugetiere sind Wirbeltiere”), drückt man einen entsprechenden Sachverhalt merkmalseitig eher als Implikation aus (“Übergewicht bedeutet Infarktgefahr”)."

Da zur Erklärung des mathematischen Implikationenmodells der Bezug zu formalen Kontexten unnötig ist und deshalb das Wesen dieses Modells durch die Beschränkung auf eine Mengendarstellung besser herausgestellt wird, ist hier der Nachteil in Kauf genommen, daß für die Behandlung auf beliebigen Mengen definierter Implikationen zusätzlichen Bezeichnungen eingeführt werden müssen. Die dargestellten Sachverhalte sind dafür sofort auf Merkmals- und Gegenstandsimplikationen anwendbar, ohne immer auf die Dualität der beiden hinweisen zu müssen.

Die zentrale Aussage des folgenden ist, daß es zu den betrachteten Implikationenfamilien kanonische Basen gibt, d.h. Teilmengen, aus denen alle Implikationen der Gesamtfamilie ableitbar sind, die aber selber keine Redundanz enthalten (siehe Satz 11 auf Seite 100 und Satz 14 auf Seite 109). Solche Basen ermöglichen durch die gelieferte Beschränkung der Anzahl zu betrachtender Implikationen erst die praktische Durchführung entsprechender Analysen.

3.4.1 Implikationen auf Mengen

Sofort geht es hier mit der Definition von Implikationen in medias res.

Definition 24 (*Implikation, respektieren, gelten*)

Sei X eine Menge.

Dann nennt man $(V, W) \in \mathfrak{P}(X)^2$ eine *Implikation* auf X und schreibt $V \to W$.

V heißt die *Prämisse* der Implikation und W die *Konklusion*.

Man sagt, eine Teilmenge $R \subseteq X$ *respektiert*, die Implikation $V \to W$, wenn $V \not\subseteq R$ oder $W \subseteq R$, d.h. $(V \subseteq R) \Rightarrow (W \subseteq R)$.

Weiter *respektiert* $R \subseteq X$ eine Menge $\mathfrak{I} \subseteq \mathfrak{P}(X)^2$ von Implikationen, wenn R jede Implikation aus $\mathfrak{I}$ respektiert.

Eine Implikation auf X *gilt* (ist *gültig*) in einem System $\mathfrak{R} \subseteq \mathfrak{P}(X)$, wenn jedes $R \in \mathfrak{R}$ die Implikation respektiert.

Eine Implikationenfamilie $\mathfrak{I} \subseteq \mathfrak{P}(X)^2$ *gilt* in einem System $\mathfrak{R} \subseteq \mathfrak{P}(X)$, wenn jedes $R \in \mathfrak{R}$ die Implikationenmenge $\mathfrak{I}$ respektiert.

([G-W 96], 2.3, Definition 36, S. 80)

Das Konzept der respektierenden Mengen ist folgendes:
In einem gegebenen Kontext (hier umgangssprachlich gemeint, aber innerhalb der Formalen Begriffsanalyse durch einen formalen Kontext beschrieben) treten immer nur gewisse Kombinationen von Elementen aus der Menge X auf. Die auftre-

tenden Kombinationen sind die respektierenden Mengen aus $\mathfrak{R}$. Dann impliziert in diesem Kontext eine Teilmenge V eine andere W, wenn in allen möglichen Kombinationen aus $\mathfrak{R}$, die V enthalten, auch schon alle Elemente von W vorkommen. Man kann in diesem Fall allein durch die Beobachtung von V auf das Vorliegen von W schließen.

Betrachtet man als X eine Menge von aussagenlogischen Variablen und $V, W, R \subseteq X$, so respektiert R die Implikation $V \to W$ genau dann, wenn $\chi_R : X \to \{T, F\}$ ein Modell für die aussagenlogische Formel ($\neg V$ oder W) ist. Die aussagenlogische Untersuchung solcher Implikationen (insbesondere eine aussagenlogische Formulierung von Satz 14 auf Seite 109) findet man in [GKr 99].

Im Bereich der relationalen Datenbanken werden Implikationen als funktionale Abhängigkeiten zwischen Attributen eines Relationsschemas betrachtet. Eine Implikation $V \to W$ bedeutet in diesem Zusammenhang, daß die Wertebelegung der Attribute aus W schon durch die Belegung der Attribute aus V bestimmt ist. In dieser Weise kann man semantische Bedingungen an die Ausprägungen eines Relationsschemas fordern (Siehe [Vos 99]). Respektierende Mengen werden dabei als abgeschlossen bzgl. der entsprechenden Implikationen bezeichnet ([BDFS 84]).

Beispiel 8 (*Implikationen auf {a,b,c}*)

Betrachte $X = \{a, b, c\}$ und $\mathfrak{R} = \{\{a\}, \{a, b\}, \{a, c\}\}$. Dann sind die in $\mathfrak{R}$ gültigen Implikationen Abbildung 3.8 zu entnehmen (Prämissen in Zeilen, Konklusionen in Spalten).

→	∅	{a}	{b}	{c}	{a,b}	{a,c}	{b,c}	{a,b,c}
∅	×	×						
{a}	×	×						
{b}	×	×	×		×			
{c}	×	×		×		×		
{a,b}	×	×	×		×			
{a,c}	×	×		×		×		
{b,c}	×	×	×	×	×	×	×	×
{a,b,c}	×	×	×	×	×	×	×	×

Abb. 3.8 Implikationen auf {a,b,c}

In Definition 24 ist keine Einschränkung an die Menge X gemacht. Aber selbst wenn man X als endlich annimmt und $n \in \mathbb{N}$ die Elementanzahl von X ist, umfaßt $\mathfrak{P}(X)^2$ 2^{2^n} Implikationen, von denen viele uninteressant sind. Einige dieser uninteressanten Exemplare charakterisiert der folgende Hilfssatz.

Hilfssatz 6 (*Triviale Implikationen*)

Ist X eine Menge und $V, W \subseteq X$ mit $W \subseteq V$ dann gilt $V \to W$ in $\mathfrak{P}(X)$.
([G-W 96], 2.3, S. 81)

Beweis: Ist $R \in \mathfrak{P}(X)$ und $V \subseteq R$, so ist auch $W \subseteq V \subseteq R$.

Eine Implikation $V \to W$, bei der die Prämisse die Konklusion umfaßt, gilt also in jedem denkbaren Mengensystem $\mathfrak{R} \subseteq \mathfrak{P}(X)$. Interessant sind nur Implikationen, deren Konklusion gegenüber der Prämisse auch etwas Neues bringt.

Definition 25 (*Informative Implikation*)
Ist X eine Menge, so heißt eine Implikation $V \to W$ auf X *informativ*, wenn $W \not\subseteq V$.
([G-D 86], S. 9)

Beispiel 9 (*Informative Implikationen aus Beispiel 8*)
In Beispiel 8 sind gültig und informativ:
$\emptyset \to \{a\}, b \to a, b \to \{a, b\}, c \to a, c \to \{a, c\},$
$\{b, c\} \to a, \{b, c\} \to \{a, b\}, \{b, c\} \to \{a, c\}, \{b, c\} \to \{a, b, c\}$

Einige Implikationen können aus anderen abgeleitet werden (z.B. transitiv siehe Hilfssatz 8 auf Seite 95). Eine solche "logische Ableitung" kann wie folgt formalisiert werden.

Definition 26 (*Semantisch folgen, vollständige Implikationenfamilie*)
Sei X eine Menge, $V \to W$, $Y \to Z$ Implikationen auf X und $\mathfrak{I}, \mathfrak{K} \subseteq \mathfrak{P}(X)^2$ Implikationenfamilien.
Dann *folgt* $V \to W$ (*semantisch*) aus $\mathfrak{I}$, wenn jede Menge aus $\mathfrak{P}(X)$, die $\mathfrak{I}$ respektiert, auch $V \to W$ respektiert.
Abkürzend sagt man, daß $V \to W$ aus $Y \to Z$ folgt, wenn $V \to W$ aus $\{Y \to Z\}$ folgt.
$\mathfrak{K}$ folgt aus $\mathfrak{I}$, wenn jede Implikation in $\mathfrak{K}$ aus $\mathfrak{I}$ folgt.
$\mathfrak{I}$ heißt für ein System $\mathfrak{R} \subseteq \mathfrak{P}(X)$ *vollständig*, wenn $\mathfrak{I}$ in $\mathfrak{R}$ gilt und jede Implikation, die in $\mathfrak{R}$ gilt, aus $\mathfrak{I}$ folgt.
(Vergl. [G-W 96], 2.3, Definition 37, S. 81)

Für die oben angesprochenen funktionalen Abhängigkeiten in relationalen Datenbanken etabliert [H-F 85] die entsprechende Charakterisierung des Begriffs des semantischen Folgens.

Die definierte Relation des semantischen Folgens verhält sich entsprechend der üblichen Regeln der Logik, sie ist insbesondere transitiv.

Hilfssatz 7 (*Transitivität von "semantisch folgen"*)
Sei X eine Menge.
Dann ist die durch
$$\mathfrak{I}\ T\ \mathfrak{K} \ :\Leftrightarrow\ \mathfrak{I} \text{ folgt aus } \mathfrak{K}$$
definierte binäre Relation auf $\mathfrak{P}(\mathfrak{P}(X)^2)$ transitiv.

Beweis: Seien $\mathfrak{I}, \mathfrak{K}, \mathfrak{L} \subseteq \mathfrak{P}(X)^2$ Implikationenfamilien auf X und $\mathfrak{K}$ folge aus $\mathfrak{I}$ und $\mathfrak{L}$ aus $\mathfrak{K}$.
Respektiert $R \in \mathfrak{P}(X)$ die Implikationenfamilie $\mathfrak{I}$, so auch alle Implikationen in $\mathfrak{K}$ und damit auch $\mathfrak{K}$ im Ganzen.
Weiter respektiert dann R auch alle Implikationen in $\mathfrak{L}$.
D.h. $\mathfrak{L}$ folgt aus $\mathfrak{I}$.

Genauso kann man – wie schon oben angedeutet – auch auf X selber transitiv schließen.

Hilfssatz 8 (*Transitiv gefolgerte Implikationen*)
Sei X eine Menge und $U, V, W \subseteq X$.
Aus $\{U \to V, V \to W\}$ folgt dann die Implikation $U \to W$.
(Vergl. [Duq 87], Rule 0, S. 222)

Beweis: $R \in \mathfrak{P}(X)$ respektiere die Implikationen $U \to V$ und $V \to W$.
Ist dann $U \subseteq R$, so nach $U \to V$ auch $V \subseteq R$ und damit nach $V \to W$ sofort $W \subseteq R$.
D.h. R respektiert $U \to W$.

Eine besonders einfache Methode, aus einer gültigen Implikation andere ebenfalls gültige Implikationen abzuleiten, zeigt Hilfssatz 9.

Hilfssatz 9 (*Schwächere Implikationen*)
Sei X eine Menge und $V \to W$ eine Implikation auf X.
Sind $Y, Z \subseteq X$ mit $V \subseteq Y$ und $Z \subseteq W$, dann folgt $Y \to Z$ aus $V \to W$.
(Vergl. [G-W 96], 2.3, S. 82)

Beweis: Nach Voraussetzung gelten die Implikationen $Y \to V$ und $W \to Z$ nach Hilfssatz 6 in $\mathfrak{P}(X)$.
Nach Hilfssatz 8 folgt damit die Implikation $Y \to W$ und dann auch $Y \to Z$.

Im Sinne von Hilfssatz 9 schwächere Implikationen folgen aus den entsprechenden stärkeren. Um die Menge der betrachteten Implikationen zu verkleinern, könnte man also die schwächeren Implikationen weglassen und nur Implikationen betrachten, deren Prämisse nicht verkleinert und deren Konklusion nicht vergrößert werden kann, ohne die Gültigkeit der Implikation zu verlieren.

Hilfssatz 10 (*Maximale Konklusion*)
Sei X eine Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Ist $V \subseteq X$, so gilt mit $k_{\mathfrak{R}}(V) := \bigcap \{R \in \mathfrak{R} \mid V \subseteq R\}$ in $\mathfrak{R}$ die Implikation $V \to W$ genau dann, wenn $W \subseteq k_{\mathfrak{R}}(V)$.
(Dabei ist der Schnitt über eine leere Menge von respektierenden Mengen

ganz X.)
(Vergl. [G-D 86], S. 10)

Beweis: Für jedes $R \in \mathfrak{R}$ mit $V \subseteq R$ gilt natürlich $k_{\mathfrak{R}}(V) \subseteq R$, weil R an der betreffenden Durchschnittsbildung beteiligt ist.
D.h. $V \to k_{\mathfrak{R}}(V)$.
Nach Hilfssatz 9 gilt also $V \to W$ in $\mathfrak{R}$ für alle $W \subseteq k_{\mathfrak{R}}(V)$.
Sei nun $V \to W$ eine beliebige in $\mathfrak{R}$ gültige Konklusion.
Dann gilt $W \subseteq R$ für alle $R \in \mathfrak{R}$ mit $V \subseteq R$.
Also ist auch $W \subseteq k_{\mathfrak{R}}(V)$.

Entsprechend Hilfssatz 10 definiert man:

Definition 27 (*Maximale Konklusion*)

Ist X eine Menge, $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen und $V \subseteq X$, so heißt $k_{\mathfrak{R}}(V) := \bigcap\{R \in \mathfrak{R} \mid V \subseteq R\}$ die *maximale Konklusion* von V in $\mathfrak{R}$.
(Vergl. [G-D 86], S. 9)

Es gibt also für jedes $V \in \mathfrak{P}(X)$ eine Implikation mit maximaler Konklusion. So ist die Anzahl der zu betrachtenden Implikationen bei einer n-elementigen Menge X schon einmal auf 2^n reduziert, weil nach Hilfssatz 10 und Hilfssatz 9 alle anderen gültigen Implikationen aus solchen mit maximaler Konklusion folgen. In Beispiel 8 auf Seite 93 sind die Implikationen mit maximaler Konklusion gerade durch die in den Zeilen jeweils am weitesten rechts stehenden Kreuze gegeben.

Die maximalen Konklusionen bilden ein Hüllensystem.

Hilfssatz 11 (*Hüllensystem der maximalen Konklusionen*)

Ist X eine Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen, so ist $k_{\mathfrak{R}} : \mathfrak{P}(X) \to \mathfrak{P}(X)$ ein Hüllenoperator auf X.
(Vergl. [G-W 96], 1.1, S.19)

Beweis: Seien $V, W \subseteq X$ mit $V \subseteq W$.
Ist dann $R \in \mathfrak{R}$ mit $W \subseteq R$, so ist auch $V \subseteq R$.
D.h. $k_{\mathfrak{R}}(V) = \bigcap\{R \in \mathfrak{R} \mid V \subseteq R\} \subseteq \bigcap\{R \in \mathfrak{R} \mid W \subseteq R\} = k_{\mathfrak{R}}(W)$.
$k_{\mathfrak{R}}$ ist also monoton.
Für $V \subseteq X$ ist $V \subseteq k_{\mathfrak{R}}(V)$ nach Konstruktion und somit $k_{\mathfrak{R}}$ extensiv.
Damit folgt für $V \subseteq X$ auch $k_{\mathfrak{R}}(V) \subseteq k_{\mathfrak{R}}(k_{\mathfrak{R}}(V))$.
Ist aber umgekehrt $R \in \mathfrak{R}$ mit $V \subseteq R$, so gilt
$k_{\mathfrak{R}}(V) = \bigcap\{S \in \mathfrak{R} \mid V \subseteq S\} \subseteq R$, weil R selbst an der Schnittbildung beteiligt ist.
Es folgt
$k_{\mathfrak{R}}(k_{\mathfrak{R}}(V)) = \bigcap\{S \in \mathfrak{R} \mid k_{\mathfrak{R}}(V) \subseteq S\} \subseteq \bigcap\{R \in \mathfrak{R} \mid V \subseteq R\} = k_{\mathfrak{R}}(V)$.

D.h. $k_{\mathfrak{R}}(k_{\mathfrak{R}}(V)) = k_{\mathfrak{R}}(V)$ und $k_{\mathfrak{R}}$ ist idempotent.
Insgesamt ist $k_{\mathfrak{R}}$ ein Hüllenoperator.

Die maximalen Konklusionen bilden die Gesamtheit der Mengen, die alle gültigen Implikationen respektieren.

Satz 10 (*Maximale Konklusionen = respektierende Mengen*)
Sei X eine Menge, $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen und $\mathfrak{I} \subseteq \mathfrak{P}(X)^2$ die Menge der in $\mathfrak{R}$ gültigen Implikationen auf X.
Dann sind die maximalen Konklusionen $k_{\mathfrak{R}}(V)$ mit $V \subseteq X$ genau die Teilmengen von X, die $\mathfrak{I}$ respektieren.
(Vergl. [G-W 96], 2.3, Hilfssatz 20, S. 80)

Beweis: Die Implikation $V \to W$ gelte in $\mathfrak{R}$.
Ist $U \subseteq X$ und $V \subseteq k_{\mathfrak{R}}(U)$, so gilt $V \subseteq R$ für alle $R \in \mathfrak{R}$ mit $U \subseteq R$.
Nach Voraussetzung ist also auch $W \subseteq R$ für alle $R \in \mathfrak{R}$ mit $U \subseteq R$.
D.h. $W \subseteq \bigcap \{R \in \mathfrak{R} \mid U \subseteq R\} = k_{\mathfrak{R}}(U)$.
Die maximalen Konklusionen respektieren also $\mathfrak{I}$.
Respektiert andersherum $R \subseteq X$ die Implikationenfamilie $\mathfrak{I}$, so ist nach Hilfssatz 10 $(R \to k_{\mathfrak{R}}(R)) \in \mathfrak{I}$.
Weil R $\mathfrak{I}$ respektiert, folgt also $k_{\mathfrak{R}}(R) \subseteq R$.
Nach Hilfssatz 11 ist damit $R = k_{\mathfrak{R}}(R)$ eine maximale Konklusion.

Maximale Konklusionen können als Prämissen informativer Implikationen ausgeschlossen werden.

Folgerung 8 (*Prämissen informativer Konklusionen*)
Sei X eine Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Ist die Implikation $V \to W$ auf X informativ und gilt in $\mathfrak{R}$, so ist $V \subsetneqq k_{\mathfrak{R}}(V)$.
Eine Implikation $V \to k_{\mathfrak{R}}(V)$ ist genau dann informativ, wenn $V \neq k_{\mathfrak{R}}(V)$.

Beweis: (Vergl. [G-D 86], S. 9 und Proposition 1, S. 10)
Sei also $V \to W$ informativ und gelte in $\mathfrak{R}$.
Nach Konstruktion gilt $V \subseteq k_{\mathfrak{R}}(V)$ für jede Menge V.
Zu zeigen ist nur $V \neq k_{\mathfrak{R}}(V)$.
Nach Hilfssatz 10 ist $W \subseteq k_{\mathfrak{R}}(V)$.
Laut Voraussetzung gilt weiter $W \not\subseteq V$.
Also gibt es ein $x \in W \backslash \mathrm{V} \subseteq k_{\mathfrak{R}}(V) \backslash \mathrm{V}$.
D.h. $V \subsetneqq k_{\mathfrak{R}}(V)$.
Ist andererseits $V \neq k_{\mathfrak{R}}(V)$, so folgt $k_{\mathfrak{R}}(V) \not\subseteq V$ wegen $V \subseteq k_{\mathfrak{R}}(V)$.
D.h. $V \to k_{\mathfrak{R}}(V)$ ist informativ.

Um eine gültige Implikationenfamilie festzulegen, reicht es aus, informative Implikationen mit maximaler Konklusion anzugeben.

Folgerung 9 (*Vollständigkeit der Implikationen mit maximaler Konklusion*)
Ist X eine Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen, so ist die Menge der informativen Implikationen mit in $\mathfrak{R}$ maximaler Konklusion $\{V \to k_{\mathfrak{R}}(V) \mid V \subseteq X \wedge k_{\mathfrak{R}}(V) \neq V\}$ vollständig für $\mathfrak{R}$.
Genauer folgt jede in $\mathfrak{R}$ gültige Implikation aus einer informativen Implikation mit in $\mathfrak{R}$ maximaler Konklusion.

Beweis: Nach Hilfssatz 10 auf Seite 95 gelten alle Implikationen der Form $V \to k_{\mathfrak{R}}(V)$ in $\mathfrak{R}$.
Sei $V \to W$ eine beliebige in $\mathfrak{R}$ gültige Implikation.
Ist $V \to W$ nicht informativ, so folgt $V \to W$ nach Hilfssatz 6 auf Seite 93 aus jeder Implikationenfamilie.
Sei nun also $V \to W$ informativ.
Nach Hilfssatz 10 ist $W \subseteq k_{\mathfrak{R}}(V)$.
Nach Hilfssatz 9 auf Seite 95 folgt daher $V \to W$ aus $V \to k_{\mathfrak{R}}(V)$.
Wäre $k_{\mathfrak{R}}(V) \subseteq V$, so auch $W \subseteq k_{\mathfrak{R}}(V) \subseteq V$, also $V \to W$ nicht informativ.
Also ist auch $V \to k_{\mathfrak{R}}(V)$ informativ.
Damit folgt $V \to W$ aus einer informativen Implikation mit in $\mathfrak{R}$ maximaler Konklusion.

Beispiel 10 (*Informative Implikationen mit maximaler Konklusion*)
In Beispiel 8 auf Seite 93 sind folgende Implikationen informativ und besitzen jeweils eine maximale Konklusion:

$$\emptyset \to \{a\},\ b \to \{a, b\},\ c \to \{a, c\},\ \{b, c\} \to \{a, b, c\}$$

Nachdem nun die Konklusionen maximiert wurden, liegt es nach Hilfssatz 9 auf Seite 95 nahe, jetzt auch die Prämissen zu minimieren. Das ist zumindest möglich, wenn X endlich ist. Unter dieser Voraussetzung ist die entstehende Implikationenfamilie dann immer noch vollständig. (Die Minimierung der Prämissen läßt sich aber weniger griffig beschreiben.) In der so konstruierbaren Implikationenfamilie folgen dann keine Implikationen als schwächere (Hilfssatz 9) oder transitiv (Hilfssatz 8 auf Seite 95) aus anderen. Trotzdem lassen sich in der Regel einige Implikationen doch noch aus anderen folgern (vergl. [G-D 86]). So folgen in Beispiel 10 alle Implikationen (deren Prämissen schon minimal sind) aus $\emptyset \to \{a\}$.

Dieser Umstand ist unabhängig von diesem Beispiel der folgenden Schlußregel zuzuschreiben:

Hilfssatz 12 (*Beidseitig erweiterte Implikationen*)

Sei X eine Menge, T eine beliebige Indexmenge und $V_t, W_t \subseteq X$ für alle $t \in T$.

Dann folgt $\bigcup_{t \in T} V_t \to \bigcup_{t \in T} W_t$ aus $\{V_t \to W_t \mid t \in T\}$.

(Vergl. [G-W 96], 2.3, S. 82)

Beweis: $R \in \mathfrak{P}(X)$ respektiere $\{V_t \to W_t \mid t \in T\}$.

Ist dann $\bigcup_{t \in T} V_t \subseteq R$, so ist $V_t \subseteq R$ für alle $t \in T$.

Nach Voraussetzung ist dann $W_t \subseteq R$ für alle $t \in T$, also auch $\bigcup_{t \in T} W_t \subseteq R$.

D.h. R respektiert $\bigcup_{t \in T} V_t \to \bigcup_{t \in T} W_t$.

Interessiert ist man, minimale Implikationenfamilien zu betrachten, in denen keine Implikation schon aus den anderen gefolgert werden kann.

Definition 28 (*Redundante Implikationenfamilie, Basis*)

Sei X eine Menge, $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen und $\mathfrak{I} \subseteq \mathfrak{P}(X)^2$ eine Implikationenfamilie auf X.

$\mathfrak{I}$ heißt *redundant*, wenn es eine Implikation $V \to W \in \mathfrak{I}$ gibt, so daß $V \to W$ aus $\mathfrak{I} \setminus \{V \to W\}$ folgt.

Dagegen heißt $\mathfrak{I}$ eine *Basis* der in $\mathfrak{R}$ gültigen Implikationen, wenn $\mathfrak{I}$ für $\mathfrak{R}$ vollständig und nicht redundant ist.

(Vergl. [G-D 86], S. 12 und [Duq 87], S. 223)

Um eine Basis zu bekommen, werden im folgenden nur bestimmte Prämissen betrachtet. Das Ziel ist, Redundanzen aufgrund von Hilfssatz 12 zu vermeiden. Dazu wählt man nur solche informativen Implikationen mit maximaler Konklusion aus, bei denen echte Teilmengen der Prämisse, die selber wieder Prämissen der zu konstruierenden Basis sind, nichts über die Gesamtprämisse hinausgehendes implizieren. Schon aus dieser Paraphrasierung des Konstruktionsprinzips für eine Basis ist die rekursive Anlage der Basiskonstruktion ersichtlich. Ein leichter nachzuvollziehendes konstruktives Prinzip wird im Anschluß an Definition 29 vorgestellt. Wegen der Möglichkeit der einfacheren und vor allem eleganteren algorithmischen Bestimmung einer Implikationenbasis wird hier Definition 29 vorgezogen.

Definition 29 (*Saturierte Prämisse*)

Sei X eine **endliche** Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.

Dann heißt $V \subseteq X$ *saturierte Prämisse* auf X, wenn $V \neq k_{\mathfrak{R}}(V)$ und für jede saturierte Prämisse $U \subsetneq V$ schon $k_{\mathfrak{R}}(U) \subseteq V$ gilt.

(Vergl. [G-W 96], 2.3, Definition 40, S. 83)

Damit die Rekursion für jede betrachtete Menge abbricht, muß die betrachtete Menge endlich sein.

Guigues und Duquenne definieren saturierte Prämissen in folgender Weise:
Zu $V \subseteq X$ sei $V^\circ := V \cup \bigcup \{k_{\mathfrak{R}}(U) \mid U \subseteq V, k_{\mathfrak{R}}(U) \neq k_{\mathfrak{R}}(V)\}$ und
$p_{\mathfrak{R}}(V) := \bigcup_{n \in \mathbb{N}_0} V_n$ mit $V_0 := V, V_{n+1} := V_n^\circ$ für $n \in \mathbb{N}_0$.
Saturierte Prämissen nach Guigues und Duquenne sind dann solche, die $V = p_{\mathfrak{R}}(V)$ erfüllen.
(Vergl. [G-D 86], S.12/13 und [Duq 87], S. 223)
Das erlaubt die Definition auch im Fall unendlicher Mengen X. Doch für die Beschreibung einer Implikationenbasis machen auch sie die Endlichkeitsvoraussetzung. Die saturierten Prämissen nach Definition 29 sind auch nach der Definition von Guigues und Duquenne saturiert. Mit zusätzlichen Einschränkungen an die Prämissen erhalten diese beiden Autoren die gleiche Implikationenbasis wie die im folgenden Satz beschriebene (siehe Anhang A.2 auf Seite 256). Die Definition von Ganter und Wille erlaubt den eleganteren Beweis für Satz 11. Außerdem können die saturierten Prämissen berechnet werden, in dem man in der Konstruktion von V° nur solche Teilmengen U betrachtet, die selber saturierte Prämissen sind.

Nimmt man die Implikationen mit saturierter Prämisse und maximaler Konklusion, so bekommt man eine Basis der Implikationen auf einer endlichen Menge.

Satz 11 (*Duquenne-Guigues-Basis*)

Sei X eine **endliche** Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Dann ist

$$\mathfrak{DQ}_{\mathfrak{R}}(X) := \{V \to k_{\mathfrak{R}}(V) \mid V \subseteq X \text{ ist saturierte Prämisse}\}$$

eine Basis der in $\mathfrak{R}$ gültigen Implikationen.

Beweis: (Vergl. [G-W 96], 2.3, Satz 8, S. 83)

Nach Hilfssatz 10 auf Seite 95 sind alle Implikationen $V \to k_{\mathfrak{R}}(V)$ mit $V \subseteq X$ in $\mathfrak{R}$ gültig.

Zeige: $\mathfrak{DQ}_{\mathfrak{R}}(X)$ ist vollständig.

Sei dazu $R \subseteq X$, so daß $\mathfrak{DQ}_{\mathfrak{R}}(X)$ von R respektiert wird.

Zeige: R ist eine maximale Konklusion.

Annahme: $k_{\mathfrak{R}}(R) \neq R$.

Ist dann U eine saturierte Prämisse mit $U \subsetneq R$, so gilt $k_{\mathfrak{R}}(U) \subseteq R$, denn R respektiert $U \to k_{\mathfrak{R}}(U)$.

D.h. R ist selber eine saturierte Prämisse.

R respektiert aber $R \to k_{\mathfrak{R}}(R)$ nach Annahme nicht – im Widerspruch zur Wahl von R.

Also ist $R = k_{\mathfrak{R}}(R)$ eine maximale Konklusion.

$\mathfrak{DQ}_{\mathfrak{R}}(X)$ wird somit nur von maximalen Konklusionen respektiert.

Nach Satz 10 auf Seite 97 folgen damit alle in $\mathfrak{R}$ gültigen Implikationen auf X aus $\mathfrak{DQ}_{\mathfrak{R}}(X)$.

Zeige: $\mathfrak{DQ}_{\mathfrak{R}}(X)$ ist nicht redundant.
Sei also V eine saturierte Prämisse.
Dann ist $V \neq k_{\mathfrak{R}}(V)$ und deshalb respektiert V die Implikation $V \to k_{\mathfrak{R}}(V)$ nicht.
Ist aber U eine saturierte Prämisse mit $U \subsetneq V$, so ist $k_{\mathfrak{R}}(U) \subseteq V$.
Also wird $\mathfrak{DQ}_{\mathfrak{R}}(X) \setminus \{V \to k_{\mathfrak{R}}(V)\}$ von V respektiert.
Damit folgt $V \to k_{\mathfrak{R}}(V)$ nicht aus $\mathfrak{DQ}_{\mathfrak{R}}(X) \setminus \{V \to k_{\mathfrak{R}}(V)\}$.
Da V beliebig war, ist $\mathfrak{DQ}_{\mathfrak{R}}(X)$ nicht redundant.
Insgesamt ist $\mathfrak{DQ}_{\mathfrak{R}}(X)$ eine Basis der in $\mathfrak{R}$ gültigen Implikationen.

Satz 11 berechtigt zu der folgenden Definition.

Definition 30 (*Duquenne-Guigues-Basis*)
Ist X eine endliche Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen, so heißt $\mathfrak{DQ}_{\mathfrak{R}}(X)$ aus Satz 11 *Duquenne-Guigues-Basis* oder *Stammbasis* der in $\mathfrak{R}$ gültigen Implikationen.
(Vergl. [G-W 96], 2.3, S. 84)

Die Betrachtung einer solchen Basis bedeutet eine einschneidende Einschränkung der Anzahl der untersuchten Implikationen.

Beispiel 11 (*Duquenne-Guigues-Basis*)
In Beispiel 8 auf Seite 93 besteht die Duquenne-Guigues-Basis aus der Implikation $\varnothing \to \{a\}$. Alle anderen informativen Implikationen mit maximaler Konklusion besitzen keine saturierte Prämisse, denn ihre Prämissen enthalten $k_{\mathfrak{R}}(\varnothing) = \{a\}$ nicht.
Alle 36 gültigen Implikationen folgen aus $\varnothing \to \{a\}$, denn alle sind von der Form $V \to W$ oder $V \to W \cup \{a\}$ mit $W \subseteq V$.

Die Überprüfung der enthaltenen saturierten Prämissen entfällt natürlich bei der Betrachtung der leeren Menge.

Bemerkung 6 (*Leere Prämisse*)
Sei X eine Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Dann ist $k_{\mathfrak{R}}(\varnothing) = \bigcap \mathfrak{R}$, also $\varnothing \to k_{\mathfrak{R}}(\varnothing)$ genau dann informativ, wenn ein $x \in X$ existiert mit $x \in R \quad \forall R \in \mathfrak{R}$.
In diesem Fall ist $\varnothing \to k_{\mathfrak{R}}(\varnothing)$ Element von $\mathfrak{DQ}_{\mathfrak{R}}(X)$, wenn X endlich ist.
(Vergl. [G-W 96], 2.3, Definition 38, S. 82)

Saturierte Prämissen stehen in engem Zusammenhang mit den maximalen Konklusionen.

Hilfssatz 13 (*Schnitt von saturierten Prämissen und maximalen Konklusionen*)
Sei X eine endliche Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Sind $V, W \subseteq X$ saturierte Prämisse oder maximale Konklusion und gilt $V \not\subseteq W$ und $W \not\subseteq V$, dann ist $V \cap W$ eine maximale Konklusion.

Beweis: (Vergl. [G-W 96], 2.3, Hilfssatz 24, S. 84)
Nach Hilfssatz 10 auf Seite 95 und Satz 10 auf Seite 97 respektieren maximale Konklusionen $\mathfrak{DQ}_{\mathfrak{R}}(X)$.
Nach dem Beweis von Satz 11 respektieren saturierte Prämissen V, W immerhin $\mathfrak{DQ}_{\mathfrak{R}}(X) \setminus \{V \to k_{\mathfrak{R}}(V)\}$ bzw. $\mathfrak{DQ}_{\mathfrak{R}}(X) \setminus \{W \to k_{\mathfrak{R}}(W)\}$.
$\mathfrak{DQ}_{\mathfrak{R}}(X) \setminus \{V \to k_{\mathfrak{R}}(V), W \to k_{\mathfrak{R}}(W)\}$ wird somit in jedem Fall von V und W respektiert und somit auch von $V \cap W$.
Wegen $V \not\subseteq W$ und $W \not\subseteq V$ gilt $V, W \not\subseteq V \cap W$.
Deshalb respektiert $V \cap W$ auch $\{V \to k_{\mathfrak{R}}(W), W \to k_{\mathfrak{R}}(W)\}$, d.h. ganz $\mathfrak{DQ}_{\mathfrak{R}}(X)$.
Nach Satz 10 ist $V \cap W$ eine maximale Konklusion.

Vincent Duquenne bezeichnet die Duquenne-Guigues-Basis als kanonische saturierte Basis ([Duq 87], S. 225). In welchem Sinn sie kanonisch ist, erläutert der folgende Satz.

Satz 12 (*Kanonische saturierte Basis*)
Sei X eine endliche Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Ist eine Implikationenfamilie $\mathfrak{I} \subseteq \mathfrak{P}(X)^2$ für $\mathfrak{R}$ vollständig, so gibt es zu jeder Implikation $V \to k_{\mathfrak{R}}(V)$ aus $\mathfrak{DQ}_{\mathfrak{R}}(X)$ eine Implikation $(U \to W) \in \mathfrak{I}$ mit $U \subseteq V$ und $k_{\mathfrak{R}}(U) = k_{\mathfrak{R}}(V)$.
Haben alle Implikationen aus $\mathfrak{I}$ eine saturierte Prämisse, so gilt sogar $\mathfrak{DQ}_{\mathfrak{R}}(X) \subseteq \mathfrak{I}$.

Beweis: (Vergl. [G-W 96], 2.3, Hilfssatz 25, S. 84)
Sei also V eine saturierte Prämisse.
Dann gilt $V \to k_{\mathfrak{R}}(V)$ nach Hilfssatz 10 auf Seite 95, folgt also nach Voraussetzung aus $\mathfrak{I}$.
Wegen $V \neq k_{\mathfrak{R}}(V)$ respektiert V die Implikation $V \to k_{\mathfrak{R}}(V)$ nicht.
Damit kann V auch nicht $\mathfrak{I}$ respektieren, weil $\mathfrak{I}$ vollständig ist.
D.h. es gibt $(U \to W) \in \mathfrak{I}$ mit $U \subseteq V$ und $W \not\subseteq V$.
Nach Hilfssatz 10 ist $W \subseteq k_{\mathfrak{R}}(U)$, also $k_{\mathfrak{R}}(U) \not\subseteq V$.
Betrachte $k_{\mathfrak{R}}(U) \cap V$.
Es gilt $U \subseteq k_{\mathfrak{R}}(U) \cap V$ und $W \not\subseteq k_{\mathfrak{R}}(U) \cap V$.
$k_{\mathfrak{R}}(U) \cap V$ respektiert also $U \to W$ nicht.
Nach Satz 10 auf Seite 97 ist $k_{\mathfrak{R}}(U) \cap V$ demnach keine maximale Konklusion.

Aus Hilfssatz 13 folgt deshalb $V \subseteq k_{\mathfrak{R}}(U)$.
Nach Hilfssatz 11 auf Seite 96 gilt damit $k_{\mathfrak{R}}(V) \subseteq k_{\mathfrak{R}}(k_{\mathfrak{R}}(U)) = k_{\mathfrak{R}}(U)$.
Wegen $U \subseteq V$ ist nach Hilfssatz 11 $k_{\mathfrak{R}}(U) \subseteq k_{\mathfrak{R}}(V)$, also $k_{\mathfrak{R}}(V) = k_{\mathfrak{R}}(U)$.
Ist U eine saturierte Prämisse, gilt sogar $U = V$, weil aus $U \subsetneqq V$ sonst $k_{\mathfrak{R}}(U) \subseteq V \subsetneqq k_{\mathfrak{R}}(V)$ folgen würde.

Damit ist insbesondere eine in der Kardinalität minimale Basis gefunden.

Folgerung 10 (*Minimale Kardinalität von Implikationenbasen*)
Sei X eine endliche Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Dann hat jede für $\mathfrak{R}$ vollständige Implikationenfamilie $\mathfrak{I} \subseteq \mathfrak{P}(X)^2$ mindestens die Kardinalität von $\mathfrak{DQ}_{\mathfrak{R}}(X)$.
(Vergl. [G-W 96], 2.3, S. 84)

Beweis: Sei also $\mathfrak{I} \subseteq \mathfrak{P}(X)^2$ eine für $\mathfrak{R}$ vollständige Implikationenfamilie.
Nach Satz 12 gibt es für jede Implikation $(V \to k_{\mathfrak{R}}(V)) \in \mathfrak{DQ}_{\mathfrak{R}}(X)$ eine Implikation $(U_V \to W_V) \in \mathfrak{I}$ mit $U_V \subseteq V$ und $k_{\mathfrak{R}}(U_V) = k_{\mathfrak{R}}(V)$.
Seien $(V_1 \to k_{\mathfrak{R}}(V_1)), (V_2 \to k_{\mathfrak{R}}(V_2)) \in \mathfrak{DQ}_{\mathfrak{R}}(X)$ mit $V_1 \neq V_2$.
Annahme: $(U_{V_1} \to W_{V_1}) = (U_{V_2} \to W_{V_2})$.
Dann ist $U_{V_1} = U_{V_2} \subseteq V_1 \cap V_2$ und
$k_{\mathfrak{R}}(V_1) = k_{\mathfrak{R}}(U_{V_1}) = k_{\mathfrak{R}}(U_{V_2}) = k_{\mathfrak{R}}(V_2)$.
Da V_1 und V_2 saturierte Prämissen sind, sind sie unvergleichbar, denn bei $V_1 \subsetneqq V_2$ wäre $k_{\mathfrak{R}}(V_1) \subseteq V_2 \subsetneqq k_{\mathfrak{R}}(V_2)$.
Nach dem Beweis von Hilfssatz 13 respektiert also $V_1 \cap V_2$ ganz $\mathfrak{DQ}_{\mathfrak{R}}(X)$.
Die Implikation $U_{V_1} \to k_{\mathfrak{R}}(U_{V_1})$ wird von $V_1 \cap V_2$ nicht respektiert, folgt aber nach Hilfssatz 10 auf Seite 95 und Satz 11 auf Seite 100 aus $\mathfrak{DQ}_{\mathfrak{R}}(X)$.
Dies ist ein Widerspruch.
Also ist $U_{V_1} \neq U_{V_2}$, womit auch die betrachteten Implikationen verschieden sind.
Daraus folgt, daß $\mathfrak{I}$ mindestens die Kardinalität von $\mathfrak{DQ}_{\mathfrak{R}}(X)$ hat.

In diesem Sinne ist also $\mathfrak{DQ}_{\mathfrak{R}}(X)$ minimal.
Bei der Angabe von Implikationen reicht es natürlich, nur den "informativen Teil" der Konklusion anzugeben.

Folgerung 11 (*Darstellung der Duquenne-Guigues-Basis*)
Sei X eine endliche Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Dann ist $\mathfrak{IDQ}_{\mathfrak{R}}(X) := \{V \to k_{\mathfrak{R}}(V) \setminus V \mid (V \to k_{\mathfrak{R}}(V)) \in \mathfrak{DQ}_{\mathfrak{R}}(X)\}$
eine Basis der in $\mathfrak{R}$ gültigen Implikationen.
(Vergl. [G-W 96], 2.3, S. 84)

Beweis: Nach Hilfssatz 9 auf Seite 95 folgt $V \to k_{\mathfrak{R}}(V) \setminus V$ aus $V \to k_{\mathfrak{R}}(V)$.
Also gilt $\mathfrak{ID}\mathfrak{Q}_{\mathfrak{R}}(X)$ nach Satz 11 auf Seite 100 in $\mathfrak{R}$.
Laut Hilfssatz 12 auf Seite 99 folgt $V \to k_{\mathfrak{R}}(V)$ aus $V \to k_{\mathfrak{R}}(V) \setminus V$ und $V \to V$, also nach Hilfssatz 6 auf Seite 93 aus $V \to k_{\mathfrak{R}}(V) \setminus V$ allein.
Nach Hilfssatz 8 auf Seite 95 ist damit $\mathfrak{ID}\mathfrak{Q}_{\mathfrak{R}}(X)$ vollständig.
Sei $(V \to k_{\mathfrak{R}}(V) \setminus V) \in \mathfrak{ID}\mathfrak{Q}_{\mathfrak{R}}(X)$ beliebig.
Würde $V \to k_{\mathfrak{R}}(V) \setminus V$ aus $\mathfrak{ID}\mathfrak{Q}_{\mathfrak{R}}(X) \setminus \{V \to k_{\mathfrak{R}}(V) \setminus V\}$ folgen, so folgte $V \to k_{\mathfrak{R}}(V)$ aus $\mathfrak{D}\mathfrak{Q}_{\mathfrak{R}}(X) \setminus \{V \to k_{\mathfrak{R}}(V)\}$, denn aus $\mathfrak{D}\mathfrak{Q}_{\mathfrak{R}}(X) \setminus \{V \to k_{\mathfrak{R}}(V)\}$ folgt nach Hilfssatz 9 $\mathfrak{ID}\mathfrak{Q}_{\mathfrak{R}}(X) \setminus \{V \to k_{\mathfrak{R}}(V) \setminus V\}$, daraus dann $V \to k_{\mathfrak{R}}(V) \setminus V$ und daraus wiederum $V \to k_{\mathfrak{R}}(V)$ nach oben.
$\mathfrak{D}\mathfrak{Q}_{\mathfrak{R}}(X)$ ist aber nach Satz 11 nicht redundant.
Somit ist auch $\mathfrak{ID}\mathfrak{Q}_{\mathfrak{R}}(X)$ nicht redundant, also insgesamt eine Basis.

Um die Lesbarkeit der Implikationen zu erhöhen, kann man sich auf einelementige Konklusionen beschränken.

Folgerung 12 (*Einfache Repräsentation der Duquenne-Guigues-Basis*)
Sei X eine endliche Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Dann ist
$$\mathfrak{EDQ}_{\mathfrak{R}}(X) := \{V \to \{x\} \mid (V \to k_{\mathfrak{R}}(V)) \in \mathfrak{DQ}_{\mathfrak{R}}((X), x \in k_{\mathfrak{R}}(V) \setminus V)\}$$
für $\mathfrak{R}$ vollständig.

Beweis: Für $(V \to k_{\mathfrak{R}}(V)) \in \mathfrak{DQ}_{\mathfrak{R}}(X)$ und $x \in k_{\mathfrak{R}}(V) \setminus V$ folgt $\{V \to \{x\} \mid x \in k_{\mathfrak{R}}(V) \setminus V\}$ nach Hilfssatz 9 auf Seite 95 aus $V \to k_{\mathfrak{R}}(V)$.
Also gilt $\mathfrak{EDQ}_{\mathfrak{R}}(X)$ nach Satz 11 auf Seite 100 in $\mathfrak{R}$.
Sei $(V \to k_{\mathfrak{R}}(V)) \in \mathfrak{DQ}_{\mathfrak{R}}(X)$.
Nach Hilfssatz 12 auf Seite 99 folgt $V \to k_{\mathfrak{R}}(V) \setminus V$ aus $\{V \to \{x\} \mid x \in k_{\mathfrak{R}}(V) \setminus V\}$.
Also folgt $\mathfrak{ID}\mathfrak{Q}_{\mathfrak{R}}(X)$ aus $\mathfrak{EDQ}_{\mathfrak{R}}(X)$.
Damit ist nach Folgerung 11 und Hilfssatz 7 auf Seite 94 $\mathfrak{EDQ}_{\mathfrak{R}}(X)$ für $\mathfrak{R}$ vollständig.

Im allgemeinen erhält man durch den Übergang zu einelementigen Konklusionen aber Redundanz.

Beispiel 12 (*Einfache Repräsentation der Duquenne-Guigues-Basis*)
Sei $X = \{a, b, c, d\}$ und $\mathfrak{R} = \{\{a\}, \{c, d\}, \{a, b, c, d\}\}$.
Dann ist $\mathfrak{DQ}_{\mathfrak{R}}(X) = \{b \to \{a, b, c, d\}, c \to \{c, d\}, d \to \{c, d\}\}$ und
$\mathfrak{EDQ}_{\mathfrak{R}}(X) = \{b \to a, b \to c, b \to d, c \to d, d \to c\}$.
$\mathfrak{EDQ}_{\mathfrak{R}}(X)$ ist redundant, weil z.B. $b \to d$ aus $\{b \to c, c \to d\}$ folgt (Hilfssatz 8 auf Seite 95).

Natürlich kann man die Duquenne-Guigues-Basis leicht aus $\mathfrak{FDO}_{\mathfrak{R}}(X)$ gewinnen und umgekehrt.

3.4.2 Implikationen von formalen Kontexten

Üblicherweise betrachtet man zu einem formalen Kontext Implikationen zwischen Mengen formaler Merkmale, dual kann man aber auch solche zwischen Mengen formaler Gegenstände untersuchen. Folgende Definition legt die Sprechweisen für beide Fälle fest.

Definition 31 (*Merkmals-/Gegenstandsimplikation*)

Sei (G, M, I) ein formaler Kontext.
$B \to D$ mit $B, D \subseteq M$ heißt *Merkmalsimplikation* von (G, M, I) und dual $A \to C$ mit $A, C \subseteq G$ *Gegenstandsimplikation* von (G, M, I).
Eine Merkmalsimplikation $B \to D$ gelte in (G, M, I), wenn $B \to D$ in $\mathfrak{M} := \{g' \mid g \in G\}$ gilt.
Dual gelte eine Merkmalsimplikation $A \to C$ in (G, M, I), wenn $A \to C$ in $\mathfrak{G} := \{m' \mid m \in M\}$ gilt.
([G-W 96], 2.3, Definition 36, S. 80)

Wie eingangs erwähnt kann man sich auch auf die Betrachtung von etwa Merkmalsimplikationen beschränken, ohne die Allgemeinheit der bisherigen Ausführungen zu verlieren.

Bemerkung 7 (*Implikationen als Merkmalsimplikationen*)

Sei X eine Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Setzt man $G := \mathfrak{R}$, $M := X$ und $R\,I\,x := (x \in R)$, so hat man einen formalen Kontext (G, M, I), dessen gültige Merkmalsimplikationen gerade die in $\mathfrak{R}$ gültigen Implikationen auf X sind.

Die maximalen Konklusionen sind bei Implikationen eines formalen Kontexts gerade die Begriffsinhalte bzw. Begriffsumfänge.

Hilfssatz 14 (*Maximale Konklusionen*)

Sei (G, M, I) ein formaler Kontext und $A \subseteq G$, $B \subseteq M$.
Dann ist $k_{\mathfrak{M}}(B) = B''$ und $k_{\mathfrak{G}}(A) = A''$.

Beweis: Betrachte $B' = \bigcup_{g \in B'} \{g\}$.
Nach Folgerung 2.e) und a) auf Seite 82 ist
$k_{\mathfrak{M}}(B) = \bigcap\{g' \mid g \in G,\ B \subseteq g'\} = \bigcap\{g' \mid g \in B'\} = B''$.
Analog gilt $k_{\mathfrak{G}}(A) = A''$.

Intuitiv erwartet man, daß eine Merkmalsimplikation $B \to D$ eines formalen Kontexts (G, M, I) besagt, daß allen formalen Gegenständen aus G, die alle formalen Merkmale aus B tragen, auch alle formalen Merkmale aus D zukommen. Daß dies durch die Definition 31 schon ausgesagt wird, zeigt Satz 13 im Punkt c). Zudem bringt er mit der Eigenschaft b) das schon aus dem Fall auf einer beliebigen Menge erklärter Implikationen bekannte handliche Kriterium für die Gültigkeit einer Implikation.

Satz 13 (*Gültigkeit von Implikationen in einem formalen Kontext*)
Sei (G, M, I) ein formaler Kontext.
Für eine Merkmalsimplikation $B \to D$ (Gegenstandsimplikation $A \to C$) sind äquivalent:

a) $B \to D$ ($A \to C$) gilt in (G, M, I).

b) $D \subseteq B''$ ($C \subseteq A''$)

c) $B' \subseteq D'$ ($A' \subseteq C'$)

d) $B \to D$ ($A \to C$) gilt im System aller Begriffsinhalte (Begriffsumfänge) von (G, M, I).

([G-D 86], S. 9 und [G-W 96], 2.3, Hilfssatz 19, S. 80)

Beweis: Nach Hilfssatz 14 und Hilfssatz 10 auf Seite 95 sind a) und b) äquivalent.
b) $\Rightarrow$ c): Ist $D \subseteq B''$, so ist nach Folgerung 2.d) und b) auf Seite 82:
$B' = B''' \subseteq D'$.
c) $\Rightarrow$ b): Gilt $B' \subseteq D'$, so nach Folgerung 2.b) und c) auch $D \subseteq D'' \subseteq B''$.
Nach Hilfssatz 14 und Satz 10 auf Seite 97 sind a) und d) äquivalent.
Für Gegenstandsimplikationen genauso.

Die Kennzeichnung in Satz 13 b) zeigt nun sofort, daß die Prämisse informativer Merkmals-(Gegenstands-)Implikationen eines formalen Kontexts kein Begriffsinhalt(-umfang) sein kann.

Folgerung 13 (*Prämissen informativer Implikationen eines formalen Kontexts*)
Gilt eine informative Merkmalsimplikation $B \to D$ (Gegenstandsimplikation $A \to C$) in einem formalen Kontext (G, M, I), so ist $B \neq B''$ ($A \neq A''$).

Beweis: Direkt aus Hilfssatz 14 und Folgerung 8 auf Seite 97.

Die Begriffsinhalte (-umfänge) sind genau die Mengen sind, die alle gültigen Merkmalsimplikationen (Gegenstandsimplikationen) eines formalen Kontexts respektieren.

Folgerung 14 (*Begriffsinhalte (-umfänge)* $\leftrightarrow$ *Implikation*en)

Ist (G, M, I) ein formaler Kontext und $\mathfrak{I} \subseteq \mathfrak{P}(M)^2$ ($\mathfrak{L} \subseteq \mathfrak{P}(G)^2$) die Menge der gültigen Merkmalsimplikationen (Gegenstandsimplikationen) von (G, M, I).

Dann ist $\mathfrak{R} := \{R \subseteq \mathfrak{P}(M) \mid R \text{ respektiert } \mathfrak{I}\}$ gerade die Menge der Begriffsinhalte von (G, M, I).

($\{S \subseteq \mathfrak{P}(G) \mid S \text{ respektiert } \mathfrak{L}\}$ die Menge der Begriffsumfänge)

(Vergl. [G-W 96], 2.3, Hilfssatz 20, S. 80)

Beweis: Direkt aus Hilfssatz 14 und Satz 10 auf Seite 97.

Hat man also die Menge der gültigen Merkmalsimplikationen (Gegenstandsimplikationen), so kann man auch die Begriffsinhalte (-umfänge) – und damit bis auf Isomorphie den Begriffsverband – berechnen.

Andersherum kann man Implikationen aus dem Liniendiagramm eines Begriffsverbands ablesen. Nach Folgerung 12 auf Seite 104 reicht es aus, dies für Implikationen mit einelementiger Konklusion zu klären.

Folgerung 15 (*Ablesen von Implikationen im Liniendiagramm*)

Ist (G, M, I) ein formaler Kontext und $B \subseteq M$, $m \in M$ ($A \subseteq G$, $g \in G$).

Dann ist $B \to m$ ($A \to g$) genau dann gültig, wenn $\bigwedge_{n \in B} \mu n \leq \mu m$ ($\bigvee_{h \in A} \gamma h \geq \gamma g$).

(Vergl. [G-W 96], 2.3, S. 80)

Beweis: Nach Satz 13 gilt $B \to m$ genau dann, wenn $m \in B''$.

Nach Folgerung 5 auf Seite 88 ist $B' = \bigcap_{n \in B} n'$ der Begriffsumfang von $\bigwedge_{n \in B} \mu n$.

Also gilt

$$
\begin{aligned}
& m \in B'' \\
\Leftrightarrow\ & g\, I\, m \quad \forall g \in B' \\
\Leftrightarrow\ & g \in m' \quad \forall g \in B' \\
\Leftrightarrow\ & \bigcap_{n \in B} n' = B' \subseteq m' \\
\Leftrightarrow\ & \bigwedge_{n \in B} \mu n \leq (m', m'') = \mu m
\end{aligned}
$$

Das liefert die Behauptung.

Für $A \to g$ dual mit Argumentation über die entsprechenden Begriffsinhalte.

Demnach liest man aus einem Liniendiagramm eines Begriffsverbands eine Merk-

malsimplikation $B \to m$ (Gegenstandsimplikation $A \to g$) dadurch ab, daß der mit m (g) bezeichnete Begriff über dem Infimum (unter dem Supremum) aller mit einem Merkmal aus B (Gegenstand aus A) bezeichneten Begriffe liegt.

Beispiel 13 (*Beispiel einer Merkmalsimplikation*)

Das in Abbildung 3.7 auf Seite 89 angegebene Liniendiagramm zum formalen Kontext aus Abbildung 3.5 auf Seite 80 ist hier noch einmal wiedergegeben.

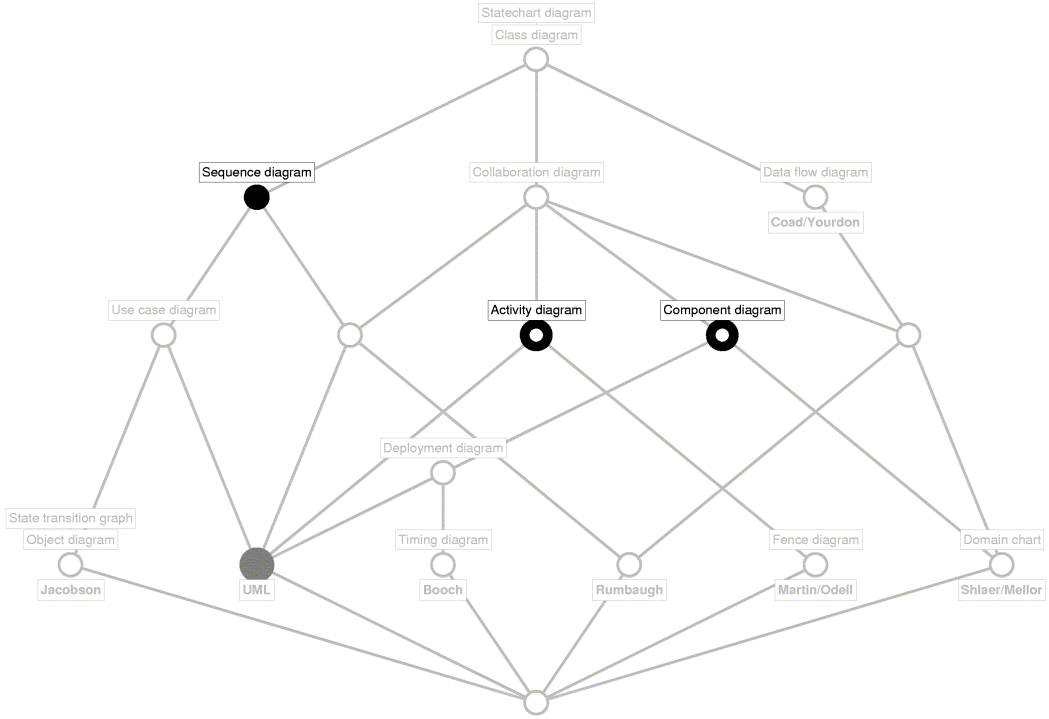


Abb. 3.9 Liniendiagramm mit Beispiel-Implikation

Durch die farblichen Hervorhebungen ist die Implikation
{*Activity diagram, Component diagram*} → {*Sequence diagram*}
dargestellt. Sie ist aus dem Diagramm abzulesen, weil Merkmalsbegriffe zu *Activity diagram* und *Component diagram* (Knoten mit dickem Rand) ein Infimum (großer dunkelgrauer Knoten) unterhalb des Merkmalsbegriffs zu *Sequence diagram* (schwarzer ausgefüllter Knoten) besitzen.

Für den Fall eines endlichen Kontexts läßt sich wie oben die Duquenne-Guigues-Basis konstruieren. Die saturierten Prämissen der Merkmalsimplikationen dieser Basis heißen in diesem Fall nach Bernhard Ganter *Pseudoinhalte*. Dementsprechend werden sie hier für Gegenstandsimplikationen *Pseudoumfänge* genannt.

Definition 32 (*Pseudoinhalt, Pseudoumfang*)

Sei (G, M, I) ein endlicher formaler Kontext.
Dann heißt $P \subseteq M$ *Pseudoinhalt* von (G, M, I), wenn $P \neq P''$ und für jeden Pseudoinhalt $Q \subsetneq P$ schon $Q'' \subseteq P$ gilt.
Analog heißt $P \subseteq G$ *Pseudoumfang* von (G, M, I), wenn $P \neq P''$ und für jeden Pseudoumfang $Q \subsetneq P$ schon $Q'' \subseteq P$ gilt.
(Vergl. [G-W 96], 2.3, Definition 40, S. 83)

Diese Definition entspricht nach Hilfssatz 14 auf Seite 105 genau Definition 29 auf Seite 99.

Die Implikationen mit Pseudoinhalten bzw. Pseudoumfängen als Prämisse und maximaler Konklusion ergeben die Duquenne-Guigues-Basis der Merkmals- bzw. Gegenstandsimplikationen.

Satz 14 (*Duquenne-Guigues-Basis für formale Kontexte*)

Sei (G, M, I) ein endlicher formaler Kontext.
Dann ist $\mathfrak{DQ}_{\mathfrak{G}}(M) = \{ P \to P'' \mid P$ ist Pseudoinhalt von $(G, M, I) \}$ und
$\mathfrak{DQ}_{\mathfrak{M}}(G) = \{ P \to P'' \mid P$ ist Pseudoumfang von $(G, M, I) \}$.
Diese Basis ist in ihrer Kardinalität minimal.

Beweis: ([G-W 96], 2.3, Satz 8, S. 83)
Folgt direkt aus Hilfssatz 14 auf Seite 105 und Folgerung 10 auf Seite 103.

Die Existenz solcher Basen erlaubt es, wenige Implikationen zu präsentieren, die die gesamte Merkmalslogik (Gegenstandsklassifikation) – und somit bis auf Isomorphie den Begriffsverband – wiedergeben, selber aber keine Redundanz enthalten. So können gezielt Überprüfungsfragen generiert werden, um die Gegenstand-Merkmal-Zuordnung des unterliegenden formalen Kontexts zu überprüfen.

Häufig angewandt wird diese Technik im Rahmen der Merkmalexploration, bei der die Klassifikation durch eine fest vorgegebene Menge von Merkmalen und ein Satz von Gegenständen als Beispiele für diese Klassifikation hergeleitet wird. Dazu beginnt man mit einem erst unvollständigen Satz von Gegenständen und deren Merkmalszuordnung. Die daraus generierten Implikationen kann der Analytiker aufgrund seines Fachwissens für alle möglichen Gegenstände bejahen oder durch ein Gegenbeispiel widerlegen. Die entsprechenden Gegenbeispiele führen zu einem vollständigen Beispielsatz für die Klassifikation nach den betrachteten Merkmalen. (Siehe [Bur 91], [G-W 96], [Gan 99])

Beispiel 14 (*Implikationenbasis*)

Zum formalen Kontext in Abbildung 3.5 auf Seite 80 (Beispiel 5) bzw. dem entsprechenden Begriffsverband in Abbildung 3.7 auf Seite 89 (Beispiel 7) besteht die Basis der Gegenstandsimplikationen aus Satz 14 aus den folgen-

den Implikationen:

1. {Martin/Odell,Shlaer/Mellor} → {UML,Booch,Rumbaugh}
2. {Martin/Odell,Rumbaugh} → {UML,Booch,Shlaer/Mellor}
3. {Jacobson,Shlaer/Mellor} → {UML,Booch,Coad/Yourdon,Martin/Odell,Rumbaugh}
4. {Jacobson,Rumbaugh} → {UML}
5. {Jacobson,Martin/Odell} → {UML,Booch,Coad/Yourdon,Rumbaugh,Shlaer/Mellor}
6. {Coad/Yourdon} → {Rumbaugh,Shlaer/Mellor}
7. {Booch,Shlaer/Mellor} → {UML}
8. {Booch,Rumbaugh} → {UML,Martin/Odell,Shlaer/Mellor}
9. {Booch,Martin/Odell} → {UML,Rumbaugh,Shlaer/Mellor}
10. {Booch,Jacobson} → {UML,Coad/Yourdon,Martin/Odell,Rumbaugh,Shlaer/Mellor}
11. {UML,Shlaer/Mellor} → {Booch}
12. {UML,Booch,Coad/Yourdon,Martin/Odell,Rumbaugh,Shlaer/Mellor} → {Jacobson}

Die Implikationen 4., 8., und 10. zeigen zum Beispiel, daß alle Beschreibungsmittel, die von mindestens zwei der Autoren der UML gleichzeitig in ihren eigenen Methoden betrachtet wurden, in die UML integriert worden sind, weil der formale Gegenstand *UML* jeweils in der Konklusion enthalten ist.

3.5 Blockrelationen

Die formalen Begriffe eines formalen Kontexts (G, M, I) gruppieren Gegenstände aus G und Merkmale aus M zu Einheiten. Zu den typischen Anwendungen der Formalen Begriffsanalyse gehört es, eine Klassifikation der untersuchten Gegenstände bezüglich der betrachteten Merkmale anhand dieser Einheiten zu gewinnen. Formale Begriffe als Klassifikationseinheiten haben die Eigenschaft, daß jeder zugehörige Gegenstand alle enthaltenen Merkmale trägt. Für verschiedene Anwendungen stellt sich diese Bedingung als zu restriktiv heraus.

Blockrelationen sind ein Mittel, diese Einschränkung abzumildern. Mit ihrer Hilfe erhält man Klassifikationseinheiten, in denen zu jeder Teilmenge von Gegenständen auch alle gemeinsamen Merkmale erfaßt werden, aber vielleicht zusätzlich noch andere Merkmale. Dual ist sicher gestellt, daß zu einer Kombination von Merkmalen alle Gegenstände, die alle diese Merkmale tragen, in der entsprechenden Einheit zusammengefaßt werden, aber wiederum eventuell auch noch zusätzliche Gegenstände. Darüber hinaus entstehen keine Zusammenstellungen von formalen Gegenständen oder formalen Merkmalen, die nicht schon auch als Begriffsumfänge oder Begriffsinhalte vorgekommen wären. In der Tat werden wieder formale Begriffe betrachtet, wobei jedoch eine andere Inzidenzrelation – eine Blockrelation – zugrunde gelegt wird.

Formale Begriffe kann man nicht nur im Liniendiagramm eines Begriffsverbands, sondern (in etwas mühsamerer Weise) auch im zugehörigen formalen Kontext ablesen. Sie entsprechen in der Kreuztabelle nach entsprechenden Zeilen- und Spaltenvertauschungen maximalen mit Kreuzen gefüllten Rechtecken.

Beispiel 15 (*Begriffe ≙ Maximale gefüllte Rechtecke*)

Der nebenstehende formale Kontext entstand aus der Untersuchung der von verschiedenen Autoren vorgeschlagenen Techniken für die Suche nach Klassenkandidaten (vergl. 2.6 ab Seite 48). Der Übersichtlichkeit wegen ist hier nur ein Teil der in 2.6 berücksichtigten Autoren erfaßt.

	Check-Listen	Grammatikalische Analyse	Bereichsanalyse	Anwendungsfall-getriebene Analyse
Booch	1 5 9		2 5 9	4 9
Coad/Yourdon	1 5		2 5	
Jacobson		3 8		4 8
Martin/Odell		3 7	2 7	
Rumbaugh	1 6	3 6		
Shlaer/Mellor	1			
Wirfs-Brock	1 6	3 6		

Abb. 3.10 Formale Begriffe in der Kreuztabelle

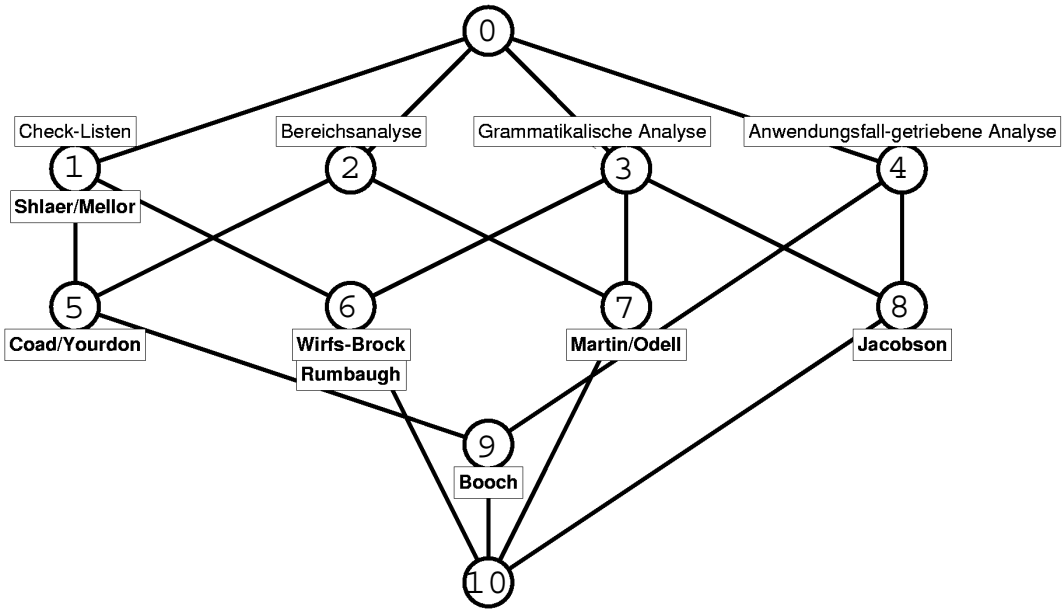


Abb. 3.11 Autoren und ihre Vorschläge zur Klassenidentifikation

In der abgebildeten Kreuztabelle sind statt Kreuzen die auch im zugehörigen Liniendiagramm benutzten Nummern der entsprechenden Begriffe in der folgenden Weise wiedergegeben. Tabellenzellen, die normalerweise ein Kreuz

erhalten hätten, enthalten die Nummern aller Begriffe, zu denen sie beitragen, d.h. daß das formale Merkmal der entsprechenden Tabellenspalte im Inhalt und der formale Gegenstand der Tabellenzeile im Umfang des Begriffs enthalten ist.
Zu jeder aufgeführten Nummer bilden die sie enthaltenden Tabellenzellen (nach entsprechenden Zeilen- und Spaltenvertauschungen) ein Rechteck, das vollständig mit eben dieser Nummer – also eigentlich einem Kreuz – ausgefüllt ist. Die Nummern 0 und 10 von Supremum und Infimum tauchen nicht auf, weil kein formaler Gegenstand alle formale Merkmale trägt und dual kein formales Merkmal auf alle Gegenstände zutrifft. Die zugehörigen Rechtecke haben für das Supremum die Breite 0 und Höhe 7 bzw. für das Infimum die Höhe 0 und Breite 4.

Die Idee der Blockrelationen ist es, die ursprüngliche Kreuztabelle so aufzufüllen, daß größere Rechtecke entstehen. So werden zum Umfang eines formalen Begriffs auch Gegenstände hinzugenommen, die nicht alle im Inhalt enthaltenen Merkmale tragen und umgekehrt. Um die ursprüngliche Klassifikation der Gegenstände und Merkmale nicht zu verfälschen, dürfen bei diesem Auffüllen keine neuen Begriffsumfänge (Kombinationen von Gegenständen) und auch keine neue Begriffsinhalte (Kombinationen von Merkmalen) entstehen. Angestrebt ist ja nur eine Vergröberung der Klassifikation durch die formalen Begriffe. Würde man neue Begriffsumfänge oder -inhalte zulassen, wären Klassifikationseinheiten entstanden, die dem ursprünglichen Kontext nicht zu entnehmen sind. Die Zuordnung der Umfänge und Inhalte zu formalen Begriffe wird bei der Bildung einer Blockrelation verändert: Zu einem Begriffsumfang wächst der entsprechende Begriffsinhalt und umgekehrt vergrößert sich zu einem Begriffsinhalt der zugehörige Umfang. Man erhält damit weniger Begriffe und vergröbert die Klassifikation.

Definition 33 (*Blockrelation*)

Eine *Blockrelation* eines formalen Kontexts (G, M, I) ist eine Relation $J \subseteq G \times M$ mit

a) $I \subseteq J$

b) Für alle $g \in G$ ist g^J ein Begriffsinhalt von (G, M, I).

c) Für alle $m \in M$ ist m^J ein Begriffsumfang von (G, M, I).

([G-W 96], 3.4, Definition 54, S. 122)

Mit einer Blockrelation J entsteht ein neuer formaler Kontext (G, M, J). Die Schreibweisen g^J und m^J sind in Definition 19 auf Seite 81 erklärt. So ist g^J die Menge aller formalen Merkmale, die g durch J zugesprochen werden und dual umfaßt m^J die formalen Gegenstände, auf die m nach J zutrifft. Da nach

Folgerung 2.a) auf Seite 82 zu einem formalen Kontext die Begriffsinhalte durch Schnitte von Gegenstandsinhalten und die Begriffsumfänge durch Schnitte von Merkmalsumfängen entstehen, besitzt nach den Bedingungen b) und c) aus Definition 33 der Kontext (G, M, J) gegenüber (G, M, I) keine neuen Begriffsinhalte oder Begriffsumfänge.

Bemerkung 8 (*Triviale Blockrelationen*)

Zu einem formalen Kontext (G, M, I) sind die Relationen I und $G \times M$ immer Blockrelationen.

Deshalb werden sie hier als *triviale Blockrelationen* bezeichnet.

Beispiel 16 (*Blockrelation zu Beispiel 15*)

Der formale Kontext aus Abbildung 3.10 besitzt genau eine interessante Blockrelation außer den beiden trivialen.

	Check-Listen	Grammatikalische Analyse	Bereichsanalyse	Anwendungsfall-getriebene Analyse
Booch	×		×	×
Coad/Yourdon	×		×	⊗
Jacobson	⊗	×	⊗	×
Martin/Odell	⊗	×	×	⊗
Rumbaugh	×	×	⊗	⊗
Shlaer/Mellor	×		⊗	⊗
Wirfs-Brock	×	×	⊗	⊗

Abb. 3.12 Eine Blockrelation

Diese ist in nebenstehender Abbildung angegeben. Originale Einträge der Inzidenzrelation sind als einfache Kreuze, durch die Blockrelation ergänzte zusätzlich mit einem Kreis eingetragen.

Begriffsumfänge des neuen Kontexts sind:

- {*Jacobson, Martin/Odell, Rumbaugh, Wirfs-Brock*}
  (schon vorher Merkmalsumfang von *Grammatikalische Analyse*)
- die Menge aller Autoren
  (immer Begriffsumfang)

Begriffsinhalte entsprechend der Blockrelation sind

- {*Check-Listen, Bereichsanalyse, Anwendungsfall-getriebene Analyse*}

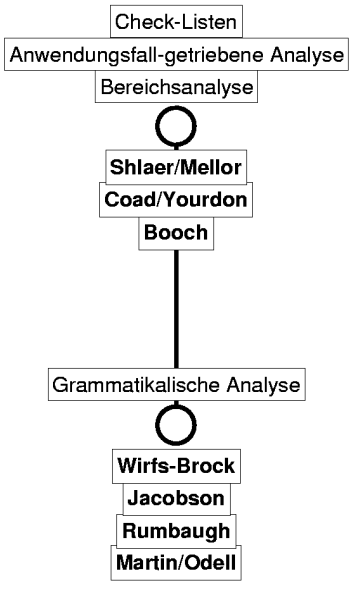


Abb. 3.13 Begriffsverband der Blockrelation

(im Original Gegenstandsinhalt von *Booch*)
- die Menge aller betrachteten Techniken
(immer ein Begriffsinhalt)

Die entstandenen formalen Begriffe sind in nebenstehendem Liniendiagramm wiedergegeben. Wie oben gesehen sind die Begriffsumfänge und -inhalte jeweils für sich allein genommen gegenüber dem ursprünglichen Begriffsverband aus Abbildung 3.11 auf Seite 111 nicht neu, die formalen Begriffe aber schon.
So sind jetzt alle Begriffe aus dem Intervall zwischen dem Gegenstandsbegriff zu *Booch* (Nummer 9) und dem globalen Supremum (Nummer 0) bzw. alle Begriffe zwischen dem globalen Infimum (Nummer 10) und dem Merkmalsbegriff zu *Grammatikalische Analyse* (Nummer 3) zusammengefaßt.

Die in Beispiel 16 aufgezeigte Zusammenfassung von Intervallen des zugrunde liegenden Begriffsverbands durch eine Blockrelation spielt in Anwendungen die entscheidende Rolle. Um zu zeigen, daß wirklich immer Intervalle des Ausgangsverbands zusammengefaßt werden und wie man diese Intervalle im ursprünglichen Begriffsverband beschreiben kann, muß nun weiter ausgeholt werden.

Zunächst einmal sei festgehalten, daß die Blockrelationen eines gegebenen formalen Kontexts wieder einen vollständigen Verband bilden.

Satz 15 (*Verband der Blockrelationen*)
Die Menge der Blockrelationen eines formalen Kontexts bildet mit der Inklusionsordnung einen vollständigen Verband.

Beweis: ([G-W 96], 3.4, S. 122)
Sei also (G, M, I) ein formaler Kontext.
Seien weiter T eine beliebige Indexmenge und für jedes $t \in T$ die Relation $J_t \subseteq G \times M$ eine Blockrelation von (G, M, I).
Dann ist $I \subseteq \bigcap_{t \in T} J_t$, weil $I \subseteq J_t$ für alle $t \in T$.
Weiter gilt für $g \in G$, daß

$$g^{\bigcap_{t \in T} J_t} = \{m \in M \mid (g, m) \in \bigcap_{t \in T} J_t\} = \{m \in M \mid (g, m) \in J_t \ \forall t \in T\}$$

$$= \bigcap_{t \in T} \{m \in M \mid (g, m) \in J_t\} = \bigcap_{t \in T} g^{J_t}$$

Dieser Schluß gilt auch im Fall $T = \varnothing$, dann ist

$$g^{\bigcap_{t \in T} J_t} = g^{G \times M} = M = \bigcap_{t \in T} g^{J_t}$$

Also ist mit den Merkmalsmengen g^{J_t} auch $g^{\bigcap_{t \in T} J_t}$ ein Begriffsinhalt von (G, M, I).

Analog zeigt man, daß $m^{\bigcap_{t \in T} J_t}$ ein Begriffsumfang von (G, M, I) ist.
Insgesamt ist $\bigcap_{t \in T} J_t$ wieder eine Blockrelation von (G, M, I).

Also ist das System der Blockrelationen von (G, M, I) ein Hüllensystem auf $G \times M$ und bildet nach Satz 5 auf Seite 78 mit der Inklusionsordnung einen vollständigen Verband.

Die oben angesprochenen Intervalle des Begriffsverbands von (G, M, I), die den formalen Begriffen von (G, M, J) entsprechen, wenn J eine Blockrelation von (G, M, I) ist, sind sogenannte Blöcke von Toleranzrelationen auf $\underline{\mathfrak{B}}(G, M, I)$.

Definition 34 (*Toleranzrelation*)

Sei V ein vollständiger Verband.
Eine Relation $\Theta \subseteq V \times V$ heißt *vollständige Toleranzrelation* auf V, wenn sie reflexiv, symmetrisch und mit Infima und Suprema verträglich ist, d.h. für jede Indexmenge T gilt:

$$x_t \Theta y_t \ \forall t \in T \quad \Rightarrow \quad (\bigwedge_{t \in T} x_t) \Theta (\bigwedge_{t \in T} y_t) \quad \text{und} \quad (\bigvee_{t \in T} x_t) \Theta (\bigvee_{t \in T} y_t)$$

Die Menge der Toleranzrelationen eines vollständigen Verbands bildet selber wieder einen vollständigen Verband.

Bemerkung 9 (*Verband der Toleranzrelationen*)

Das System der vollständigen Toleranzrelationen eines vollständigen Verbands ist mit der Inklusionsordnung ein vollständiger Verband, denn:
Reflexivität, Symmetrie und die Verträglichkeit mit Infima und Suprema übertragen sich direkt auf beliebige Schnitte von Toleranzrelationen.
Also bilden diese ein Hüllensystem und damit nach Satz 5 auf Seite 78 mit der Inklusionsordnung einen vollständigen Verband.

Mit zwei Elementen, die in einer Toleranzrelation stehen, erhält man schon ein ganzes Intervall, dessen Elemente paarweise in dieser Toleranzrelation stehen.

Hilfssatz 15 (*Toleranzrelationen in Intervallen*)

Seien V ein vollständiger Verband und Θ eine vollständige Toleranzrelation auf V.
Sind $a, b \in V$ mit $a \Theta b$, so gilt schon $x \Theta y$ für alle $x, y \in [a \wedge b, a \vee b]$.

Beweis: ([G-W 96], 3.4, Hilfssatz 54, S. 120)

Weil Θ reflexiv ist, gilt $a \Theta a$ und $b \Theta b$.
Weiter gilt $b \Theta a$, weil Θ symmetrisch ist.
Nach der Infimumsverträglichkeit von Θ gilt also $a = (a \wedge a) \Theta (a \wedge b)$

und $b = (b \wedge b)\,\Theta\,(a \wedge b)$.
Mit der Supremumsverträglichkeit von Θ folgt daraus
$(a \vee b)\,\Theta\,((a \wedge b) \vee (a \wedge b)) = (a \wedge b)$.
Sind $x, y \in [a \wedge b, a \vee b]$, so sind $x\Theta x$ und $y\Theta y$ wieder durch die Reflexivität von Θ gegeben.
Mit der Supremumsverträglichkeit von Θ ergibt sich damit
$x = x \vee (a \wedge b)\,\Theta\,x \vee (a \vee b) = (a \vee b)$ und analog $(a \vee b)\,\Theta\,y$.
Durch die Infimumsverträglichkeit von Θ folgt
$x = x \wedge (a \vee b)\,\Theta\,(a \vee b) \wedge y = y$.

Man hat nun eine Isomorphie zwischen dem Verband der Toleranzrelationen und dem Verband der Blockrelationen.

Satz 16 (*Toleranzrelationen* $\leftrightarrow$ *Blockrelationen*)
Sei (G, M, I) ein formaler Kontext.
Dann ist der Verband der vollständigen Toleranzrelationen auf $\underline{\mathfrak{B}}(G, M, I)$ isomorph zum Verband aller Blockrelationen von (G, M, I).
Ein entsprechender Isomorphismus β kann folgendermaßen definiert werden: $g\ (\beta(\Theta))\ m \;:\Leftrightarrow\; \gamma g\ \Theta\ (\gamma g \wedge \mu m)$ für $g \in G, m \in M$ und eine vollständige Toleranzrelation Θ auf $\underline{\mathfrak{B}}(G, M, I)$.
Die Umkehrabbildung ist gegeben durch
$(A, B)\ \beta^{-1}(J)\ (C, D) \;:\Leftrightarrow\; A \times D \cup C \times B \subseteq J$ für
$(A, B), (C, D) \in \underline{\mathfrak{B}}(G, M, I)$ und Blockrelationen J von (G, M, I).

Beweis: ([G-W 96], 3.4, Satz 15, S. 123)
Siehe Anhang A, Seite 264.

Durch die Verträglichkeit einer Toleranzrelation mit Infima und Suprema kann die Abbildung β, die einer vollständigen Toleranzrelation die zugehörige Blockrelation zuordnet, auch durch eine duale Vorschrift definiert werden.

Bemerkung 10 (*Duale Definition von* β)
Sei (G, M, I) ein formaler Kontext.
Dann gilt mit den Bezeichnungen aus Satz 16 für jede vollständige Toleranzrelation Θ auf $\underline{\mathfrak{B}}(G, M, I)$:
$g\ (\beta(\Theta))\ m \;\Leftrightarrow\; (\gamma g \vee \mu m)\ \Theta\ \mu m$ für alle $g \in G, m \in M$.
([G-W 96], 3.4, Satz 15, S. 123)

Beweis: Nach Satz 16 ist β definiert durch
$g\ (\beta(\Theta))\ m \;:\Leftrightarrow\; \gamma g\ \Theta\ (\gamma g \wedge \mu m)$.
Gilt für $g \in G, m \in M$, daß $g\ (\beta(\Theta))\ m$, so ist also $\gamma g\ \Theta\ (\gamma g \wedge \mu m)$ und wegen der Supremumsverträglichkeit von Θ auch
$(\gamma g \vee \mu m)\ \Theta\ (\gamma g \wedge \mu m) \vee \mu m = \mu m$.

Gilt andersherum $(\gamma g \vee \mu m)\ \Theta\ \mu m$, so folgt mit der Infimumsverträglichkeit von Θ, daß $\gamma g = \gamma g \wedge (\gamma g \vee \mu m)\ \Theta\ (\gamma g \wedge \mu m)$, d.h. $g\ (\beta(\Theta))\ m$.

Entsprechend Hilfssatz 15 werden die Blöcke einer Toleranzrelation wie folgt definiert.

Definition 35 (*Blöcke einer Toleranzrelation*)
Seien V ein vollständiger Verband und Θ eine vollständige Toleranzrelation auf V.
Für $a \in V$ definiere $a_\Theta := \wedge\{x \in V \mid x\Theta a\}$ und $a^\Theta := \vee\{x \in V \mid x\Theta a\}$.
Ein Intervall der Form $[a]_\Theta := [a_\Theta, (a_\Theta)^\Theta]$ heißt dann Block von Θ.
Dual sei als Schreibweise $[a]^\Theta := [(a^\Theta)_\Theta, a^\Theta]$ vereinbart.
([G-W 96], 3.4, Definition 52, S. 120)

Wegen der Reflexivität von Θ sind $[a]_\Theta$ und $[a]^\Theta$ Intervalle. Man bekommt dadurch gerade die maximalen Teilmengen, deren Elemente paarweise in Relation Θ stehen (siehe Anhang A, Bemerkung A2 auf Seite 272). Damit ergibt sich eine Faktorisierung des untersuchten vollständigen Verbands nach Θ (siehe Satz 17 auf Seite 120).

a_Θ ist das kleinste und a^Θ das größte Element, die mit a in Relation Θ stehen.

Hilfssatz 16 (*Kleinste und größte in Toleranzrelation stehende Elemente*)
Seien V ein vollständiger Verband und Θ eine vollständige Toleranzrelation auf V.
Dann gilt für alle $a \in V$, daß $a\ \Theta\ a_\Theta$ und $a\ \Theta\ a^\Theta$ und $a \in [a]_\Theta$.
([G-W 96], 3.4, S. 121)

Beweis: Wegen der Verträglichkeit von Θ mit Infima gilt
$a_\Theta = \wedge\{x \in V \mid x\Theta a\}\ \Theta \bigwedge_{x \in V} a = a$.
Dual folgt $a\ \Theta\ a^\Theta$ aus der Verträglichkeit mit Suprema.
Wegen der Reflexivität von Θ ist $a_\Theta = \wedge\{x \in V \mid x\Theta a\} \le a$.
Aus $a\ \Theta\ a_\Theta$ folgt weiter $(a_\Theta)^\Theta = \vee\{x \in V \mid x\Theta a_\Theta\} \ge a$.
D.h. $a \in [a]_\Theta$.

a_Θ und a^Θ müssen untereinander nicht in Relation Θ stehen. (Θ war nicht als transitiv angenommen worden.) Deshalb ergibt sich die etwas komplizierte Definition der Blöcke, deren Elemente alle paarweise in Relation Θ stehen sollen.

Die denkbare duale Definition über $[a]^\Theta$ ergibt die gleichen Mengen.

Hilfssatz 17 (*Duale Blockdefinition*)

Seien V ein vollständiger Verband und Θ eine vollständige Toleranzrelation auf V.
Dann gilt für alle $a \in V$, daß $((a_\Theta)^\Theta)_\Theta = a_\Theta$ und $((a^\Theta)_\Theta)^\Theta = a^\Theta$.
([G-W 96], 3.4, S. 121)

Beweis: Nach Hilfssatz 16 gilt $a_\Theta \,\Theta\, (a_\Theta)^\Theta$.
Also ist $((a_\Theta)^\Theta)_\Theta = \wedge\{x \in V \mid x \,\Theta\, (a_\Theta)^\Theta\} \leq a_\Theta$.
Weiter gilt nach Hilfssatz 16 $(a_\Theta)^\Theta \,\Theta\, ((a_\Theta)^\Theta)_\Theta$, $a \,\Theta\, a_\Theta$ und $a \leq (a_\Theta)^\Theta$.
Deshalb ist nach der Infimumsverträglichkeit von Θ
$a = (a \wedge (a_\Theta)^\Theta) \,\Theta\, (a_\Theta \wedge ((a_\Theta)^\Theta)_\Theta) = ((a_\Theta)^\Theta)_\Theta$.
Damit ist $a_\Theta = \wedge\{x \in V \mid x\Theta a\} \leq ((a_\Theta)^\Theta)_\Theta$.
Insgesamt gilt $((a_\Theta)^\Theta)_\Theta = a_\Theta$.
$((a^\Theta)_\Theta)^\Theta = a^\Theta$ zeigt man dual.

Weil also $[a]_\Theta = [a_\Theta, (a_\Theta)^\Theta] = [((a_\Theta)^\Theta)_\Theta, (a_\Theta)^\Theta] = [a_\Theta]^\Theta$ und andersherum $[a]^\Theta = [a^\Theta]_\Theta$ gilt, ergeben sich durch die mögliche duale Definition die selben Blöcke.

Die Blöcke einer Toleranzrelation brauchen nicht disjunkt sein und können "sich überkreuzen".

Bemerkung 11 (*Blöcke nicht disjunkt*)

Die Blöcke einer vollständigen Toleranzrelation Θ auf einem vollständigen Verband V brauchen nicht disjunkt zu sein.
Für $a, b \in V$ gilt $[a]_\Theta \cap [b]_\Theta \neq \emptyset \quad \Leftrightarrow \quad a_\Theta \leq (b_\Theta)^\Theta$ und $b_\Theta \leq (a_\Theta)^\Theta$.

Beweis: Siehe Anhang A, Seite 273.

Die unteren und oberen Intervallgrenzen der Blöcke sind gleich geordnet.

Hilfssatz 18 (*Untere und obere Intervallgrenzen von Blöcken*)

Seien V ein vollständiger Verband, Θ eine vollständige Toleranzrelation auf V und $a, b \in V$.
Dann gilt $a_\Theta \leq b_\Theta \;\Leftrightarrow\; (a_\Theta)^\Theta \leq (b_\Theta)^\Theta$.
([G-W 96], 3.4, S. 122)

Beweis: "$\Rightarrow$" Es gelte $a_\Theta \leq b_\Theta$.
Dann gilt $b_\Theta = (a_\Theta \vee b_\Theta) \,\Theta\, ((a_\Theta)^\Theta \vee (b_\Theta)^\Theta)$.
Also ist $(b_\Theta)^\Theta = \vee\{x \in V \mid x\Theta b_\Theta\} \geq (a_\Theta)^\Theta \vee (b_\Theta)^\Theta \geq (a_\Theta)^\Theta$.
"$\Leftarrow$" Es gelte $(a_\Theta)^\Theta \leq (b_\Theta)^\Theta$.
Dann gilt $(a_\Theta)^\Theta = ((a_\Theta)^\Theta \wedge (b_\Theta)^\Theta) \,\Theta\, (a_\Theta \wedge b_\Theta)$.
Nach Hilfssatz 17 ist damit
$a_\Theta = ((a_\Theta)^\Theta)_\Theta = \wedge\{x \in V \mid x\Theta(a_\Theta)^\Theta\} \leq a_\Theta \wedge b_\Theta \leq b_\Theta$.

Dementsprechend betrachtet man auf der Menge der Blöcke einer Toleranzrelation die folgende Ordnung.

Definition 36 (*Ordnung der Blöcke*)

Seien V ein vollständiger Verband, Θ eine vollständige Toleranzrelation auf V und $a, b \in V$.

Dann definiere $[a]_\Theta \leq [b]_\Theta \;:\Leftrightarrow\; a_\Theta \leq b_\Theta$.

V/Θ sei die Menge der Blöcke von Θ mit dieser Ordnung.

Die in Definition 36 definierte Relation ist wirklich eine Ordnung: Reflexivität und Transitivität übertragen sich direkt von der Ordnung auf V. Die Antisymmetrie folgt mit Hilfssatz 18 und der Antisymmetrie der Ordnung auf V.

Hilfssatz 19 (*Blockintervallgrenzen $\leftrightarrow$ Homomorphismen*)

Seien V ein vollständiger Verband und Θ eine vollständige Toleranzrelation auf V.

Dann ist die Abbildung $\varphi : V \to V, x \mapsto x_\Theta$ ein vollständiger $\vee$-Morphismus und die Abbildung $\psi : V \to V, x \mapsto x^\Theta$ ein vollständiger $\wedge$-Morphismus.

([G-W 96], 3.4, Hilfssatz 56, S. 121)

Beweis: Seien T eine beliebige Indexmenge und $a_t \in V$ für alle $t \in T$.

Zeige: $(\bigvee_{t \in T} a_t)_\Theta = \bigvee_{t \in T} a_{t\Theta}$.

Wegen der Supremumsverträglichkeit von Θ gilt $\bigvee_{t \in T} a_{t\Theta}\ \Theta\ \bigvee_{t \in T} a_t$.

Also ist $(\bigvee_{t \in T} a_t)_\Theta = \wedge\left\{x \in V \mid x\,\Theta\,(\bigvee_{t \in T} a_t)\right\} \leq \bigvee_{t \in T} a_{t\Theta}$.

Ist andersherum $y \in V$ mit $y\,\Theta\,(\bigvee_{t \in T} a_t)$, so gilt für jedes $s \in T$, daß

$y \wedge a_{s\Theta} = y \wedge \wedge\{x \in V \mid x\,\Theta\,a_s\}\,\Theta\,((\bigvee_{t \in T} a_t) \wedge \bigwedge_{x \in V} a_s) = a_s$.

Für jedes $s \in T$ gilt also $y \geq y \wedge a_{s\Theta} \geq \wedge\{x \in V \mid x\,\Theta\,a_s\} = a_{s\Theta}$.

Damit ist $y \geq \bigvee_{s \in T} a_{s\Theta}$.

Schließlich folgt hieraus $(\bigvee_{t \in T} a_t)_\Theta = \wedge\{x \in V \mid x\,\Theta\,(\bigvee_{t \in T} a_t)\} \geq \bigvee_{s \in T} a_{s\Theta}$.

Insgesamt gilt die behauptete Gleichheit.

Für $(\bigwedge_{t \in T} a_t)^\Theta = \bigwedge_{t \in T} a_t^\Theta$ dual.

Jetzt kann man feststellen, daß auch die Blöcke einer Toleranzrelation einen vollständigen Verband bilden.

Satz 17 (*Verband der Blöcke*)

Sind V ein vollständiger Verband und Θ eine vollständige Toleranzrelation auf V, so ist V/Θ ein vollständiger Verband mit $\bigvee_{t \in T}[a_t]_\Theta = [\bigvee_{t \in T} a_t]_\Theta$ und $\bigwedge_{t \in T}[a_t]^\Theta = [\bigwedge_{t \in T} a_t]^\Theta$ für jede beliebige Indexmenge T und Elemente $a_t \in V$.

Beweis: ([G-W 96], 3.4, Satz 14, S. 122)

Nach Hilfssatz 19 ist $(\bigvee_{t \in T} a_t)_\Theta = \bigvee_{t \in T} a_{t\Theta}$.

Also ist $[\bigvee_{t \in T} a_t]_\Theta \geq [a_s]_\Theta$ für alle $s \in T$.

Ist andersherum $[b]_\Theta$ ein Block mit $[b]_\Theta \geq [a_s]_\Theta$ für alle $s \in T$, so gilt $b_\Theta \geq a_{t\Theta}$ für alle $s \in T$ und deshalb $b_\Theta \geq \bigvee_{s \in T} a_{s\Theta} = (\bigvee_{t \in T} a_t)_\Theta$.

D.h. $[b]_\Theta \geq [\bigvee_{t \in T} a_t]_\Theta$.

Damit existiert $\bigvee_{t \in T}[a_t]_\Theta$ und es ist $\bigvee_{t \in T}[a_t]_\Theta = [\bigvee_{t \in T} a_t]_\Theta$.

Dual gilt $\bigwedge_{t \in T}[a_t]^\Theta = [\bigwedge_{t \in T} a_t]^\Theta$.

Im Falle eines Begriffsverbands lassen sich die Intervallgrenzen von Blöcken einer Toleranzrelation mit Hilfe der originalen Inzidenzrelation und der zugehörigen Blockrelation darstellen.

Hilfssatz 20 (*Blöcke ↔ Inzidenz- + Blockrelation*)

Seien (G, M, I) ein formaler Kontext, Θ eine vollständige Toleranzrelation auf $\underline{\mathfrak{B}}(G, M, I)$ und $J := \beta(\Theta)$ die zugehörige Blockrelation von (G, M, I) (mit β aus Satz 16 auf Seite 116).

Dann gilt für einen formalen Begriff $(A, B) \in \underline{\mathfrak{B}}(G, M, I)$:

a) $(A, B)_\Theta = (A^{JI}, A^J)$

b) $(A, B)^\Theta = (B^J, B^{JI})$

c) $((A, B)_\Theta)^\Theta = (A^{JJ}, A^{JJI})$

Beweis: ([G-W 96], 3.4, Korollar 57, S. 125)

Sei $(A, B) \in \underline{\mathfrak{B}}(G, M, I)$.

Nach Folgerung 2.c),a) auf Seite 82 ist $A^J = \bigcap_{g \in A} g^J$ als Schnitt von Begriffsinhalten von (G, M, I) selber ein Begriffsinhalt von (G, M, I).

Also ist $(A^{JI}, A^J) \in \underline{\mathfrak{B}}(G, M, I)$.

Weiter ist
$$\begin{aligned}(A \times A^J) \cup (A^{JI} \times B) &= (A \times A^J) \cup (A^{JI} \times A^I) \\ &\subseteq (A \times A^J) \cup (A^{JJ} \times A^J) \subseteq J\end{aligned}$$

Nach Satz 16 gilt also $(A, B)\,\Theta\,(A^{JI}, A^J)$.

Genauso ist $(B^J, B^{JI}) \in \underline{\mathfrak{B}}(G, M, I)$ mit $(A, B)\,\Theta\,(B^J, B^{JI})$.

Sei $(C, D) \in \underline{\mathfrak{B}}(G, M, I)$ mit $(A, B)\,\Theta\,(C, D)$.
Dann ist nach Satz 16 $(A \times D) \cup (C \times B) \subseteq J$.
D.h. $D \subseteq A^J$ und $C \subseteq B^J$.
Also gilt $(A^{JI}, A^J) \leq (C, D) \leq (B^J, B^{JI})$.
Es folgt $(A, B)_\Theta = (A^{JI}, A^J)$ und $(A, B)^\Theta = (B^J, B^{JI})$.
Damit ist $((A, B)_\Theta)^\Theta = ((A^{JI}, A^J))^\Theta = (A^{JJ}, A^{JJI})$.

Nun bekommt man auch eine Isomorphie zwischen den formalen Begriffen eines aus einer Blockrelation neu gebildeten Kontexts und den Blöcken der entsprechenden Toleranzrelation.

Satz 18 (*Blockrelation ↔ Toleranzrelation*)
Seien (G, M, I) ein formaler Kontext, Θ eine vollständige Toleranzrelation auf $\underline{\mathfrak{B}}(G, M, I)$ und $J := \beta(\Theta)$ die zugehörige Blockrelation von (G, M, I) (mit β aus Satz 16 auf Seite 116).
Dann sind $\underline{\mathfrak{B}}(G, M, I)/\Theta$ und $\underline{\mathfrak{B}}(G, M, J)$ isomorph.
Genauer gilt:

a) $[(B^I, B), (A, A^I)]$ ist genau dann ein Block von Θ, wenn (A, B) ein formaler Begriff von (G, M, J) ist.

b) Die Abbildung $[(B^I, B), (A, A^I)] \mapsto (A, B)$ ist ein Isomorphismus zwischen $\underline{\mathfrak{B}}(G, M, I)/\Theta$ und $\underline{\mathfrak{B}}(G, M, J)$.

c) Für $(A, B) \in \underline{\mathfrak{B}}(G, M, J)$ gilt für den zugehörigen Block von Θ, daß $[(B^I, B), (A, A^I)] = \mathfrak{B}(A, B, I \cap (A \times B))$.

Beweis: ([G-W 96], 3.4, Korollar 57, S. 125)
Siehe Anhang A, Seite 269.

Die Bildung von Blockrelationen eines formalen Kontexts und Toleranzrelationen des zugehörigen Begriffsverbands sind gleichbedeutend. Die formalen Begriffe des durch eine Blockrelation neu definierten Kontexts entsprechen bestimmten Intervallen des originalen Begriffsverbands, nämlich gerade den Blöcken der zugehörigen Toleranzrelation. Insbesondere haben formale Kontexte mit isomorphen Begriffsverbänden auch isomorphe Verbände von Blockrelationen.

Satz 19 (*Blockrelationen und isomorphe Begriffsverbände*)
Seien (G, M, I) und (H, N, R) zwei formale Kontexte, so daß $\underline{\mathfrak{B}}(G, M, I) \cong \underline{\mathfrak{B}}(H, N, R)$.
Dann sind auch die Verbände der Blockrelationen von (G, M, I) und (H, N, R) isomorph.
Sei weiter J eine Blockrelation von (G, M, I) und S die nach dieser Isomorphie zugehörige Blockrelation von (H, N, R).

Dann sind die durch die Blockrelationen definierten Begriffsverbände $\underline{\mathfrak{B}}(G, M, J)$ und $\underline{\mathfrak{B}}(H, N, S)$ isomorph.

Beweis: Siehe Anhang A, Seite 271.

Zu jeder Blockrelation sind nach Satz 18 zwei Liniendiagramme interessant. Das eine stellt den durch die Blockrelation neu definierten Begriffsverband dar (grobe Sicht) und das andere zeigt die entsprechenden Blöcke im ursprüngliche Begriffsverband auf.

Beispiel 17 (*Liniendiagramme zu Beispiel 15 und Beispiel 16*)

Nebenstehend ein Liniendiagramm zur Blockrelation in Abbildung 3.12 auf Seite 113.
Abbildung 3.15 gibt ein Liniendiagramm des Ausgangskontexts aus Abbildung 3.10 auf Seite 111 wieder. Gegenüber Abbildung 3.11 auf Seite 111 ist die Anordnung geändert, um die Blöcke der zu der Blockrelation gehörigen Toleranzrelation besser herausstellen zu können.
Die Kennzeichnung der Knoten in Abbildung 3.14 und Abbildung 3.15 verdeutlicht die in Satz 18 hergeleitete Zuordnung von formalen Begriffen im Blockrelations-Liniendiagramm zu entsprechenden Blöcken des originalen Begriffsverbands.

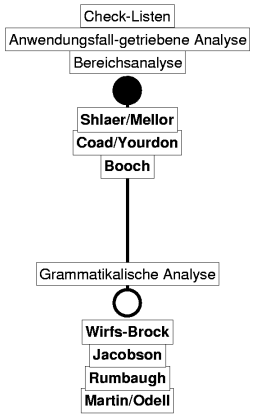


Abb. 3.14 Liniendiagramm zur Blockrelation

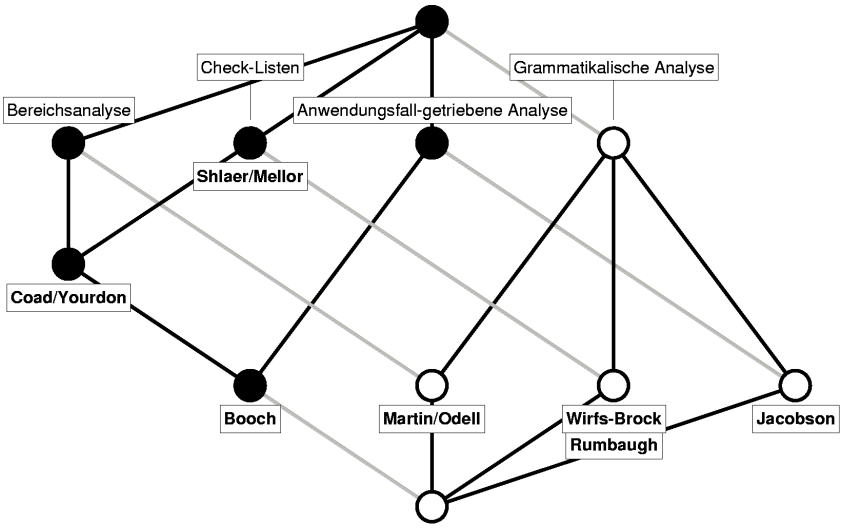


Abb. 3.15 Liniendiagramm mit Blöcken der Toleranzrelation

Dem Supremum aus Abbildung 3.14 entspricht das Intervall der formalen Begriffe mit schwarz ausgefüllt gezeichneten Knoten in, dem Infimum das der nicht ausgefüllt gezeichneten Knoten.
Wie schon in Beispiel 16 auf Seite 113 erläutert, sind durch den Übergang zur Blockrelation keine neuen Begriffsinhalte oder - umfänge entstanden, aber zwei neue Begriffe. Sie tragen jeweils den Umfang des größten formalen Begriffs des zugehörigen Blocks und den Inhalt des kleinsten. Diese Umfang-Inhalt-Paare kommen im originalen Begriffsverband nicht vor.

4 BASE - ein begriffsbasiertes Analyseverfahren für die Software-Entwicklung

Dieses Kapitel stellt BASE als Analyseverfahren vor. Um das Vorgehen zu veranschaulichen, wird ein durchgehendes Beispiel verwendet. Dieses wird zunächst in 4.1 erläutert. Der folgende Abschnitt 4.2 erklärt den grundsätzlichen Ansatz von BASE. Im wesentlichen wird dort beschrieben, wie ein formaler Kontext innerhalb von BASE zusammengestellt wird und welche (ersten) Schlüsse man aus dem entstehenden Begriffsverband bzw. Liniendiagramm ableiten kann. Nach dem ersten Ansatz wird das erstellte Modell durch funktionale Zerlegung der betrachteten Anwendungsfälle verfeinert. Dies beschreibt der Abschnitt 4.3. Insbesondere wird eine Möglichkeit vorgestellt, die Granularität der Verfeinerung zu kontrollieren. Das dazu benutzte Vorgehen ermöglicht auch die Generierung von Fragen zur Modellüberprüfung (ausgeführt in Abschnitt 4.4). In 4.5 wird aufgezeigt, wie sich aus dem funktional weit aufgefächerten Modell Klassenkandidaten ableiten lassen. Diese Betrachtungen werden in 4.6 durch einen weiteren Ansatz ergänzt, der die Systemzerlegung in einzelne Bausteine (Komponenten oder Klassen) unterstützt. Dabei stößt man auf Möglichkeiten, nur Teile der betrachteten Diagramme zu präsentieren, um deren Übersichtlichkeit zu wahren. Entsprechende Verfahren sind in 4.7 zusammengefaßt. Der abschließende Abschnitt 4.8 stellt ein Vorgehensmodell für die Anwendung von BASE vor.

Zur Beschreibung der Interna von BASE werden hier die in Kapitel 3 eingeführten mathematischen Konzepte benutzt. Bei der ersten Erwähnung eines entsprechenden Terminus findet sich jeweils eine Referenz auf die entsprechende Definition in

Kapitel 3. Außerdem sind Referenzen auf Sätze aus Kapitel 3 eingeflochten, wenn dort bewiesene wichtige Eigenschaften innerhalb von BASE ausgenutzt werden. Dies ist für die Erklärung des Ansatzes essentiell. Dem Benutzer von BASE können die mathematischen Einzelheiten jedoch verborgen bleiben, wie in Kapitel 5 zu sehen ist, in dem das Analyse-Werkzeug für BASE beschrieben wird.

4.1 Das Analysebeispiel

Das für die Beschreibung von BASE benutzte Beispiel entstand in einem Studentenpraktikum an der Philipps-Universität Marburg. Die Ergebnisse dieses Praktikums sind im WWW dokumentiert (siehe [FGT 98]). Aufgabe der Studenten war die Erstellung von Schulungsunterlagen zur Unified Modeling Language (UML) in der Version 1.1 (vergl. [BJR 97]) in Form einer WWW-Präsentation.

In einem ersten Schritt wurde dazu ein Beispielsystem mit UML modelliert. Entnommen war dieses Beispiel dem FRISCO-Report ([FHL+ 98]). Untersuchungsgegenstand ist dabei das Geschäft einer imaginären Firma JWI (Japan Wines Inc.). Grundlage der Modellierung war die Beschreibung des Geschäfts von JWI in 28 umgangssprachlichen Aussagen. Diese sind im Anhang C wiedergegeben. JWI erhält Wein von Winzereien und liefert ihn an Einzelhändler weiter. Ein neu zu erstellendes Software-System soll die Aufnahme und Abwicklung von Kundenbestellungen auf der einen Seite und die Bestellung bei und Verbuchung von Lieferungen von Winzereien auf der anderen Seite unterstützen.

Dieses Beispiel wurde für die Darstellung an dieser Stelle ausgewählt, weil es ohne die Verwendung von BASE ausgehend von den Anwendungsfällen des Systems entwickelt wurde und es so möglich ist, die Anwendung von BASE anhand eines zunächst anderweitig und von anderen Personen modellierten Systems aufzuzeigen. Alle im folgenden getroffenen Entwurfsentscheidungen sind – soweit dort schon getroffen – vom Praktikum übernommen worden, um damit den Einfluß anderer Beteiligter auf die Modellierung zu simulieren.

4.2 Grundlegender Ansatz

4.2.1 Anwendungsfälle und ihre Behandlung in BASE

Das zentrale Ziel eines Software-Entwicklungsprojekts ist die Erfüllung der funktionalen Anforderungen. (Martin Fowler und Kendall Scott bringen diesen Um-

stand auf den Punkt: "Diagrams are, after all, just pretty pictures. No user is going to thank you for pretty pictures; what a user wants is software that executes.", [F-S 97], Chapter 1) Deshalb sollten die funktionalen Anforderungen an den Anfang der Entwicklung gestellt werden. Anwendungsfälle nach Ivar Jacobson ([JCJÖ 94], siehe auch 2.5) beschreiben die geforderte Funktionalität. Sie gelten als erfolgversprechende Technik für die frühe Analysephase, weil sie aus der Perspektive der Anwender heraus formuliert werden. Außerdem benötigt der Anwender keinen Einblick in objektorientierte Modellierungstechniken, er muß nur die Abläufe in seiner Sprache beschreiben. Erst nach der Erarbeitung der wichtigsten Anwendungsfälle beginnen die Entwickler, auf deren Grundlage ein objektorientiertes Analysemodell zu erstellen. Viele Autoren haben die Anwendungsfälle in ihre Methoden aufgenommen. Auch innerhalb der *Unified Modeling Language* (UML, [BJR 99a]) bilden sie einen Schwerpunkt.

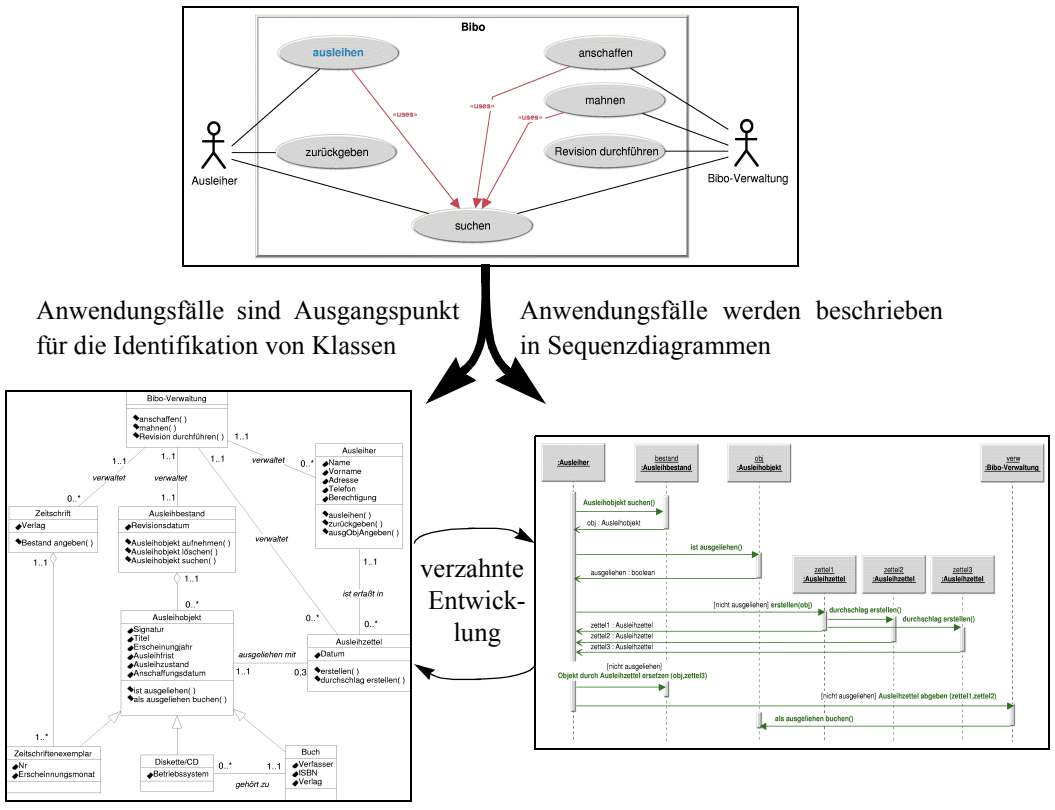


Abb. 2.7 Anwendungsfall-getriebene Analyse

In 2.6.2 wurde schon das Anwendungsfall-getriebene Verfahren zur Identifikation von Klassenkandidaten diskutiert. Auch BASE fußt auf Anwendungsfällen als einer anerkannten Modellierungstechnik für die Festschreibung von funktionalen Anforderungen.

Abbildung 2.7 stammt aus Abschnitt 2.5. Sie stellt die Idee für die erste Phase der

Anwendungsfall-getriebenen Analyse dar. (Die einzelnen Diagramme finden sich in hinreichend großer und bis in die Einzelheiten lesbarer Darstellung in Kapitel 2: Abbildung 2.6 auf Seite 46, Abbildung 2.3 auf Seite 36 und Abbildung 2.4 auf Seite 39)

Zunächst sind die Anwendungsfälle zu identifizieren und zu beschreiben. Die Gesamtheit der Anwendungsfälle (jeweils in Ovalen angegeben) ergibt die Funktionalität des Systems (Kasten um die Ovale). Sie wird genutzt durch Akteure außerhalb des Systems. Die Beschreibung der einzelnen Anwendungsfälle geschieht in Prosa, gewünscht ist aber eine Beschreibung formalerer Art, die im Hinblick auf den späteren Systementwurf weniger Mehrdeutigkeiten enthält. Weitverbreitetes Mittel hierzu sind Interaktionsdiagramme, wie das in Abbildung 2.7 rechts stehende Sequenzdiagramm. In ihnen wird die Interaktion zwischen Objekten des Systems beschrieben, die zur Bearbeitung eines Anwendungsfalls durch das System nötig ist. Die betreffenden Klassen und ihre Beziehungen untereinander werden im Klassendiagramm (in Abbildung 2.7 links) dokumentiert. Das Anliegen von BASE ist es, die Gewinnung von Klassen aus dem Anwendungsfallmodell zu unterstützen und auch die Modellierung in Sequenzdiagrammen dadurch zu erleichtern, daß zunächst nur die Einzelschritte identifiziert werden müssen und nachträglich auf der Grundlage eines schon weiter ausgeprägten Systemverständnisses den gefundenen Klassen zugeordnet werden können.

Die angesprochenen Modelle beschreiben anwendungsspezifisches Wissen. Dieses besitzen die Entwickler in der Regel nicht. Die Mitwirkung von Fachexperten und späteren Anwendern ist in dieser Phase für einen Projekterfolg unabdingbar. Objektorientierte Konzepte bieten aber nicht unbedingt eine gute Basis für die Kommunikation mit den Fachexperten (vergl. [FKM+ 96]). BASE schaltet der Entwicklung von objektorientierten Modellen eine kombinierte Untersuchung von funktionalen und Daten-Aspekten vor. Dieses Vorgehen kommt den an der Entwicklung beteiligten Anwendern ebenfalls entgegen. So sind die funktionale Zerlegung und die entsprechenden Modelle aus funktionaler bzw. ablauforientierter Sicht zur Beschreibung ihrer Geschäftsprozesse gewohnt. Deshalb sollte es in der Regel einfacher sein, die Einzelschritte innerhalb der Anwendungsfälle aus rein funktionaler Sicht zu identifizieren als sofort die Diskussion um Objekte und Klassen zu beginnen und dann die Einzelschritte als Operationsaufrufe von gewissen Objekten an andere zu interpretieren. Wie schon in 1.6.1 auf Seite 16 erwähnt, dienen Liniendiagramme von Begriffsverbänden als Darstellungs- und Kommunikationsmittel. Diese bieten nach Erfahrungen der Darmstädter Arbeitsgruppe um Rudolf Wille eine geeignete Basis zur Verständigung auch mit mathematisch wenig versierten Gesprächspartnern.

Idee von BASE ist der durch die Anwendungsfälle gegebenen funktionalen Sicht

eine datenorientierte Sicht zur Seite zu stellen und aus diesen komplementären Sichten Klassenkandidaten abzuleiten. Als Datenelemente werden "Dinge" des Untersuchungsbereichs betrachtet. "Dinge" müssen dabei nicht unbedingt materiell sein, sondern können auch in Abstrakta wie Beziehungen, Rollen etc. bestehen. Deshalb wird hier die Bezeichnung immer in Anführungszeichen gebraucht. Für die Entwickler sind sie neben den Anwendungsfällen auf der anderen Seite die entscheidenden Orientierungspunkte beim Durchdringen des Untersuchungsbereichs. Im Hinblick auf das zu entwickelnde Klassenmodell sind sie Kandidaten für Objekte, Klassen, Attribute und Rollen. Mit ihnen ist eine Systembeschreibung aus Datensicht möglich. Haben Entwickler und Fachexperten zusammen Schwierigkeiten, "Dinge" festzulegen, können – wie in 2.6 dargestellt – die bei Shlaer/Mellor, Coad/Yourdon, Fusion oder Booch vorgeschlagenen Listen Anregungen bieten. Als erste Klassenkandidaten werden auch bei den meisten anderen Methoden solche in Erwägung gezogen, die ein direktes Gegenstück in der realen Welt repräsentieren. In BASE wird aber die Entscheidung, welche "Dinge" später als Klassen modelliert werden und welche etwa als Attribute, verschoben. Die Betrachtung der Gegenstandsbegriffe in dem mit Hilfe der Formalen Begriffsanalyse entwickelten Liniendiagramm hilft bei dieser Entscheidung.

Die Funktionalität ist durch die Anwendungsfälle beschrieben. Mit den Fachexperten sind nun die Anwendungsfälle durchzugehen und daraufhin zu untersuchen, welche "Dinge" jeweils involviert sind. Die Formulierung unterstellt hier eine Reihenfolge "Anwendungsfälle vor betroffenen Dingen". Das ist in dem Sinne gemeint, daß zunächst mit einer gewissen Anzahl von Anwendungsfällen begonnen wird, um so die beschriebene Funktionalität in den Mittelpunkt der Betrachtung zu stellen. Zu allen diesen als Ausgangspunkt dienenden Anwendungsfällen sollten von den Fachexperten Prosa-Beschreibungen erstellt werden. Indem die Entwickler diese Beschreibungen mit den Anwendern durcharbeiten, bilden sie ein grobes Verständnis des Untersuchungsbereichs als Kommunikationsgrundlage aus. Danach ist von beiden Gruppen zusammen die Zuordnung von "Dingen" zu Anwendungsfällen, in die sie involviert sind, zu leisten. Um die von Jacobson erwähnte häufige Revision schon getroffener Entwurfsentscheidungen zu verringern, ist dieser Schritt hinter das erste gemeinsame Durcharbeiten aller zu Anfang betrachteten Anwendungsfälle gesetzt (vergl. Abbildung 4.1).

Mit der Auflistung der beteiligten "Dinge" wird die erste grobe Systembeschreibung verfeinert. Dabei kommen in der Regel auch neue Anwendungsfälle – etwa zur Pflege von Datenbeständen oder zur Umsetzung gemeinsamer Teilaufgaben schon betrachteter Anwendungsfälle – ins Blickfeld. Diese werden dann in die Betrachtung eingeschlossen. Da davon ausgegangen werden kann, daß sich zu diesem Zeitpunkt schon ein Entwicklern und Fachexperten gemeinsames Grundverständ-

nis des Untersuchungsbereichs herausgebildet hat, kann bei den neu gewonnenen Anwendungsfällen die Zuordnung der involvierten "Dinge" zu den betreffenden Anwendungsfällen direkt mit der Beschreibung zusammen geschehen.

Damit erfolgt die Bearbeitung der Anwendungsfälle in BASE nach folgendem Schema:

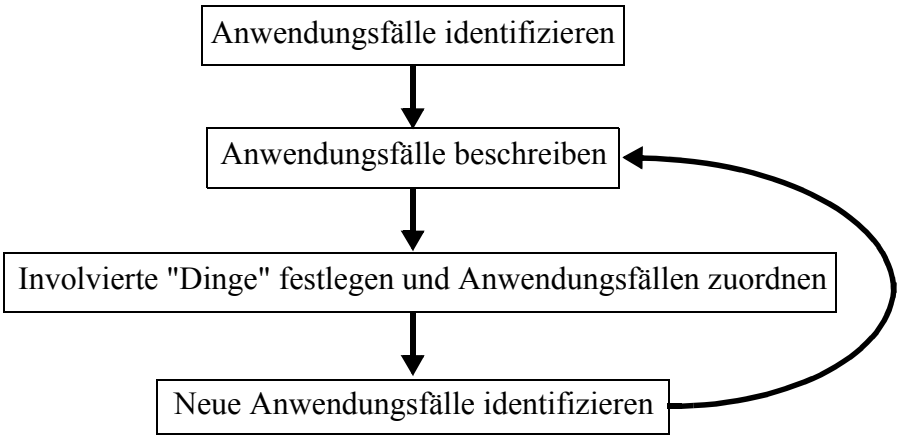


Abb. 4.1 Identifizieren und Indizieren von Anwendungsfällen

4.2.2 Die Anwendungsfälle des Analysebeispiels

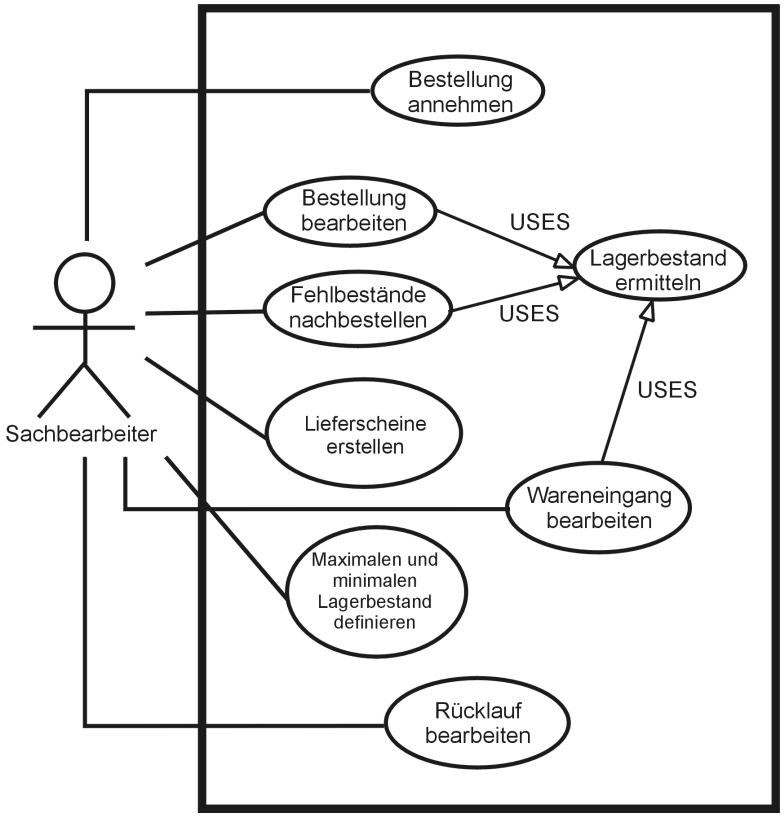


Abb. 4.2 Anwendungsfälle des JWI-Beispiels

Im Praktikum wurde das in Abbildung 4.2 angegebene Anwendungsfallmodell erstellt.

Die Beschreibungen der Anwendungsfälle wurden aus den zugrunde liegenden Aussagen (siehe Anhang C) entwickelt. Sie sind im folgenden – wie im Praktikum formuliert, aber mit zusätzlichen Hervorhebungen (vergl. unten) – wiedergegeben. Nur der erste Satz der Beschreibung des Anwendungsfalls *Bestellung bearbeiten* wurde hier um die Erwähnung des *Produkts* gekürzt, um die Behandlung von "*uses*"-Beziehungen zwischen Anwendungsfällen aufzuzeigen (siehe 4.2.3 auf Seite 135). Dadurch ergibt sich zunächst eine unvollständige Zuordnung von "Dingen" zu Anwendungsfällen – in dem Sinne, daß das "Ding" *Produkt* eigentlich dem Anwendungsfall *Bestellung bearbeiten* aus fachlichen Gründen zugesprochen werden müßte, dies aber bei der durchgeführten Indizierung unterbleibt. Diese Behandlung der "*uses*"-Beziehung zwischen den Anwendungsfällen *Bestellung bearbeiten* und *Lagerbestand ermitteln* führt gerade zur Korrektur dieses Fehlers.

Brainstorming zur Indizierungserstellung

Die folgenden Anwendungsfallbeschreibungen dienen als Grundlage für die weitere Modellierung mit BASE. Als Ersatz für den Diskurs mit Fachexperten wurde hier eine Indizierung der Anwendungsfälle mit den beteiligten "Dingen" allein anhand des Textes vorgenommen. Die entsprechende Auswahl ist die erste Entwurfsentscheidung, die nachträglich getroffen wurde und nicht dem Praktikum direkt entnommen werden konnte. Sie ist in den Beschreibungen durch die fett und kursiv gedruckten Wörter angedeutet.

In einem ersten Schritt wurde alle möglicherweise als interessante "Dinge" in Frage kommenden Wörter markiert. Speziell in Fällen, in denen Entwicklern sich mehreren Fachexperten gegenüber sehen, wird auch in realen Projekten erst einmal eine solche Sammlung – oder neudeutsch ein Brainstorming – vorgenommen.

Die unten benutzten Angaben in Klammern sind Referenzen auf die betreffenden Aussagen der zugrundeliegenden Beschreibung (siehe Anhang C).

Anwendungsfall *Bestellung annehmen*

- Das ***Zentrum*** empfängt ***Bestellungen*** der ***Einzelhändler*** telefonisch von 9 bis 17 Uhr (S7).
- Die eingehenden ***Bestellungen*** der ***Kunden*** werden vom ***Sachbearbeiter*** in ***Bestellformularen*** erfaßt (S8).
- Eine ***Kundenbestellung*** kann aus mehreren ***Bestellposten*** bestehen. Diese beziehen sich auf einzelne ***Produkte*** (S9).
- Jeder ***Bestellposten*** wird auf dem ***Formular*** in einer Zeile notiert (S10).

Anwendungsfall *Bestellung bearbeiten*

- Der Anwendungsfall "Lagerbestand ermitteln" wird *benutzt* ("*uses*"-Beziehung), um für jeden ***Posten*** der gerade bearbeiteten ***Kundenbestellung*** den aktuellen ***Lagerbestand*** zu ermitteln (S11).
- Falls für einen ***Bestellposten*** genügend ***Waren*** auf ***Lager*** sind, dann wird dieser in die ***Lieferkartei*** eingetragen und der ***Lagerbestand*** entsprechend korrigiert (S14). Anderenfalls wird der ***Bestellposten*** in die ***Wartekartei*** eingetragen (S15).

Anwendungsfall *Fehlbestände nachbestellen*

- Der Anwendungsfall "Lagerbestand ermitteln" wird *benutzt* ("*uses*"-Beziehung), um für jedes ***Produkt*** den aktuellen ***Lagerbestand*** zu ermitteln. Das Ergebnis dient dazu, ***Fehlbestände*** aufzufinden.
- Täglich nach 17 Uhr (d.h. nur einmal täglich) werden die notwendigen ***Bestellungen*** an die ***Weinproduzenten*** wie folgt erstellt (S16):
- ***Bestellungen*** an ***Weinproduzenten*** werden für solche ***Produkte*** erteilt, deren aktueller ***Lagerbestand*** verringert um die ***noch nicht nachbestellten Mengen*** aus der ***Wartekartei*** unter den ***Minimallagerbestand*** gesunken ist. Dabei wird so viel bestellt, daß der entsprechende ***Maximalbestand*** erreicht wird (S17, S28).
- Die ***Nachbestellungen*** werden vom ***Sachbearbeiter*** in ***Bestellformularen*** an ***Weinproduzenten*** erfaßt (siehe AN-2.1, S18).

Anwendungsfall *Lagerbestand ermitteln*

- Es wird der aktuelle mengenmäßige ***Lagerbestand*** für ein ***Produkt*** ermittelt.
- "Lagerbestand ermitteln" wird von den Anwendungsfällen "Bestellung bearbeiten", "Fehlbestände nachbestellen" und "Wareneingang bearbeiten" *benutzt* ("*uses*"-Beziehung).

Anwendungsfall *Lieferscheine erstellen*

- Es werden ***Lieferscheine*** für jeden ***Lieferwagen*** erstellt. Dabei werden den ***Lieferwagen*** die ***Bestellposten*** aus der ***Lieferkartei*** entsprechend ihrer ***Fahrtziele*** zugeordnet (S20).
- Ein Muster für einen ***Lieferschein*** ist in (S21) zu finden.

Anwendungsfall *Wareneingang bearbeiten*

- Frische ***Produkte*** von den ***Weinproduzenten*** treffen zwischen 10 und 16 Uhr ein (S27).
- Nach Eintreffen werden die frischen ***Produkte*** (nach dem Grundsatz "zuerst bestellt - zuerst beliefert") den ***Bestellposten*** der ***Wartekartei*** zugeordnet. Alle nunmehr lieferbaren ***Posten*** werden von dort in die ***Lieferkartei*** übertragen. Der Rest der frischen ***Ware*** wird dem ***Lagerbestand*** zugeführt (S28).

Anwendungsfall *Maximalen und minimalen Lagerbestand definieren*

- Der ***Sachbearbeiter*** definiert den ***maximalen*** und ***minimalen Lagerbestand*** für ein ***Produkt*** anhand seiner bzw. ihrer persönlichen Erfahrung oder externer Vorgaben.

Anwendungsfall *Rücklauf bearbeiten*

- ***Produkte*** mit dem ***Vermerk*** "Retoure" (siehe ***Lieferschein***) werden ins ***Lagerhaus*** zurückgebracht (S24).
- Der ***Sachbearbeiter*** löscht alle (gemäß dem ***Vermerk*** auf dem ***Lieferschein***) ordnungsgemäß ausgelieferten ***Bestellposten*** aus der ***Lieferkartei*** (S25).
- ***Bestellposten*** mit dem ***Vermerk*** "Retoure" verbleiben in der ***Lieferkartei*** und werden am Folgetag in die ***Lieferscheine*** aufgenommen (S26).

Behandlung von Synonymen

Bei der Indizierung auf der Basis von Anforderungsdokumenten sind in der Regel einheitliche Bezeichnungen für die auftauchenden "Dinge" festzulegen. Dies ist auch nötig, um eine "Projektsprache zu etablieren. Die dabei ausgezeichneten Termini sind deshalb jeweils mit einer Erklärung für das Projekt zu dokumentieren.

Da das Beispiel im wesentlichen ein Vertriebszentrum ohne näheren Bezug auf dessen spezielles ***Produkt*** "Wein" beschreibt, werden für die belieferten *Einzelhändler* die Bezeichnung ***Kunde*** und die liefernden *Weinproduzenten* der Terminus ***Lieferant*** benutzt. Die *Bestellungen* von Kunden an JWI und von JWI an Lieferanten besitzen für das Geschäft von JWI verschiedenen Stellenwert. Deshalb werden im folgenden die Bezeichnungen ***Kundenbestellung*** und ***Nachbestellung*** verwendet, um sie auseinander zu halten. Der ***Bestellposten*** bietet gegenüber seinem Synonym *Posten* die präzisere Formulierung. *Ware* wird synonym zu ***Produkt*** gebraucht und deshalb nicht als eigenes "Ding" betrachtet. Die *noch nicht nachbestellte Menge* ist der Kürze halber als ***vorgemerkte Menge*** bezeichnet, von den verschiedenen Bezeichnungen des minimalen und maximalen Lagerbestands eines Produkts durchgehend nur ***Minimalbestand*** und ***Maximalbestand*** benutzt und der *Vermerk*, der auf dem Lieferschein Auskunft darüber gibt, ob die Lieferung erfolgreich erfolgt ist, wird als ***Lieferscheinvermerk*** bezeichnet, um ihn auch außerhalb des Kontexts der obigen Anwendungsfallbeschreibungen eindeutig referenzieren zu können.

Beschränkung auf die wichtigen "Dinge"

Nicht alle "Dinge", die sich auf den ersten Blick zur Indizierung anbieten, erscheinen bei genauerer Betrachtung zur Beschreibung der Anwendungsfälle hilfreich. So wird hier zum Beispiel *Zentrum* nicht als wichtiges "Ding" behandelt, weil es

das umfassende System beschreibt. Aus ähnlichem Grund ist der *Sachbearbeiter* auch nicht weiter erwähnt, weil er ein Aktor ist, der außerhalb des Systems steht. Diese Entwurfentscheidung könnte anders ausfallen, wenn das System wiederum Daten über ihn verwalten müßte. Das *Bestellformular* wird nicht weiter beachtet, weil eine Abgrenzung gegen die in ihm dokumentierte *Bestellung* schwerfällt. Dagegen wird der ***Lieferschein*** als "Ding" behandelt, weil eine "Lieferung" nicht erwähnt wird. Die ungleiche Behandlung dieser "Dinge" drückt auch die verschieden wichtige Bedeutung für das Geschäft von JWI aus.

Die Indizierung muß nicht starr in den geschilderten drei Schritten ablaufen, sondern die verschiedenen Überlegungen laufen in der Regel verschränkt ab. Anhand des Beispiels soll nur klar werden, was beachtet werden muß. Außerdem macht diese Darstellung klar, daß der resultierende formale Kontext, der ja Grundlage für das gesamte weitere Verfahren ist, von den Entwurfsentscheidungen der Analytiker abhängt. Dies muß kein Nachteil sein, weil die mit Hilfe von BASE aufgrund verschiedener Vorstellungen erstellten Liniendiagramme die Diskussion über diese differierenden Bilder vom Untersuchungsbereich fördern können (siehe 7.2.1 auf Seite 247).

4.2.3 Formaler Kontext

Im Sinne der Formalen Begriffsanalyse werden nun die Anwendungsfälle als *formale Merkmale* behandelt und den in ihnen vorkommenden "Dingen" zugeordnet, die ihrerseits als *formale Gegenstände* interpretiert werden. So ergibt sich ein *formaler Kontext* (Definition 18 auf Seite 80).

In den ursprünglichen Arbeiten zu BASE (siehe [D-H 98a] und [D-H 98b]) waren die Rollen von formalen Gegenständen und formalen Merkmalen vertauscht. Typische mit Formaler Begriffsanalyse bearbeitete Klassifikationsaufgaben haben als Ausgangspunkt eine Menge von Gegenständen. Bei BASE bilden die Anwendungsfälle den Ausgangspunkt der Untersuchung. Deshalb wurden sie ursprünglich als formale Gegenstände behandelt. Wie in 1.6.1 auf Seite 16 dargelegt und durch Satz 9 auf Seite 85 mathematisch untermauert, spielt die Rollenvertauschung keine wesentliche Rolle für die sich ergebenden Begriffsverbände. Man betrachtet nach einer Vertauschung lediglich einen dualen Verband bzw. ein auf den Kopf gestelltes Liniendiagramm. Da in der praktischen Anwendung von BASE dem Modellierer die mathematischen Grundlagen verborgen bleiben sollen, ist das Argument bezüglich der Reihenfolge der Betrachtung von formalen Gegenständen und formalen Merkmalen hinfällig. Der Anwender von BASE soll sich nur mit Anwendungsfällen und den in sie involvierten "Dingen" beschäftigen, wobei es ihm gleichgültig sein kann, was intern als formale Gegenstände und was als formale

Merkmale behandelt wird.

Die Rollen von formalen Gegenständen und Merkmalen zu vertauschen, hatte mehrere Gründe. Zum einen erschien es unnatürlich, "Dinge" des Untersuchungsbereichs als "Merkmale" zu behandeln, während auch in der mathematischen Formalisierung von "Gegenständen" die Rede ist. Dieser Umstand fiel besonders bei der Vorstellung des Ansatzes in englischer Sprache und der damit verbundenen Suche nach geeigneten Termini ins Auge ([D-H 98b]: "In terms of FCA we now treat the use cases as formal things and the marked things as formal features."). Zum anderen entspricht die Anordnung der entstehenden Liniendiagramme nach dem Rollentausch besser den üblichen Schichtenmodellen, in denen die Datenzugriffsschicht unten, darüber die Anwendungsschicht und oben die Benutzerschicht dargestellt wird. Analog finden sich bei BASE die grundlegenden "Dinge" (als Datenelemente) unten und die allgemeinen Systemfunktionen (ausgedrückt durch Anwendungsfälle), die im späteren Software-System über die Benutzerschnittstelle angeboten werden, oben im Liniendiagramm (siehe Abbildung 4.5 auf Seite 137 und die nachfolgende Diskussion).

"*uses*"- und "*extends*"-Beziehungen zwischen Anwendungsfällen werden nach der Indizierung wie folgt berücksichtigt: Benutzt oder erweitert ein Anwendungsfall *a* einen anderen Anwendungsfall *b*, so berührt er auch alle "Dinge", die in *b* eine Rolle spielen. Diese werden ihm deshalb zusätzlich zu den ihm explizit zugeordneten ebenfalls zugesprochen. Das ist gerechtfertigt, denn eine "*uses*"-Beziehung besagt, daß *b* innerhalb von *a* eine Teilaufgabe übernimmt. "Dinge", die von *b* berührt werden, sind damit mittelbar auch in *a* involviert. Eine "*extends*"-Beziehung besagt dagegen, daß *a* eine Spezialisierung von *b* ist. Deshalb muß man davon ausgehen, daß mindestens alle "Dinge", die in *b* eine Rolle spielen, auch von *a* berührt werden.

Abb. 4.3 Behandlung von "uses"- und "extends"-Beziehungen

Im Sinne der formalen Begriffsanalyse wird dies durch eine *Merkmalsimplikation* $b \rightarrow a$ ausgedrückt, d.h. allen formalen Gegenständen, die das formale Merkmal *b* tragen, wird auch *a* zugesprochen (Definition 31 auf Seite 105). Im formalen Kontext werden einfach die Kreuze der *b*-Spalte ebenfalls in die *a*-Spalte eingetragen.

Im JWI-Beispiel entstand folgender formaler Kontext aus der oben angedeuteten Indizierung. Den Effekt der "*uses*"-Beziehungen sieht man am Beispiel des Anwendungsfalls *Bestellung bearbeiten*. Das fett gedruckte Kreuz geht nicht direkt aus der oben vorgenommenen Indizierung hervor, weil *Produkt* nicht in der Beschreibung von *Bestellung bearbeiten* vorkommt. Es wurde vom benutzten Anwendungsfall *Lagerbestand ermitteln* übernommen.

	Bestellung annehmen	Bestellung bearbeiten	Fehlbestände nachbestellen	Lagerbestand ermitteln	Lieferscheine erstellen	Maximalen und minimalen Lagerbestand definieren	Wareneingang bearbeiten	Rücklauf bearbeiten
Kundenbestellung	X	X						
Kunde	X							
Bestellposten	X	X			X		X	X
Produkt	X	**X**	X	X		X	X	X
Lagerbestand		X	X	X			X	
Lieferkartei		X			X		X	X
Wartekartei		X					X	
Vorgemerkte Menge			X					
Nachbestellung			X					
Lieferant			X				X	
Minimalbestand			X			X		
Maximalbestand			X			X		
Lieferschein					X			X
Lieferwagen					X			
Fahrtziel					X			
Lieferscheinvermerk								X

Abb. 4.4 Formaler Kontext zum JWI-Beispiel

Ein weiterer identifizierter Anwendungsfall "Reklamation bearbeiten" wurde nicht berücksichtigt, weil das zugrunde liegende Dokument über ihn keinerlei Informationen enthält. Daß die betrachteten Anwendungsfälle und damit die funktionalen Anforderungen zu Anfang der Analyse unvollständig sind, ist für reale Projekte eher typisch (vergl. [Stan 95]). Auch Jacobson sieht die inkrementelle Vervollständigung des Anwendungsfallmodells als den Normalfall an ([JCJÖ 94], S. 162/163: "We have as yet only discussed the identification of use cases. This is often a very iterative process where several attempts are made. [...] Since use cases often focus on a particular functionality of the system, it is possible to analyze the total functionality of the system in an incremental way. In this manner we can develop use cases for different functionality areas independently and later join these use cases together to form the complete requirements model."). Im Unterschied zum realen Projekt ist im hier betrachteten Beispiel jedoch kein Nachfragen bei den Fachexperten möglich und so muß die Reklamation bei der Betrachtung des Geschäfts von JWI außen vor bleiben.

Der in der Analyse aktuell betrachtete Ausschnitt des Untersuchungsbereichs ist im formalen Kontext explizit beschrieben. Vergrößert man ihn, erweitert man den Kontext.

4.2.4 Liniendiagramm des Begriffsverbands

Aus dem formalen Kontext mit Anwendungsfällen und in ihnen beteiligten "Dingen" kann ein Liniendiagramm des zugehörigen Begriffsverbands berechnet werden (Begriffsverband: Definition 22 auf Seite 84, Liniendiagramm: Definition 10 auf Seite 71 und Beispiel 7 auf Seite 89 inklusive nachfolgender Erläuterung).

Folgendes Diagramm ist aus dem Kontext in Abbildung 4.4 entstanden:

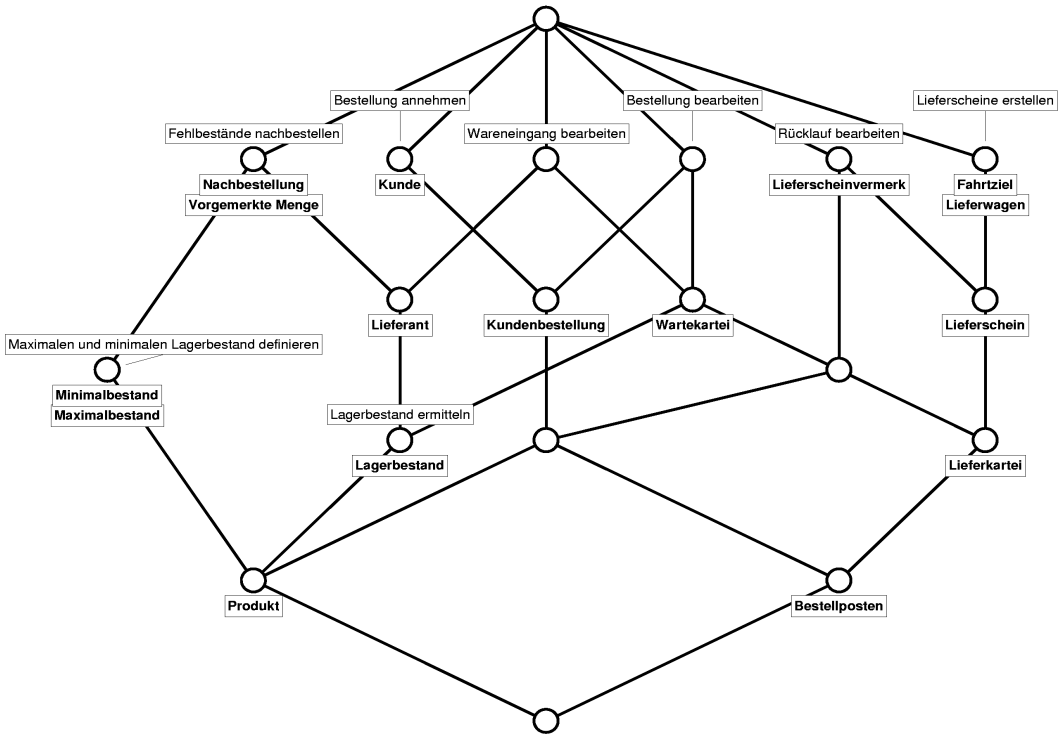


Abb. 4.5 Liniendiagramm zum JWI-Beispiel

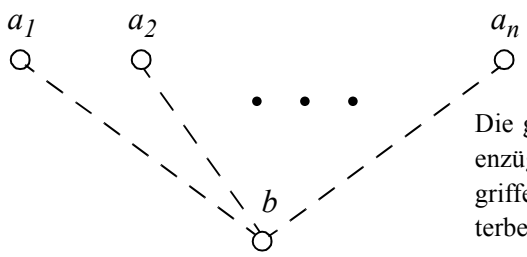
Die formalen Begriffe entstehen durch die gegenseitigen Abhängigkeiten von Anwendungsfällen und "Dingen". Zu Begriffsinhalten werden solche Anwendungsfälle gruppiert, die gemeinsam gewisse "Dinge" berühren. Dual ergeben sich die Begriffsumfänge als Mengen von "Dingen", die zusammen in einige Anwendungsfälle involviert sind. (Formaler Begriff: Definition 20 auf Seite 82)

Analyse aus funktionaler Sicht

Betrachtet man das Liniendiagramm von oben, erhält man eine *funktionale Sicht* auf das System. Die Funktionalität wird ausgedrückt durch die Anwendungsfälle. Diese stehen um so höher im Diagramm, je mehr der betrachteten "Dinge" sie berühren. Dies sind in der Regel die übergeordneten Systemfunktionen, welche die Systemfunktionalität ausmachen. Nur lokal wichtige Funktionen tauchen weiter unten auf. Von oben nach unten (bzw. zur "Diagrammitte" hin) gelesen beschreibt

das Diagramm die Systemfunktionalität also in top-down-Manier. Dies sieht man im Beispiel sehr gut. Die Coatome (untere Nachbarn des globalen Supremums, Definition 12 auf Seite 73) des Verbands sind gerade die Merkmalsbegriffe der Anwendungsfälle, die das Geschäft von JWI ausmachen. Nur die zwei Anwendungsfälle *Minimalen und maximalen Lagerbestand definieren* und *Lagerbestand ermitteln* stehen im Diagramm weiter unten. Sie betreffen unterstützende Hilfsfunktionen, die nur intern von Bedeutung, für eine Beschreibung des Geschäfts von JWI nach außen aber unerheblich sind. Bezüglich des Kriteriums, welche und wie viele "Dinge" betroffen sind, ergibt sich eine Hierarchie der Anwendungsfälle. In der Anordnung des Liniendiagramms wurde versucht, dies durch die Anordnung der Knoten in Ebenen zu unterstreichen.

Eine Unterordnung eines Anwendungsfalls unter einen anderen im Diagramm kann Ausdruck einer bestehenden "*uses*"- oder "*extends*"-Beziehung sein. In den Fällen "*a uses b*" bzw. "*a extends b*" steht jeweils aufgrund der Forderung der Implikation $b \to a$ im Liniendiagramm a über b. Dies sieht man im Beispiel gut an den Anwendungsfällen *Bestellung bearbeiten*, *Fehlbestände nachbestellen* und *Wareneingang bearbeiten*, die jeweils den Anwendungsfall *Lagerbestand ermitteln* benutzen und deshalb im Liniendiagramm über ihm stehen. Andersherum kann die Anordnung im Liniendiagramm auch Anlaß zur Überprüfung geben, ob "*uses*"- oder "*extends*"-Beziehungen bestehen, die noch nicht beschrieben sind.



Untersuchung auf:
a_1 *uses* b, ... , a_n *uses* b

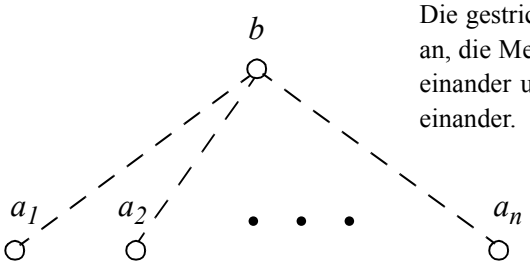
Die gestrichelten Linien deuten aufsteigende Linienzüge an, die mit $a_1, \ldots, a_n$ beschrifteten Begriffe stehen nebeneinander und nicht in Ober-/Unterbegriffsbeziehung zueinander.

Abb. 4.6 "*uses*"-Beziehungen im Liniendiagramm

"*uses*"-Beziehungen werden vor allem dann modelliert, wenn ein Anwendungsfall für mehrere andere Leistungen erbringt. Besonders wichtig für die Modellierung sind solche Situationen, wenn die erbrachte Leistung gerade die Gemeinsamkeit sonst unvergleichbarer Anwendungsfälle ausdrückt. Steht also ein Anwendungsfall unter mehreren anderen (d.h. sein Merkmalsbegriff ist $\wedge$-reduzibel, aber nicht das Supremum des Begriffsverbands, Definition 11 auf Seite 72), ist zu untersuchen, ob dies Ausdruck von bestehenden, aber noch nicht erkannten "*uses*"-Beziehungen ist. In der Situation von Abbildung 4.6 wäre also zu fragen, ob die Anwendungsfälle $a_1, \ldots, a_n$ den Anwendungsfall b benutzen, oder die aufgezeigten – über die untersuchten "Dinge" gestifteten – Datenabhängigkeiten einen anderen

Grund haben.

"*extends*"-Beziehungen finden dann Anwendung, wenn ein Anwendungsfall grundsätzliches Systemverhalten beschreibt, das in mehreren verschiedenen Spezialisierungen angeboten werden soll. Steht also ein Anwendungsfall *b* über mehreren anderen, ist zu untersuchen, ob dies Ausdruck von "*extends*"-Beziehungen ist. (Nach Definition 11 auf Seite 72 mathematisch formuliert liegt dieser Fall vor, wenn der Merkmalsbegriff μb $\vee$-reduzibel ist und es Anwendungsfälle a_1, a_2 gibt, so daß $\mu a_1, \mu a_2 < \mu b$ und $\mu a_1 \not\leq \mu a_2, \mu a_2 \not\leq \mu a_1$.[1]) In der Situation von Abbildung 4.7 wäre also zu fragen, ob die Anwendungsfälle $a_1, \ldots, a_n$ den Anwendungsfall *b* erweitern, oder ob die aufgezeigten – über die untersuchten "Dinge" gestifteten – Datenabhängigkeiten einen anderen Grund haben.



Die gestrichelten Linien deuten aufsteigende Linienzüge an, die Merkmalsbegriffe von $a_1, \ldots, a_n$ stehen nebeneinander und nicht in Ober-/Unterbegriffsbeziehung zueinander.

Untersuchung auf:
a_1 *extends b*, ... , a_n *extends b*

Abb. 4.7 "*extends*"-Beziehungen im Liniendiagramm

1. Die mathematische Formulierung für die Situation möglicher "*extends*"-Beziehungen ist komplizierter als die für "*uses*"-Beziehungen. Zusätzlich zu der Reduzibilität ist die Existenz von Anwendungsfällen a_1, a_2 mit $\mu a_1, \mu a_2 < \mu b$ und $\mu a_1 \not\leq \mu a_2, \mu a_2 \not\leq \mu a_1$ gefordert. Die entsprechende Bedingung in der Situation von "*uses*"-Beziehungen wäre, daß es Anwendungsfälle a_1, a_2 mit $\mu a_1, \mu a_2 > \mu b$ und $\mu a_1 \not\leq \mu a_2, \mu a_2 \not\leq \mu a_1$ gibt. Dies folgt aber schon aus der $\wedge$-Reduzibilität von μb, wenn man allein zusichert, daß μb nicht das globale Supremum ist. Diesen Sachverhalt sieht man wie folgt:

   Daß der Merkmalsbegriff zu einem Anwendungsfall *b* $\wedge$-reduzibel ist, heißt $\mu b = \wedge\{(A, B) \in \mathfrak{B}(G, M, I) \mid (A, B) > \mu b\}$. Ist μb nicht das Supremum des untersuchten Begriffsverbands $\mathfrak{B}(G, M, I)$ so hat μb mindestens zwei obere Nachbarn $(A_1, B_1), (A_2, B_2)$, denn bei nur einem oberen Nachbarn (A_1, B_1) wäre $\wedge\{(A, B) \in \mathfrak{B}(G, M, I) \mid (A, B) > \mu b\} = (A_1, B_1) \neq \mu b$. Weil (A_1, B_1) und (A_2, B_2) unvergleichbar sind, gibt es zwei Anwendungsfälle a_1, a_2, so daß $a_1 \in B_1 \backslash B_2$ und $a_2 \in B_2 \backslash B_1$. Wegen $B_1, B_2 \subseteq b''$ folgt $\mu a_1, \mu a_2 > \mu b$. Weiter ist $\mu a_1 \not\leq \mu a_2, \mu a_2 \not\leq \mu a_1$, weil etwa in dem Fall $\mu a_1 \leq \mu a_2$ im Widerspruch zur Wahl von a_1, a_2 auch $a_2 \in a_2'' \subseteq a_1'' \subseteq B_1$ wäre.

   In der Überprüfung auf "*extends*"-Beziehungen kann aber die Zusatzbedingung nicht durch die einfachere Bedingung "μb ist nicht das Infimum des Begriffsverbands" ersetzt werden. Mehrere untere Nachbarn von μb besitzen untereinander unvergleichbare Untermengen von b' als Begriffsumfänge. Diese Begriffsumfänge gruppieren "Dinge" die der Anwendungsfall *b* mit anderen gemeinsam hat. Die Merkmalsbegriffe der anderen beteiligten Anwendungsfälle sind aber nicht die unteren Nachbarn von μb selber, wenn noch andere "Dinge" in diese Anwendungsfälle involviert sind. Dann sind nämlich die Merkmalsbegriffe zu *b* und den angesprochenen anderen Anwendungsfällen unvergleichbar. Dies sieht man in Abbildung 4.5 auf Seite 137 am Beispiel des Merkmalsbegriffs zum Anwendungsfall *Rücklauf bearbeiten*. Er besitzt zwei untere Nachbarn, ist also $\vee$-reduzibel, aber kein kleinerer Begriff ist ein Merkmalsbegriff.

Analyse aus Datensicht

Dual zur oben diskutierten funktionalen Sicht erhält man von unten betrachtet eine *Datensicht* auf das System. Die "Dinge", die im Untersuchungsbereich wichtig sind (und deshalb hoffentlich bei der Indizierung der Anwendungsfälle von den Fachexperten genannt wurden), sind von unten nach oben in ihrer Bedeutung für das System angeordnet. Unten stehen die grundlegenden "Dinge", die durch weite Teile der Systemfunktionalität betroffen sind. Die unten im Diagramm auftretenden "Dinge" sind damit die prominentesten Klassenkandidaten. Klassenkandidaten, die nur für kleinere Teile der Systemfunktionalität wichtig sind, stehen weiter oben. Sind sie in nur einen Anwendungsfall involviert, so sind sie diesem direkt zugeordnet. Primäres Anliegen sollte es aber sein, zuerst die systemweit wichtigen Klassen zu modellieren, also von unten im Diagramm zu beginnen. Von unten nach oben (bzw. zur "Diagrammitte" hin) gelesen ergibt sich ein top-down-Bild der Datensicht – von den allgemeinen zu den speziellen Dingen.

So erhalten im Beispiel die Atome (obere Nachbarn des globalen Infimums, Definition 12 auf Seite 73) die Beschriftung *Produkt* bzw. *Bestellposten*. Diese "Dinge" spielen für das Geschäft von JWI die zentrale Rolle. Wichtig sind die Weinsorten, die von JWI vertrieben werden, und die jeweiligen Bestellmengen für eine Sorte. Naheliegend ist es, solche Angaben in Klassen *Produkt* und *Bestellposten* zu modellieren.

Zu jeder Weinsorte ist auch die Lagerverwaltung zu erledigen. Natürliche Attribute einer Klasse *Produkt* sind dafür *Lagerbestand*, *Minimalbestand*, *Maximalbestand*, *Vorgemerkte Menge*, *Lieferant* (letztere zwei für die Abwicklung von Nachbestellungen für das Lager). Sie stehen alle im Diagramm oberhalb von *Produkt*. Geht man davon aus, daß Attribute einer (potentiellen) Klasse zusammen mit dieser auch schon als "interessante Dinge" in die Analyse einbezogen wurden, und daß bei der Indizierung der Anwendungsfälle jedesmal angegeben wurde, daß mit einem Attribut auch die betreffende Klasse involviert ist, erhält man diese typische Anordnung im Diagramm. Mathematisch ausgedrückt sind dann die Gegenstandsbegriffe der Attribute im Hauptfilter des Gegenstandsbegriffs der Klasse enthalten (Definition 5 auf Seite 66). Legt man andersherum ein Attribut einer Klasse fest, das im Diagramm nicht oberhalb der Klasse vorkommt, gemahnt das Diagramm zu einer Untersuchung, ob diese Modellierung gerechtfertigt ist. Es ist dann nachzufragen, ob die Klasse nur bei der Indizierung nicht beachtet wurde, aber eigentlich immer mitbetroffen ist, oder ob das Attribut in einigen Anwendungsfällen wirklich als eigenständig behandelt wird. Im zweiten Fall ist zu erwägen, ob eine Modellierung als eigene Klasse nicht vorzuziehen ist, um die Eigenständigkeit herauszustellen. In diesem Sinne hilft BASE auch, diesen Aspekt bestehender oder im Aufbau befindlicher Modelle zu überprüfen.

Die oben erwähnte Annahme über die Rolle von potentiellen Klassen und ihren Attributen bei der Indizierung muß nicht immer zutreffen. Sie erscheint aber sinnvoll, denn das Involviertsein eines "Dings" in einen Anwendungsfall bedeutet oft eine Aktion, die dieses "Dings" verändern kann. Handelt es sich dabei um ein Attribut einer Klasse, wurde mit dem entsprechenden Attributwert auch das betreffende Objekt verändert, womit die Klasse auch betroffen ist. Andersherum könnte man argumentieren, daß eine Klasse immer alle ihre Attribute umfaßt und so diese auch immer mit der Klasse zusammen involviert sind. Dies wird aber bei der in der Indizierung nicht berücksichtigt, da nur die für einen spezifischen Anwendungsfall interessanten Attribute diesem zugeordnet werden. Häufig treten aber Objekte der gleichen Klasse in verschiedenen Rollen auf und die betreffende Klasse umfaßt mehrere Teile, die auch einzeln in Funktionen benötigt werden. In einem solchen Fall werden bei der Indizierung eben nur die Attribute aus dem einen interessanten Teil erfaßt, die übrigen aber nicht. (Vergl. auch 6.2 auf Seite 228)

Aus dem Diagramm läßt sich eine Ordnung der "Dinge" nach ihrer Bedeutung für das System ablesen. Sie in dieser Ordnung zu behandeln, erscheint sinnvoll, um zuerst die grundsätzliche Systemstruktur zu modellieren, bevor man sich auf Einzelheiten einläßt (vergl. auch 4.5.1 auf Seite 159). Das Diagramm veranschaulicht Datenabhängigkeiten der Anwendungsfälle, die über die betrachteten "Dinge" entstehen. Bei der Modellierung oder Veränderung von Klassen, die ein direktes Gegenstück unter den betrachteten "Dingen" besitzen, sind so die betroffenen Anwendungsfälle direkt ablesbar.

4.3 Funktionale Zerlegung

4.3.1 Funktionale Verfeinerung des Anwendungsfallmodells

Die bisher betrachteten Klassenkandidaten sind wenig überraschend – sie entsprechen ausnahmslos solchen "Dingen", die bei der Indizierung der Anwendungsfälle schon als wichtig herausgestellt worden waren. Zu erwarten ist, daß zwischen diesen "Dingen" schon viele der Entitätsklassen im Sinne von Jacobson (siehe 2.6.1, Seite 54) auftauchen. Möglicherweise werden bei dieser Betrachtung auch schon Attribute behandelt. Insgesamt ist die Sicht auf das System aber noch sehr grob. Eine weitergehende und im Sinne der Objektorientierung zentrale Idee besteht darin, auch die Identifikation von Operationen der Klassen anzugehen. Zu einer vollständigen Klassenspezifikation reicht nicht allein die Angabe von Attributen, sondern auch die Operationen müssen festgelegt werden. Zudem möchte man den Ablauf von Anwendungsfällen mit Hilfe von Sequenzdiagrammen formaler beschrei-

ben, als das bisher durch Umgangssprache geschehen ist. Deshalb soll nun eine funktionale Zerlegung der Anwendungsfälle betrachtet werden.

Dazu werden die Anwendungsfälle so weit in Einzelschritte zerlegt, bis diese als Operationen gewissen Klassen zugeteilt werden können. Die Abfolge der Einzelschritte innerhalb eines Anwendungsfalls kann dann durch die Aufzeichnung von Operationsaufrufen bzw. Nachrichten zwischen den beteiligten Objekten in einem Sequenzdiagramm dargestellt werden.

In BASE wird die Festlegung der Klassen zeitlich relativ weit nach hinten verschoben, um sie erst vorzunehmen, wenn schon ein tieferes Verständnis des Untersuchungsbereichs erreicht ist. Die untersuchten Anwendungsfälle können aber nach dem in 2.2.1 dargestellten Vorgehen auch ohne Bezug auf Klassen eines Klassenmodells funktional zerlegt werden. Wie in 4.2.1 auf Seite 128 diskutiert, ist auch für die an der Entwicklung beteiligten Fachexperten die funktionale Zerlegung eine gewohnte Vorgehensweise – wie etwa bei der Beschreibung von Geschäftsprozessen durch Ereignis-Prozeß-Ketten ([KNS 92]) praktiziert.

Man zerlege also die Anwendungsfälle in Schritte und führe diese Zerlegung rekursiv weiter durch. Die Zerlegung führt so zu funktional untergeordneten "*Unter-Anwendungsfällen*", die selber weiter zerlegt werden und *Funktionen*, die selber nicht weiter zerlegt werden. (Der Terminus "Funktion" soll unterstreichen, daß noch keine Zuordnung von Operationen zu Klassen vorliegt.) Für den betrachteten formalen Kontext aus Anwendungsfällen und involvierten "Dingen" bedeutet dies, daß mit jeder Zerlegung eines Anwendungsfalls neue Unter-Anwendungsfälle und Funktionen als formale Merkmale hinzukommen.

Die folgende Darstellung bezieht sich immer auf einen ersten einfachen Zerlegungsschritt, in dem Anwendungsfälle in Funktionen zerlegt werden. So können die Rollen von funktional übergeordneten Elementen (Anwendungsfällen) und untergeordneten Elementen (Funktionen) sprachlich unterschieden werden. Dies dient nur der einfacheren Formulierung.[1]

Für die jeweils schon vor einem Zerlegungsschritt betrachteten "Dinge" (formale Gegenstände) muß untersucht werden, ob sie von der neuen Funktion berührt werden. Außerdem wird eine solche Zerlegung zur Identifikation neuer "Dinge" führen, auf welche die Behandlung der neuen Funktionen die Aufmerksamkeit von Entwicklern und Fachexperten lenkt. Da der funktional übergeordnete Anwendungsfall und die neu betrachtete Funktion in einer "*uses*"-Beziehung stehen (Eine

1. Mit den Festlegungen, daß Funktionen in weiteren Zerlegungsschritten zu Unter-Anwendungsfällen werden können und daß Unter-Anwendungsfälle innerhalb der Zerlegung selber wieder Anwendungsfälle sind, ist die Darstellung auf Zerlegungsschritte tieferer Ebenen übertragbar.

in der Zerlegungshierarchie niedriger stehende Funktion wird von den übergeordneten benutzt, um Teilaufgaben zu erfüllen bzw. Einzelschritte auszuführen.), müssen die neu identifizierten "Dinge" in der Inzidenzrelation des formalen Kontexts auch dem übergeordneten Anwendungsfall zugeschrieben werden (vergl. 4.2.3 auf Seite 135). Die funktionale Verfeinerung erfolgt in einer top-down-Strategie und ist dementsprechend im Liniendiagramm von oben zur "Diagrammitte" hin nachzuvollziehen (siehe Diskussion von "*uses*"-Beziehungen in 4.2.4 auf Seite 138).

Für die anderen untersuchten Anwendungsfälle ist eigens zu untersuchen, welche der neu identifizierten "Dinge" sie betreffen. So erhält man durch die Zerlegung einen neuen formalen Kontext und damit auch ein neues Liniendiagramm.

Im folgenden werden die verschiedenen Situationen, die durch funktionale Zerlegung von Anwendungsfällen entstehen können, anhand von Beispielen dargestellt.

Eine besonders einfache Zerlegung

Im Beispiel von JWI muß im Anwendungsfall *Bestellung annehmen* eine *Kundenbestellung* angelegt werden. Innerhalb dieses Anlegens werden einzelne *Bestellposten* erzeugt. Dies beschreibt eine erste Zerlegung dieses Anwendungsfalls.

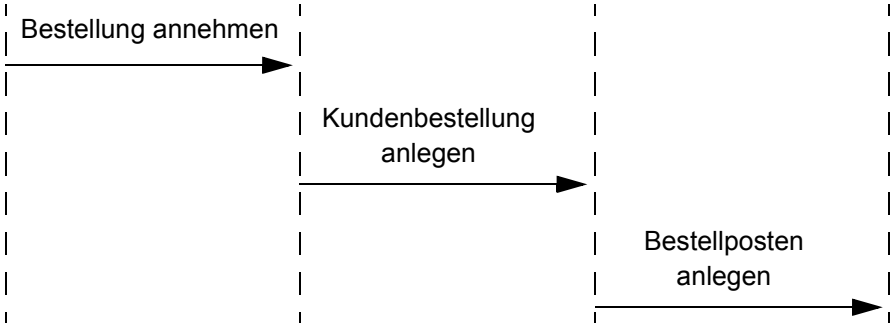


Abb. 4.8 Funktionale Zerlegung des Anwendungsfalls *Bestellung annehmen*

Hier – wie auch im folgenden – ist statt der üblichen baumartigen Darstellung von funktionalen Zerlegungen eine den Sequenzdiagrammen (siehe 2.4.2 auf Seite 38) angelehnte Notation benutzt. Der Grund dafür ist zum einen, daß (vollständige) Sequenzdiagramme ein Ergebnis der Analyse sein sollen und sie im wesentlichen aus einer solchen funktionalen Zerlegung entwickelt werden. Deshalb ist es sinnvoll schon die zeitliche Abfolge festzuhalten, auch wenn sie für die Indizierung in BASE keine Rolle spielt. Zum anderen ergibt sich die übliche baumartige Anordnung sowieso schon im Liniendiagramm. Der entscheidende Unterschied der hier benutzten Darstellung zu (vollständigen) Sequenzdiagrammen ist, daß die senkrechten Linien nicht Lebenslinien von Objekten darstellen, sondern allein zur Be-

schreibung der Aufrufhierarchie dienen. Insbesondere kann es vorkommen, daß mehrere vertikale Linien aus diesen Zerlegungsdiagrammen im späteren Sequenzdiagramm zusammenfallen, wenn nämlich die entsprechenden Funktionen von ein und demselben Objekt ausgeführt werden.

Das *Anlegen einer Kundenstellung* betrifft den *Kunden* und die *Kundenbestellung* selber. Ein *Bestellposten* bezieht sich auf ein *Produkt*, von dem eine gewisse Menge bestellt wird. Der Bestellposten wird innerhalb der betreffenden Kundenbestellung angelegt. Bei der *Erzeugung eines Bestellpostens* sind damit die "Dinge" *Bestellposten*, *Produkt* und *Kundenbestellung* involviert. Außerdem spielt ein neues "Ding" – die *Bestellmenge* – eine Rolle. Da die Funktion *Bestellposten anlegen* der Funktion *Kundenbestellung anlegen* funktional untergeordnet ist, wird eine Implikation *Bestellposten anlegen* → *Kundenbestellung anlegen* eingeführt und damit alle in *Bestellposten anlegen* betroffenen "Dinge" an *Kundenbestellung anlegen* "vererbt" (siehe Diskussion von "*uses*"-Beziehungen in 4.2.3 auf Seite 135). Analog ergibt sich die Implikation *Kundenbestellung anlegen* → *Bestellung annehmen* mit der entsprechenden Konsequenz. Das neue "Ding" *Bestellmenge* wird so auch dem übergeordneten Anwendungsfall *Bestellung annehmen* zugeordnet. Für die anderen Anwendungsfälle muß erst untersucht werden, ob in ihnen die *Bestellmenge* eine Rolle spielt. Bei dem Anwendungsfall *Bestellung bearbeiten* ist dies der Fall, weil anhand der *Bestellmenge* entschieden werden muß, ob von dem bestellten *Produkt* genug auf Lager ist, um den Bestellposten direkt ausliefern zu können. Nach der Verfeinerung ergibt sich das Liniendiagramm in Abbildung 4.9.

Durch die funktional detailliertere Betrachtung wurde das Modell um ein neu zu betrachtendes "Ding" als zusätzlichen Klassen- bzw. Attributkandidat und zwei Operationskandidaten bereichert. Die Struktur des Liniendiagramms ist in diesem Fall jedoch unverändert geblieben. (Gegenüber Abbildung 4.5 auf Seite 137 unveränderte Diagrammelemente sind grau dargestellt und die neuen schwarz hervorgehoben.)

Die Funktion *Kundenbestellung anlegen* benutzt genau die "Dinge", die in den Anwendungsfall *Bestellung annehmen* involviert sind. Das neue "Ding" *Bestellmenge* ist genau den Anwendungsfällen und Funktionen zugesprochen worden, welche die *Kundenbestellung* berühren. Schließlich betrifft die neue Funktion *Bestellposten anlegen* gerade die "Dinge" *Produkt*, *Bestellposten*, *Kundenbestellung* (und *Bestellmenge*). Im bisherigen Modell war die entsprechende Gegenstandsmenge ohne den neuen formalen Gegenstand *Bestellmenge* der Begriffsumfang, der die Gemeinsamkeit der Anwendungsfälle *Bestellung annehmen* und *Bestellung bearbeiten* beschrieb. Also könnte *Bestellposten anlegen* eine gemeinsam von beiden Anwendungsfällen genutzte Funktion sein (Vergl. 4.2.4 auf Seite 138), was hier jedoch nicht der Fall ist, weil innerhalb der Bearbeitung von Bestellungen keine Be-

stellposten neu angelegt werden. Bis auf diese Hinzunahme der neuen Funktionen und des neuen "Dings" haben sich daher keine neuen Kombinationen von Anwendungsfällen und Funktionen ergeben, die gemeinsam gewisse "Dinge" angehen, und es sind auch keine neuen Gruppierungen von "Dingen" entstanden, die alle zusammen in einige Anwendungsfälle oder Funktionen involviert sind.

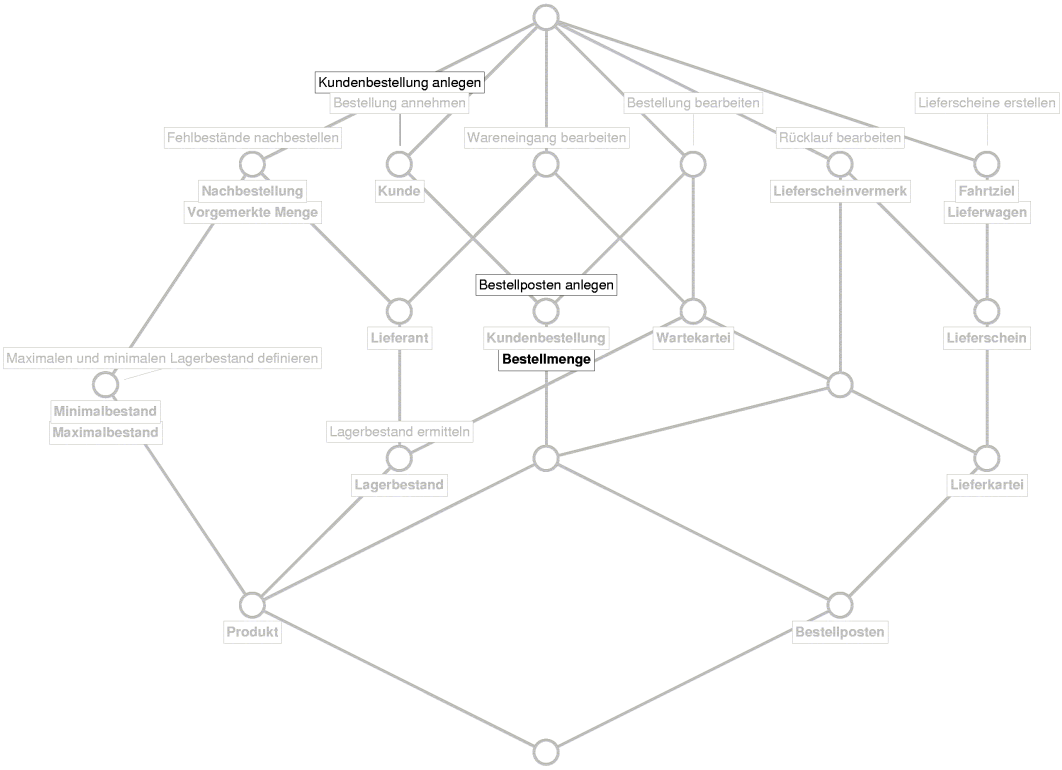


Abb. 4.9 Anwendungsfall *Bestellung annehmen* verfeinert

Eine diagrammändernde Zerlegung

Aus der Betrachtung neuer Funktionen können aber auch neue Begriffsinhalte und Begriffsumfänge resultieren, wenn nämlich eine neue Funktion eine Kombination von "Dingen" betrifft, die im bisherigen Begriffsverband noch nicht als Begriffsumfang vorkam oder wenn die funktional detailliertere Betrachtung einem schon vorher betrachteten Anwendungsfall ein zusätzliches "Ding" zuordnet, so daß bei ihm ein neuer Begriffsumfang entsteht. Außerdem kann ein neu betrachtetes "Ding" in eine Menge von Anwendungsfällen involviert sein, die vorher noch nicht als Begriffsinhalt vorgekommen ist. In diesen Fällen ändert sich die Struktur des Liniendiagramms, so daß die neu aufgespürten Abhängigkeiten widergespiegelt werden.

In dem Fall, daß nur neue Funktionen ohne neue "Dinge" zusätzlich betrachtet werden oder die neu gefundenen "Dinge" nur den neuen Funktionen und dem gera-

de zerlegten Anwendungsfall – aber keinem anderen schon vorher betrachteten Anwendungsfall – zugesprochen werden, beschränken sich die Änderungen des Diagramms auf die Teilstruktur unterhalb des Merkmalsbegriffs des zerlegten Anwendungsfalls (mathematisch gesprochen auf das Hauptideal dieses Begriffs, Definition 5 auf Seite 66). Dies sieht man am folgenden Beispiel.

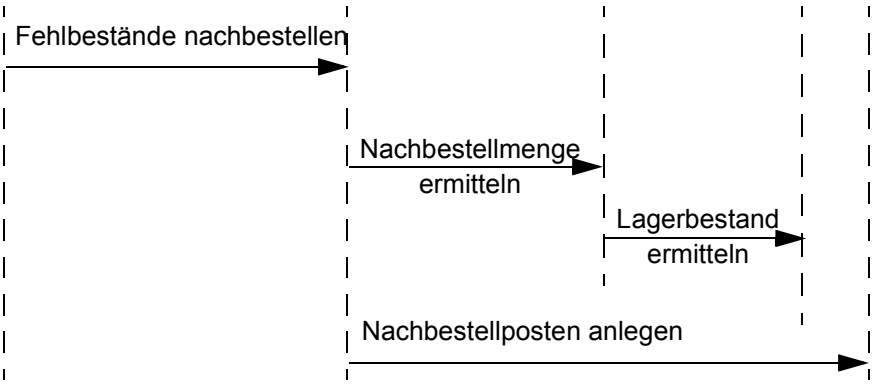


Abb. 4.10 Funktionale Zerlegung des Anwendungsfalls *Fehlbestände nachbestellen*

Betrachtet wird eine funktionale Zerlegung des Anwendungsfalls *Fehlbestände nachbestellen*. Als Einzelschritte werden dabei das Berechnen der nachzubestellenden Menge (*Nachbestellmenge = Maximalbestand + Vorgemerkte Menge - Lagerbestand*, wenn *Vorgemerkte Menge* > 0 oder *Lagerbestand* < *Minimalbestand*, sonst 0) und das Anlegen des entsprechenden Nachbestellpostens für ein Produkt betrachtet.

Das resultiert – wiederum ausgehend vom ursprünglichen unverfeinerten Liniendiagramm in Abbildung 4.5 auf Seite 137 – in dem Liniendiagramm in Abbildung 4.11. (Änderungen sind wieder schwarz hervorgehoben und übernommene Diagrammelemente grau dargestellt.)

Die beiden durch die Zerlegung entstandenen Funktionen trennen die "Dinge" *Nachbestellung* und *Vorgemerkte Menge*, die vor der Zerlegung nur gemeinsam in den Anwendungsfall *Fehlbestände nachbestellen* involviert waren. Also sind neue Begriffsumfänge entstanden und damit die Struktur des Liniendiagramms verändert worden. Als neues "Ding" wurde die *Nachbestellmenge* berücksichtigt, aber nur den neuen Funktionen (und damit auch dem zerlegten Anwendungsfall) zugesprochen. Das Diagramm zeigt deutlich, wie sich alle Änderungen gegenüber dem ursprünglichen Liniendiagramm aus Abbildung 4.5 auf Seite 137 auf das Hauptideal des Merkmalsbegriffs zu *Fehlbestände nachbestellen* beschränken. In diesem Sinne sind die Änderungen lokal.

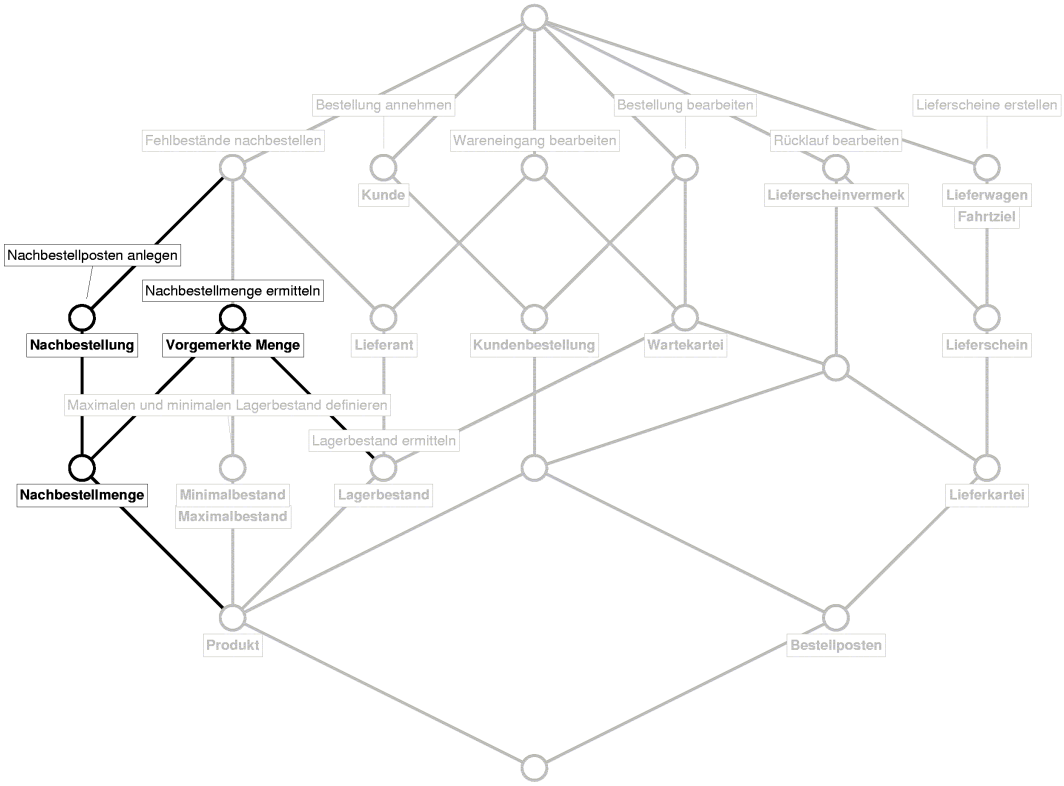


Abb. 4.11 Anwendungsfall *Fehlbestände nachbestellen* verfeinert

Eine Zerlegung mit globalem Effekt

In der Praxis kann es auch durchaus vorkommen, daß sich die ursprüngliche Indizierung als zum Teil unzutreffend erweist, weil einer bei der Zerlegung entdeckten Funktion ein schon vorher betrachtetes "Ding" zugeschrieben wird, das aber vorher nicht als im übergeordneten Anwendungsfall involviert erkannt worden war. In diesem Fall und in dem oben angesprochenen Fall, daß schon betrachteten Anwendungsfällen neu entdeckte "Dinge" zugesprochen werden, können sich auch Änderungen außerhalb des Hauptideals des zerlegten Anwendungsfalls ergeben. Neue Knoten des Liniendiagramms können auch in diesen Fällen nur unterhalb des zerlegten Anwendungsfalls entstehen, weil neue Begriffsinhalte nur durch Hinzunahme dieses Anwendungsfalls (abgesehen von den neuen Funktionen) zu erreichen sind. Gelöscht werden können nur obere Nachbarn (Definition 9 auf Seite 70) von Knoten unterhalb des Anwendungsfalls, weil Begriffsumfänge nur dadurch verloren werden können, daß durch Hinzunahme des zerlegten Anwendungsfalls ein schon vorher vorhandener Inhalt entsteht. (Da sich diese Inhalte nur durch ein Element unterscheiden, müssen die betreffenden Begriffe Nachbarn sein.)

Der Anwendungsfall *Bestellung bearbeiten* kann z.B. wie folgt zerlegt werden:

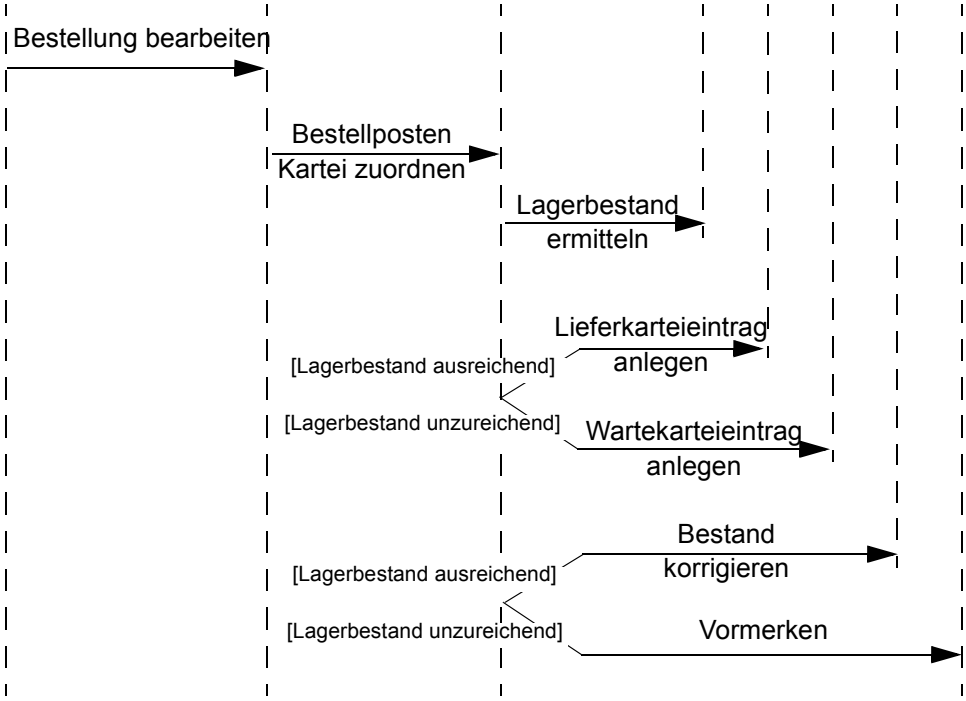


Abb. 4.12 Funktionale Zerlegung des Anwendungsfalls *Bestellung bearbeiten*

Abbildung 4.13 enthält ein entsprechend der angegebenen Zerlegungen der Anwendungsfälle *Bestellung annehmen* und *Bestellung bearbeiten* verfeinertes Liniendiagramm. Diesmal wurde die Verfeinerung ausgehend von den schon einmal verfeinerten Modell in Abbildung 4.9 auf Seite 145 vorgenommen, weil der Anwendungsfall *Bestellung bearbeiten* ebenfalls das bei der Verfeinerung von *Bestellung annehmen* neu identifizierte "Ding" *Bestellmenge* betrifft. (Änderungen sind wieder schwarz hervorgehoben und übernommene Diagrammelemente grau dargestellt.)

Bei diesem Verfeinerungsschritt hat sich die Struktur des Liniendiagramms erheblich geändert. Durch die genauere funktionale Betrachtung sind zusätzliche Schichten zur Anordnung der Knoten hinzugekommen. Die starke Verzweigung unterhalb von *Bestellung bearbeiten* dokumentiert die weitgehenden Datenabhängigkeiten der Einzelschritte in dieser Bearbeitung. Dies spiegelt wider, daß bei der Bearbeitung von Bestellungen die wesentlichen Komponenten des späteren Systems zusammenwirken müssen. Die Annahme von Bestellungen, die Verwaltung des Lagers und der Bestände der einzelnen Produkte und die Auslieferung von Bestellungen können weitgehend voneinander getrennt behandelt werden. Bei der Bestellungsbearbeitung sind jedoch alle diese Tätigkeitsgebiete mit betroffen.

Aufgedeckt wurde bei der Zerlegung das Versäumnis, die *Vorgemerkte Menge* in-

nerhalb des Anwendungsfalls *Bestellung bearbeiten* zu beachten (Ist nicht genügend Wein auf Lager, um einen Bestellposten sofort ausliefern zu können, muß bei der Bestellungsbearbeitung die *Vorgemerkte Menge* entsprechend heraufgesetzt werden.). Durch die Beachtung dieses Umstands erfolgt die Verbindung von *Fehlbestände nachbestellen* und *Bestellung bearbeiten* im neuen Diagramm über einen neuen höheren Knoten und der Gegenstandsbegriff zur *Vorgemerkten Menge* liegt nun im Gegensatz zu vorher im Hauptideal von *Bestellung bearbeiten*. Sein oberer Nachbar ist neu erzeugt worden, da {*Fehlbestände nachbestellen*, *Bestellung bearbeiten*} zuvor kein Begriffsinhalt war. Alle anderen Veränderungen des Diagramms sind allein der funktionalen Zerlegung zuzuschreiben und finden sich im Hauptideal zu *Bestellung bearbeiten*. Das liegt darin begründet, daß die ursprüngliche Indizierung nur in bezug auf die *Vorgemerkte Menge* geändert wurde. (Allerdings sind die Positionen der Merkmalsbegriffe zu den Anwendungsfällen *Bestellung annehmen* und *Bestellung bearbeiten* vertauscht, um die Abhängigkeiten zwischen *Bestellung bearbeiten* und *Fehlbestände nachbestellen* übersichtlicher darstellen zu können.)

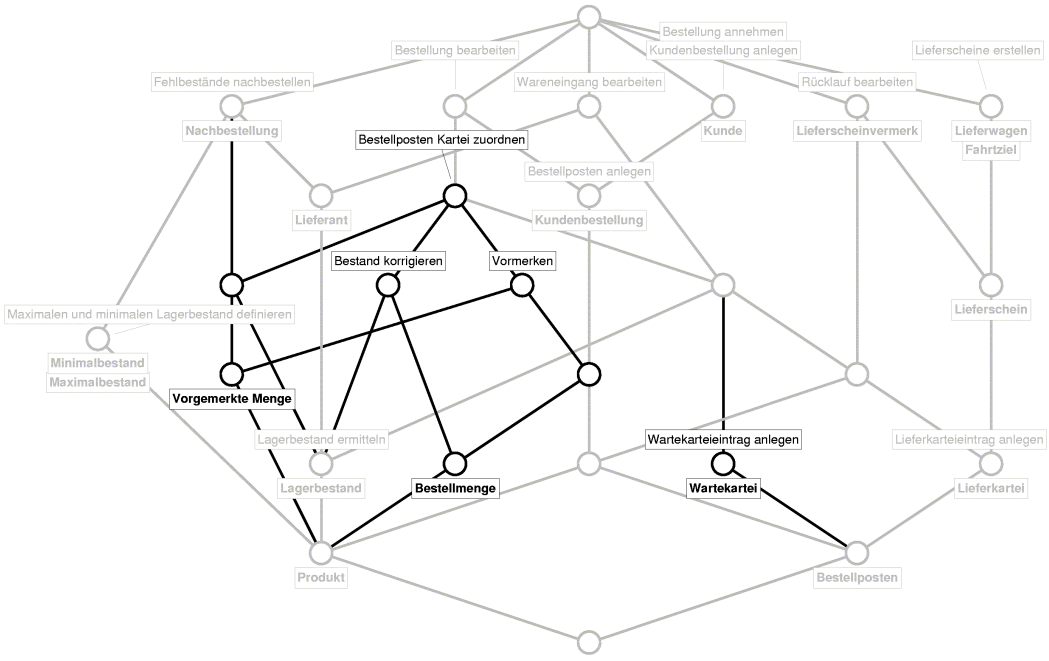


Abb. 4.13 Anwendungsfälle *Bestellung annehmen* und *Bestellung bearbeiten* verfeinert

Abbildung 4.14 zeigt die Änderung aufgrund der korrigierten Indizierung im nicht verfeinerten Diagramm noch einmal einzeln. Ihr zugrunde liegt der ursprüngliche Kontext aus Abbildung 4.4 auf Seite 136, der durch die zusätzliche Zuordnung der *Vorgemerkten Menge* zum Anwendungsfall *Bestellung bearbeiten* korrigiert wur-

de. So sieht man deutlich die Diagrammänderung, die nicht direkt der funktionalen Zerlegung von *Bestellung bearbeiten* zuzuschreiben ist.

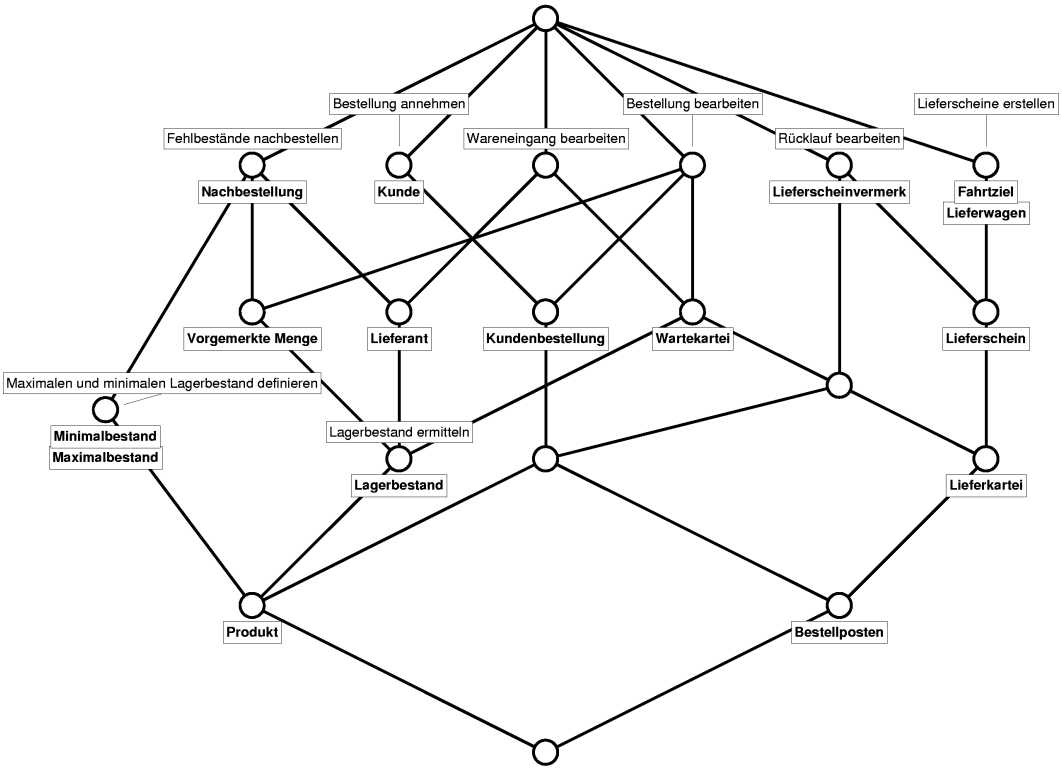


Abb. 4.14 Korrigiertes Liniendiagramm zum JWI-Beispiel

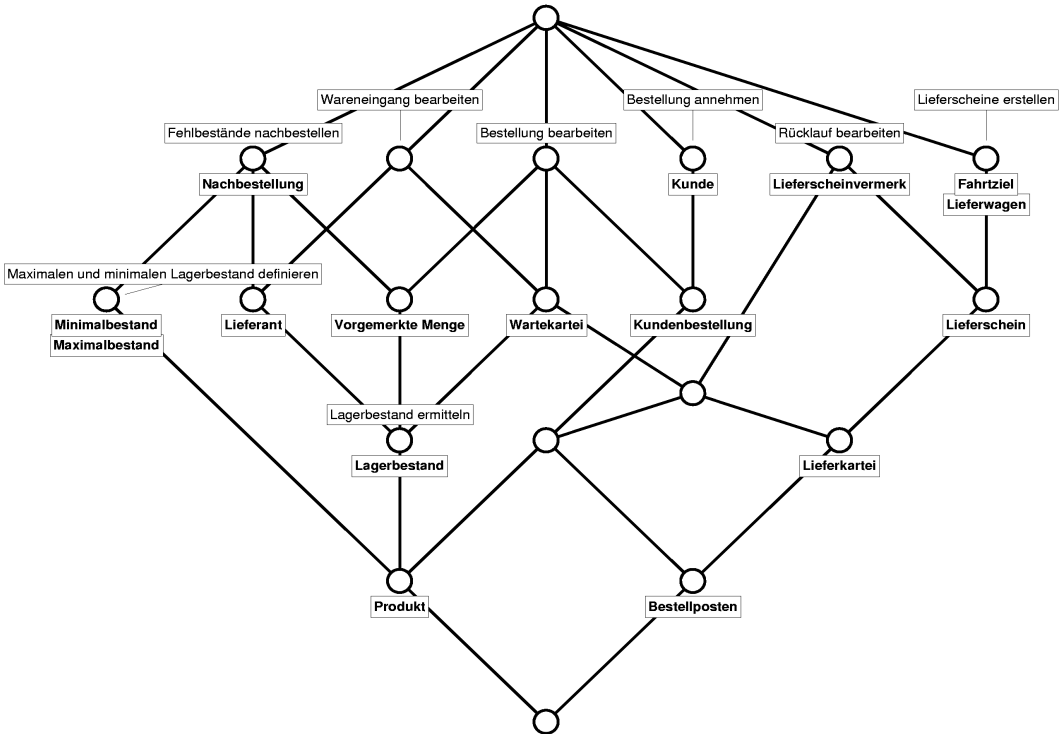


Abb. 4.15 Optimiertes Liniendiagramm zum JWI-Beispiel

Der einzige Unterschied zu Abbildung 4.5 auf Seite 137 ist der neue Gegenstandsbegriff zur *Vorgemerkten Menge*.

Um eine ansprechendere Struktur zur erhalten, kann man die Anordnung – insbesondere die Reihenfolge der Anwendungsfälle – im Liniendiagramm etwas verändern. (siehe Abbildung 4.15)

4.3.2 Abbruchkriterium für die funktionale Zerlegung

Für einen rekursiven Zerlegungsprozeß wie den hier betrachteten benötigt man ein Kriterium, um die Rekursion an den passenden Stellen abzubrechen. In dem hier vorliegenden Fall ist also die Frage zu beantworten, wann die funktionale Zerlegung für die Analyse hinreichend detailliert vorgenommen ist. Das angestrebte Kriterium dafür ist, daß die Einzelschritte durch Operationen implementierbar sind, die jeweils eindeutig einer Klasse zugeordnet werden können. Die Erfüllung dieses Kriteriums kann vollständig nur überprüft werden, wenn schon ein Klassenmodell vorliegt. Davon kann man in BASE eben nicht ausgehen. Auch in den herkömmlichen Methoden ist das nur bedingt der Fall, weil die Beschäftigung mit den Anwendungsfällen ja gerade zu einem tragfähigen Klassenmodell führen soll. Letztendlich bleibt also die Entscheidung, wann der Prozeß der funktionalen Zerlegung zu beenden ist, allein der Intuition des Entwicklers überlassen. BASE bietet ihm jedoch an dieser Stelle ein Hilfsmittel.

Ziel ist es, die innerhalb der Zerlegung auftretenden Funktionen entsprechenden Klassen zuzuordnen. Wichtigste erste Kandidaten für Klassen sind die in der Behandlung der Anwendungsfälle einbezogenen "Dinge". Nimmt man an, daß eines dieser "Dinge" später als Klasse modelliert wird und daß weiter eine Funktion betrachtet wird, die nur dieses "Ding" berührt, so ist diese Funktion wahrscheinlich gerade als Operation dieser Klasse festzulegen. In diesem Sinne sollten also die betrachteten "Dinge" mit Hilfe der betrachteten Funktionen getrennt werden. Kombinationen von "Dingen", die von jeder betrachteten Funktion immer gemeinsam berührt werden, sollten zusammen in einer Klasse modelliert werden können. Für solche Kombinationen ist demnach zu fragen, ob sie für den gesamten Untersuchungsbereich als zusammengehörig anzusehen sind. Ist das nicht der Fall, sollte in die funktionale Zerlegung eine Funktion eingeschlossen werden, die einige "Dinge" aus der Kombination berührt, andere aber nicht. Als mathematisches Hilfsmittel zur Überprüfung dieses Kriteriums dienen Gegenstandsimplikationen (Definition 31 auf Seite 105).

Gegenstandsimplikationen innerhalb des Umfangs des Merkmalsbegriffs des zerlegten Anwendungsfalls werden berechnet. Jede Implikation bedeutet eine Frage an Entwicklern und Fachexperten. Diese entscheiden dann für die Implikationen,

ob diese fachlich begründet sind. Eine Gegenstandsimplikation $X \rightarrow Y$ zwischen zwei Gegenstandsmengen X und Y bedeutet hier, daß alle Funktionen, die alle "Dinge" aus X berühren, auch alle "Dinge" aus Y betreffen. Entsprechend werden Entwickler und Fachexperten gefragt, ob dies zutreffend ist. Gibt es innerhalb der Abfolge der Schritte des betrachteten Anwendungsfalls noch nicht erfaßte Funktionen, für die das nicht gilt, müssen diese bei der Verfeinerung betrachtet werden. Können dagegen alle Implikationen bejaht werden, kann die funktionale Zerlegung des betreffenden Anwendungsfalls in bezug auf die bis dahin betrachteten involvierten "Dinge" als abgeschlossen betrachtet werden.

Die Duquenne-Guigues-Basis der Gegenstandsimplikationen (Satz 14 auf Seite 109) zum Liniendiagramm in Abbildung 4.15 bzw. dem zugrundeliegenden formalen Kontext besteht aus:

1. {Lieferscheinvermerk} → {Bestellposten,Produkt,Lieferkartei,Lieferschein}
2. {Fahrtziel} → {Bestellposten,Lieferkartei,Lieferschein,Lieferwagen}
3. {Lieferwagen} → {Bestellposten,Lieferkartei,Lieferschein,Fahrtziel}
4. {Lieferschein} → {Bestellposten,Lieferkartei}
5. {Maximalbestand} → {Produkt,Minimalbestand}
6. {Minimalbestand} → {Produkt,Maximalbestand}
7. {Lieferant} → {Produkt,Lagerbestand}
8. {Nachbestellung} →
   {Produkt,Lagerbestand,Vorgemerkte Menge,Lieferant,Minimalbestand,Maximalbestand}
9. {Vorgemerkte Menge} → {Produkt,Lagerbestand}
10. {Wartekartei} → {Bestellposten,Produkt,Lagerbestand,Lieferkartei}
11. {Lieferkartei} → {Bestellposten}
12. {Lagerbestand} → {Produkt}
13. {Produkt,Lagerbestand,Minimalbestand,Maximalbestand} →
    {Vorgemerkte Menge,Nachbestellung,Lieferant}
14. {Produkt,Lagerbestand,Vorgemerkte Menge,Lieferant} →
    {Nachbestellung,Minimalbestand,Maximalbestand}
15. {Bestellposten,Produkt,Minimalbestand,Maximalbestand} →
    {Kundenbestellung,Kunde,Lagerbestand,Lieferkartei,Wartekartei,
    Vorgemerkte Menge,Nachbestellung,Lieferant,Lieferschein,Lieferwagen,Fahrtziel,
    Lieferscheinvermerk}
16. {Bestellposten,Produkt,Lieferkartei,Lieferschein} → {Lieferscheinvermerk}
17. {Bestellposten,Produkt,Lieferkartei,Lieferschein,Lieferwagen,Fahrtziel,Lieferscheinvermerk}
    → {Kundenbestellung,Kunde,Lagerbestand,Wartekartei,Vorgemerkte Menge,
    Nachbestellung,Lieferant,Minimalbestand,Maximalbestand}
18. {Bestellposten,Produkt,Lagerbestand} → {Lieferkartei,Wartekartei}
19. {Bestellposten,Produkt,Lagerbestand,Lieferkartei,Wartekartei,Lieferschein,
    Lieferscheinvermerk} →
    {Kundenbestellung,Kunde,Vorgemerkte Menge,Nachbestellung,Lieferant,
    Minimalbestand,Maximalbestand,Lieferwagen,Fahrtziel}
20. {Bestellposten,Produkt,Lagerbestand,Lieferkartei,Wartekartei,Vorgemerkte Menge} →
    {Kundenbestellung}
21. {Kunde} → {Kundenbestellung,Bestellposten,Produkt}
22. {Kundenbestellung} → {Bestellposten,Produkt}
23. {Kundenbestellung,Bestellposten,Produkt,Lieferkartei} →
    {Lagerbestand,Wartekartei,Vorgemerkte Menge}

24. {Kundenbestellung,Kunde,Bestellposten,Produkt,Lagerbestand,Lieferkartei,Wartekartei, Vorgemerkte Menge} →
{Nachbestellung,Lieferant,Minimalbestand,Maximalbestand,Lieferschein,Lieferwagen, Fahrtziel,Lieferscheinvermerk}

Diese Implikationen führen zu Fragen an die Fachexperten. Zu beurteilen ist, ob jede Funktion, die alle "Dinge" der jeweiligen Prämisse betrifft, auch alle "Dinge" der Konklusion angeht. Dabei können die formalen Gegenstände der Konklusionen – wie in Folgerung 12 auf Seite 104 dargestellt – auch einzeln abgefragt werden.

In dem dargestellten Begriffsverband gilt (wie in allen bisher betrachteten) die Implikation *Lieferschein* → *Lieferkartei* (Implikation 4). Diese ist inhaltlich aber nicht haltbar, weil nach der Erstellung der Lieferscheine aus der Lieferkartei heraus die Lieferscheine allein weiter bearbeitet werden können, ohne noch einmal auf die Lieferkartei zugreifen zu müssen. Also ist das betrachtete Modell in bezug auf die Verarbeitung von Lieferscheinen noch zu ungenau und deshalb der entsprechende Anwendungsfall funktional zu zerlegen. In der Tat wird diese Implikation bei der folgenden Verfeinerung des Anwendungsfalls *Lieferscheine erstellen* aufgelöst.

Für die Zerlegung ist davon ausgegangen, daß die Bestellposten der Lieferkartei durchgegangen werden und für jeden zu beliefernden Kunden ein Lieferschein ausgestellt wird, dem alle betreffenden Bestellposten angehängt werden. Am Ende werden die erstellten Lieferscheine schließlich den Lieferwagen zugeordnet, die unter Umständen auch mehrere Fahrtziele gemeinsam bedienen.

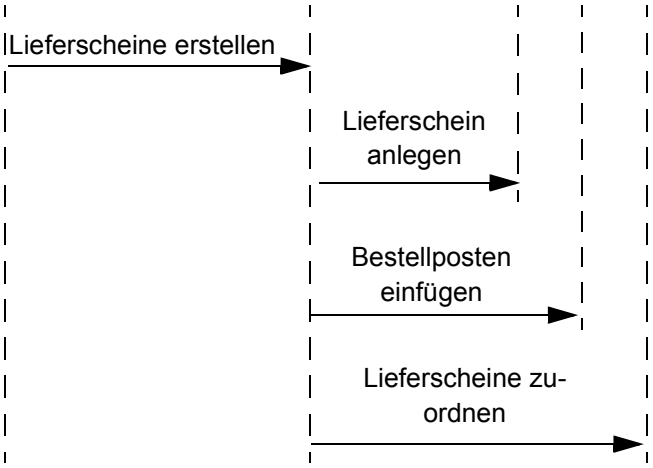


Abb. 4.16 Funktionale Zerlegung des Anwendungsfalls *Lieferscheine erstellen*

Diese Zerlegung führte ausgehend von Abbildung 4.15 auf Seite 150 zu dem fol-

genden Liniendiagramm. (Änderungen sind wieder schwarz hervorgehoben und übernommene Diagrammelemente grau dargestellt.)

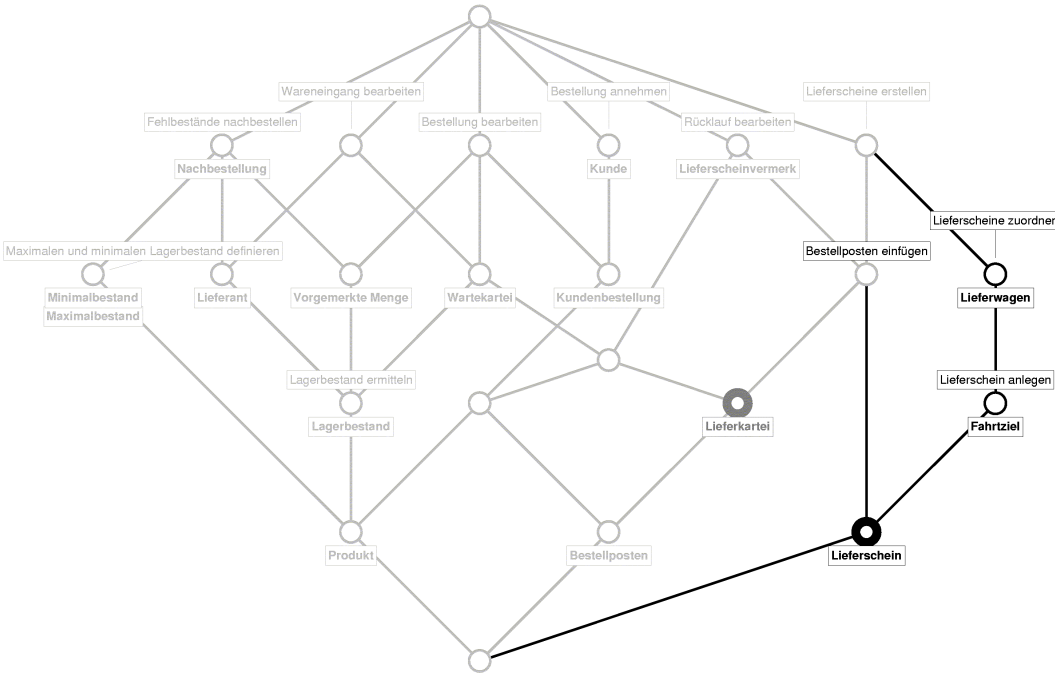


Abb. 4.17 Anwendungsfall *Lieferscheine erstellen* verfeinert

Die Implikation *Lieferschein* → *Lieferkartei* gilt nach der Verfeinerung nicht mehr (siehe dick gezeichnete Knoten), weil in den Teilschritten des Anlegens eines (leeren) Lieferscheins und der Zuordnung der Lieferscheine zu den Lieferwagen die Lieferkartei keine Rolle spielt.

Die beispielhaft betrachtete Implikation *Lieferschein* → *Lieferkartei* ist von besonders einfacher Gestalt, weil sie aus dem Liniendiagramm in Abbildung 4.15 auf Seite 150 einfach dadurch abgelesen werden kann, daß *Lieferschein* im Diagramm über *Lieferkartei* steht. In 3.4 wurden auch komplexere Implikationen mit einer Kombination mehrerer formaler Merkmale bzw. Gegenstände als Prämisse betrachtet. Auch solche Implikationen können an dieser Stelle hilfreich sein. In allen hier betrachteten Begriffsverbänden gilt beispielsweise die Implikation
{*Produkt*, *Lieferkartei*} → *Bestellposten*. Diese Implikation drückt auch eine zutreffende Modellierung aus. Von der *Lieferkartei* kommt man nur zu einem darin aufgeführten *Produkt*, indem man sich einen betreffenden *Bestellposten* anschaut – allgemeiner betreffen alle Funktionen innerhalb von JWI, die gleichzeitig ein *Produkt* und die *Lieferkartei* berühren, auch einen oder mehrere *Bestellposten*.

Auf ein anderes Beispiel für eine komplexere Implikation stößt man durch die oben erfolgte Zerlegung des Anwendungsfalls *Fehlbestände nachbestellen*. Nachfolgend das schon in 4.3.1 angegebene Liniendiagramm (Abbildung 4.11 auf

Seite 147) zur funktionalen Zerlegung dieses Anwendungsfalls. Im Unterschied zu vorher ist in Abbildung 4.18 lediglich der später gefundene Indizierungsfehler im Ausgangskontext behoben.

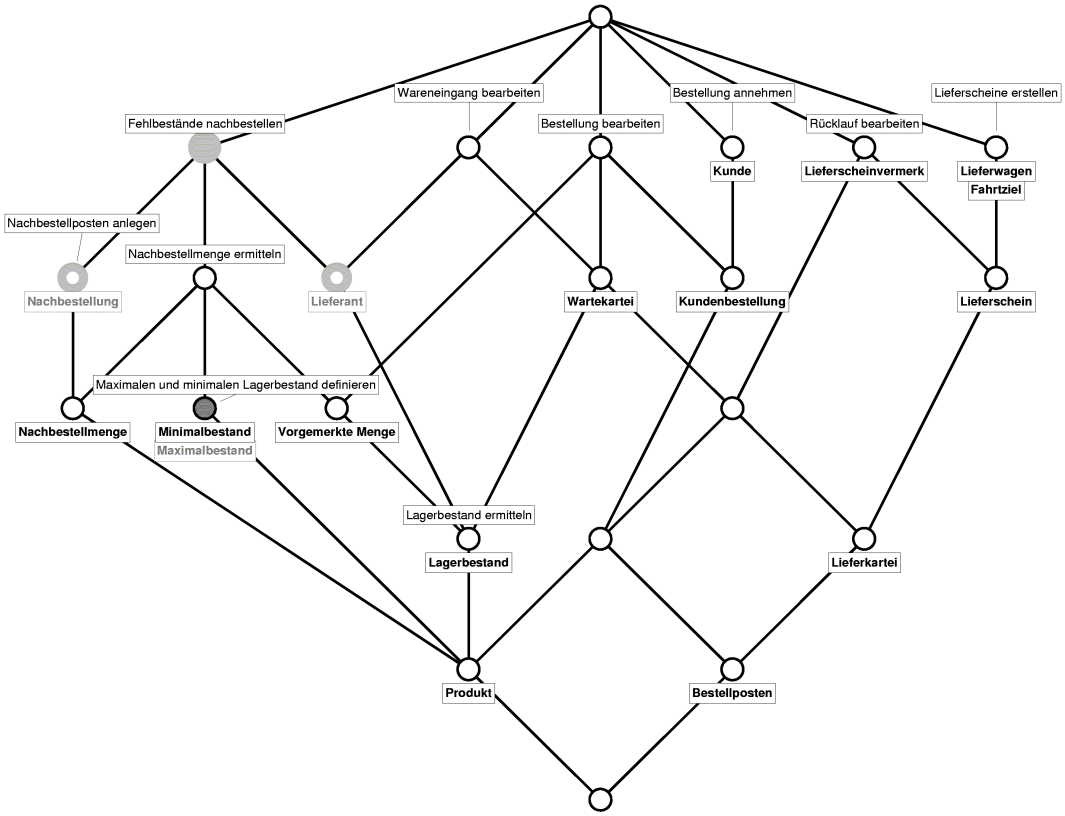


Abb. 4.18 Anwendungsfall *Fehlbestände nachbestellen* verfeinert

In dem entsprechenden Begriffsverband gilt – wie in allen bisher betrachteten – die Implikation {*Nachbestellung*, *Lieferant*} → *Maximalbestand* (durch die grauen Knoten des Diagramms angedeutet). Dies stellt aber einen unzutreffenden Zusammenhang her. Der Einzelschritt, erst eine leere Nachbestellung für einen Lieferanten einzurichten, wurde in der Zerlegung nicht berücksichtigt. Die entsprechende Funktion *Nachbestellung anlegen* löst die Implikation auf, weil sie eine Nachbestellung und einen Lieferanten, aber nicht den Maximalbestand eines Produkts betrifft. Nach ihrer Ergänzung ergibt sich das Liniendiagramm in Abbildung 4.19. (Änderungen gegenüber Abbildung 4.18 sind wieder schwarz hervorgehoben und übernommene Diagrammelemente grau dargestellt.)

Wie die markierten Knoten zeigen, liegt das Supremum der Gegenstandsbegriffe von *Nachbestellung* und *Lieferant* nicht mehr über dem Gegenstandsbegriff zu *Maximalbestand*. Also gilt die angesprochene Implikation nach der weiteren Zerlegung nicht mehr.

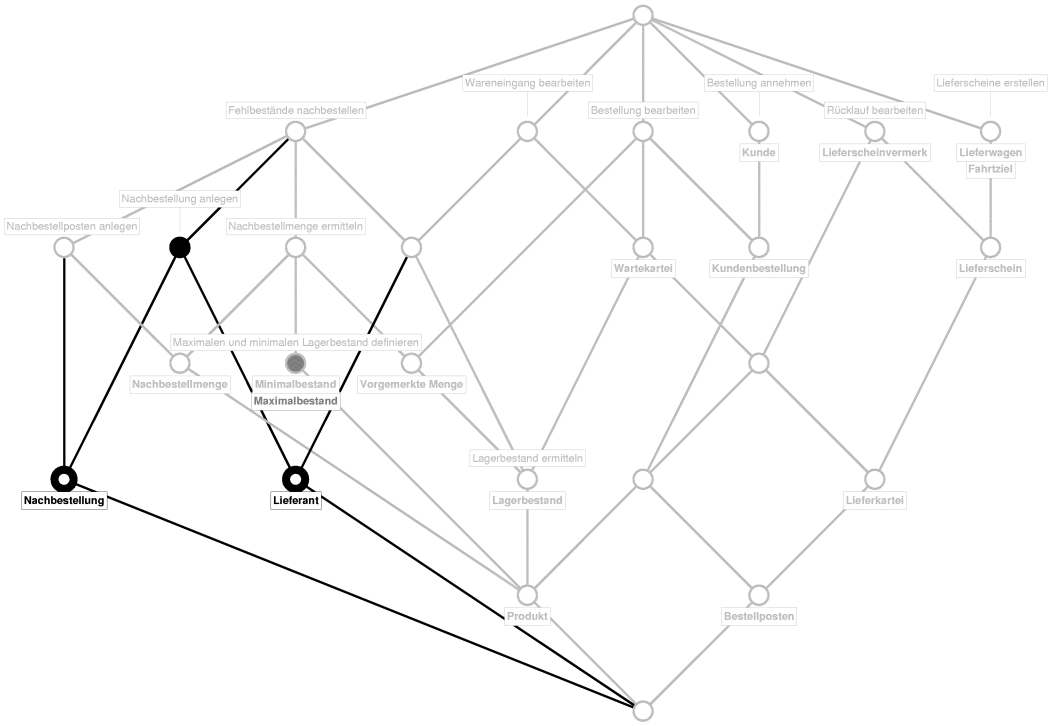


Abb. 4.19 Anwendungsfall *Fehlbestände nachbestellen* weiter verfeinert

An diesem Beispiel werden aber auch die Grenzen dieses Überprüfungsansatzes klar. Um die Anzahl der Überprüfungsfragen zu minimieren und Redundanz in ihnen auszuschließen, wird nur die Duquenne-Guigues-Basis der Gegenstandsimplikationen betrachtet. Gibt es unter allen gültigen Gegenstandsimplikationen des zugrunde liegenden formalen Kontexts eine fachlich unzutreffende, so enthält die Basis eine Implikation, die ebenfalls fachlich nicht haltbar ist. Daß die in der Basis enthaltenen unzutreffenden Implikationen gerade die Formulierung der Überprüfungsfragen liefern, die für Entwickler und Fachexperten die eingängigste ist, um den fachlichen Mangel des Modells zu erkennen, ist jedoch nicht gesagt.

Die Implikation $\{Nachbestellung, Lieferant\} \rightarrow Maximalbestand$, die hier den Hinweis lieferte, daß die Zerlegung noch unvollständig war, ist z.B. kein Element der Duquenne-Guigues-Basis zum Begriffsverband in Abbildung 4.18. Sie folgt jedoch aus den Implikationen:
$Nachbestellung \rightarrow \{Produkt, Nachbestellmenge\}$,
$Lieferant \rightarrow \{Produkt, Lagerbestand\}$ und
$\{Produkt, Lagerbestand, Nachbestellmenge\}$
$\rightarrow \{Vorgemerkte\ Menge, Minimalbestand, Maximalbestand\}$,
die alle in der Basis zum Begriffsverband in Abbildung 4.18 enthalten sind (Ableitung der Implikation nach Hilfssatz 12 auf Seite 99 und Hilfssatz 8 auf Seite 95). Von diesen sind die ersten beiden ebenfalls fachlich unzutreffend. Dies kann man

im Nachhinein dadurch erkennen, daß sie ebenso durch die Betrachtung der Funktion *Nachbestellung anlegen* aufgelöst werden. Fachlich gesehen steht das folgende dahinter: Für das Anlegen einer leeren Nachbestellung sind nur die *Nachbestellung* selber und der *Lieferant*, an den sie gerichtet wird, nötig, ohne daß die nachzubestellenden *Produkte* und die betreffenden *Nachbestellmengen* einzutragen wären, so daß auch der *Lagerbestand* nicht abgefragt werden muß.

Vorstellbar wäre, daß den an der Entwicklung beteiligten zwar von der Implikation $\{Nachbestellung, Lieferant\} \rightarrow Maximalbestand$ klar ist, daß die fachlich unzutreffend ist, von den Implikationen $Nachbestellung \rightarrow \{Produkt,Nachbestellmenge\}$ und $Lieferant \rightarrow \{Produkt,Lagerbestand\}$ ihnen das aber nicht sofort ins Auge springt.

Sicher sein, eine passende Formulierung zu finden, kann man aber nur, wenn man alle im Kontext gültigen Implikationen präsentiert. Nur würde durch die starke Redundanz ein solches Vorgehen kaum die Akzeptanz der Beteiligten finden und schon allein durch die Anzahl der zu betrachtenden Implikationen unpraktikabel sein.

4.4 Überprüfung des Modells

4.4.1 Überprüfung durch Implikationen

Mit Hilfe von Gegenstandsimplikationen wird – wie in 4.3.2 dargestellt – die Suche nach funktional untergeordneten Funktionen unterstützt. Immer wenn eine aus dem vorliegenden Kontext abgeleitete Implikation inhaltlich nicht haltbar ist, wird nach einer neuen Funktion in der Zerlegung gesucht, die die Implikation auflöst. Nun kann es aber auch der Fall sein, daß eine solche unter den bisher schon untersuchten Anwendungsfällen und Funktionen gefunden wird, wenn bei der Indizierung Fehler vorkamen. In diesem Fall sind die Beschreibungen der betroffenen Anwendungsfälle und die entsprechende Indizierung zu korrigieren.

Also dienen Gegenstandsimplikationen auch zur Überprüfung des Anwendungsfallmodells und der vorgenommenen Indizierung. Dazu können dual auch Merkmalsimplikationen herangezogen werden (Definition 31 auf Seite 105). Eine Merkmalsimplikation $V \rightarrow W$ zwischen zwei Merkmalsmengen V und W bedeutet im Kontext von BASE, daß alle "Dinge", die in alle Anwendungsfälle oder Funktionen aus V involviert sind, auch in allen Anwendungsfällen oder Funktionen aus W eine Rolle spielen. Die durch die Implikation so ausgedrückte Datenabhängigkeit muß von den Fachexperten beurteilt werden. Trifft sie fachlich nicht zu, sind

bei der Indizierung Fehler gemacht oder wichtige "Dinge" vergessen worden.

Das bei der funktionalen Zerlegung des Anwendungsfalls *Bestellung bearbeiten* aufgedeckte Versäumnis, diesem Anwendungsfall als involviertes "Ding" die *Vorgemerkte Menge* zuzusprechen, kann auch auf diesem Weg bemerkt werden. Die Duquenne-Guigues-Basis der Merkmalsimplikationen zum ursprünglichen formalen Kontext in Abbildung 4.4 auf Seite 136 bzw. dem in Abbildung 4.5 auf Seite 137 dargestellten zugehörigen Begriffsverband besteht aus den folgenden Implikationen:

1. {Wareneingang bearbeiten,Rücklauf bearbeiten} → {Bestellung bearbeiten}
2. {Maximalen und minimalen Lagerbestand definieren} → {Fehlbestände nachbestellen}
3. {Lieferscheine erstellen,Wareneingang bearbeiten} →
   {Bestellung bearbeiten,Rücklauf bearbeiten}
4. {Lagerbestand ermitteln} →
   {Bestellung bearbeiten,Fehlbestände nachbestellen,Wareneingang bearbeiten}
5. {Fehlbestände nachbestellen,Rücklauf bearbeiten} →
   {Bestellung annehmen,Bestellung bearbeiten,Lagerbestand ermitteln,
   Maximalen und minimalen Lagerbestand definieren,Wareneingang bearbeiten}
6. {Fehlbestände nachbestellen,Maximalen und minimalen Lagerbestand definieren,
   Wareneingang bearbeiten} →
   {Bestellung annehmen,Bestellung bearbeiten,Lagerbestand ermitteln,Rücklauf bearbeiten}
7. {Fehlbestände nachbestellen,Lieferscheine erstellen} →
   {Bestellung annehmen,Bestellung bearbeiten,Lagerbestand ermitteln,
   Maximalen und minimalen Lagerbestand definieren,Wareneingang bearbeiten,
   Rücklauf bearbeiten}
8. {Bestellung bearbeiten,Rücklauf bearbeiten} → {Wareneingang bearbeiten}
9. {Bestellung bearbeiten,Lieferscheine erstellen} →
   {Wareneingang bearbeiten,Rücklauf bearbeiten}
10. {Bestellung bearbeiten,Fehlbestände nachbestellen} →
    {Lagerbestand ermitteln,Wareneingang bearbeiten}
11. {Bestellung annehmen,Rücklauf bearbeiten} →
    {Bestellung bearbeiten,Wareneingang bearbeiten}
12. {Bestellung annehmen,Wareneingang bearbeiten} →
    {Bestellung bearbeiten,Rücklauf bearbeiten}
13. {Bestellung annehmen,Lieferscheine erstellen} →
    {Bestellung bearbeiten,Wareneingang bearbeiten,Rücklauf bearbeiten}
14. {Bestellung annehmen,Fehlbestände nachbestellen} →
    {Bestellung bearbeiten,Lagerbestand ermitteln,Maximalen und minimalen Lagerbestand
    definieren,Wareneingang bearbeiten,Rücklauf bearbeiten}

Die 10. Implikation führt etwa zu der Frage, ob alle in die Anwendungsfälle *Bestellung bearbeiten* und *Fehlbestände nachbestellen* involvierten "Dinge" auch bei der Ermittlung des Lagerbestands und der Bearbeitung des Wareneingangs betroffen sind. Da *Lagerbestand ermitteln* von *Wareneingang bearbeiten* benutzt wird, kann bei der Untersuchung, ob dies fachlich zutrifft, der Anwendungsfall *Wareneingang bearbeiten* vernachlässigt werden (Er betrifft ja automatisch alle in *Lagerbestand ermitteln* involvierten "Dinge".) Wäre die Implikation fachlich zu halten, so kämen den beiden Anwendungsfällen *Bestellung bearbeiten* und *Fehlbestände*

nachbestellen gemeinsam keine anderen als die in *Lagerbestand ermitteln* involvierten "Dinge" (*Produkt* und *Lagerbestand*) zu. Die Überlegung (auf fachlicher Ebene), daß die nachzubestellenden Mengen gerade bei der Bearbeitung der Bestellungen der Einzelhändler auflaufen, also eine Abhängigkeit der beiden betrachteten Anwendungsfälle ergeben, führt direkt dazu, die in 4.3.1 auf Seite 149 dargestellte Korrektur der Indizierung vorzunehmen.

In dieser Weise können Gegenstands- und Merkmalsimplikationen dazu dienen, das erstellte Modell zu überprüfen. Drückt eine Implikation einen fachlich unzutreffenden Sachverhalt aus, so wurde ein Fehler der Indizierung aufgedeckt oder bei einer Gegenstandsimplikation ein neuer Anwendungsfall (eine neue Funktion) bzw. bei einer Merkmalsimplikation ein neues "Ding" identifiziert.

4.4.2 Konsistenzprüfung anhand von "*uses*"-Beziehungen

Außerhalb der Mittel der Formalen Begriffsanalyse bietet sich eine Konsistenzprüfung funktional verfeinerter Modelle an. Wurde bei der groben Modellierung der Anwendungsfälle eine "uses"-Beziehung zwischen zwei Anwendungsfällen attestiert, so muß der benutzte Anwendungsfall in der funktionalen Zerlegung des benutzenden auftauchen.

4.5 Klassenkandidaten im Liniendiagramm

Durch das Vorgehen wie in 4.2 und 4.3 wurden Liniendiagramme erstellt, die Datenelemente, Funktionen und deren Abhängigkeiten wiedergeben. Daten- und funktionale Sicht wurden aufgefächert. In einem objektorientierten Modell werden Datenelemente und Funktionen zu Klassen als Einheiten zusammengestellt. Diese Aufgabe obliegt nun dem Entwickler. Ihn bei dieser Aufgabe zu unterstützen, ist das zentrale Anliegen von BASE.

4.5.1 Klassenkandidaten unter den "Dingen"

Die "einfachsten" Klassenkandidaten findet er unter den betrachteten "Dingen". Dabei handelt es sich in der Regel um Entitätsklassen im Sinne von Jacobson (2.6.1, Seite 54). Wie in 4.2.4 auf Seite 140 dargelegt, erscheint es sinnvoll, die "Dinge" im Liniendiagramm von unten nach oben darauf zu überprüfen, ob sie wertvolle Klassenkandidaten bilden, weil unten im Diagramm die anwendungsfallübergreifend interessanten "Dinge" stehen. Von der Klärung der Frage, wie sie modelliert werden sollen, hängen große Teile des Gesamtmodells ab. Dieses Vorgehen

entspricht einer top-down-Strategie bei der Festlegung der Klassen. Die grundlegenden Klassen werden zuerst modelliert, später dann die spezielleren, deren Definition von ersteren abhängt.

Die Entscheidung, ob ein "Ding" als Klassenkandidat modelliert wird, wird im wesentlichen danach getroffen, ob der entsprechende Klassenkandidat eine wertvolle Abstraktion und eine für sich interessante Einheit (von späteren Attributen und Operationen) im Untersuchungsbereich bildet (siehe 2.3.1 auf Seite 31). Die Modellierung der einzelnen ausgewählten Klassenkandidaten erfolgt – wie in 2.3.1 auf Seite 31 erläutert – dann auch nach einer top-down-Strategie, denn ausgehend von der Bedeutung und dem Zweck des Klassenkandidatens wird nach Attributen und Operationen zur genaueren Spezifikation gesucht. Die Suche wird ebenfalls durch das Liniendiagramm geleitet.

Ist ein "Ding" als Klassenkandidat ausgewählt, bietet der Hauptfilter des entsprechenden Gegenstandsbegriffs Hinweise für die Festlegung seiner Attribute und Operationen. Wie in 4.2.4 auf Seite 140 diskutiert, sind als "Dinge" behandelte Attribute typischerweise in ihm zu finden. Das ist zumindest der Fall, wenn die vorgenommene Indizierung der Anwendungsfälle (implizit) berücksichtigt, daß ein entsprechendes Attribut nur über seine Klasse ansprechbar ist. ("implizit", weil zum Zeitpunkt der Indizierung noch keine Klassen und Attribute festgelegt sind) Genauso finden sich die Operationskandidaten im Hauptfilter des Gegenstandsbegriffs des Klassenkandidaten, denn von der Operation einer Klasse kann man erwarten, daß sie die Klasse auch betrifft.

In Situationen, in denen von zwei unter den "Dingen" vorhandenen Klassenkandidaten einer von einer Teilmenge der Anwendungsfälle und Funktionen betroffen ist, in die der andere involviert ist, steht aber auch der erste im Liniendiagramm über dem zweiten, ohne daß es sich um ein Attribut des zweiten handeln muß. Weiter sind alle Funktionen, die einen Klassenkandidaten berühren, im Hauptfilter seines Gegenstandsbegriffs enthalten. Sie müssen aber nicht notwendigerweise alle der entsprechenden Klasse als Operationen zugeordnet werden. Es kann auch der Fall sein, daß die Anwendungsfälle oder Funktionen aus dem Hauptfilter die Klasse nur benutzen.

Deshalb macht es Sinn, im Hauptfilter eines Klassenkandidatens nach Attributen und Operationen zu suchen. Den ganzen Hauptfilter automatisch als Aufzählung der Attribute und Operationen zu betrachten, ist dagegen nicht sinnvoll.

4.5.2 Begriffe als Klassenkandidaten

Zur Motivation eines begriffsbasierten Ansatzes wurde in 1.3 auf Seite 11 und 2.3.1 auf Seite 32 darauf hingewiesen, daß laut verschiedener führender Autoren

Klassen Begriffe des Untersuchungsbereichs implementieren sollen. In 2.3.1 und 1.6.2 wird auch diskutiert, wie eine entsprechende natürliche Formalisierung mit Hilfe von Formaler Begriffsanalyse aussehen könnte: Objekte sind formale Gegenstände, Attribute und Operationen formale Merkmale.

Der Ansatz in BASE ist anders angelegt, damit eine Analyse auf Typebene durchgeführt werden kann (vergl. Diskussion in 1.6.2 auf Seite 16). Aber dennoch scheint es naheliegend, die innerhalb von BASE betrachteten formalen Begriffe als Klassenkandidaten zu betrachten. Schließlich gruppieren sie Datenelemente und Funktionen. Diese Betrachtungsweise führt aber nur zu sehr eingeschränkten Klassenkandidaten. In einer solchen Klasse müßte nämlich jede Operation alle Attribute benutzen. Dies ist – wie schon in 4.2.4 auf Seite 141 ansatzweise diskutiert – aber nicht zu erwarten. Treten nämlich Objekte einer Klasse in verschiedenen Rollen auf und unterteilt sich die Klasse dementsprechend in verschiedene Teile, so ist diese Unterteilung auch bei der Zuordnung von Operationen und involvierten Attributen zu beobachten. Aber selbst Klassen, die als monolithischer Block behandelt werden, können Operationen enthalten, die nicht alle Attribute betreffen. Gemäß dem Modularisierungsaspekt von Klassen sind Operationen nämlich typischerweise so kleine funktionale Einheiten, daß sie direkt umgesetzt werden. Komplexe Funktionen werden in kleine Operationen zerlegt, die dann auch noch verschiedenen Klassen zugeteilt werden können. Wie in 2.2.2 auf Seite 25 erläutert, wird die Systemfunktionalität erst durch das Zusammenspiel aller Objekte erbracht.

So werden im JWI-Beispiel die "Dinge" *Lieferant*, *Lagerbestand* und *Produkt* zu einem Begriffsumfang zusammengefaßt (Gegenstandsbegriff zu *Lieferant*) und es macht auch durchaus Sinn, die ersten beiden als Attribute von *Produkt* aufzufassen. Der Begriffsinhalt des Begriffs besteht aber aus den allgemeinen Anwendungsfällen *Fehlbestände nachbestellen* und *Wareneingang bearbeiten*, die *Produkt* benutzen, aber schwerlich als Operationen einer Klasse *Produkt* zu verstehen sind. (Dieser Begriff bleibt bei der beispielhaft durchgeführten Zerlegung aller Anwendungsfälle genauso erhalten.) Der spezifische Anwendungsfall *Lagerbestand ermitteln*, der gut als Operation einer Klasse *Produkt* zu deuten ist, benutzt nur das (potentielle) Attribut *Lagerbestand*. Andererseits gibt es mit den Gegenstandsbegriffe zu *Minimalbestand* und *Vorgemerkte Menge* andere formale Begriffe, die sinnvolle Attribute von *Produkt* kapseln (Begriffsumfänge: {*Maximalbestand*, *Minimalbestand*, *Produkt*} bzw. {*Vorgemerkte Menge*, *Lagerbestand*, *Produkt*}) und die Verwendung von *Produkt* in anderen Zusammenhängen widerspiegeln. (Diese Begriffe werden in Folge der funktionalen Zerlegung des Anwendungsfalls *Fehlbestände nachbestellen* lediglich in ihren Inhalten um die Funktion *Nachbestellmenge ermitteln* erweitert.)

Deshalb sind formale Begriffe als Klassenkandidaten zu einschränkend.

Konzentriert man sich allein auf die reduzierte Beschriftung des Liniendiagramms, so liest man die stärkste Zuordnung von Funktionen und "Dingen" zueinander ab. Wenn der Merkmalsbegriff zu einer Funktion *f* und der Gegenstandsbegriff zu einem "Ding" *d* gleich sind, d.h. die entsprechenden Beschriftungen im Liniendiagramm am selben Knoten stehen, sind *f* und *d* (durch den zugrunde liegenden formalen Kontext) in sehr strengem Sinn einander zugeordnet. In dieser Situation betrifft *f* genau die "Dinge", die immer gemeinsam mit *d* auftreten und *d* ist in genau die Funktionen involviert, die alle von *f* benutzten Dinge berühren. Zu erwarten ist, daß diese starke Zuordnung auch in der späteren Klassenstruktur erhalten bleibt.

Interessant ist in diesem Zusammenhang die "vertikale Mitte" des Diagramms. Wie in 4.2.4 auf Seite 137 ausgeführt, ergibt sich von oben nach unten die funktionale Zerlegung des Systems. Die am weitesten spezialisierten, kleinen Funktionen – die am eindeutigsten späteren Klassen zugeordnet werden können – stehen unter den übergreifenden allgemeineren Anwendungsfällen. Dual fächert sich das Diagramm von unten anhand der Datensicht auf das System auf. Wie in 4.2.4 auf Seite 140 erläutert, stehen die anwendungsfallübergreifend interessanten "Dinge" unten und über ihnen folgen die in spezielleren Zusammenhängen zu betrachtenden "Dinge". Die Spezialisierung der beiden Sichten verläuft im Diagramm in entgegengesetzten vertikalen Richtungen. "In der Mitte" treffen sich die speziellsten Funktionen mit den speziellsten "Dingen". Diese lassen sich am einfachsten in eindeutiger Weise einander zuordnen, wobei erst einmal dahin gestellt sei, ob die entsprechenden "Dinge" zu Klassen oder Attributen werden. (Im Sinne der Diskussion unter 4.5.1 ist zu erwarten, daß es sich vor allem um Attribute handeln wird.)

Erbringt die Untersuchung der "Dinge" im unteren Diagrammbereich nach 4.2.4 auf Seite 140 keine brauchbaren Klassenkandidaten, denen in einer top-down-Strategie danach noch Attribute und Operationen zuzuordnen sind, so kann "in der Mitte" des funktional verfeinerten Diagramms nach Knoten gesucht werden, die gleichzeitig mit Funktionen und "Dingen" beschriftet sind. Ausgehend von diesen Einheiten kann versucht werden, Klassen in einer bottom-up-Strategie zu entwikkeln. (Vergl. auch 2.3.1 auf Seite 31)

Also betrachtet man statt der kompletten formalen Begriffe nur die an einem Knoten des Liniendiagramms zusammentreffenden "Dinge" und Funktionen. Dies ist auch einfacher aus dem Diagramm abzulesen. Gegenüber dem vollen Begriff werden "Dinge" ausgelassen, die zwar in am Knoten angegebene Funktionen involviert sind, die aber auch von anderen nicht im (vollen) Begriffsinhalt enthaltenen Funktionen benutzt werden. Dual verzichtet man auf solche Funktionen und Anwendungsfälle, die alle dem Knoten zugeordneten "Dinge" betreffen – darüber hinaus aber auch noch andere, die nicht im (vollen) Begriffsumfang enthalten sind.

Dies sind oft übergreifende Anwendungsfälle, die nur schwer einer Klasse zuzuordnen sind, bevor das Klassenmodell nicht schon weit ausgearbeitet ist.

Aus dieser Betrachtung erhält man Nuclei bestehend aus zusammengehörigen Funktionen und "Dingen", die im späteren Klassenmodell komplett einer Klasse zugeordnet werden. Deshalb werden die Interna der einzelnen Klassen bei diesem Vorgehen in einer bottom-up-Strategie entwickelt, denn sie werden aus solchen Nuclei zusammengestückelt. Für die Reihenfolge der Festlegung der Klassen selber ist nach 4.2.4 auf Seite 140 und 4.5.1 eine top-down-Strategie zu bevorzugen. Deshalb ist hier die Rede von Nuclei, die zu Teilen von Klassen werden, und diese Nuclei werden nicht als Klassenkandidaten behandelt, die später nur weiter zu ergänzen sind, denn so ergeben sich nur sehr spezialisierte Klassenkandidaten. Das bietet sich erst an, wenn das Klassenmodell schon so weit feststeht, daß solche Einzelheiten "als Zusatz" modelliert werden können, ohne die Grundstruktur des Systemmodells zu stören.

Dementsprechend ist zur Festlegung der Klassenkandidaten dem Vorgehen unter 4.5.1 Vorrang einzuräumen. Im folgenden Abschnitt wird noch eine Möglichkeit aufgezeigt, grobe Strukturen innerhalb des Systemmodells zu erkennen.

4.6 Blockrelationen in BASE

4.6.1 Motivation und Idee des Vorgehens

In 3.5 wurden Blockrelationen als Konzept zur Zusammenfassung von formalen Gegenständen und formalen Merkmalen zu allgemeineren Einheiten vorgestellt, als es formale Begriffe sind. Die Idee, sie in BASE zu benutzen, ist aus der Untersuchung von Modularisierungsvorschlägen für Altsysteme durch Christian Lindig und Gregor Snelting in [L-S 97] entstanden (siehe 6.3 auf Seite 232). Die Autoren erstellen in der referenzierten Arbeit (Re-) Strukturierungsvorschläge für bestehende Software. Sie behandeln dabei Programmfunktionen als formale Gegenstände und Programmvariablen als formale Merkmale und untersuchen den formalen Kontext, der angibt, welche Variable in welcher Funktion benutzt wird. Dies ist ein ganz ähnlicher Ansatz zu BASE (vergl. auch die Diskussion um die Rollen von formalen Gegenständen und Merkmalen in 4.2.3 auf Seite 134). Lindig und Snelting prägen die Sprechweise vom "Modul maximaler Kohäsion", in dem jede Funktion alle Variablen benutzt, und vom "Modul regulärer Kohäsion", von dessen Funktionen eine alle enthaltenen Variablen benutzt und von dessen Variablen eine von allen Funktionen des Moduls benutzt wird.

Im Begriffsverband entsprechen Module maximaler Kohäsion den formalen Begriffen. Beim Übergang zu Blockrelationen entstehen an den Stellen Module regulärer Kohäsion, an denen der entsprechende Block als obere Intervallgrenze einen Merkmals- und als untere Intervallgrenze einen Gegenstandsbegriff trägt. In [L-S 97] bedauern die Autoren das Fehlen eines praktischen Anwendungsbeispiels für diese Technik. Für BASE war die Überlegung nun, daß im Falle von Klassen als Modularisierungseinheiten eine größere Wahrscheinlichkeit vorliegt, sie als Module regulärer Kohäsion zu finden. So betreffen alle Operationen (hoffentlich) die Klasse selber und ein eventuell modellierter Konstruktor sollte alle Attribute initialisieren. In Analogie zu [L-S 97] ist aber möglich, daß sich Komponenten als größere Modularisierungseinheiten ergeben. Im folgenden ist deshalb immer von "Komponenten" die Rede. Eine eventuell identifizierte Klasse wird damit sprachlich als eine Komponente behandelt, die nur aus einer Klasse besteht. Diese einheitliche Sprechweise erleichtert die Darstellung. Außerdem zeigen die bisher durchgeführten Untersuchungen mit dem im folgenden geschilderten Blockrelationenansatz, daß meist grobe Systemzerlegungen erzeugt werden.

Im folgenden ist die Verwendung von Blockrelationen innerhalb von BASE anhand des JWI-Beispiels erklärt. Bei der Darstellung der Ergebnisse werden die trivialen Blockrelationen nach Bemerkung 8 auf Seite 113 – d.h. die ursprüngliche Inzidenzrelation und die vollständige Relation $G \times M$ – nicht weiter betrachtet. Sie liefern nichts Neues oder die gröbstmögliche Sicht auf das System, in der keine Modellelemente unterschieden werden.

4.6.2 Alternative Modularisierungsvorschläge

Der unverfeinerte Begriffsverband aus Abbildung 4.15 auf Seite 150 hat drei nicht triviale Blockrelationen. Die entsprechenden formalen Kontexte sind in Abbildung 4.20 wiedergegeben. In den Kreuztabellen der Blockrelationen sind neu erzeugte Kreuze dick, schon in der originalen Inzidenzrelation (nach der Berücksichtigung der "*uses*"-Beziehungen) vorhandene dünn gezeichnet.

Einfacher zu interpretieren sind natürlich wieder die entsprechenden Liniendiagramme. Sie sind im folgenden zusammen mit den Blöcken im originalen Liniendiagramm angegeben. Dabei werden die formalen Kontexte aus Abbildung 4.20 in der Reihenfolge von links nach rechts betrachtet.

Abbildung 4.21 gibt den Begriffsverband zur ersten Blockrelation aus Abbildung 4.20 wieder und Abbildung 4.22 die zugehörigen Blöcke des originalen Begriffsverbands.

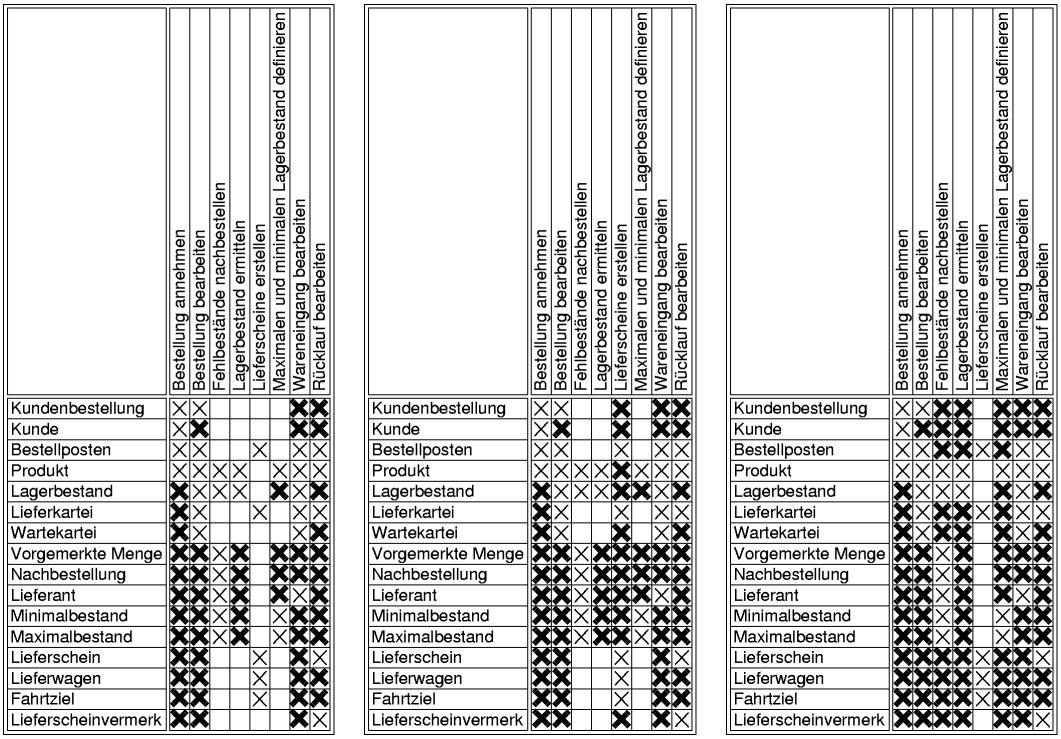


Abb. 4.20 Blockrelationen zum Begriffsverband aus Abbildung 4.15

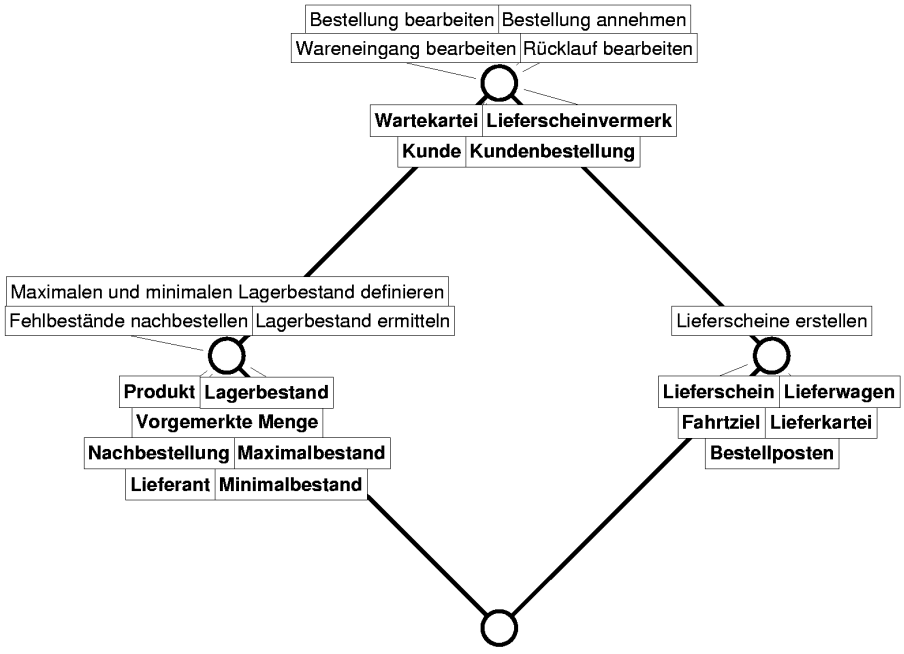


Abb. 4.21 Begriffsverband der ersten Blockrelation aus Abbildung 4.20

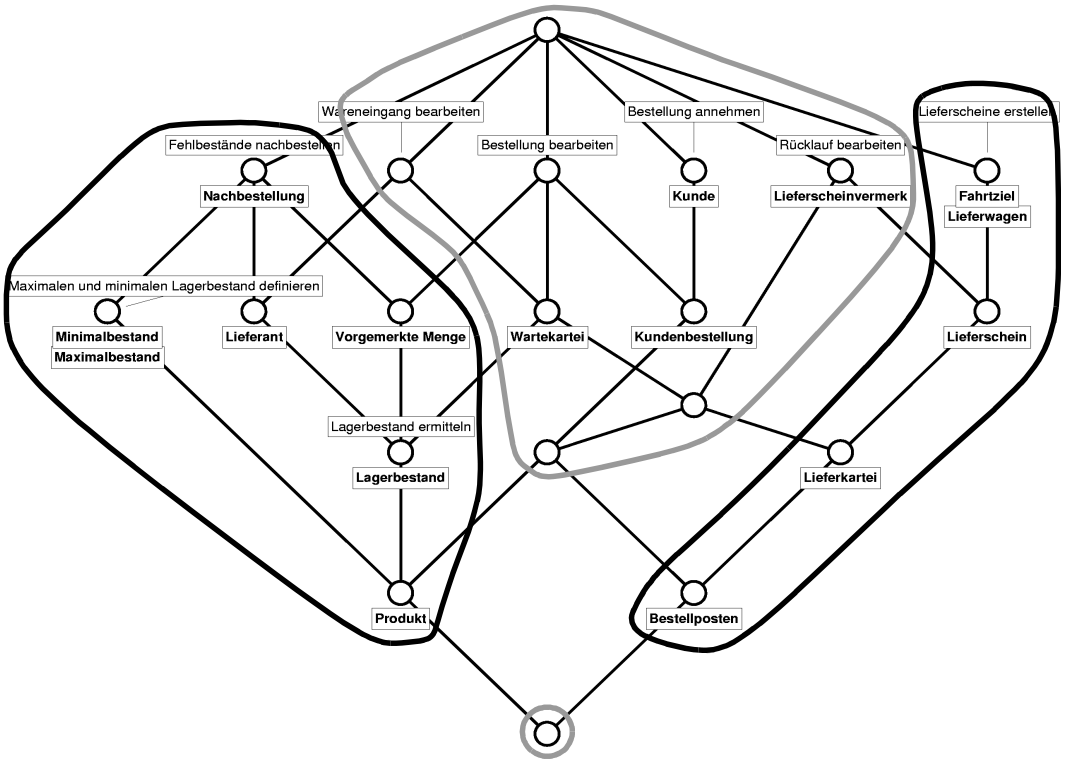


Abb. 4.22 Blöcke zur ersten Blockrelation aus Abbildung 4.20

Wie in 4.5.2 in bezug auf die Identifikation von Klassenkandidaten diskutiert, kann man sich auch im Liniendiagramm zu einer Blockrelation auf die Betrachtung der reduzierten Beschriftung konzentrieren, statt die kompletten Umfänge und Inhalte der formalen Begriffe zu untersuchen. In Abbildung 4.21 bzw. Abbildung 4.22 erkennt man dann die drei Einheiten

- ({*Kundenbestellung*, *Kunde*, *Lieferscheinvermerk*, *Wartekartei*} {*Bestellung annehmen*, *Bestellung bearbeiten*, *Rücklauf bearbeiten*, *Wareneingang bearbeiten*})
- ({*Lieferant*, *Minimalbestand*, *Maximalbestand*, *Nachbestellung*, *Vorgemerkte Menge*, *Lagerbestand*, *Produkt*} {*Maximalen und minimalen Lagerbestand definieren*, *Lagerbestand ermitteln*, *Fehlbestände nachbestellen*})
- ({*Bestellposten*, *Lieferkartei*, *Fahrtziel*, *Lieferwagen*, *Lieferschein*} {*Lieferscheine erstellen*})

Ihnen ist eine mögliche Komponentenzerlegung des Systems zu entnehmen. Der oberste Block beschreibt eine Komponente zur Koordinierung der Geschäftsaktivitäten von JWI an den Schnittstellen zu den Geschäftspartnern. Diese benutzt zwei andere Komponenten, die durch den linken und rechten Block beschrieben werden. Links finden sich alle Elemente zur JWI-internen Lagerverwaltung und rechts zur

Lieferungsabwicklung. Letztgenannte Komponente stellt auch das grundlegende "Ding" Bestellposten zur Verfügung. Links sind dagegen schon die Attribute der (potentiellen) Klasse *Produkt* zusammengefaßt, wobei nur das "Ding" Nachbestellung aus der Reihe fällt. Es ergibt sich in diesem Beispiel nicht eine Klassenstruktur für das zu entwickelnde System, aber eine sinnvolle Komponentenaufteilung.

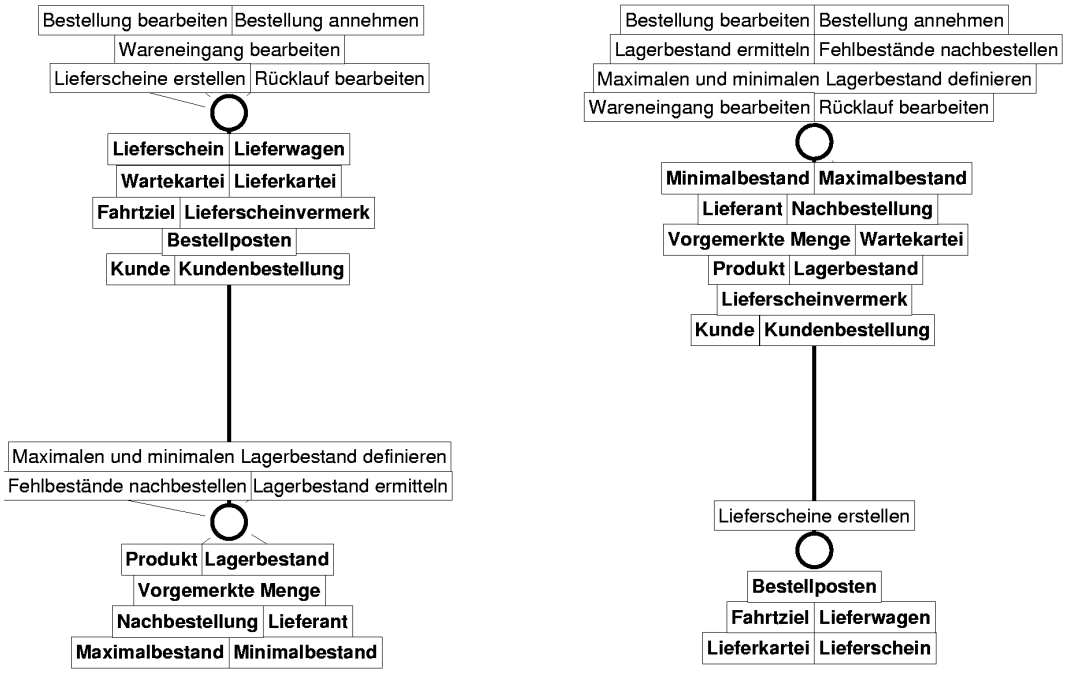


Abb. 4.23 Begriffsverbände der zweiten und dritten Blockrelation aus Abbildung 4.20

Alternativ zu den Begriffsverbänden der Blockrelationen lassen sich auch zu diesen Blockrelationen die Blöcke im Originaldiagramm veranschaulichen. (Abbildung 4.24 und Abbildung 4.25)

In diesem Beispiel sind jeweils zwei der Blöcke zu der ersten Blockrelation durch die beiden weiteren Blockrelationen auf zwei verschiedene Weisen noch einmal zusammengefaßt worden.

In der Situation, daß mehrere nicht-triviale Blockrelationen existieren, kann sich der Modellierer für eine der Komponentenzerlegungen entscheiden. So werden ihm mehrere Möglichkeiten aufgezeigt, das System zu modularisieren. Das Liniendiagramm des Begriffsverbands zur entsprechenden Blockrelation gibt eine Übersicht über die Komponentenzerlegung. Dabei sind Benutzungsbeziehungen der Komponenten durch die Ober-/Unterbegriffsrelation ausgedrückt – die benutzte Komponente ist durch den Unterbegriff gegeben. In Bezug auf die betrachteten Funktionen ist die Anordnung schon in Verbindung mit der Darstellung von

"*uses*"- und "*extends*"-Beziehungen im Liniendiagramm in 4.2.4 auf Seite 138 diskutiert worden. Benutzte Funktionen stehen unterhalb der benutzenden. Das gilt auch für solche Funktionen, die durch eine funktionale Zerlegung entstanden sind (vergl. 4.3.1 auf Seite 142). In Bezug auf die "Dinge" ist festzuhalten, daß die in der Benutzungshierarchie unten stehenden (benutzten) Komponenten die grundlegenden Datenstrukturen zur Verfügung stellen müssen, während in der Hierarchie höher gelegene nur noch speziellere Datenstrukturen ergänzen, die allein für gewisse Systemteile interessant sind. Nach der Diskussion in 4.2.4 auf Seite 140 entspricht dies gerade der Anordnung der Dinge im Diagramm. Somit ist die Aussage über die Darstellung von Benutzungsbeziehungen zwischen Komponenten im Liniendiagramm der entsprechenden Blockrelation gerechtfertigt. Daß diese Anordnung der üblichen grafischen Darstellung einer Komponentenzerlegung entspricht, war ein Grund, die Rollen von formalen Gegenständen und Merkmalen in BASE gegenüber dem ersten Ansatz zu vertauschen (siehe Diskussion in 4.2.3 auf Seite 134).

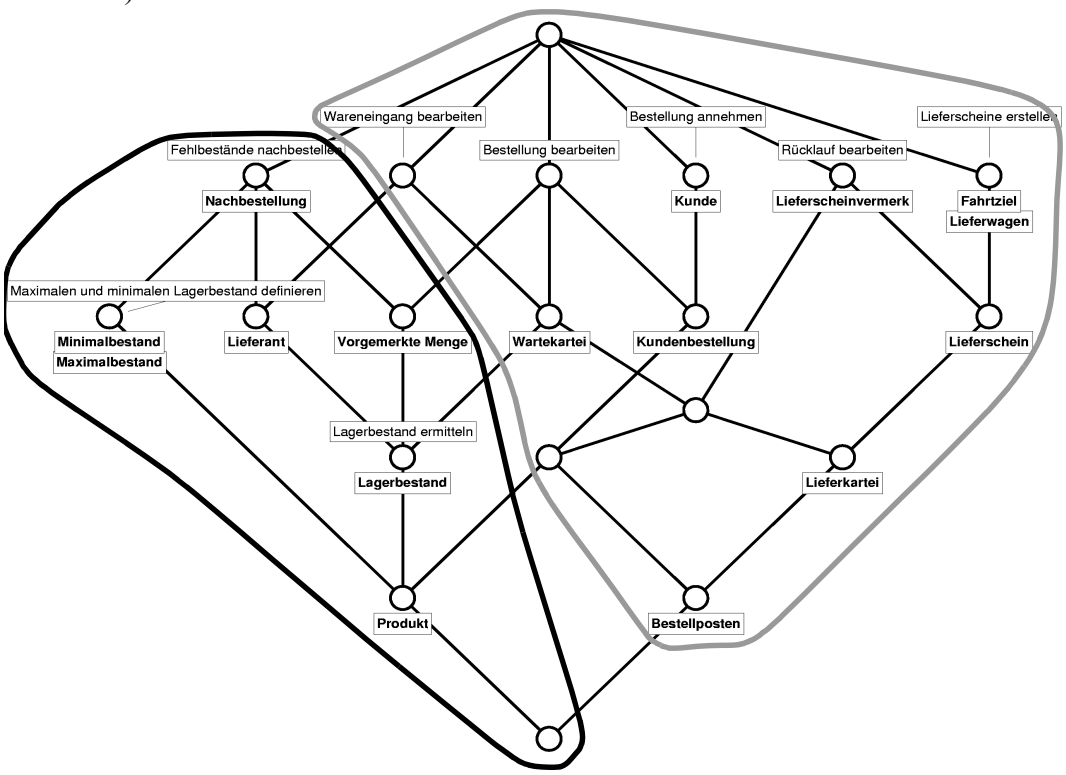


Abb. 4.24 Blöcke zur zweiten Blockrelation aus Abbildung 4.20

Die interne Struktur der Komponenten wird durch das originale Liniendiagramm beschrieben, in dem Komponenten als Blöcke identifiziert werden können. So hat der Entwickler die Möglichkeit, sein Modell in bezug auf eine Komponentenzerlegung in grober oder feiner Sicht zu betrachten. Dadurch bietet sich auch ein Ansatz, die Übersichtlichkeit der Liniendiagramme zu wahren, indem nur der Block

der gerade untersuchten Komponente angezeigt wird, wenn schon eine Komponentenzerlegung festgelegt wurde. (Siehe auch 4.7 auf Seite 177)

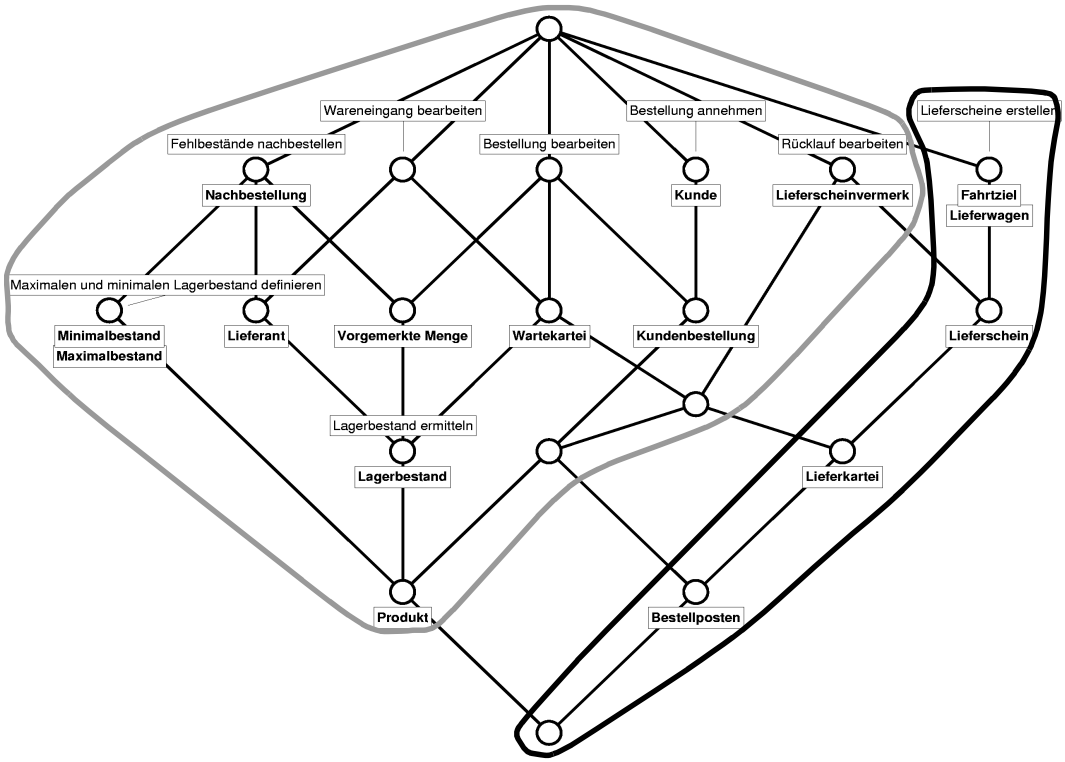


Abb. 4.25 Blöcke zur dritten Blockrelation aus Abbildung 4.20

4.6.3 "Erzwungene" Komponenten

Die Zerlegung in Blöcke ist gegenüber der in BASE vorgenommenen Schritten der Modellverfeinerung leider nicht stabil. Nicht in allen Verfeinerungsschritten bleibt eine einmal festgestellte Zerlegung in Blöcke erhalten. Im Beispiel sind auch nach der in 4.3.1 vorgenommenen Zerlegung des Anwendungsfalls *Bestellung annehmen* alle oben dargestellten Blockrelationen (mit Erweiterung um die neuen Funktionen und "Dinge") erhalten geblieben (und keine neuen entstanden), weil die Begriffsverbände des verfeinerten und des unverfeinerten Anwendungsfallmodells isomorph sind (siehe Abbildung 4.9 auf Seite 145 und Satz 19 auf Seite 121). Dagegen hat der in Abbildung 4.13 auf Seite 149 dargestellte Begriffsverband, der durch die zusätzliche funktionale Zerlegung des Anwendungsfalls *Bestellung bearbeiten* entstanden ist, keine nicht-trivialen Blockrelationen mehr.

Die vorher schon im unverfeinerten Diagramm gefundenen Blöcke findet man aber dennoch "von Hand" im verfeinerten Liniendiagramm (Abbildung 4.26) leicht wieder. Die oberen Intervallgrenzen lassen sich nämlich über ihren Begriffsumfang und die unteren über ihren Begriffsinhalt wiederfinden. Diese sind zwar im verfei-

nerten Begriffsverband erweitert worden, doch man findet sie eindeutig wieder, indem man jeweils den Abschluß im verfeinerten formalen Kontext bildet. Sucht man also die obere Intervallgrenze eines Blocks, geht man von seinem originalen Umfang über zum kleinsten der neuen Umfänge, der ihn enthält. Ist der ursprüngliche Begriffsumfang A, so bildet man also A^{LL}, wenn L die verfeinerte Relation ist. Das heißt, man sucht im verfeinerten Liniendiagramm das Supremum der Gegenstandsbegriffe der Gegenstände des ursprünglichen Begriffsumfangs. Dual findet man die unteren Intervallgrenzen als Infimum der Merkmalsbegriffe der Merkmale der ursprünglichen Begriffsinhalte dieser formalen Begriffe.

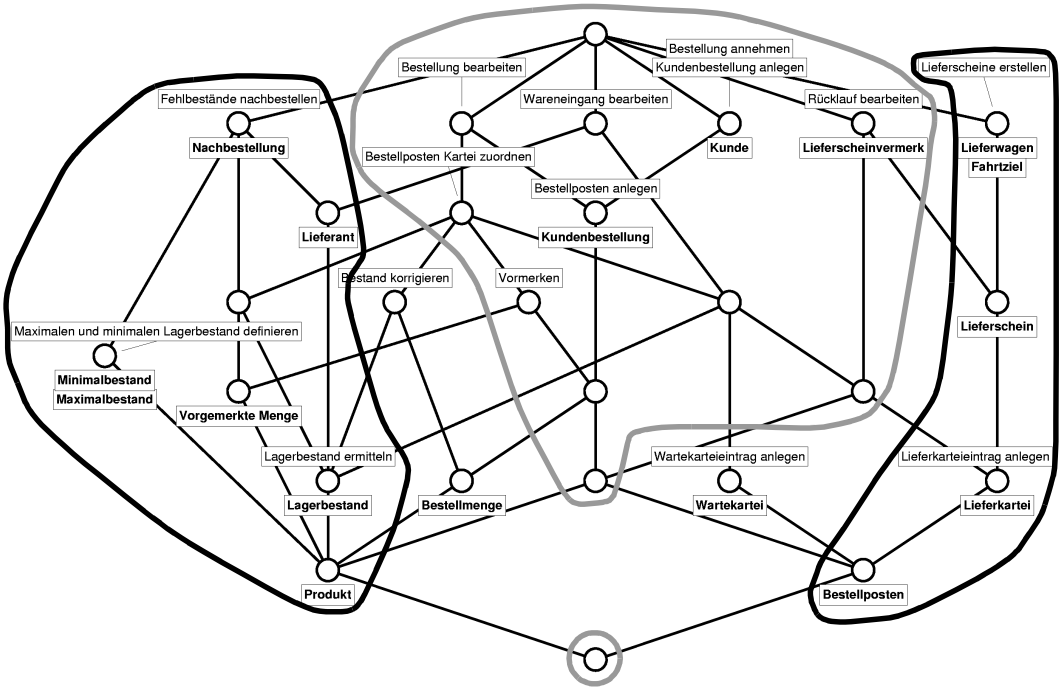


Abb. 4.26 Blöcke aus Abbildung 4.22

In Abbildung 4.26 identifiziert man dann sofort diejenigen drei Begriffe, die eine Zerlegung mit Hilfe einer Blockrelation verhindern. Es sind die Merkmalsbegriffe zu *Bestand korrigieren* und *Wartekarteieintrag anlegen* und der Gegenstandsbegriff zur *Bestellmenge*. Sie sind nämlich die einzigen, die nicht in einem der Blokkintervalle liegen. Sie bilden auch keine eigenen zusätzlichen Blöcke, weil bei einer solchen Aufteilung die in Hilfssatz 18 auf Seite 118 festgestellte Eigenschaft der gleichartigen Ordnung von oberen und unteren Intervallgrenzen verletzt wäre.

Der Entwickler kann aber hergehen und sie Blöcken "künstlich" zuordnen. Erst einmal sollen die beiden störenden Knoten zwischen dem linken und dem oberen Block aus Abbildung 4.26 behandelt werden. Ihre Zuordnung zum linken Block ergibt sich in diesem Fall sogar in natürlicher Weise aus der weiteren Analyse der

Anwendungsfälle von JWI. Man stellt nämlich fest, daß die Funktion *Bestand korrigieren* auch innerhalb der Zerlegung des Anwendungsfalls *Wareneingang bearbeiten* benutzt wird.

Der Anwendungsfall *Wareneingang bearbeiten* kann z.B. wie folgt zerlegt werden:

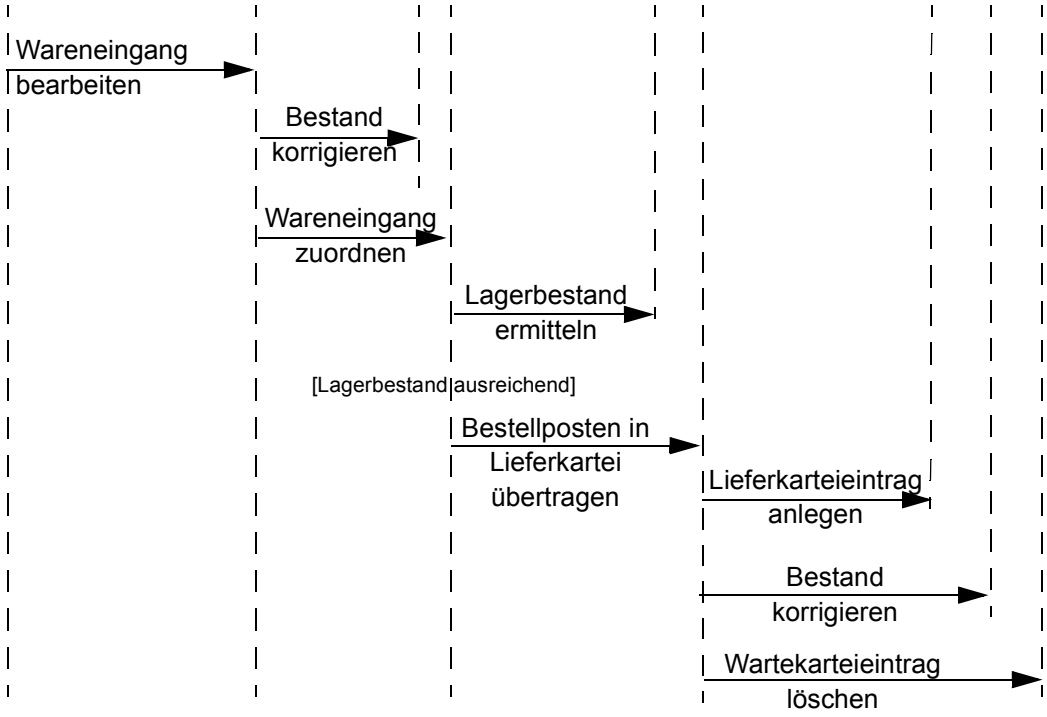


Abb. 4.27 Funktionale Zerlegung des Anwendungsfalls *Wareneingang bearbeiten*

In diesem Fall ist die Bezeichnung *Bestellmenge* für die eingetroffene Menge einer Weinsorte, um die der Lagerbestand erhöht werden muß, unzutreffend. Man ändere also diese Bezeichnung in *Produktmenge*. Danach fällt es leicht, zu sagen, daß dieses "Ding" auch in den Anwendungsfall *Lagerbestand ermitteln* involviert ist, denn dort wird eine Produktmenge festgestellt. (Nimmt man diese Zuordnung vor, spricht man aufgrund der bestehenden "*uses*"-Beziehungen die *Produktmenge* auch den Anwendungsfällen *Fehlbestände nachbestellen* und *Wareneingang bearbeiten* zu.). Aus dieser Korrektur erhält man das folgende Liniendiagramm mit den schon bekannten Blöcken. (Veränderungen gegenüber Abbildung 4.26 bzw. Abbildung 4.13 auf Seite 149 sind schwarz gekennzeichnet, unveränderte Diagrammelemente grau gezeichnet.)

Diese Modularisierungsmöglichkeit ergibt sich also allein durch die zusätzliche Zuordnung der *Produktmenge* zum Anwendungsfall *Lagerbestand ermitteln*. Das Erzwingen der Modularisierung hat in diesem Fall zu einer im Hinblick auf die weitere Analyse klareren Modellierung geführt.

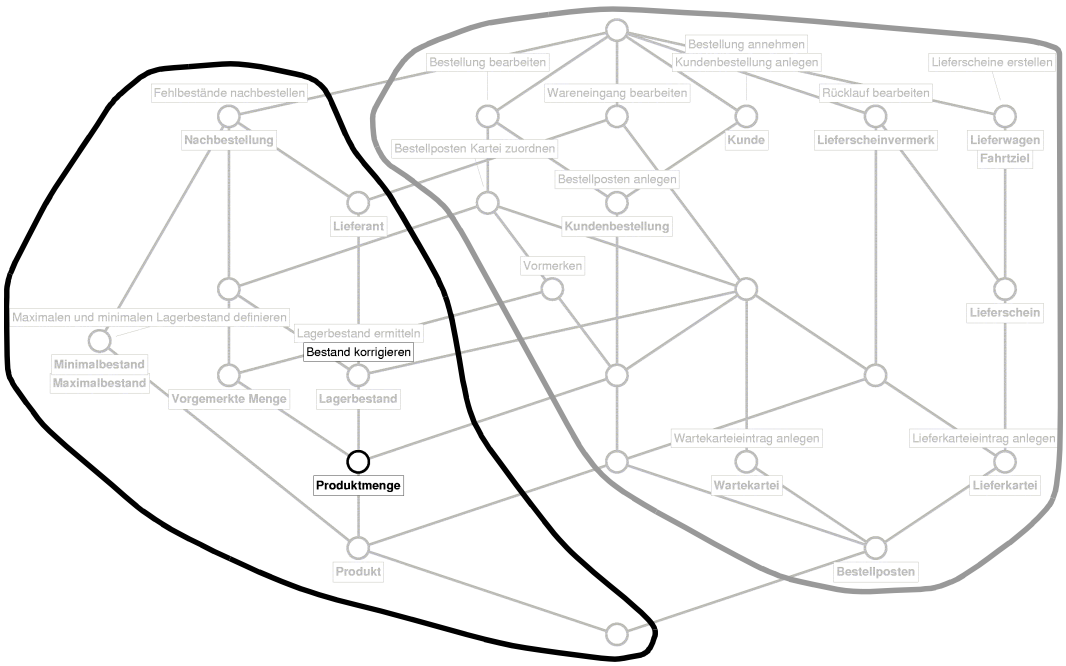


Abb. 4.28 Erste erzwungene Blockzerlegung

Genauso – nur schwerer inhaltlich zu begründen – kann man den störenden Knoten zwischen dem oberen und dem rechten Block behandeln und damit die feinere Blockzerlegung aus Abbildung 4.22 auf Seite 166 erzwingen.

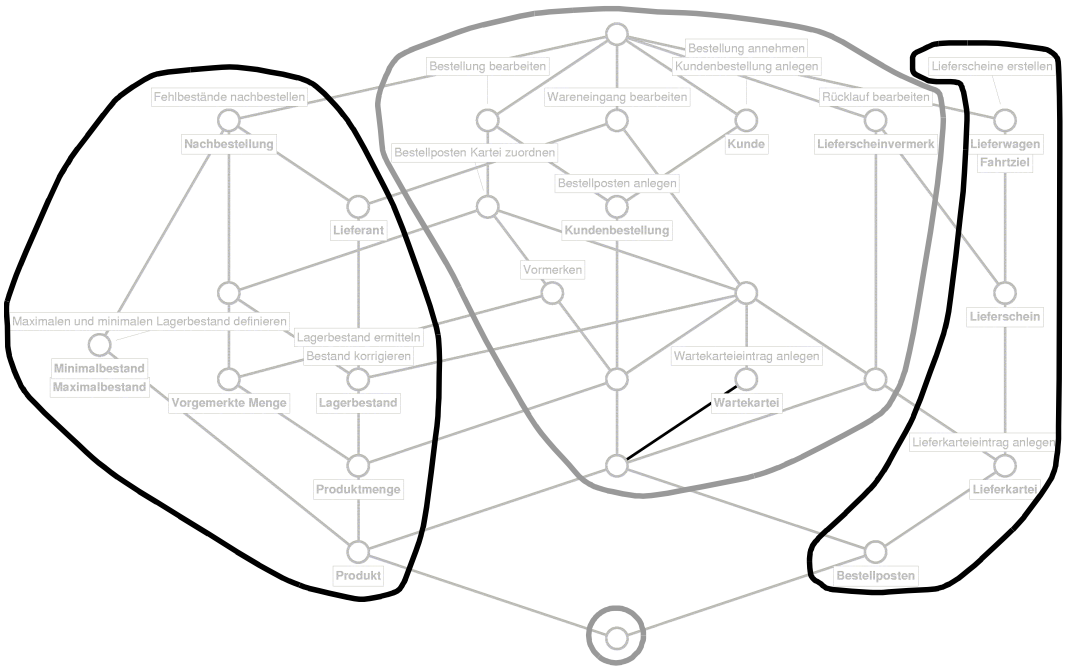


Abb. 4.29 Zweite erzwungene Blockzerlegung

Indem man etwa der Funktion *Wartekarteieintrag anlegen* zusätzlich das in einem

Eintrag referenzierte *Produkt* zuordnet, erhält man das Diagramm in Abbildung 4.29 mit den wohlbekannten Blöcken. (Die gegenüber Abbildung 4.28 veränderte Kante ist schwarz auf grau hervorgehoben.)

Schon festgestellte Zerlegungen können durch solch einfache Maßnahmen auch in verfeinerten Diagrammen erzwungen werden. Dabei trifft jedoch der Entwickler die Entscheidung über die Komponentenzerlegung und diese wird nicht mehr vollständig automatisch vorgeschlagen. Auf der anderen Seite fußen die automatisch generierten Modularisierungsvorschläge auch auf der Entwurfsentscheidung des Entwicklers, die er mit der Festlegung der Indizierung getroffen hat. Die Aufteilung in Komponenten ist einfach eine weitere Entwurfsentscheidung seinerseits. Durch die Festlegung von Komponenten wird das Modell einfacher.

Diese Betrachtung liefert jedoch noch kein oder höchstens ein sehr ineffizientes Mittel zur Hand, die formalen Begriffe bzw. Ober-/Unterbegriffsbeziehungen aufzuzeigen, die in einem Begriffsverband eine Zerlegung in Blöcke einer Toleranzrelation verhindern. Oben war dies nur dadurch einfach möglich, daß die Zielzerlegung durch die vorherige Analyse vorgegeben war. Alle möglichen Zerlegungen zu testen, ist praktisch nicht möglich und nicht sinnvoll, weil man innerhalb der Menge von $2^{\frac{(k-1)k}{2}}$ möglichen binären, reflexiven, symmetrischen Relationen auf der Menge der formalen Begriffe suchen müßte (mit k als Anzahl der Begriffe). "Nimmt man den Umweg" über die Blockrelationen, um eine Zerlegung anhand einer Toleranzrelation zu bestimmen, muß man "nur" 2^{mn-i} mögliche Relationen, wenn m die Anzahl der formalen Gegenstände, n die Anzahl der formalen Merkmale und i die Kardinalität der Inzidenzrelation – d.h. Anzahl der Kreuze in der Kontexttabelle – sind. (Diese Zahl ist in der Regel kleiner als die vorher angegebene, siehe B.2.1 auf Seite 283). Entscheidend ist, daß sich der zur Berechnung nötige Hüllenoperator für die Blockrelationen verhältnismäßig einfach beschreiben läßt (siehe B.4.1 auf Seite 293). In den erwähnten Untersuchungen zur Modularisierung von Altsystemen durch Lindig und Snelting ([L-S 97]) werden vornehmlich andere Verbandszerlegungen betrachtet. In erster Linie spielen Zerlegungen in horizontale Summanden – Diagrammteile die nach Entfernen des globalen Supremums und Infimums ohne Verbindung zueinander sind – eine Rolle. Da diese Zerlegung anhand des Liniendiagramms mit Hilfe von Graphenalgorithmen geschieht (vergl. [FLS 95]), können auch Diagrammteile ausgemacht werden, welche die Zerlegung verhindern. In solch einem Fall kann der Benutzer der entsprechenden Software entscheiden, ob er die Abhängigkeit über Komponentengrenzen hinweg in Kauf nimmt oder auf die weitere Modularisierung an der Stelle verzichtet. Eine entsprechende Technik für Zerlegungen durch Blockrelationen existiert (noch) nicht. Die Betrachtung horizontaler Summanden wäre auch innerhalb von BASE

möglich und entsprechende Komponentenzerlegungen bzw. Klassenkandidaten durch horizontale Summanden möglich. Allerdings sind weitgehend unabhängige Systemteile wie in [L-S 97] nicht zu erwarten. Im Fall der Modularisierung von Altsystemen ist das anders, weil die untersuchten Programme (hoffentlich) in gewissem Maße modular angelegt wurden.

4.6.4 Verschieden feine Komponentenzerlegungen

Die bisher angeführten Beispiele für Blockrelationen fassen immer größere Diagrammteile zusammen oder formen Blöcke mit einzelnen Knoten. Außerdem treten die Blöcke halbwegs gleichmäßig auf das Diagramm verteilt auf. Das muß nicht immer der Fall sein. Diagrammteile, die mit dem restlichen Diagramm wenig verzweigt sind (wo also nur lokale Datenabhängigkeiten vorliegen), werden durch Blockrelationen auch allein für sich gruppiert. So besitzt etwa der formale Kontext zur Zerlegung des Anwendungsfalls *Lieferscheine erstellen* (Begriffsverband dargestellt in Abbildung 4.17 auf Seite 154) eine Blockrelation, die nur zwei der originalen Begriffe in einem Intervall zusammenfaßt und alle anderen unberührt läßt. In Abbildung 4.30 ist dementsprechend nur der aus zwei Begriffe zusammengefaßte Block extra hervorgehoben.

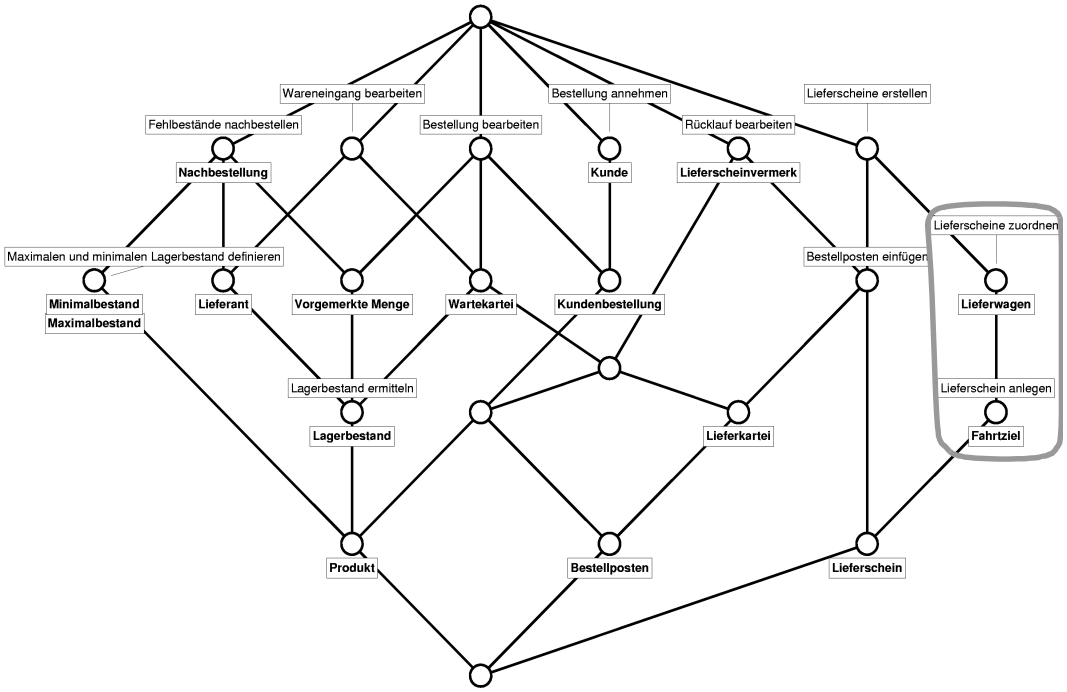


Abb. 4.30 Blöcke zur "ersten" Blockrelation zu Abbildung 4.17

In diesem Fall beschäftigt sich der neue Block gerade mit der Zusammenstellung

von Lieferungen nach den entsprechenden Fahrtzielen, die bei der Erstellung von Lieferscheinen auf diesen vermerkt werden. Ob nun die Zuordnung der Lieferscheine zu Touren und damit zu Lieferwagen erst nach Erstellung aller Lieferscheine geschieht – wie ursprünglich modelliert – oder schon sofort bei der Erstellung jedes einzelnen Lieferscheins vorgenommen wird, so daß das "Ding" Lieferwagen von der *Funktion Lieferschein* anlegen benutzt wird (wie durch die Blockrelation ergänzt), ist für den Rest des Systems unerheblich. Es bestehen keine Datenabhängigkeiten.

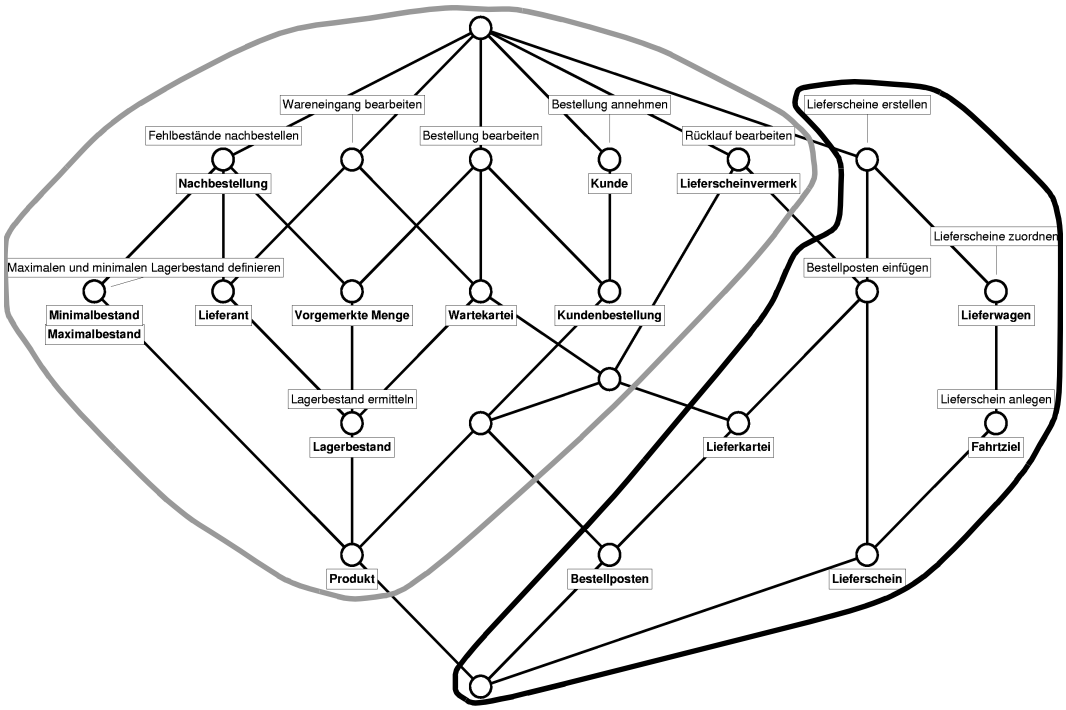


Abb. 4.31 Blöcke zur "zweiten" Blockrelation zu Abbildung 4.17

Der gerade besprochene formale Kontext zur Zerlegung des Anwendungsfalls *Lieferscheine erstellen* besitzt noch eine zweite nicht-triviale Blockrelation. Die ihr entsprechenden Blöcke sind in Abbildung 4.31 dargestellt.

Diese Aufteilung ist schon aus Abbildung 4.25 auf Seite 169 bekannt. Hier wird nur noch diese eine der beiden alternativen "Grobzerlegungen" des unverfeinerten Begriffsverbands durch Blockrelationen angeboten. Im folgenden sei erläutert, wie es dazu kommt.

Durch die Zerlegung des Anwendungsfalls *Lieferscheine erstellen* ist das Diagramm im zugehörigen Hauptideal aufgefächert worden, ohne daß die neuen formalen Begriffe viele Datenabhängigkeiten zum restlichen System aufzeigen. Faßt man wie oben sinnvoller Weise das "Ding" *Lieferschein* und den Anwendungsfall

Lieferscheine erstellen in einem Block zusammen, so liegen nach Hilfssatz 15 auf Seite 115 auch alle formalen Begriffe zwischen dem Merkmalsbegriff zu *Lieferscheine erstellen* und dem Gegenstandsbegriff zum *Lieferschein* in diesem Block. Alle diese Begriffe stehen also in der Komponentenbildung zugrundeliegenden vollständigen Toleranzrelation. Wegen der Reflexivität von Toleranzrelationen steht jeder formale Begriff zu sich selber in Relation. Wegen der Infimumsverträglichkeit stehen damit auch das globale Infimum (als Infimum der Gegenstandsbegriffe zu *Lieferkartei* und *Lieferschein*) und der Gegenstandsbegriff (als Infimum von sich selber und dem Merkmalsbegriff zu *Lieferscheine erstellen*) in Relation und damit wiederum alle formalen Begriffe dazwischen. Nimmt man weiter die Zusammenfassung der Merkmalsbegriffe zu *Bestellung annehmen* und *Bestellung bearbeiten* an – wie in den Blöcken des unverfeinerten Begriffsbands immer der Fall, so liegt nach Hilfssatz 15 auf Seite 115 das globale Supremum als Supremum der beiden zugehörigen Merkmalsbegriffe im selben Block. Damit stehen auch der Gegenstandsbegriff zum *Bestellposten* (als Infimum der Merkmalsbegriffe zu *Bestellung annehmen* und *Lieferscheine erstellen*) und der Merkmalsbegriff zu *Lieferscheine erstellen* (als Infimum des globalen Supremums und sich selbst) in Relation. Weiter stehen dann der Gegenstandsbegriff zum *Bestellposten* (als Infimum von sich selbst und dem Merkmalsbegriff zu *Lieferscheine erstellen*) und der Merkmalsbegriff zu Lieferscheine zuordnen (als Infimum des Merkmalsbegriff zu *Lieferscheine erstellen* und sich selbst) in Relation. Aus Hilfssatz 15 bekommt man dann den in Abbildung 4.31 rechts dargestellten Block.

Würde man zusätzlich das globale Infimum und den Merkmalsbegriff zu *Fehlbestände nachbestellen* wie in der zweiten alternativen Grobzerlegung einem gemeinsamen Block zuschlagen, so ständen wegen der Supremumsverträglichkeit der zugrunde liegenden Toleranzrelation auch die Merkmalsbegriffe zu Fehlbestände nachbestellen (als Supremum von sich selbst und dem globalen Infimum) und zu *Lieferscheine erstellen* (als Supremum des globalen Supremums und sich selbst) in Relation und somit hätte man nach Hilfssatz 15 auf Seite 115 nur einen Block, der alle Begriffe umfaßt.

Dementsprechend ergibt die Verfeinerung "der anderen Diagrammseite" – also des Anwendungsfalls *Fehlbestände nachbestellen* – die andere der beiden alternativen Grobzerlegungen. Die einzige Blockrelation des formalen Kontexts zu Abbildung 4.19 auf Seite 156 ergibt die Blockzerlegung in Abbildung 4.32.

Blockrelationen bieten also Modularisierungsvorschläge und helfen Klassenkandidaten bzw. Systemkomponenten festzulegen. Nicht gesagt ist, daß die untersuchten Begriffsverbände interessante Blockrelationen besitzen (siehe auch Anhang B.4.3). Ist das aber der Fall, liefert die Technik Liniendiagramme für eine grobe Sicht auf die Komponenten und für eine feine Sicht auf die Komponenteninterna.

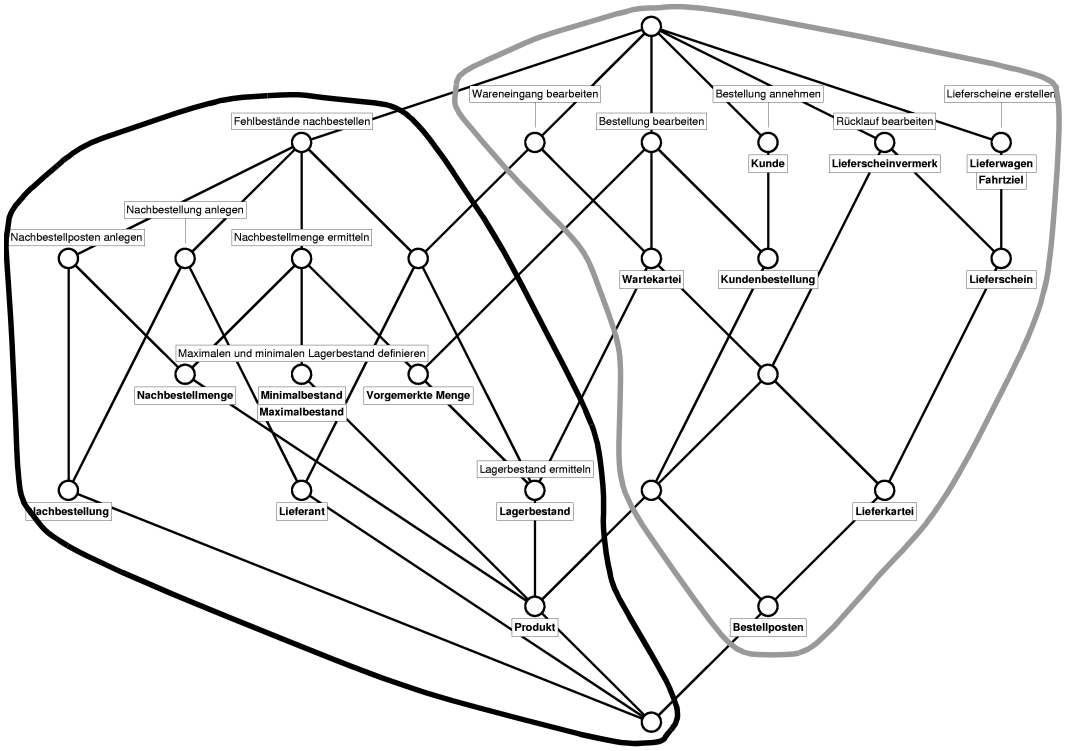


Abb. 4.32 Blöcke des Begriffsverbands aus Abbildung 4.19

Nach Bemerkung 11 auf Seite 118 müssen die Blöcke einer vollständigen Toleranzrelation nicht notwendigerweise disjunkt sein. Tritt dieser Fall auf, so teilen zwei Komponenten einen Kern von Funktionen und "Dingen". Nach Bemerkung 11 und Hilfssatz 18 auf Seite 118 liegen sie "quer" zueinander, d.h. in jeder Komponente gibt es sowohl Funktionen als auch "Dinge", die in der anderen nicht enthalten sind. In einer solchen Situation muß der Entwickler entscheiden, ob er die beiden Komponenten vereint, den gemeinsamen Kern einer der beiden Komponenten zuschlägt oder ihn zu einer eigenen Komponente formt (und dann vielleicht die beiden ursprünglichen Komponenten noch weiter aufteilt, um eine eindeutige Benutzungshierarchie ohne gegenseitige Benutzungsbeziehungen erhalten).

4.7 Zur Übersichtlichkeit des Liniendiagramms

Schon kleine Untersuchungsbereiche führen zu schwer überschaubaren Liniendiagrammen, wenn die betrachteten Anwendungsfälle viele Datenabhängigkeiten aufweisen. Dadurch wird schon das Zeichnen und vor allem die Interpretation der Diagramme stark erschwert.

Schon in 4.6 auf Seite 168 wurde die Möglichkeit angesprochen, nur eine einzelne

Komponente, die durch eine Blockzerlegung definiert wurde, als Diagramm anzuzeigen. Dies kann der Übersichtlichkeit des Liniendiagramms dienen. Nach Satz 18 auf Seite 121 bedeutet dieses Vorgehen, daß ein Liniendiagramm eines Teilkontexts gezeichnet wird. Die "Dinge" in diesem Teilkontext sind durch den Begriffsumfang des Blocksupremums und die Anwendungsfälle bzw. Funktionen durch den Begriffsinhalt des Blockinfimums gegeben. Damit werden alle "Dinge" berücksichtigt, deren Gegenstandsbegriffe innerhalb des Blocks liegen und zusätzlich alle die aus benutzten Komponenten – d.h. solche, deren Gegenstandsbegriffe außerhalb des Blocks liegen, aber Unterbegriffe des Blocksupremums sind. Dual sind in jenem Teilkontext die Anwendungsfälle bzw. Funktionen erfaßt, deren Merkmalsbegriffe innerhalb des Blocks liegen und zusätzlich alle aus benutzenden Komponenten – d.h. solche, deren Merkmalsbegriffe zwar außerhalb des Blocks liegen, aber Oberbegriffe des Blockinfimums sind. Diese zusätzlich berücksichtigten formalen Gegenstände und Merkmale erscheinen nicht als Beschriftung im Liniendiagramm, wenn man dieses einfach aus dem ursprünglichen herausschneidet, jedoch im Liniendiagramm des entsprechenden Teilkontexts. Sie sollten einbezogen werden, um die Einordnung der Komponente in das Gesamtsystem zu veranschaulichen. So sieht man, wo welche "Dinge" aus anderen Komponenten benutzt werden und welche Anwendungsfälle aus anderen Komponenten welche Komponententeile benutzen. Die entsprechende Beschriftung sollte allerdings den Unterschied zwischen der Komponente zugehörigen und externen "Dingen" und Anwendungsfällen bzw. Funktionen kenntlich machen.

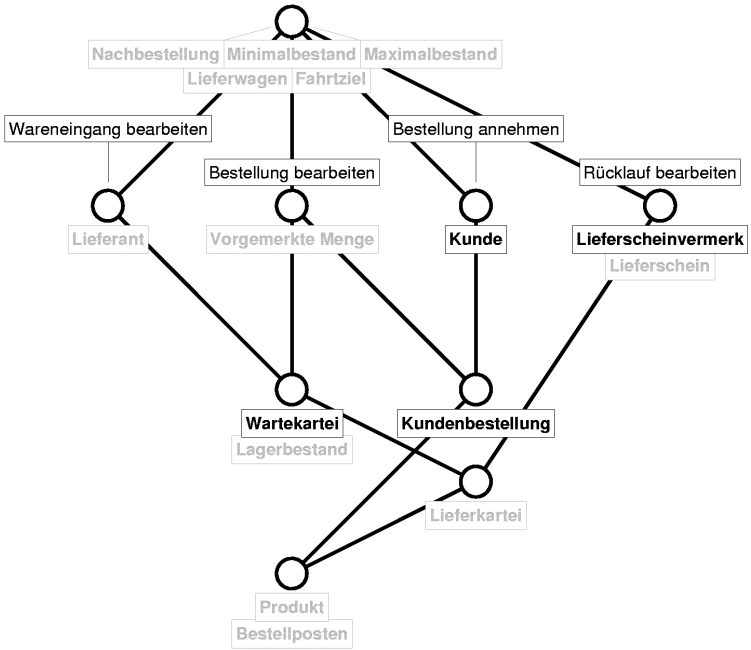


Abb. 4.33 Ein Block aus Abbildung 4.22

Abbildung 4.33 gibt in solcher Weise die oberste Komponente aus Abbildung 4.22 auf Seite 166 wieder. Die benutzten "Dinge" aus anderen Blöcken sind grau gezeichnet. Die am globalen Supremum angegebenen sind hier in keinen Anwendungsfall involviert. (Man kann "Dinge" aus anderen Blöcken auch weglassen, wenn sie unter keinen Merkmalsbegriff fallen.) Berechnet wurde in diesem Fall das Diagramm aus dem oben beschriebenen Teilkontext. So bekommt man die "Dinge" aus benutzten Blöcken automatisch dazu. Damit das Diagramm nicht neu ausgerichtet werden muß, ist es aber sinnvoll, es aus dem Gesamtdiagramm herauszuschneiden. Dann müssen die zusätzlichen "Dinge" extra gesucht werden. Dabei besteht auch die Option, Funktionen aus benutzten Komponenten zu berücksichtigen. Diese lassen sich aber keinem der Knoten des Liniendiagramms sinnvoll und korrekt zuordnen. (Z.B. hat in Abbildung 4.33 kein formaler Begriff den Umfang {*Produkt*, *Lagerbestand*}. Also gibt es für den benutzten Anwendungsfall *Lagerbestand ermitteln* keinen passenden Knoten.) Man müßte solche Funktionen außerhalb der Liniendiagrammdarstellung notieren, oder für sie neue Knoten einführen. Letzteres würde das Diagramm mitunter doch wieder stark vergrößern. Aus diesen Gründen sind hier benutzte Funktionen aus anderen Blöcken nicht wiedergegeben.

Wenn jedoch ein von anderen benutzter Block dargestellt wird, können leicht alle benutzenden Anwendungsfälle in die Darstellung integriert werden. So könnte der linke Block aus Abbildung 4.22 auf Seite 166 wie in Abbildung 4.34 präsentiert werden.

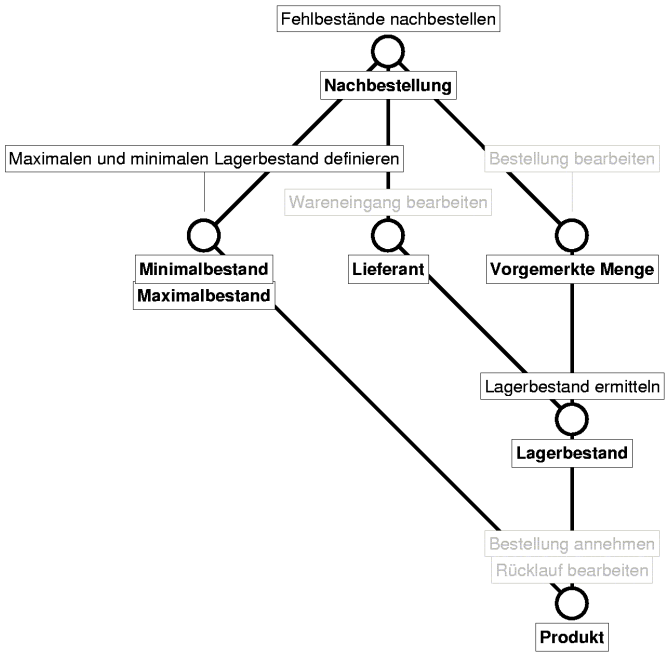


Abb. 4.34 Ein zweiter Block aus Abbildung 4.22

Steht einmal eine Komponentenstruktur, die nach einer Blockrelation gebildet ist, fest, wird es dem Entwickler häufig reichen, nur die Komponente zu sehen. Trotzdem muß das Systemmodell komponentenübergreifend gepflegt werden, um Abhängigkeiten zwischen den Komponenten im Griff zu behalten. Werden also innerhalb einer Komponente neue Funktionen oder "Dinge" betrachtet, ist auch die Indizierung der restlichen Systemelemente (die vielleicht noch nicht einmal wie in Abbildung 4.33 und Abbildung 4.34 zusätzlich angezeigt sind) in bezug auf diese neuen Elemente zu hinterfragen und festzulegen.

Als andere Strukturen, die formale Begriffe zusammenfassen, waren in 4.3.1 und 4.5.1 wiederholt Hauptideale und -filter genannt worden. Hauptideale zu Merkmalsbegriffen von Anwendungsfällen enthalten die bei der funktionalen Zerlegung des entsprechenden Anwendungsfalls entstandenen Funktionen. Das Hauptideal zur Funktion *Bestellposten anlegen* aus der Verfeinerung des Anwendungsfalls *Bestellung bearbeiten* (Abbildung 4.13 auf Seite 149) ist zum Beispiel in Abbildung 4.35 wiedergegeben. Hauptfilter können dazu dienen, Attribute und Operationen zu Klassenkandidaten zu finden.

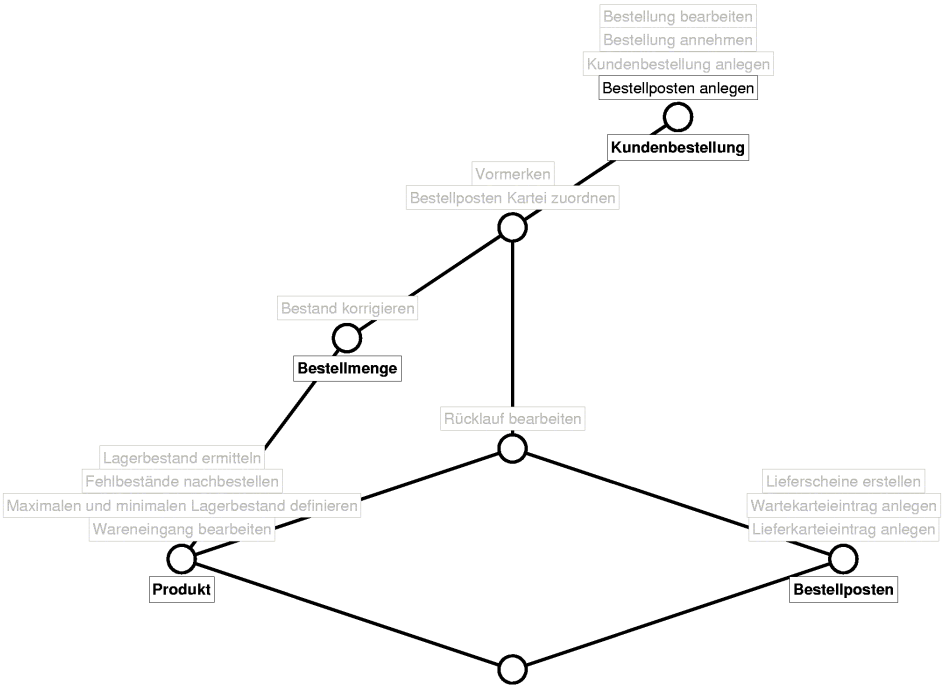


Abb. 4.35 Hauptideal zu *Bestellposten anlegen*

Die Einschränkung des Liniendiagramms auf ein solches Hauptideal ist im besonderen dann hilfreich, wenn die Diagrammkanten ineinander verschränkt verlaufen, so daß es schwierig sein kann, zu einem formalen Begriff mit einem Blick seine Ober- und Unterbegriffe zu identifizieren. Für die Durchführung einer funktiona-

len Zerlegung ist eine solche Betrachtungsweise schon ausreichend. Zu welchen Anwendungsfällen und Funktionen außerhalb des Hauptideals Datenabhängigkeiten bestehen, ist in Abbildung 4.35 wiederum durch die grau gehaltene Beschriftung angegeben. Da das Infimum in jedem Hauptideal liegt, werden so immer alle Anwendungsfälle betrachtet. Nur solche die dem Infimum selber zugeordnet sind, weisen keine Datenabhängigkeiten mit dem Anwendungsfall auf, zu dem das Hauptideal untersucht wird.

Ebenso reicht bei der Betrachtung von Klassenkandidaten unter den "Dingen" oft ein Blick auf den entsprechenden Hauptfilter aus, um ihm Attribute und Operationen zuzuordnen. In diesem Fall werden alle "Dinge" im Diagrammausschnitt repräsentiert. Der Hauptfilter des Gegenstandsbegriffs zur Wartekartei aus dem Begriffsverband in Abbildung 4.13 auf Seite 149 sieht etwa wie nebenstehend aus.

Zu allen "Dingen", die nicht dem Supremum zugeordnet sind, gibt es Anwendungsfälle oder Funktionen, die diese "Dinge" gemeinsam mit der Wartekartei berühren.

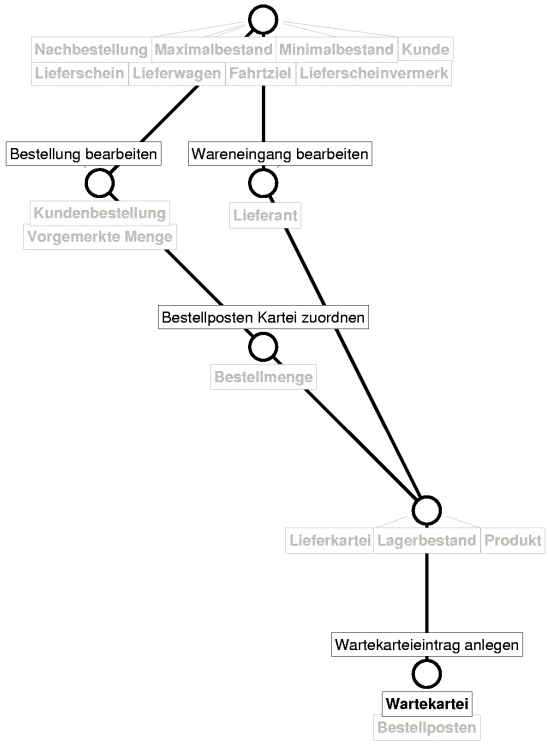


Abb. 4.36 Hauptfilter zur *Wartekartei*

4.8 Vorgehensmodell

Nachdem nun die einzelnen Techniken innerhalb von BASE beschrieben sind, soll hier zusammengefaßt werden, wie sie in der Analyse eines Software-Entwicklungsprojekts eingesetzt werden können.

Ausgangspunkt der Analyse sind die Anwendungsfälle des zu entwickelnden Systems. Es ist ein Anwendungsfallmodell nach Jacobson ([JCJÖ 93]) zu erstellen. Man beginne mit allen Anwendungsfällen, die unmittelbar von den Fachexperten genannt werden. Die wichtigsten lassen sich aus den funktionalen Anforderungen an das System ableiten. Zumindest die Grundidee dieser Anforderungen ist dem entsprechenden Projektauftrag zu entnehmen. Mit fortschreitender Analyse erge-

ben sich dann zusätzliche Anwendungsfälle, die in die Betrachtung einbezogen werden.

Wie schon in 4.2.1 auf Seite 129 erläutert, sollten zu Beginn gleich alle bekannten Anwendungsfälle in die Betrachtung eingeschlossen – und nicht erst sukzessive ergänzt – werden, um das Ausmaß der von Jacobson geschilderten Revisionen des Anwendungsfallmodells zu minimieren. Aus dem gleichen Grund sollte eine funktionale Zerlegung wie in 4.3.1 erst nach der Einbeziehung aller bekannten Anwendungsfälle geschehen. Für den Entwickler ist es erst einmal wertvoll, eine Übersicht über den ganzen Untersuchungsbereich zu bekommen, bevor er Einzelheiten mit den Fachexperten erörtern kann. Die grobe Sicht mit Hilfe der Anwendungsfälle und der darin involvierten Dinge ermöglicht schon weitgehende Einsichten in die Struktur des Untersuchungsbereichs (siehe 4.2.4).

Der Analyseprozeß unterteilt sich damit in die zwei Phasen *Grobanalyse* und *Feinanalyse*:

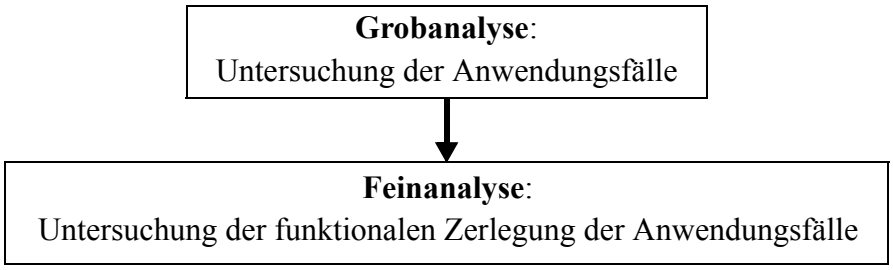


Abb. 4.37 Analysephasen in BASE

Bevor im folgenden die Abfolge der einzelnen Tätigkeiten innerhalb der beiden Phasen beschrieben wird, seien die in 4.2 bis 4.6 dargestellten Techniken noch einmal aufgelistet. Sie kommen in den beiden verschiedenen Phasen mit verschiedenem Gewicht zu tragen und erbringen auch verschiedene Ergebnisse. In folgender Tabelle ist dargelegt, welche Technik in welcher der beiden Phasen zu welchem Zweck angewandt wird. Dazu sind die Tabellenzeilen mit den Techniken beschriftet und die Tabellenspalten mit den entsprechenden Untersuchungszielen. (Die letzte Spalte gibt noch an, in welchem Abschnitt der Arbeit die jeweilige Technik beschrieben ist.) Ein "G" in einer Tabellenzelle bedeutet, daß die in der betreffenden Zeile angegebene Technik in der Grobanalyse zu dem Zweck eingesetzt wird, der in der Spalte angegeben ist. Ein "F" trifft die entsprechende Aussage für die Phase der Feinanalyse.

In der Grobanalyse wird allein der top-down-Ansatz zur Identifikation von Klassenkandidaten unter den "Dingen" verfolgt, weil die passenden Klassenoperationen noch nicht in der funktionalen Zerlegung der Anwendungsfälle hergeleitet wurden. Eben aus diesem Grund erbringt in dieser Phase die Untersuchung von

Blockrelationen in der Regel Komponenten- und nicht Klassenstrukturen zum Anschein. Eine Indizierungsüberprüfung kann zu jeder Zeit vorgenommen werden. Innerhalb der Feinanalyse werden Gegenstandsimplikationen darüber hinaus noch speziell für den Zweck eingesetzt, neue Funktionen innerhalb einer Zerlegung zu finden.

	Identifikation von Klassen-kandidaten	Erstellung einer Komponenten-zerlegung	Indizie-rungsüber-prüfung	Auffinden neuer Funktionen	Beschrie-ben in Abschnitt:
Untersuchung von Gegenstandsbegriffen zu "Dingen"	G F				4.2.4 4.5.1
Betrachtung zusam-menfallender Merk-mals- und Gegen-standsbegriffe	F				4.5.2
Untersuchung von Blockzerlegungen	F	G F			4.6
Überprüfungsfragen anhand von Gegen-standsimplikationen			G F	F	4.3 4.4
Überprüfungsfragen anhand von Merkmal-simplikationen			G F		4.4

Abb. 4.38 Techniken von BASE

Diese Zusammenstellung sollte die Verfolgung der folgenden detaillierten Phasenbeschreibung erleichtern.

Die erste Phase *Grobanalyse* zur Untersuchung der "top level"-Anwendungsfälle wird durch Abbildung 4.39 beschrieben. Das grundsätzliche Vorgehen bei der Behandlung der Anwendungsfälle wurde schon in Abbildung 4.1 auf Seite 130 wiedergegeben. Hier sind jetzt aber die in 4.2 bis 4.6 beschriebenen Techniken eingeordnet.

Nachdem der erste Grundstock von Anwendungsfällen festgelegt ist, kommen die begriffsanalytischen Mittel von BASE ins Spiel. Anhand der Indizierung wird ein Modell der Datenabhängigkeiten als Liniendiagramm erstellt. Dieses kann anhand von Gegenstands- und Merkmalsimplikationen überprüft werden. Natürlich können geübte Nutzer auch direkt anhand des Liniendiagramms diskutieren. (Solche Verkürzungen sind der Übersichtlichkeit halber in Abbildung 4.39 nicht dargestellt.) Als unzutreffend erkannte Implikationen führen zu einer Veränderung der Indizierung, nämlich der

- Behebung von Indizierungsfehlern

- Hinzunahme neuer "Dinge" aufgrund unzutreffender Merkmalsimplikationen
- Hinzunahme neuer Anwendungsfälle aufgrund unzutreffender Gegenstandsimplikationen

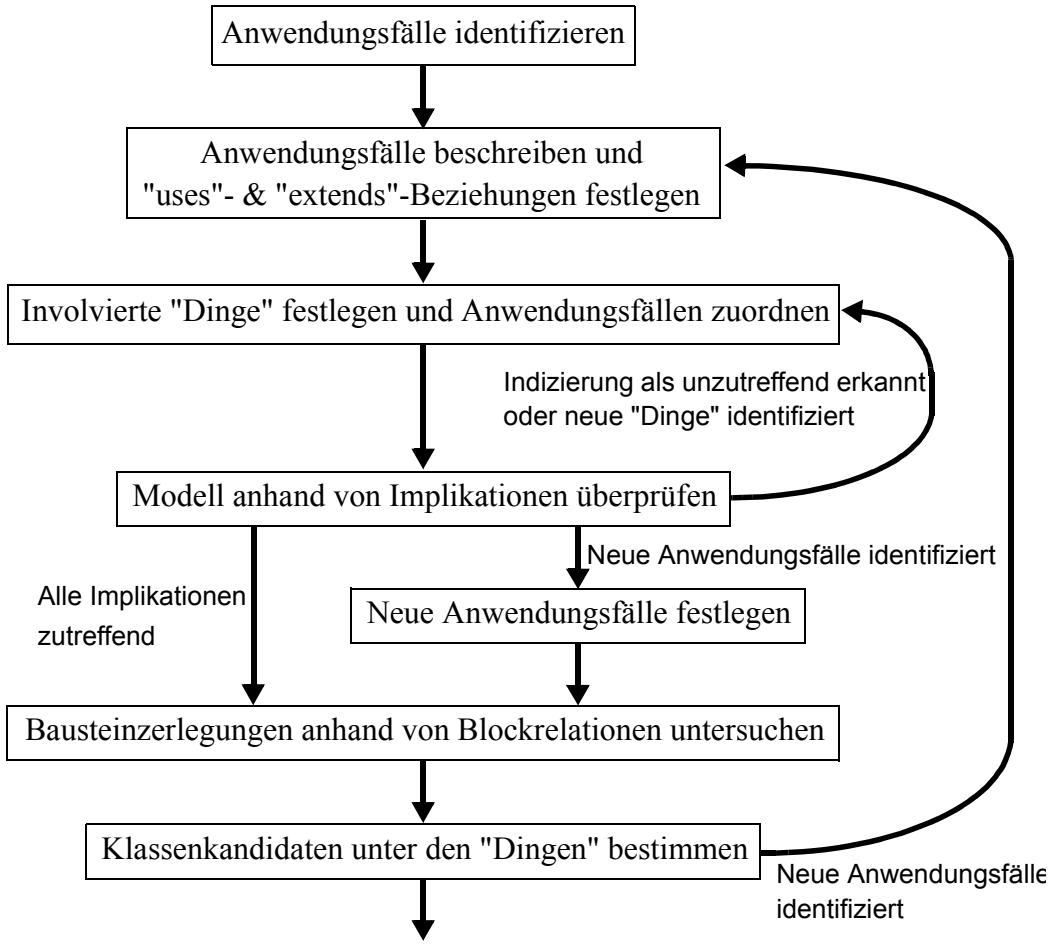


Abb. 4.39 Grobanalyse: Untersuchung der Anwendungsfälle

Die ersten beiden der oben aufgeführten Fälle führen sofort zu einer neuen Indizierung im formalen Kontext. Bei der Identifikation von neuen Anwendungsfällen ist vorgesehen, zunächst die Zerlegung via Blockrelationen und die Betrachtung von Klassenkandidaten unter den "Dingen" anhand des unveränderten Kontexts vorzunehmen. Maßgeblich dafür ist die Überlegung, daß das Modell ohne die neuen identifizierten Anwendungsfälle einfacher ist. Dennoch kann man unterstellen, daß die bisherige Auswahl der betrachteten Anwendungsfälle schon eine gute Übersicht über die Systemfunktionalität liefert. Deshalb können im ursprünglichen Kontext aufgedeckte Strukturen – insbesondere Komponentenzerlegungen – auch für die weitere Entwicklung hilfreich sein, sind aber eventuell im späteren komplizierteren Modell nur schwer aufzudecken (vergl. 4.6 auf Seite 169). Natürlich ist

es auch möglich, die neuen Anwendungsfälle direkt zu berücksichtigen – insbesondere bei einem noch kleinen und überschaubaren Begriffsverband. (Aus Gründen der Übersichtlichkeit ist diese Möglichkeit nicht in Abbildung 4.39 als eigener Pfeil eingezeichnet.)

Werden neue "Dinge" oder Anwendungsfälle mittels fachlich unzutreffender Implikationen gefunden, geschieht dies sukzessive, wenn die entsprechenden Überprüfungsfragen einzeln nacheinander bearbeitet werden und die Erweiterung des formalen Kontexts sofort nach dem Ablehnen einer Implikation vorgenommen wird. D.h. bei diesem Vorgehen werden neue "Dinge" oder Anwendungsfälle meist einzeln in das Modell aufgenommen, weil zur Auflösung der als fachlich unzutreffend erkannten Implikation ein Gegenbeispiel ausreicht. Möglich ist aber auch erst die gesamte Überprüfung und eventuell auch eine nachfolgende Untersuchung auf Blockzerlegungen durchzuführen und die vielleicht aufgelaufenen neuen "Dinge" oder Anwendungsfälle auf einen Schlag zu ergänzen. Deshalb ist in der obigen Formulierung keine Einschränkung auf "Einzelergänzungen" gemacht.

Schon angesprochen sind in einem Liniendiagramm bestehende Komponentenzerlegungen, die eventuell durch Hinzunahme neuer Anwendungsfälle später nicht mehr durch Blockrelationen rekonstruiert werden. Hat der Entwickler einmal eine ihm wertvolle Zerlegung gefunden, sollte sie dokumentiert bleiben. So kann sie in späteren Liniendiagrammen – wie in Verbindung mit Abbildung 4.26 auf Seite 170 diskutiert – im Schritt *Bausteinzerlegungen anhand von Blockrelationen untersuchen* mit berücksichtigt werden.

Die Untersuchung von Klassenkandidaten unter den betrachteten "Dingen", die in 4.2.4 und 4.5.1 als eine grundlegende Idee von BASE an den Anfang der Erörterungen gesetzt wurde, ist hier nach hinten verschoben worden, weil die Beschäftigung mit möglichen Komponentenzerlegungen im vorhergehenden Schritt noch ein weiteres Durchdringen des Untersuchungsbereichs mit sich bringt. Häufig wird das nicht nötig sein, weil schon aus der Anordnung der Gegenstandsbegriffe zu den "Dingen" eindeutige Klassenkandidaten ins Auge springen. Die ersten Klassenkandidaten werden auch aus solchen Betrachtungen resultieren, denn die einfachsten Klassenkandidaten entsprechen Jacobsons Entitätsklassen. Zu ihnen können durch die Betrachtung des entsprechenden Hauptfilters Attribute gesucht werden (siehe 4.5.1). Operationen sind in diesem Stadium noch schwer zuzuordnen, weil bisher nur "top-level"-Anwendungsfälle betrachtet wurden. Solche Anwendungsfälle kommen in den seltensten Fällen Entitätsklassen als Operationen zu, weil diese in erster Linie der Datenspeicherung dienen, während innerhalb der Anwendungsfälle viel Koordination geleistet werden muß. Im Sinne von Jacobson kommen dafür eher Kontroll- oder Schnittstellenklassen in Frage (siehe 2.6.1, Seite 54).

Der abschließende Pfeil in Abbildung 4.39 deutet an, daß nach der beschriebenen Grobanalyse noch die funktionale Zerlegung der Anwendungsfälle erfolgen soll. Das Vorgehen in dieser zweiten Phase ist ganz ähnlich, nur daß dann feiner-granulare Funktionen zusätzlich zu den übergreifenden Anwendungsfällen betrachtet werden. Es ist in Abbildung 4.40 schematisch dargestellt. Im Gegensatz zur ersten Phase, die einen breiten Überblick über den Untersuchungsbereich liefern soll, arbeitet man sich in der zweiten Phase in die Tiefe. Einzelne Anwendungsfälle werden detailliert betrachtet. So wird die Betrachtung von zusammenfallenden Merkmals- und Gegenstandsbegriffen nach 4.5.2 (Seite 162) zur Bildung von Klassenkandidaten – oder zumindest ihren Teilen – erst interessant. Außerdem ist die Granularität der Zerlegung zu steuern. Aus diesem Grund werden in der Darstellung des Vorgehens in Abbildung 4.40 Gegenstands- und Merkmalsimplikationen getrennt behandelt. (Erstere dienen nach 4.3.2 zur Überprüfung der Zerlegungsdetaillierung.)

Der eigentlichen Zerlegung vorgeschaltet ist ein Schritt zur Auswahl des zu verfeinernden Modells. Dem liegt die Idee zugrunde, daß man im allgemeinen zunächst jeden Anwendungsfall für sich einzeln betrachtet, um sich voll auf seine Details konzentrieren zu können. Man kann aber auch die enge Verbindung mehrerer Anwendungsfälle zum Anlaß nehmen, die Zerlegung aus einem Modell heraus vorzunehmen, in dem schon andere Anwendungsfälle funktional verfeinert sind – wie es etwa in Abbildung 4.13 auf Seite 149 geschehen ist. Dementsprechend ist im letzten Schritt auch eine Verzweigung vorgesehen. Von Fall zu Fall kann es entweder sinnvoll sein, sich direkt in die funktionale Zerlegung des nächsten Anwendungsfalls zu stürzen, oder vorher schon vorgenommene Zerlegungen verschiedener Anwendungsfälle in einem gemeinsamen Modell zu betrachten. Die zweite Möglichkeit bedeutet, daß man die verfeinerten Indizierungen in einem formalen Kontext zusammenführt und das Liniendiagramm zu diesem untersucht.

Als eigener Schritt ist jeweils die Beschreibung von Anwendungsfällen bzw. Funktionen erwähnt. Dahinter steckt die Idee, daß bei der Erstellung dieser Beschreibung auch die involvierten "Dinge" festgelegt werden (siehe 5.1.1). Natürlich sollten auch Beschreibungen der "Dinge" in ein Datenlexikon aufgenommen werden, um die Etablierung einer "Projektsprache" zu fördern. Nur anhand solcher Dokumente können neue Mitarbeiter später in das Projekt integriert werden. Dies ist in der Darstellung von BASE nicht weiter erwähnt, weil die Pflege eines solchen Datenlexikons innerhalb der Projektbibliothek unabhängig von der eingesetzten Modellierungsmethode immer vorgenommen werden sollte. Die Beschreibungen von "Dingen" spielen für die spezifischen Techniken in BASE selber keine Rolle. Interessant wäre natürlich die Kombination mit einem datenorientierten Ansatz (siehe 7.2.4 auf Seite 250). Auch für die Organisation von Datenlexika kann Formale

Begriffsanalyse gewinnbringend eingesetzt werden (siehe 6.5 auf Seite 237).

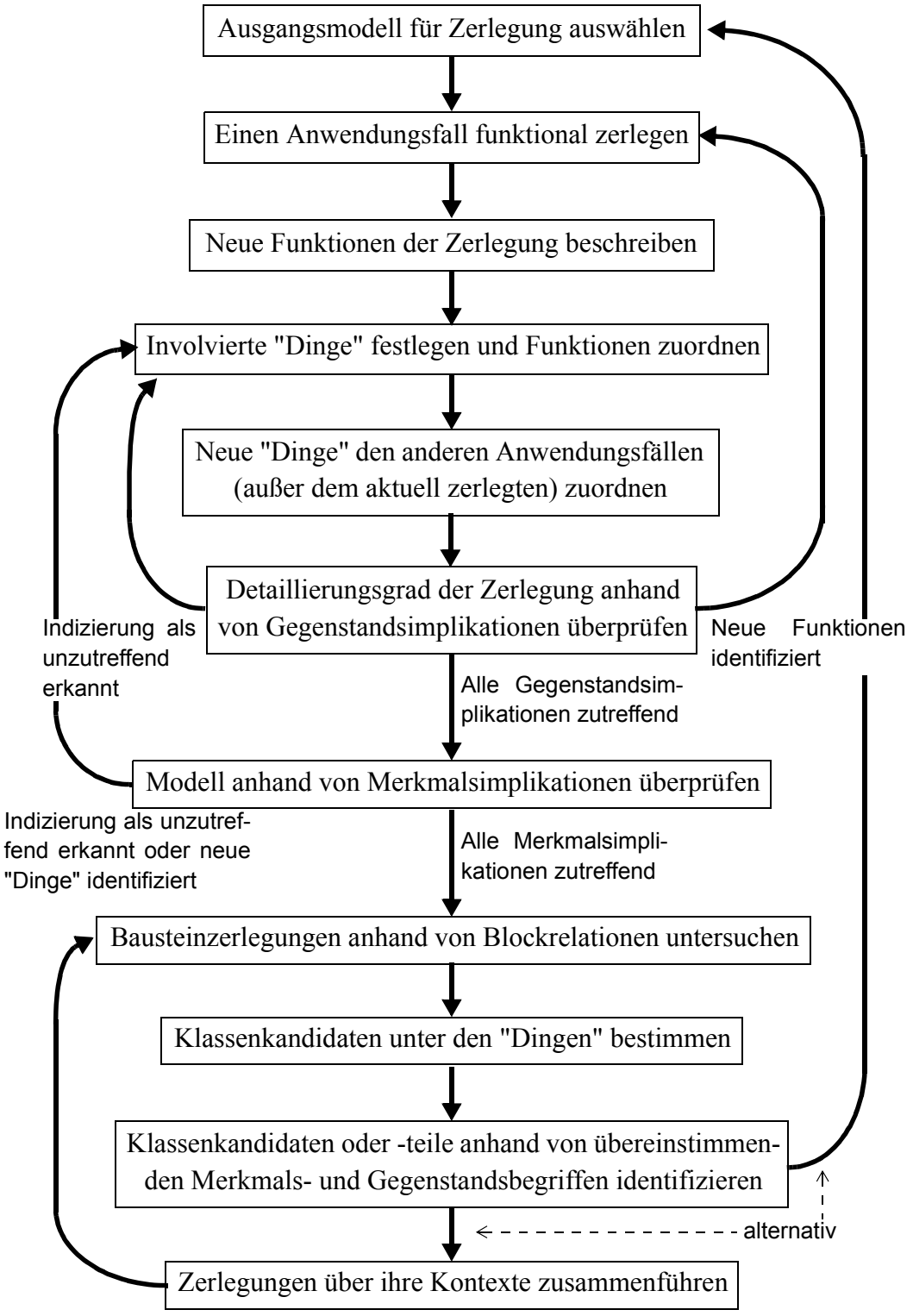


Abb. 4.40 Feinanalyse: Funktionale Zerlegung der Anwendungsfälle

Die Modellierung des Klassenmodells selber und die Erstellung von Sequenzdiagrammen wie in Abbildung 2.7 auf Seite 127 sind als Analyseziel vorgegeben, unter den angegebenen Schritte aber nicht explizit angesprochen. Der Grund dafür ist, daß in Abbildung 4.39 und Abbildung 4.40 kein Zeitpunkt festgemacht werden kann, zu dem sicher wäre, daß alle nötigen Klassen identifiziert sind. Es sollte einfach so sein, daß gefundene Klassenkandidaten in das Klassenmodell aufgenommen werden. Die Beziehungen zwischen den Klassen werden aber weitgehend nicht mit Mitteln von BASE festgelegt. Die untersuchten Liniendiagramme geben nur Hinweise auf Benutzungsbeziehungen, nicht aber ob diese sinnvoller Weise als Spezialisierung, Assoziation, Aggregation zwischen Klassen oder Botschaftsbeziehung zwischen Objekten modelliert werden sollten. Sequenzdiagramme können aus den festgelegten Zerlegungen der Anwendungsfälle entwickelt werden, wenn die enthaltenen Funktionen im Klassenmodell schon Klassen zugeordnet wurden. Es muß dann nur noch die Zuordnung der Funktionsaufrufe zu entsprechenden Objekten vorgenommen werden.

5 Ein Analyse-Werkzeug für BASE

Der Erfolg eines Ansatzes wie BASE hängt auch wesentlich von der gebotenen Unterstützung durch Software-Werkzeuge ab. In BASE erfordert insbesondere die Darstellung der Liniendiagramme eine solche Werkzeugunterstützung. Im Rahmen der vorliegenden Dissertation wurde ein solches Werkzeug entwickelt. Hier werden die Grundzüge seiner Benutzung und seines Aufbaus dargestellt, ohne ein vollständiges Benutzerhandbuch oder eine komplette technische Dokumentation zu liefern. Beispielhaft soll gezeigt werden, wie der Ansatz durch entsprechende Software unterstützt werden kann.

Wie schon in 1.6.3 auf Seite 18 und in Kapitel 4 auf Seite 126 erwähnt, soll ein Benutzer von BASE von den mathematischen Details im Hintergrund verschont bleiben. Lediglich Liniendiagramme sollte er interpretieren können, ohne deren exakte mathematische Semantik kennen zu müssen. Um klarzustellen, daß dies möglich und mit dem vorliegenden Prototypen erreicht ist, werden in 5.1 zunächst die grundlegenden Elemente der Benutzerschnittstelle und in 5.2 der Umgang mit dem Werkzeug erklärt. Abschnitt 5.3 skizziert zwei Szenarien zum Test des Ansatzes. Erst in 5.4 ist es dann wieder nötig, zur Darstellung der benutzten Datenstrukturen auf die mathematischen Grundlagen Bezug zu nehmen. Im wesentlichen fußt das vorgestellte Werkzeug auf der Klassenbibliothek *The Formal Concept Analysis Library* von Frank Vogt ([Vog 96]).

Einige Vorarbeiten zur Entwicklung des Werkzeugs wurden innerhalb von Praktika am Fachbereich Mathematik und Informatik der Philipps-Universität Marburg von Studenten geleistet. Inna Schwab, Christian Näcker und Jörn Schimmelpfeng entwickelten ein erstes Programm zur Bearbeitung von formalen Kontexten und Liniendiagrammen unter Borland C++. Nguyen Huu An, Serdal Kaya, Achim Sellmann und Ralf Wiehl portierten anschließend *The Formal Concept Analysis Library* und den entwickelten Liniendiagramm-Editor auf MS Visual C++. Jan Malcomes, Ralf Wiehl und Thorsten Züchner erstellten daraus einen ersten Prototyp für BASE. Dieser Dame und diesen Herren gilt deshalb mein besonderer Dank. Alle

erwähnten Praktika erbrachten wertvolle Beiträge für die Entwicklung des hier dargestellten Werkzeugs. Am Ende wurde jedoch eine grundlegende Revision und weitgehende Neuimplementierung unternommen.

5.1 Benutzeroberfläche

Die wesentlichen Elemente der Benutzerschnittstelle betreffen die Darstellung und Manipulation

- der Daten zu den Anwendungsfällen, inklusive deren funktionaler Zerlegung
- der erstellten Liniendiagramme, inklusive der Darstellung von Komponentenzerlegungen
- der generierten Überprüfungsfragen.

In dieser Reihenfolge sind sie im folgenden dargestellt.

5.1.1 Darstellung von Anwendungsfalldaten

Zur Darstellung, Eingabe und Veränderung der Daten zu den Anwendungsfällen dient der folgende Dialog:

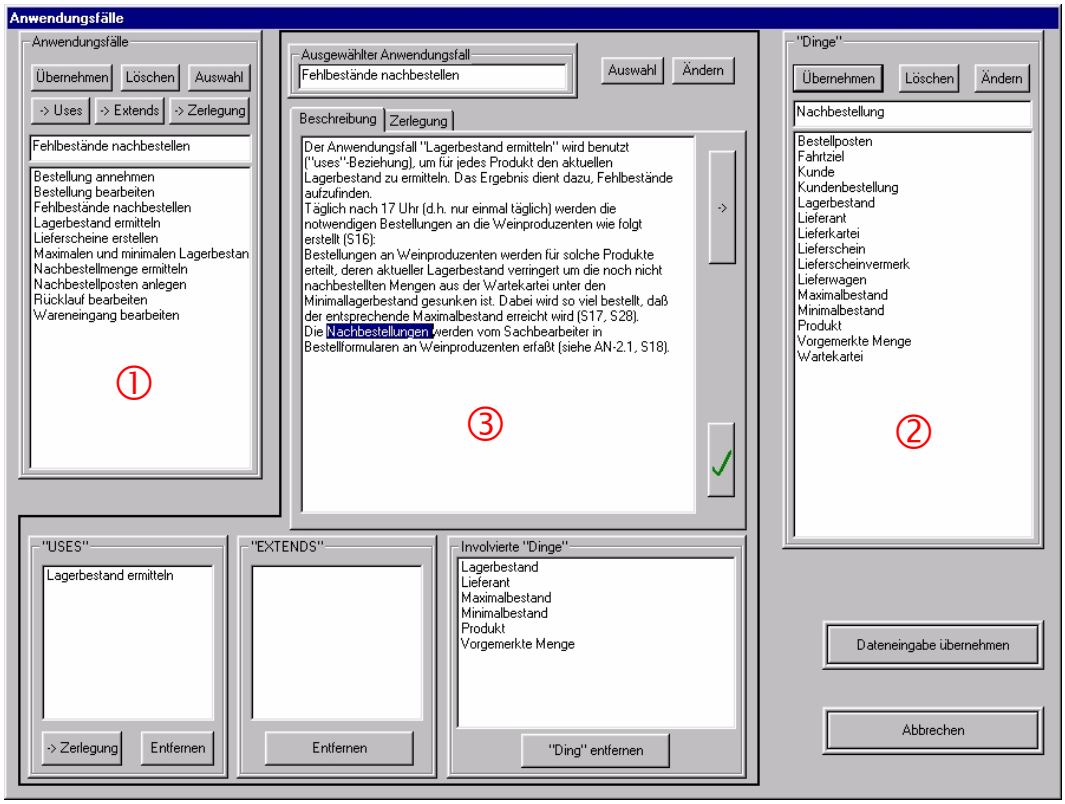


Abb. 5.1 Anwendungsfall-Editor in "Beschreibungsansicht"

Im linken mit einer ① gekennzeichneten Bereich sind die Anwendungsfälle aufgelistet. Zu der entsprechenden Listbox kommt ein Editierfeld für ihre Eingabe und Buttons, die verschiedene Aktionen auslösen. In analoger Weise sind auf der rechten Seite (siehe ②) alle betrachteten "Dinge" aufgeführt. In der Mitte – durch einen schwarzen Rahmen zusammengefaßt und in Abbildung 5.1 zusätzlich durch eine ③ gekennzeichnet – sind alle Angaben zu dem aktuell bearbeiteten Anwendungsfall. Diese sind dargestellt in

- einem Editierfeld für den Namen (oben),
- einem Eingabefeld für die Beschreibung des Anwendungsfalls (darunter),
- einer Listbox für die benutzten Anwendungsfälle ("*uses*", links unten),
- einer Listbox für die allgemeineren Anwendungsfälle, die spezialisiert werden ("*extends*", halb links unten),
- einer Listbox für die involvierten Dinge (unten Mitte)

Alternativ zu der Beschreibung kann auch die funktionale Zerlegung eingeblendet werden. Alle anderen Angaben bleiben ungeändert:

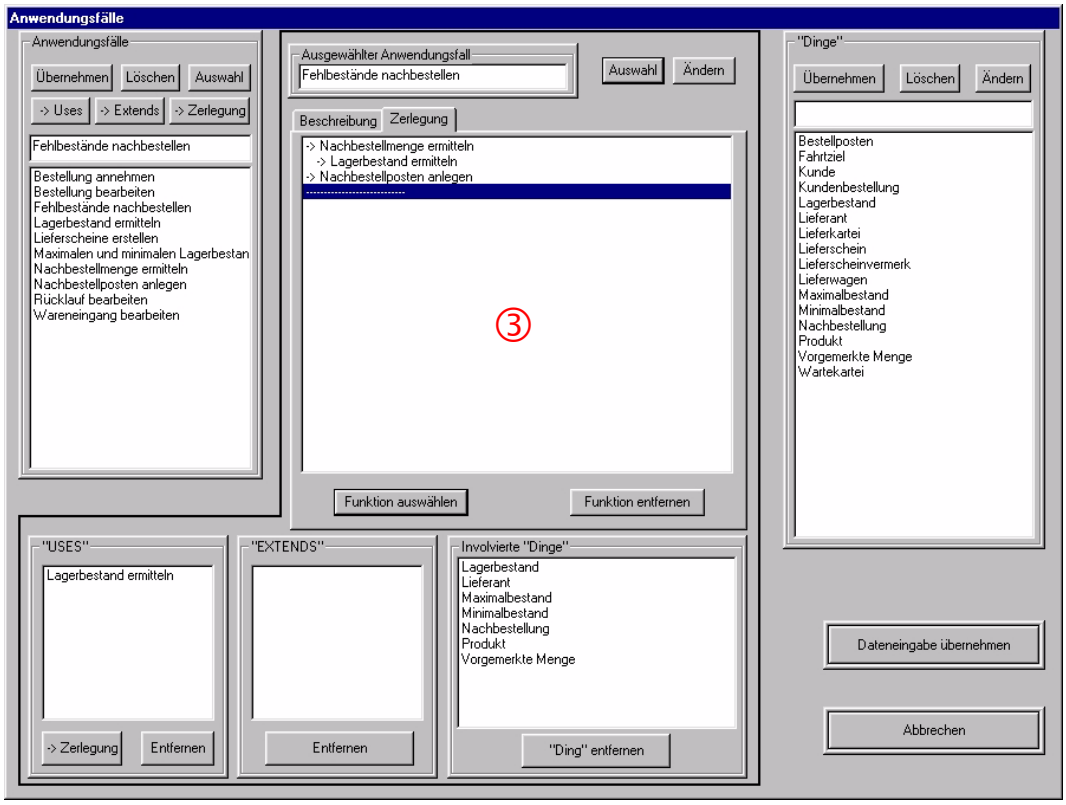


Abb. 5.2 Anwendungsfall-Editor in "Zerlegungsansicht"

Die Daten zu den Anwendungsfällen werden mit diesem Dialog in der folgenden Weise eingegeben:
Zuerst müssen links die Namen von Anwendungsfällen eingetippt und in die Liste

übernommen werden.

Wird dann einer der Anwendungsfälle aus der Liste zur Bearbeitung ausgewählt, ist es danach auch möglich, rechts "Dinge" einzugeben. Neu eingegebene "Dinge" werden immer der Liste aller "Dinge" zugeschlagen und dem aktuell in der Mitte bearbeiteten Anwendungsfall als involviertes "Ding" zugeordnet. Schon vorher eingegebene "Dinge" können ihm aber auch aus der rechten Listbox heraus zugeteilt werden. Eine weitere Erleichterung der Eingabe ist in Abbildung 5.1 angedeutet. Normalerweise wird der Benutzer den zur Bearbeitung ausgewählten Anwendungsfall erst beschreiben, um ihm dann später die involvierten "Dinge" zuzuweisen. Die Beschreibung kann er in der Dialogmitte eingeben. Aus dem eingegebenen Text kann er dann Teile selektieren und mit dem "->"-Button in das Editierfeld der "Dinge" rechts übertragen. Dort kann er mit dem "Übernehmen"-Button den Inhalt des Editierfelds als ein "Ding" in die Liste übernehmen und dem Anwendungsfall zuordnen. Der Zwischenschritt über das Editierfeld wurde eingefügt, um den als neues "Ding" selektierten Text noch verändern zu können. Dies ist etwa nötig, um Bezeichner in einheitlicher Flexion zu verwenden.

Die "*uses*"- und "*extends*"-Liste können dadurch gefüllt werden, daß aus der linken Liste ein entsprechender Anwendungsfall ausgewählt und durch den "->*uses*"- bzw. "->*extends*"-Button in die entsprechende Liste übertragen und damit dem aktuell bearbeiteten Anwendungsfall zugeordnet wird. Zirkuläre "*uses*"- und "*extends*"-Beziehungen werden jedoch beim Versuch ihrer Eingabe abgelehnt. Im Falle der "*extends*"-Beziehungen machen Zirkel offensichtlich keinen Sinn, denn spezialisieren sich zwei Anwendungsfälle gegenseitig, sind sie einfach gleich. Aber auch für "*uses*"-Beziehungen werden hier keine zirkulären Beziehungen erlaubt. In BASE und in der Benutzungshierarchie der Anwendungsfälle wären alle an einem Zirkel beteiligten Anwendungsfälle ununterscheidbar. Dies würde eine klare Modularisierung verhindern. Als Konsequenz dieser Einschränkung ist Rekursion nicht modellierbar. Für direkte rekursive Benutzung einer Funktion durch sich selbst greift diese Einschränkung nicht wirklich, weil die Indizierung mit involvierten "Dingen" mit oder ohne Berücksichtigung der "*uses*"-Beziehung zu sich selbst die gleiche bleibt. Ein Unterschied tritt erst bei verschränkter Rekursion auf, wenn sich die Rekursion über mehrere Ebenen erstreckt und so ein echter Benutzungszirkel mit mehr als einem Element entsteht. Solche Konstruktionen sind aber zunächst – wie oben angeführt – zu vermeiden. Sie werden erst bei der Implementierung der Klassenoperationen möglicherweise wichtig.

In der "Zerlegungsansicht" wie in Abbildung 5.2 ist in der Mitte die funktionale Zerlegung des aktuell bearbeiteten Anwendungsfalls in einfacher Weise wiedergegeben. Seine Teilschritte sind in der Reihenfolge ihrer Abfolge innerhalb des Anwendungsfalls aufgelistet. Sie können auch mehrfach vorkommen. Zu jedem wei-

ter zerlegten Schritt wird dessen Zerlegung eingerückt auch eingefügt. Die Liste der Schritte (auch eine leere) enthält immer eine spezielle Abschlußzeile. Zu jeder Zeit ist diese Zeile oder einer der Zerlegungsschritte auf oberster Ebene selektiert. Um Anwendungsfälle in die Zerlegung einzufügen, muß man sie nur links auswählen. Mit dem "-> Zerlegung"-Button werden sie dann vor der aktuellen Auswahl in die Zerlegungsliste mitsamt eventueller eigener Zerlegung eingefügt.[1] Ebenfalls können Zerlegungsschritte aus der "*uses*"-Liste gewählt werden. Das Einfügen geschieht immer auf der obersten Ebene der Zerlegung. Will man einen Schritt aus der Zerlegungsliste weiter zerlegen, muß man ihn durch den "Funktion auswählen"-Button vorher erst zur Bearbeitung auswählen.

Wird der Dialog durch den Button "Dateneingabe übernehmen" beendet, so wird für jeden Anwendungsfall nachgeprüft, ob alle von ihm im Sinne einer "*uses*"-Beziehung benutzten Anwendungsfälle auch (transitiv) in seiner Zerlegung vorkommen, wenn seine Zerlegung schon angegeben ist. Ist das nicht der Fall, werden dem Benutzer sukzessive die Paare von benutzendem und benutzten Anwendungsfall angezeigt, die die obige Regel verletzen. Er kann dann entscheiden, ob er die Eingabe korrigieren will. In diesem Fall wird der benutzende Anwendungsfall im Dialog angezeigt. Lehnt er die Korrektur der beanstandeten Fälle ab, kann er die Daten auch unverändert übernehmen.

Im Dialog werden nicht zulässige Buttons deaktiviert und grau angezeigt, um den Benutzer zu leiten. (In obigen Abbildungen sind nur zur besseren Lesbarkeit alle Beschriftungen schwarz gezeichnet.) Weiter können viele Aktionen auch durch die Return-Taste oder Doppelklick in einer Listbox ausgelöst werden, um die Arbeit zu beschleunigen. Zur optischen Unterstützung dieser Funktionalität werden die entsprechenden Buttons mit fetter Umrandung gezeichnet.

5.1.2 Darstellung von Liniendiagrammen

Die Darstellung der Liniendiagramme ist an das Programm *Anaconda* von Frank Vogt ([Vog 99]) aus der Darmstädter Arbeitsgruppe um Rudolf Wille angelehnt, mit dem auch die Liniendiagramme in Kapitel 4 gezeichnet wurden. Die Diagramme werden im Hauptfenster angezeigt (siehe Abbildung 5.3).

Die Daten der Anwendungsfälle – insbesondere die Indizierung mit den involvierten "Dingen" – kann nur über den Anwendungsfall-Editor verändert werden. Im Hauptfenster kann der Benutzer nur die Anordnung des Diagramms verändern, indem er einen Knoten anklickt und mit gehaltener Maustaste verschiebt. In der Menüleiste kann er zusätzlich anwählen, ob er alle unter oder über diesem Knoten lie-

1. An diesem Punkt ist die Vermeidung von Zyklen auch von technischer Bedeutung.

genden (und über Linienzüge mit ihm verbundenen) Knoten gleichzeitig mit verschieben will. Vertikal setzt das Werkzeug der Verschiebung aber Grenzen, so daß die relative vertikale Position eines Knotens im Verhältnis zu seinen (verbundenen) oberen und unteren Nachbarn nicht verändert wird. Beschriftungen können einzeln verschoben werden. Als zusätzliche Erleichterung beim Zeichnen kann der Benutzer die Beschriftung durch die Anwendungsfälle oder durch die "Dinge" ein- und ausblenden, die Diagrammgröße durch Eingabe eines Prozentwerts verändern (Die Schriftgröße bleibt dabei unverändert.), in gleicher Weise nur den Durchmesser der Knoten variieren, und für Anwendungsfälle und "Dinge" getrennt die Schriftart festlegen. Standardmäßig werden die "Dinge" durch Fettdruck von den Anwendungsfällen abgehoben.

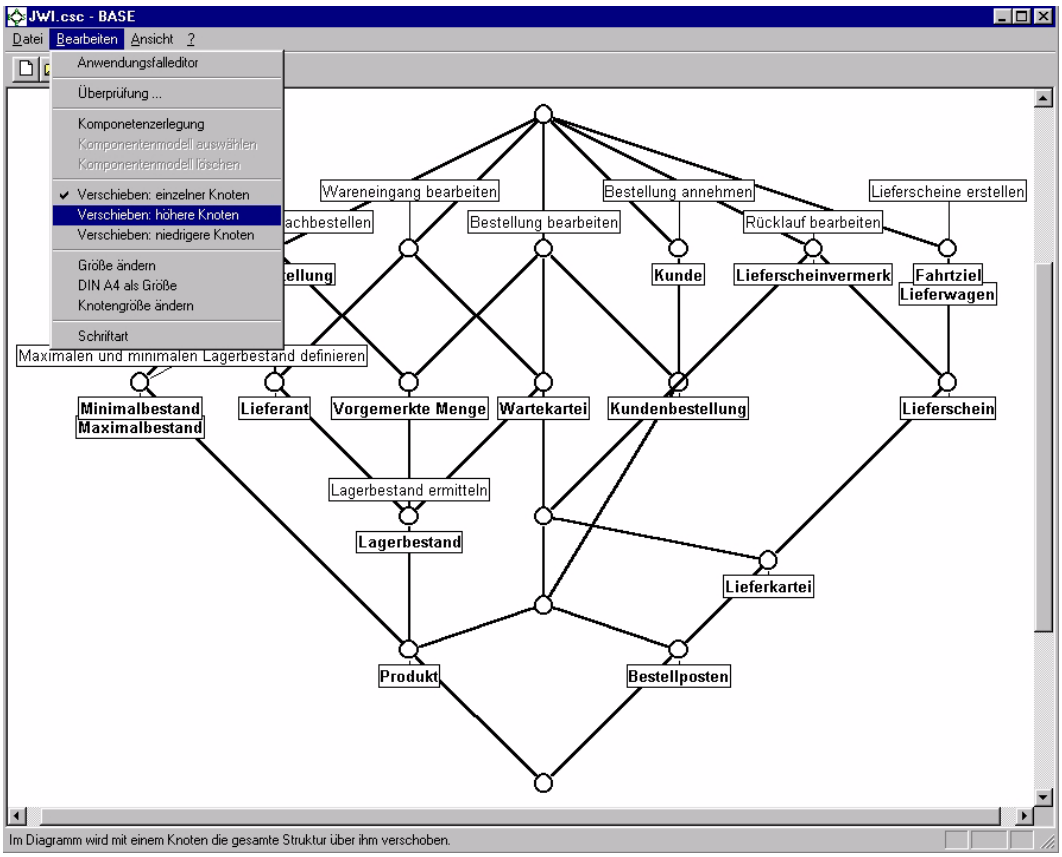


Abb. 5.3 Liniendiagramm im Hauptfenster des Werkzeugs

Es kann passieren, daß Kanten durch Knoten hindurch gezeichnet werden, zu denen sie nicht inzident sind. Dies ist dadurch zu erkennen, daß die Kante in der Darstellung den Knoten schneidet (siehe Abbildung 5.3). Inzidente Kanten sind nur bis zum Knotenrand durchgezogen und immer auf den Knotenmittelpunkt gerichtet. So können auch Kanten, die nur den Knotenrand berühren als nicht inzident erkannt werden. Durch das Verschieben von Knoten sollten diese unerwünschten

Überschneidungen beseitigt werden.

Für einen Ausdruck des Diagramms kann dieses auf DIN A4-Größe skaliert werden. Dabei bleibt die Schriftgröße unverändert und nur das Diagramm wird in seiner Größe angepaßt. Je nachdem, welches Format eine größere Darstellung ermöglicht, wird Hoch- oder Querformat gewählt. So können auch für "Offline"-Diskussionen Unterlagen erstellt werden. (In einer Produktversion des Werkzeugs sollte auch der Druck auf mehrere Seiten mit der Definition entsprechender Kleberänder ermöglicht werden.)

Die Änderungen der Diagrammgröße und Beschriftung werden gespeichert und betreffen nicht nur die aktuelle Diagrammdarstellung auf dem Bildschirm. Zu einem späteren Zeitpunkt wird das Diagramm also wieder in der zuletzt eingestellten Größe angezeigt.

Darstellung von Komponentenzerlegungen

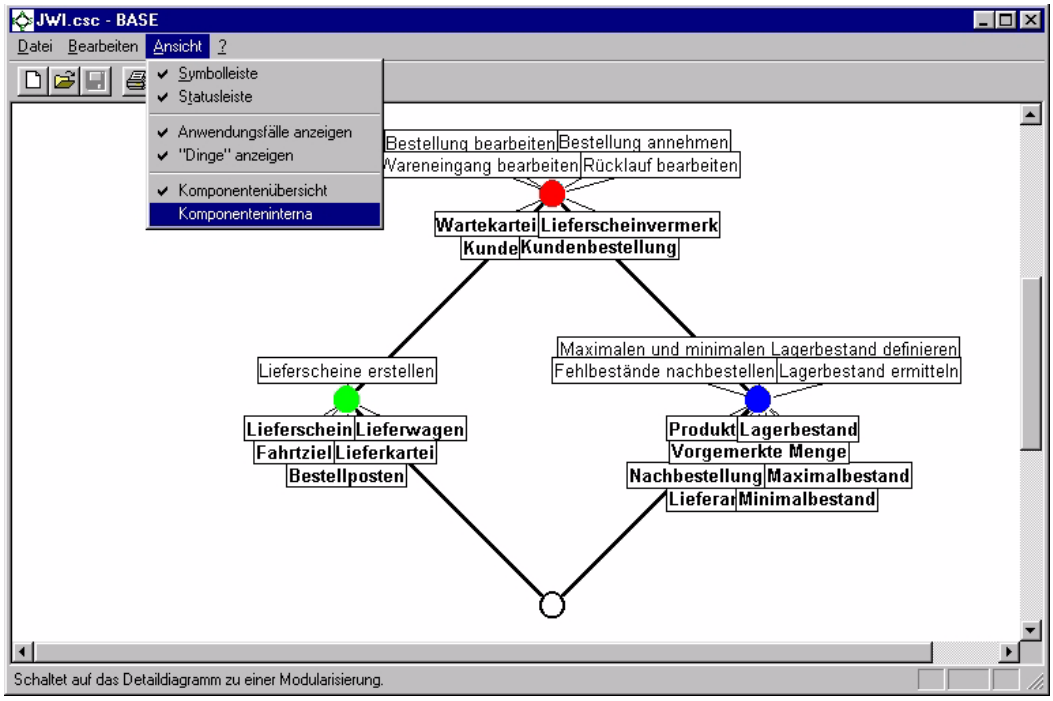


Abb. 5.4 Übersichtsdiagramm zu einer Komponentenzerlegung

Vorschläge zur Modularisierung werden wie in 4.6 auch in Liniendiagrammen dargestellt. Es werden jeweils ein Übersichtsdiagramm und das zugrundeliegende Liniendiagramm mit einer Kennzeichnung der Komponenten angeboten. Diese Kennzeichnung ist dadurch gegeben, daß jeder Komponente, die mehrere Knoten des originalen Diagramms umfaßt, eine andere Farbe zugeordnet wird. Im Übersichtsdiagramm wird der einer solche Komponente zugehörige Begriffsknoten in

dieser Farbe (ausgefüllt) dargestellt, in der feineren Sicht alle Knoten der Komponente und die innerhalb liegenden Kanten. Die Darstellung von Knoten, die direkt aus dem ursprünglichen Diagramm übernommen sind, ohne mit anderen zusammengefaßt zu werden, bleibt unverändert: ein unausgefüllter Kreis mit schwarzem Rand. So stechen die Veränderungen klar hervor.

Anhand der Farben können die einzelnen Komponenten im feineren Diagramm sofort wiedererkannt werden.

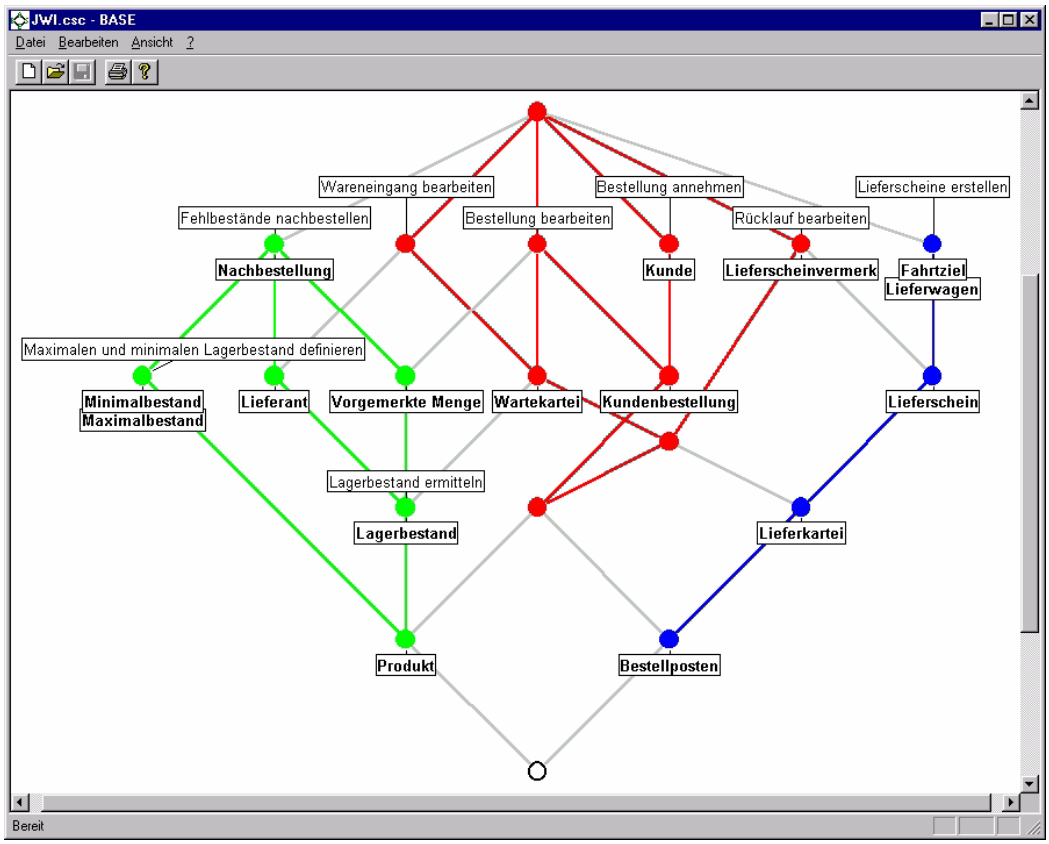


Abb. 5.5 Detaildiagramm zu einer Komponentenzerlegung

Über Kontextmenüs kann der Benutzer weitere Funktionen aufrufen, die ihm bei der Orientierung helfen sollen. In der Darstellung einer Komponentenzerlegung kann er die Darstellungsfarbe von farblich hervorgehobenen Komponenten bestimmen, um sie besser voneinander anzuheben. Ausgehend von einem Knoten oder einem "Ding" eines beliebigen Diagramms (außer Detaildiagrammen zu Modularisierungsvorschlägen) kann er die Darstellung auf die darüberliegenden Knoten reduzieren. "Dinge", die keinem dieser Knoten zukommen, werden dann wie in 4.7 grau zusätzlich eingezeichnet (siehe Abbildung 5.6). Genauso ist es möglich, von einem Knoten oder einem Anwendungsfall aus alle nicht darunter liegenden Knoten auszublenden. In diesem Fall werden Anwendungsfälle, die keinem dieser

Knoten zukommen, zusätzlich grau angezeigt.

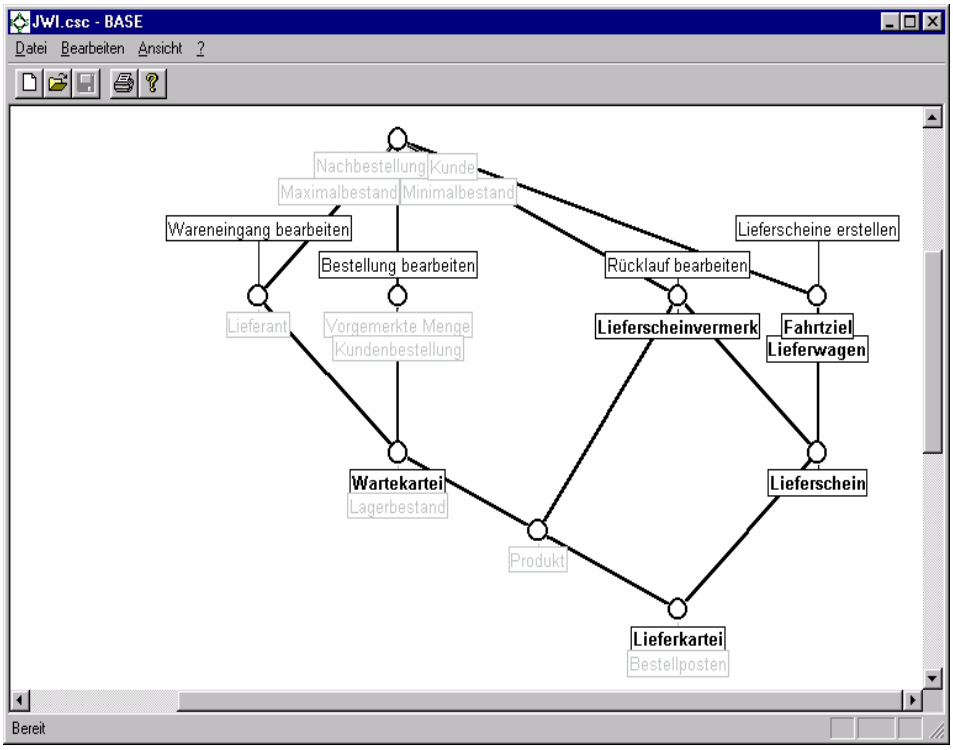


Abb. 5.6 Ausgehend von *Lieferkartei* alle höheren Knoten

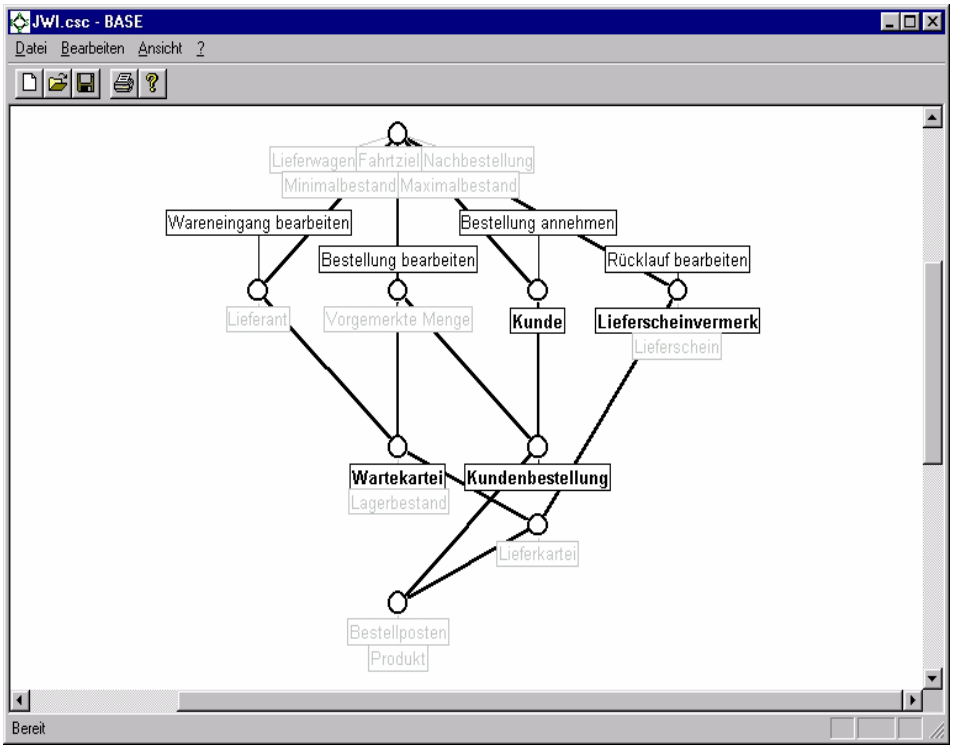


Abb. 5.7 Die obere Komponente aus Abbildung 5.5

Im Fall einer Übersichtsdarstellung zu einer Komponentenzerlegung kann auch nur eine einzelne Komponente angezeigt werden. In diesem Fall werden Komponenten-externe "Dinge" von unter einem Komponentenknoten liegenden Knoten und Komponenten-externe Anwendungsfälle von höheren Knoten grau in die Darstellung integriert. (In Abbildung 5.7 sieht man nur das anhand der "Dinge".)

5.1.3 Darstellung von Überprüfungsfragen

Überprüfungsfragen werden nacheinander dem Benutzer in einem nicht-modalen Dialog präsentiert.

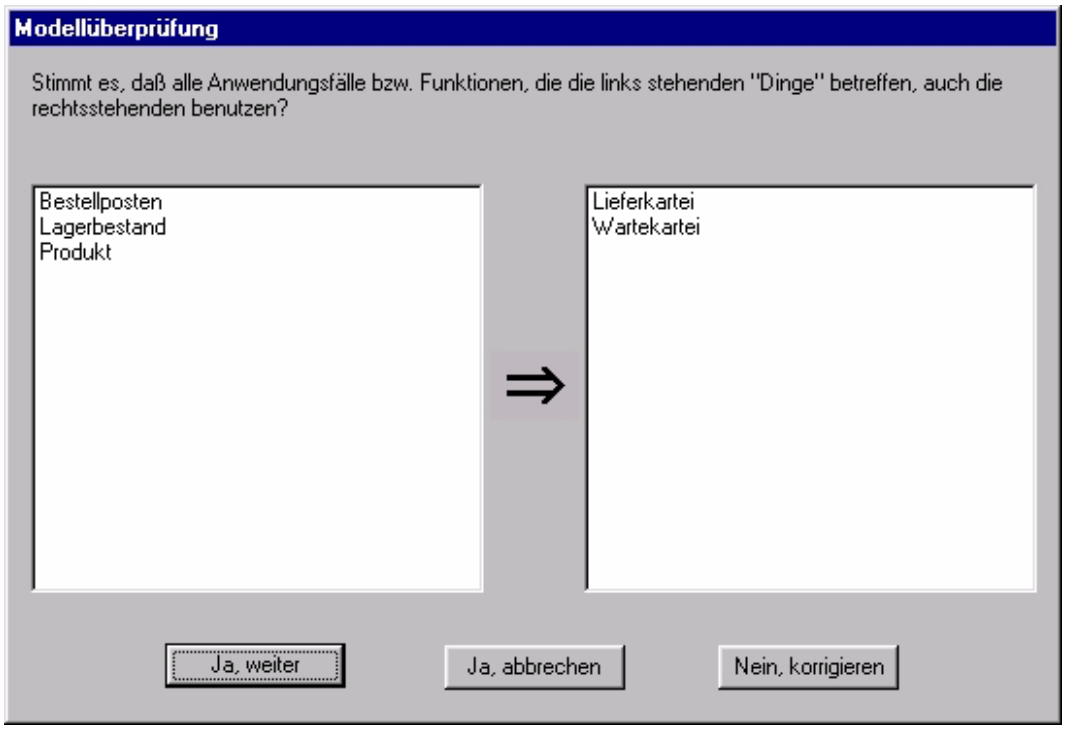


Abb. 5.8 Eine Überprüfungsfrage

Der Benutzer kann

- die Frage bejahen und bekommt dann die nächste vorgelegt
- sie bejahen und die Überprüfung abbrechen
- sie verneinen, wodurch er sofort in den Anwendungsfall-Editor gelangt.

Für den letzten Fall ist der Dialog modal angelegt, damit der Benutzer erinnert wird, was ihn zu einer Indizierungskorrektur oder Modellergänzung angeregt hat.

Bei der Anzeige der generierten Fragen wird nach deren Anzahl automatisch entschieden, ob alle nach dem Modell aus den links aufgeführten Elementen folgenden Elemente auf einmal oder einzeln abgefragt werden. Im zweiten Fall werden Überprüfungsfragen mit mehr als einelementiger rechter Seite in mehrere Fragen

aufgeteilt. Leicht in das Werkzeug zu integrieren wäre eine Möglichkeit für den Benutzer, festzulegen, welche Fragenrepräsentation er bevorzugt, oder bis zu welcher Fragenanzahl er die Fragen aufteilen lassen möchte. Im vorliegenden Werkzeug ist diese Anzahl fest auf 10 (zusammengesetzte Fragen) eingestellt.

5.2 Arbeitsweise mit dem Werkzeug

Zusätzlich zu den Anwendungsfalldaten und den Liniendiagrammen wird ein Projektname gespeichert. Legt der Benutzer ein neues Projekt an, wird er zunächst nach diesem gefragt. Danach erscheint der Anwendungsfall-Dialog wie in Abbildung 5.1 auf Seite 190 zur Eingabe der Daten zu den Anwendungsfällen. Nach Abschluß der Eingabe muß der Benutzer einen Bezeichner für das eingegebene Modell festlegen. Zu einem Projekt sind in der Regel mehrere Liniendiagramme (grobe Sicht ↔ funktionale Verfeinerung verschiedener Anwendungsfälle) interessant. Die entsprechenden Modelle werden durch die zugeordneten Bezeichner identifiziert. Als Standard wird der Projektname vorgeschlagen. Werden später schon eingegebene Anwendungsfallmodelle modifiziert, so kann der Benutzer entscheiden, ob er ein neues Diagramm und damit auch Modell anlegt oder das alte überschreibt.

5.2.1 Erste automatisch berechnete Liniendiagramme

Aus den eingegebenen Daten berechnet das Werkzeug ein erstes Liniendiagramm. In Abbildung 5.9 ist das gleiche Diagramm wie in Abbildung 5.3 auf Seite 194 zu sehen, wenngleich die Anordnung der Knoten viel unvorteilhafter erscheint. Dieses Diagramm wurde so vom Werkzeug automatisch erzeugt und danach nur noch insgesamt etwas verkleinert, um es der Fenstergröße anzupassen. Das automatische Zeichnen garantiert lediglich die Korrektheit der vertikalen Anordnung des Diagramms (siehe B.5.1 auf Seite 306). Ein "schönes" Diagramm ist nicht zu erwarten, denn es ist durch formale Regeln nicht definierbar, was ein gutes Diagramm ausmacht. Unter anderem spielen auch ästhetische Gesichtspunkte ein Rolle. Ein Ansatz besteht darin, die Kantenüberschneidungen zu minimieren. Aber es gibt Fälle, in denen Diagramme mit mehr Kantenüberschneidungen sinnvoller erscheinen, weil sie etwa mehr Symmetrie aufweisen oder wohlbekannte Strukturen enthalten. Deshalb ist es üblich, zuerst ein Diagramm automatisch zu erzeugen und es dann dem Benutzer zu überlassen, eine für ihn vorteilhafte Anordnung herzustellen (vergl. [Vog 96], 3.3, S. 57). Zum automatischen Zeichnen von Diagrammen siehe auch B.5 auf Seite 305.

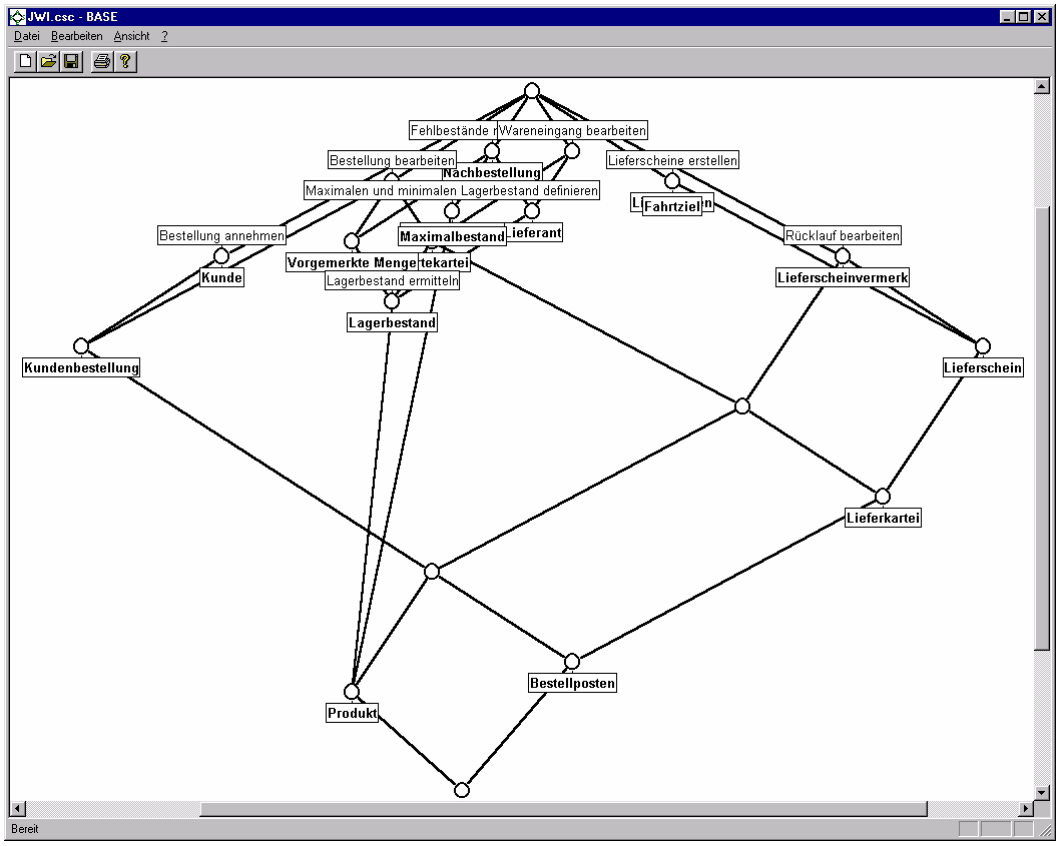


Abb. 5.9 Automatisch berechnetes Liniendiagramm

Indem man eine der Optionen "Verschieben von höheren Knoten" und "Verschieben von niedrigeren Knoten" anwählt, lassen sich große Diagrammteile schnell anordnen. Da das automatisch gezeichnete Diagramm durch das Werkzeug von oben her aufgebaut wird, erscheint es oft hilfreich, auch von oben her das Diagramm anzuordnen und dabei alle Knoten, die unter einem zum Verschieben ausgewählten Knoten hängen, mit zu verschieben. Eine Anordnung in Schichten erhöht die Übersichtlichkeit. Ist eine der Schichten sehr breit (enthält also viele Knoten) und bestehen starke Verzweigungen unterhalb dieser Knoten, sollte man auf das Verschieben einzelner Knoten zurückschalten, weil die niedrigeren Knoten bei jeder über ihnen vorgenommenen Verschiebung mit verschoben werden und so weit außerhalb des Diagramms landen können, wenn alle Verschiebungen in die gleiche Richtung führen. Ebenfalls manuell anzupassen ist die Anordnung der Beschriftung. Anwendungsfälle werden in BASE zentriert oberhalb des entsprechenden Diagrammknotens und "Dinge" unterhalb ihres Knotens gezeichnet. Beim ersten automatischen Zeichnen werden mehrere Beschriftungen zu einem Knoten vertikal etwas versetzt. Dies sieht man in Abbildung 5.9 am Knoten zu *Lieferscheine erstellen* (oben halb rechts) und insbesondere in Abbildung 5.12 auf Seite 203.

5.2.2 Diagramme zu veränderten Anwendungsfallmodellen

In der Ausrichtung der Diagramme besteht die größte Arbeit für den Benutzer. Deshalb wird beim ersten Zeichnen des Liniendiagramms zu einem funktional verfeinerten oder einem korrigierten Anwendungsfallmodell die Anordnung des originalen Diagramms berücksichtigt, damit die Arbeit zur Anordnung nicht noch einmal komplett von neuem geleistet werden muß. So muß der Benutzer nur die neuen Knoten neu ausrichten. Außerdem bleibt die grobe Struktur des Liniendiagramms erhalten. Was genau ein "neuer" bzw. "alter" Knoten ist und wie die Berücksichtigung der originalen Anordnung exakt geleistet wird, ist in Verbindung mit dem angewandten Algorithmus in Anhang B.5.3 dargestellt. Hier soll nur der Effekt an einem Beispiel gezeigt werden. Das erste in Kapitel 4 angegebene Beispieldiagramm (Abbildung 4.5 auf Seite 137) wird durch das Werkzeug wie folgt dargestellt:

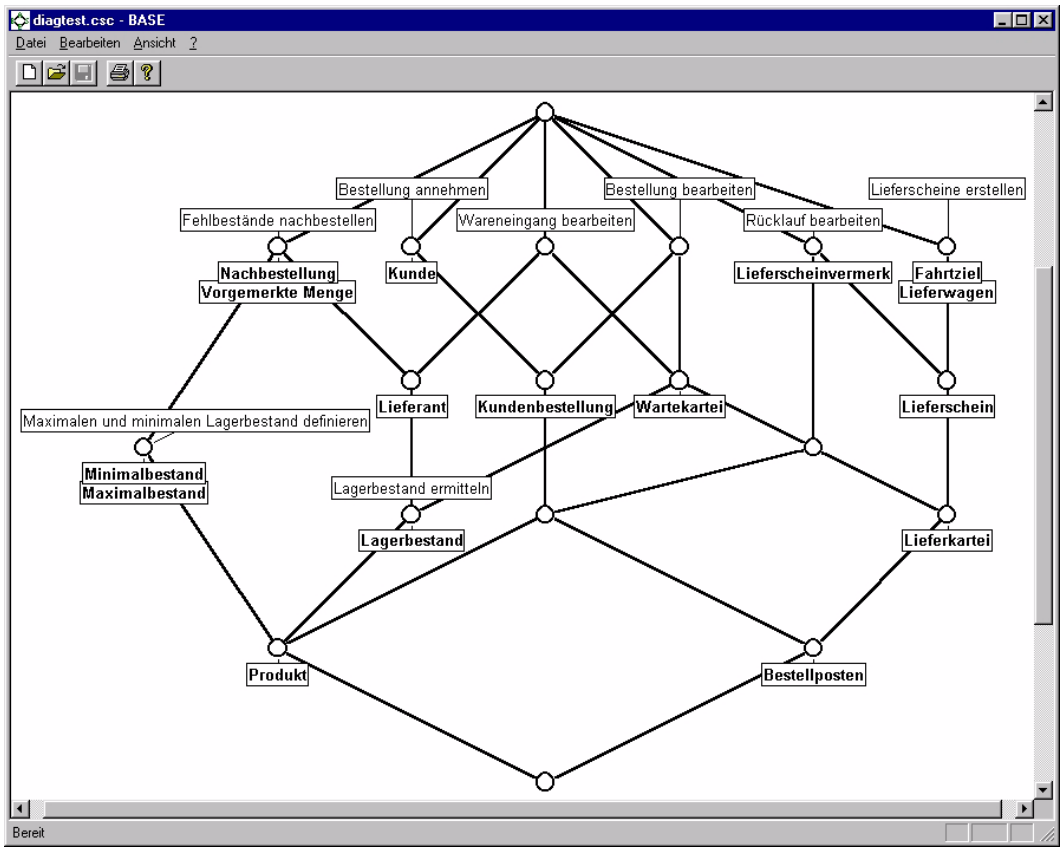


Abb. 5.10 Ausgangsdiagramm vor funktionaler Verfeinerung

Ausgehend von diesem Diagramm wurde die in 4.3.1 vorgeschlagene funktionale Zerlegung des Anwendungsfalls *Fehlbestände nachbestellen* (Abbildung 4.10 und Abbildung 4.11 auf Seite 147) mit dem Werkzeug vorgenommen. Nach der Eingabe zeichnet das Werkzeug automatisch das folgende Diagramm:

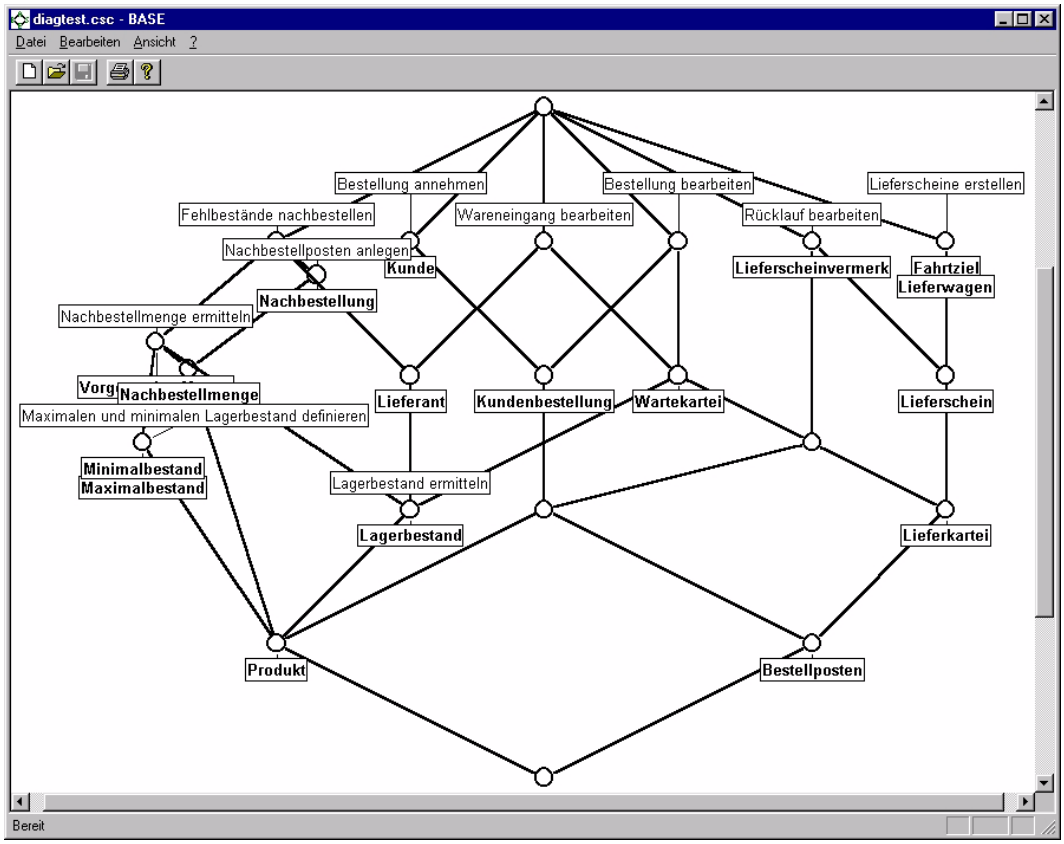


Abb. 5.11 Liniendiagramm nach funktionaler Verfeinerung

Die Struktur des Liniendiagramms wurde erhalten, nur die Positionen der Knoten zu den neuen Funktionen *Nachbestellmenge ermitteln* und *Nachbestellposten einfügen* und dem neuen "Ding" *Nachbestellmenge* müssen noch vom Benutzer korrigiert werden, um ein ansehnlicheres Diagramm zu erhalten. Wie in 5.2.1 diskutiert, ist lediglich die Korrektheit ihrer vertikalen Position garantiert.

5.2.3 Modularisierungsvorschläge

Bevor eine Komponentenzerlegung erzeugt wird, sollte das zugrunde liegende Diagramm ausgerichtet werden, weil die Anordnung der Diagramme zu einem Modularisierungsvorschlag aus der Originalanordnung berechnet wird. Für das Detaildiagramm der Komponentenzerlegung wird einfach die Originalanordnung übernommen. In der Übersichtsdarstellung befinden sich die Knoten im Mittelpunkt der Knotenmenge des Originaldiagramms, die zu der jeweiligen Komponente gehört.

Meist werden nicht alle Vorschläge für Komponentenzerlegungen dem Benutzer hilfreich erscheinen. Deshalb wird er bei der ersten Anzeige eines entsprechenden

Diagramms gefragt, ob er das Diagramm mit dem Projekt speichern will. Dies ist im besonderen der Fall, wenn er die Komponentenzerlegungen hat neu berechnen lassen.

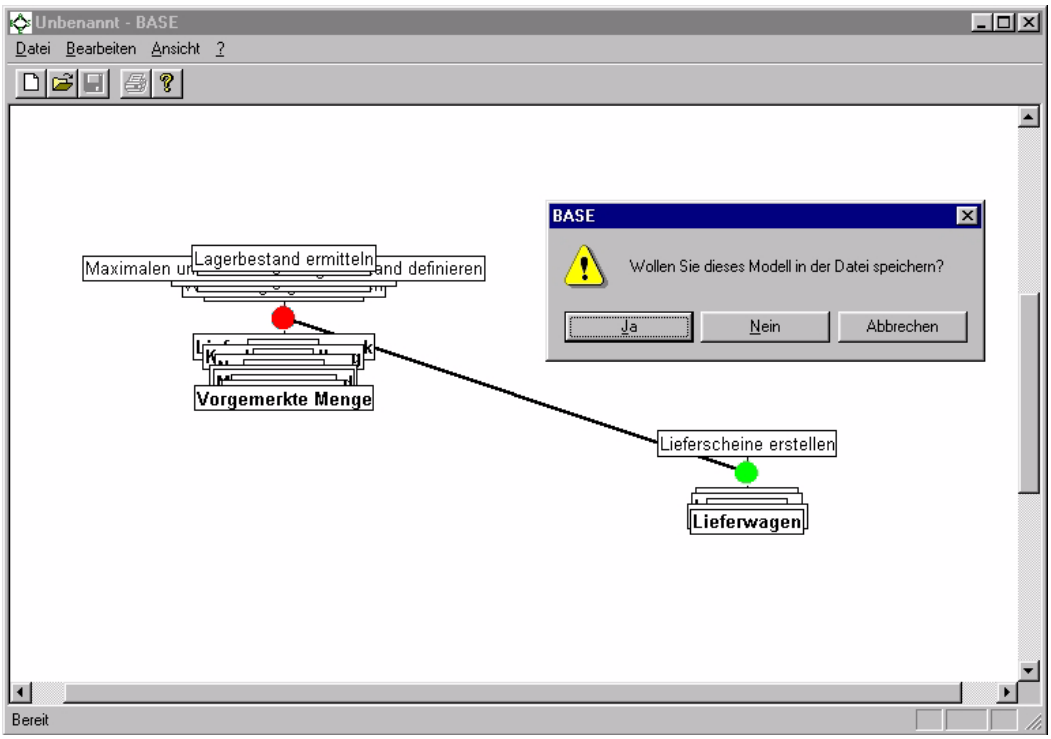


Abb. 5.12 Ein neuer Modularisierungsvorschlag

Übernimmt der Benutzer das Komponentenmodell, muß er ihm wieder einen Namen geben. Als Standard wird ihm der Name des Ausgangsmodells mit angehängtem "Ko" und einer Zahl, die die erzeugten alternativen Komponentenmodelle durchnumeriert, vorgeschlagen. Andernfalls wird das Modell bei "Nein" gelöscht und bei "Abbrechen" zwar gespeichert, aber als noch nicht bestätigt gekennzeichnet. Letzteres ermöglicht es also, die Entscheidung aufzuschieben.

5.2.4 Negativ beschiedene Überprüfungsfragen

Ist beabsichtigt, Überprüfungsfragen zu einem eingegebenen Anwendungsfallmodell zu bearbeiten, können diese sofort generiert werden, ohne vorher das Diagramm auszurichten, denn sie sind von der graphischen Repräsentation unabhängig.[1] Bei dem Auffinden einer aus fachlichen Gründen zu verneinenden Frage bleibt der anzeigende Dialog stehen, während der Anwendungsfall-Dialog geöffnet

1. Zur Generierung der Überprüfungsfragen siehe 4.3.2, 4.4.1 und Anhang B.3.

wird. Weil dieser sehr groß ist, liegen beide in der Regel übereinander. Um die Gedächtnisstütze durch die Überprüfungsfrage auszunutzen, reicht es aber, den Anwendungsfall-Dialog für einen Moment zu Seite zu schieben.

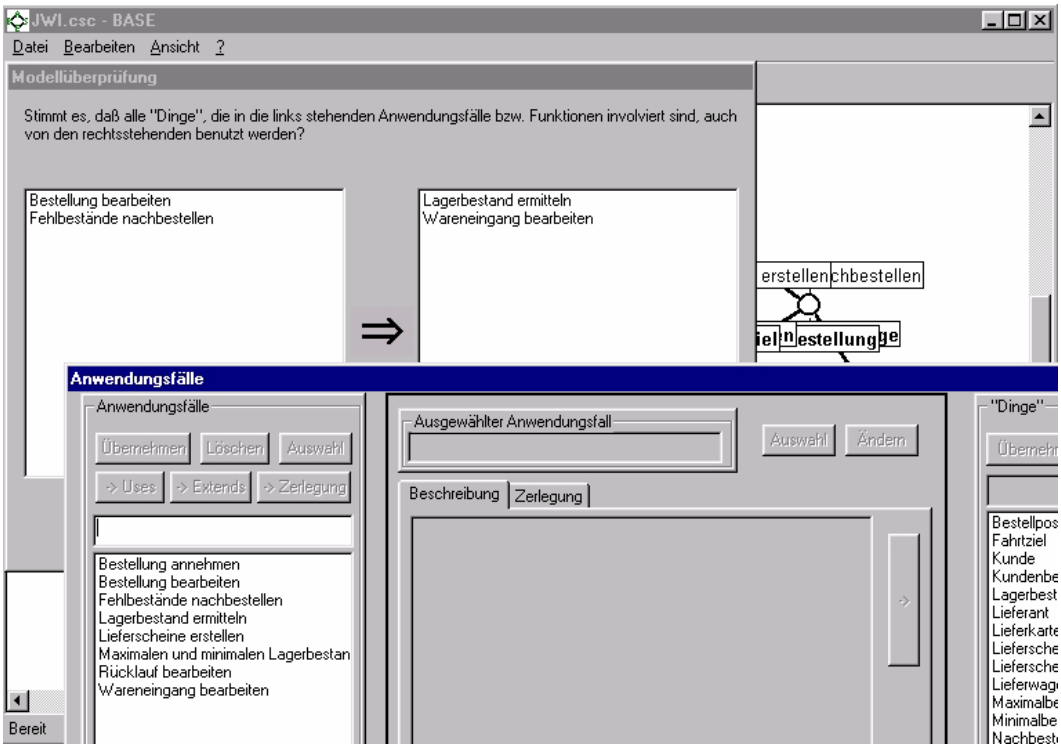


Abb. 5.13 Fragen- und Anwendungsfall-Dialog

5.3 Test des Ansatzes

Hier hätte eigentlich ein Abriß über die Ergebnisse eines ersten Einsatzes von BASE innerhalb eines Praktikums am Fachbereich Mathematik und Informatik der Philipps-Universität Marburg stehen sollen. Leider konnte der geplante Test wegen mangelnder Beteiligung von Studenten an der Lehrveranstaltung nicht stattfinden. Deshalb werden nur einige Grundsätze für einen solchen Test dargestellt.

Zunächst einmal zu den Kriterien zur Beurteilung eines solchen Tests:
Zu erwarten, daß mit Hilfe von BASE sofort und immer bessere Modelle entwikkelt würden als nach herkömmlichen Methoden, wäre sicher übertrieben. BASE verspricht jedoch, den Diskurs zwischen Fachexperten und Entwicklern wirkungsvoll zu unterstützen. Deshalb ist im wesentlichen zu untersuchen, ob durch BASE die Nachfragen von Entwicklern bei den Anwendern schneller und gezielter erfolgen und wie die Kommunikationssituation von beiden Personengruppen beurteilt wird.

In einem industriellen Projekt, in dem BASE versuchsweise von erfahrenen Entwicklern eingesetzt wird, könnte die Evaluation deshalb im wesentlichen durch Befragung der Beteiligten erfolgen. Man kann unterstellen, daß sie über hinreichende Erfahrung verfügen, die Methode im Vergleich zu den sonst von ihnen eingesetzten zu beurteilen. Interessant wäre ihre Einschätzung des Verhältnisses von Aufwand und Nutzen und der Grad ihrer Zufriedenheit in der Kommunikation miteinander.[1]

In der Universität stehen innerhalb von typischen Projekten mit Studentengruppen keine erfahrenen Entwickler für einen Test zur Verfügung. Aber es wäre möglich, zwei konkurrierende Gruppen die gleiche Modellierungsaufgabe behandeln zu lassen – eine nach einer der in 2.4 auf Seite 33 angeführten Methoden und eine nach BASE. Auftraggeber wären im Fall des geplanten Tests die Sekretärinnen des Fachgebiets Informatik gewesen, die sich ein System zur Verwaltung von Leistungs- und Prüfungsnachweisen der Studenten des Fachbereichs wünschten. Diese Konstellation ist von daher vorteilhaft, daß die Auftraggeber zwar in das Testvorhaben eingeweiht werden können, aber andererseits nicht selber an der Entwicklung von BASE teilhatten. So sind sie unvoreingenommen, können aber bei der Beurteilung der Gruppen mithelfen, indem sie über Anzahl und Art der Nachfragen der beiden Gruppen Buch führen. Gesichert müßte sein, daß die Auftraggeber den beiden Gruppen von sich aus die gleichen Informationen zukommen lassen. Erfahrungen mit konkurrierenden Studentengruppen zeigen, daß sie aufgrund von Konkurrenzverhalten wirklich eigenständig und unabhängig voneinander arbeiten. Einer Befragung wird in dieser Situation keine so große Bedeutung zukommen, weil die Studenten in der Regel Anfänger in der Modellierung von Informationssystemen sind und daher ihre Eindrücke schlecht einordnen können.

Entwicklungsergebnisse von Projekten mit ohne BASE mittels Metriken miteinander zu vergleichen, erscheint nicht sinnvoll, denn für die ganz frühe Analysephase, in der BASE zum Tragen kommt, gibt es noch keine verbreiteten Techniken und Erfahrungen dazu. Das liegt natürlich auch daran, daß zur Definition entsprechender Metriken erst einmal hinreichend formal definierte Modelle vorliegen müssen.

Als Entwicklungsgegenstand für einen Test sind Systeme geeignet, deren Funktionalität durch mehrere Anwendungsfälle beschrieben werden kann.

1. Realistisch beurteilt wird die Einarbeitung in den Umgang mit den Liniendiagrammen zunächst eine Hürde darstellen, die Techniken von BASE überhaupt anzuwenden. Diese zu überwinden, gelingt nur aus eigenem Antrieb. Andererseits erlaubt gerade die Arbeit mit dem Liniendiagramm-Editor ein spielerisches Umgehen mit den dargestellten Anwendungsfallmodellen. Deshalb kann der Umgang mit dem Werkzeug auch Spaß bereiten.

5.4 Technischer Aufbau des Werkzeugs

Das vorliegende Werkzeug – im folgenden wie die Methode "BASE" genannt – wurde unter *MS Visual C++ 4.2* mit Hilfe der *Microsoft Foundation Classes* (*MFC*, siehe [Krg 96]) und *The Formal Concept Analysis Library* (hier *FCA-L* abgekürzt, [Vog 96]) entwickelt. Dabei liegt eine Version von *The Formal Concept Analysis Library* in MS Visual C++ zugrunde, die aus der ursprünglich am Fachbereich Mathematik der Technischen Hochschule Darmstadt unter Borland C++ entwickelten Bibliothek am Fachbereich Mathematik und Informatik der Philipps-Universität Marburg durch Portierung auf MS Visual C++ entstanden ist. Diese Version benutzt auch die MFC. Für BASE wurde sie um die Behandlung von Implikationen und Blockrelationen sowie einigen weiteren kleineren Ergänzungen erweitert (siehe Anhang B).

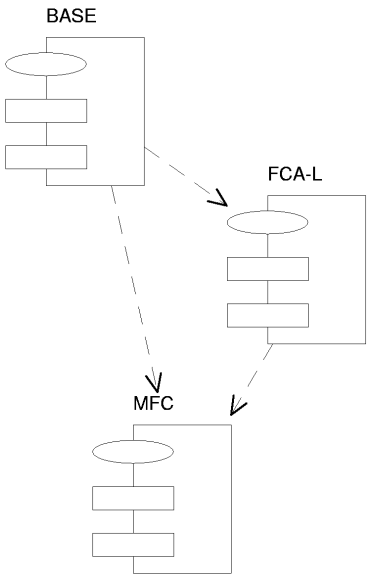


Abb. 5.14 Benutzte Bibliotheken

Die folgende Darstellung konzentriert sich auf die Datenstrukturen in BASE. Deren Erklärung ist hinreichend, um die Struktur der Anwendung darzustellen. Hinweise zu den angewandten Algorithmen finden sich in Anhang B.

Da das Werkzeug ein relativ kleines System ist, ist es nicht weiter in Komponenten zerlegt. Klassen bieten eine hinreichende Komponentenebene.

5.4.1 Die Klassen des Werkzeugs

BASE baut auf dem *SDI*-Framework (*Single Document Interface*) der MFC auf. Damit sind die wesentlichen Klassen von BASE:

- *CBASEApp* – Ihr einziges Objekt repräsentiert das Windows-Programm selber.
- *CMainFrame* – Ihr einziges Objekt repräsentiert das Hauptfenster der Anwendung.
- *CBASEDoc* – Verwaltet die Daten zu einem geladenen Projekt.
- *CBASEView* – Stellt das aktuell bearbeitete Liniendiagramm dar.

Das Klassendiagramm in Abbildung 5.15 zeigt die wesentlichen Beziehungen zwischen den Klassen. Die dunkelgrau hinterlegten Klassen sind innerhalb von BASE angelegt, die anderen in den MFC enthalten.

CBASEApp spezialisiert die Klasse *CWinApp*. Das von ihr zur Laufzeit erzeugte

Objekt kontrolliert die Objekte der anderen Klassen über die Klasse *CDocTemplate* – im Fall einer SDI-Anwendung wie BASE speziell über deren Unterklasse *CSingleDocTemplate*. Dazu wird bei Programmstart ein Objekt dieser Klasse angelegt und diesem mitgeteilt, daß die datenhaltende Klasse *CBASEDoc*, die darstellende Klasse *CBASEView* und die Klasse für das Hautfenster der Anwendung *CMainFrame* sind. Da BASE nach dem SDI-Framework entworfen ist, existiert genau ein Objekt von *CMainFrame* und jeweils nur ein Objekt von *CBASEDoc* zu dem gerade bearbeiteten Projekt bzw. der gerade geöffneten Datei.

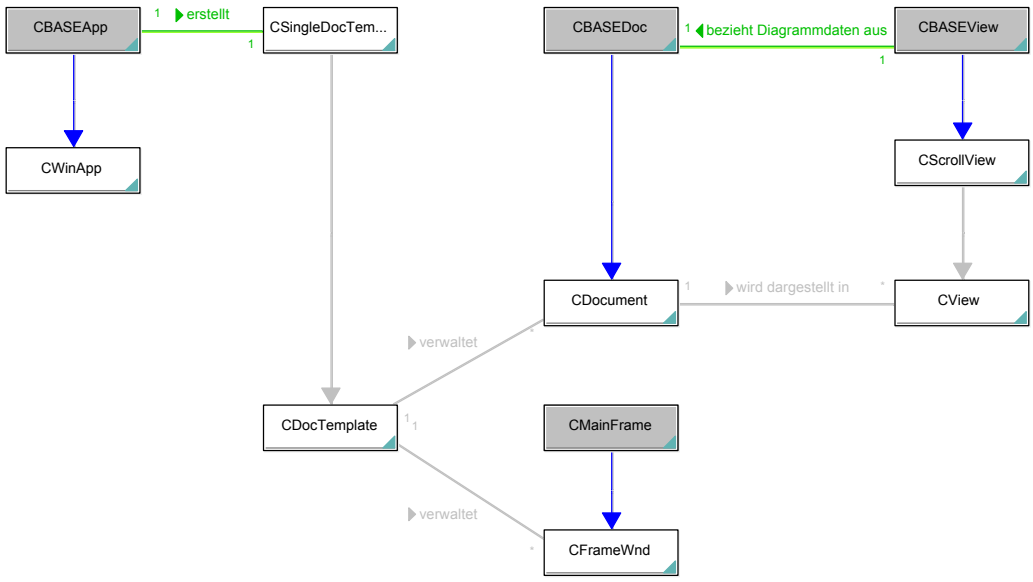


Abb. 5.15 Grundlegende Klassen von BASE

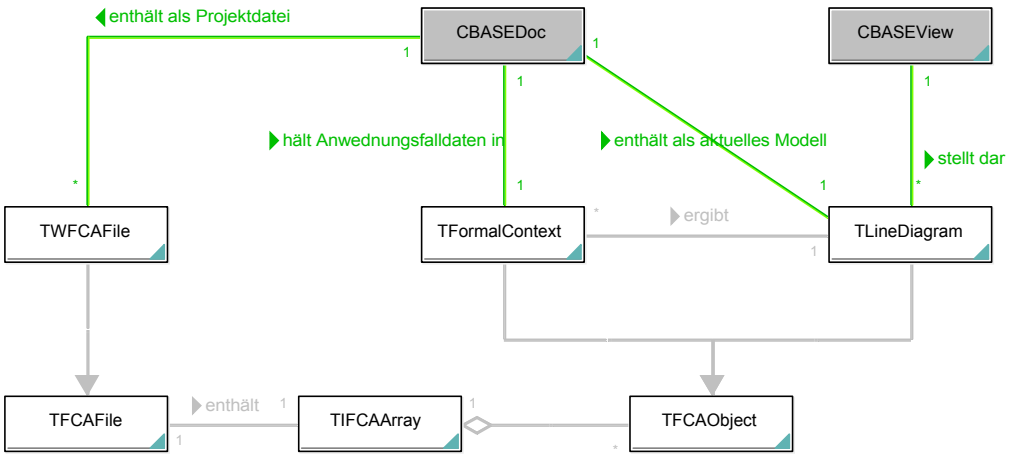


Abb. 5.16 *CBASEDoc* und *CBASEView*

Im Falle von BASE gibt es auch jeweils nur ein "View-Objekt". Nur die Diagrammansicht ist so programmiert, Daten zu den Anwendungsfällen und Überprüfungsfragen werden – wie schon in 5.1 erläutert – in Dialogen präsentiert. Weil die gewählte Fenstergröße eventuell für die Darstellung eines Diagramms nicht ausreichend ist, erbt *CBASEView* von *CScrollView*, um "Scrollen" zu ermöglichen.

Die wirklich für BASE spezifischen Klassen sind *CBASEDoc* und *CBASEView*. Deshalb werden sie im folgenden genauer erklärt. Wie sie Klassen aus *The Formal Concept Analysis Library* benutzen, zeigt Abbildung 5.16. (Dunkelgrau hinterlegten Klassen sind innerhalb von BASE angelegt, die anderen in *The Formal Concept Analysis Library* enthalten.)

Die Klasse CBASEDoc

Das Dateiformat von BASE ist *ConScript* wie von der Arbeitsgruppe "Allgemeine Algebra und Diskrete Mathematik" des Mathematik-Fachbereichs der Technischen Hochschule Darmstadt definiert (siehe 5.4.4 auf Seite 216). ConScript-Dateien speichern Strukturen der Formalen Begriffsanalyse. Dementsprechend werden in BASE die Projektdateien durch die Klasse *TWFCAFile* repräsentiert. Diese erweitert die eigentliche Kernklasse *TFCAFile* um Meldungen in einem Dialog auszugeben. Die gespeicherten begriffsanalytischen Strukturen – insbesondere formale Kontexte (*TFormalContext*) und Liniendiagramme (*TLineDiagram*) werden allgemein durch die Klasse *TFCAObject* repräsentiert. *CBASEDoc* enthält einen Zeiger auf ein Objekt von *TWFCAFile*. Damit sind alle in der Projektdatei gespeicherten Strukturen zugänglich.

Daten zu Anwendungsfällen werden in Objekten der Klasse *TFormalContext* gespeichert (siehe 5.4.2 auf Seite 210). Für jedes Anwendungsfallmodell, das der Benutzer für das Projekt speichern möchte, wird dementsprechend ein Objekt dieser Klasse erstellt. Nachdem der entsprechende formale Kontext der Projektdatei zugefügt und aus den Daten ein Liniendiagramm erzeugt wurde, hält *CBASEDoc* lediglich einen Zeiger auf das Liniendiagramm. Die Zuordnung beider Strukturen in der Datei erfolgt über ihren Namen: Beide tragen den selben Namen.

Zu erzeugten Blockzerlegungen wird lediglich das Übersichtsdiagramm mit einem Verweis auf das Liniendiagramm gespeichert, aus dem es berechnet wurde. Der zu seiner Erzeugung benutzte formale Kontext existiert nur temporär. Genauso werden Diagrammausschnitte – also Hauptfilter/-ideale und einzelne Blöcke einer Blockzerlegung – und Detaildiagramme zu Blockzerlegungen temporär als Liniendiagramme erzeugt, aber nicht gespeichert. Abhängig davon, welche Art von Liniendiagramm angezeigt ist, stehen dem Benutzer verschiedene Funktionen zur Verfügung. *CBASEDoc* speichert deshalb, ob gerade ein Blockrelationsdiagramm angezeigt wird und ob das angezeigte Diagramm nur temporär erzeugt ist.

Damit Blockrelationsdiagramme zu ein und demselben Anwendungsfallmodell nicht immer neu in der Projektdatei gesucht werden müssen, werden Zeiger auf sie zusätzlich in *CBASEDoc* gespeichert, sobald der Benutzer die Anzeige einer Blockzerlegung anfordert.

Schließlich enthält *CBASEDoc* noch einen Zeiger auf einen Dialog zur Anzeige von Überprüfungsfragen.

Damit kann *CBASEDoc* folgende Aufgaben übernehmen:

- Neuerstellung, Öffnen und Speichern von Projekten
- Auswahl und Löschen eines Anwendungsfallmodells
- Eingabe und Veränderung von Anwendungsfalldaten
- Generierung und Anzeige von Überprüfungsfragen
- Berechnung und Verwaltung von Blockzerlegungen

Anhang D enthält die Klassendeklaration von *CBASEDoc*.

Die Klasse *CBASEView*

CBASEView übernimmt das Zeichnen der Liniendiagramme. Die Daten zu einem Liniendiagramm sind in einem Objekt der Klasse *TLineDiagram* enthalten. Insbesondere trifft dies auf Koordinatenangaben für die Knoten und Beschriftungen zu. Diese in Koordinaten des Zeichenbereichs umzuwandeln, ist eine wesentliche Aufgabe der Klasse. Der Inhalt der Klasse ist um einiges "technischer" als der von *CBASEDoc*. Deshalb wird hier auf eine Erklärung der Attribute der Klasse verzichtet und nur die von der Klasse übernommenen Aufgaben werden aufgelistet. Diese sind:

- Zeichnen des aktuellen Diagramms
- Ein- und Ausblenden der Beschriftung mit Anwendungsfällen bzw. "Dingen"
- Verschieben von einzelnen Knoten, ganzen Hauptfiltern oder -idealen durch den Benutzer
- Erzeugung und Anzeige von Diagrammausschnitten (Hauptfilter/-ideale, einzelne Blöcke)
- Erzeugung und Anzeige von Detaildiagrammen zu Blockzerlegungen
- Größenänderungen des Diagramm und seiner Knoten
- Festlegung der Schriftart für die Beschriftung mit Anwendungsfällen bzw. "Dingen" durch den Benutzer
- Zuordnung einer Zeichenfarbe zu einem aus mehreren Knoten bestehenden Block durch den Benutzer.

Anhang D enthält die Klassendeklaration von *CBASEView*.

5.4.2 Verwaltung der Anwendungsfalldaten

Zu jedem Anwendungsfall und jeder Funktion sind der Name, eine Beschreibung, die involvierten "Dinge", die benutzten Anwendungsfälle, die Anwendungsfälle, die spezialisiert wurden, und die funktionale Zerlegung zu speichern. Benutzt wird dazu die Klasse *TFormalContext*. Sie hat die folgenden Attribute:

```
class TFormalContext: public TFCAObject
{
    static const char ID_string[];
       // ConScript-Schlüsselwort für formale Kontexte
    TQSObjectArray    the_objects;
       // formale Gegenstände
    TQSAttributeArray the_attributes;
       // formale Merkmale
    TRelation         the_relation;
       // Inzidenzrelation
    TArrows the_arrows;
       // Pfeilrelationen
    int arrows_up_ok;
       // Sind die gespeichgerten Aufwärtspfeile gültig?
    int arrows_down_ok;
       // Sind die gespeichgerten Abwärtspfeile gültig?
};
```

Mit Hilfe der Pfeilrelationen können formale Gegenstände mit $\bigvee$-irreduziblem Gegenstandsbegriff und formale Merkmale mit $\bigwedge$-irreduziblem Merkmalsbegriff charakterisiert werden (vergl. Definition 11 auf Seite 72). Dies ist in verschiedenen Berechnungen von Bedeutung. So können im besonderen formale Kontexte auf ihre im Sinne des vorigen Satzes irreduziblen formalen Gegenstände und formale Merkmale reduziert werden, ohne den zugehörigen Begriffsverband (bis auf Isomorphie) zu ändern. Eine solche Reduzierung erlaubt eine effizientere Berechnung des Verbands. Außerdem läßt sich mit Hilfe der Pfeilrelationen eine hier nicht betrachtete Klasse von Implikationen auszeichnen und berechnen und sogenannte doppelt fundierte Kontexte definieren, die viele Eigenschaften von endlichen Kontexten erhalten. (weitere Ausführungen in [G-W 96], 1.2, 2.3 und [Vog 96], 2.3, 16.1) Für BASE sind Pfeilrelationen nicht von herausragender Bedeutung.

Die Klasse *TRelation* speichert eine Inzidenzrelation, indem jede Zeile und Spalte der zugehörigen Kreuztabelle in ein Bit-Array abgebildet wird. (Diese redundante Speicherung der Zeilen und Spalten erlaubt einen schnellen Zugriff bei den wesentlichen Operationen der Inhalts- und Umfangsbestimmung, siehe Algorithmus B2 auf Seite 281) Einzelheiten sind in [Vog 96], 2.1, 11.1 nachzulesen.[1]

Für BASE interessant ist, genauer zu sehen, wie formale Gegenstände und Merk-

1. Die redundante Zeilen- und Spaltenspeicherung verdoppelt den Speicherbedarf. Richard Cole, Peter Eklund und Bernd Groh untersuchen in [CEG 97] Möglichkeiten, große Kontexte zu komprimieren.

male in *The Formal Concept Analysis Library* behandelt werden. Zunächst einmal der relevante Teil der Klassenhierarchie:

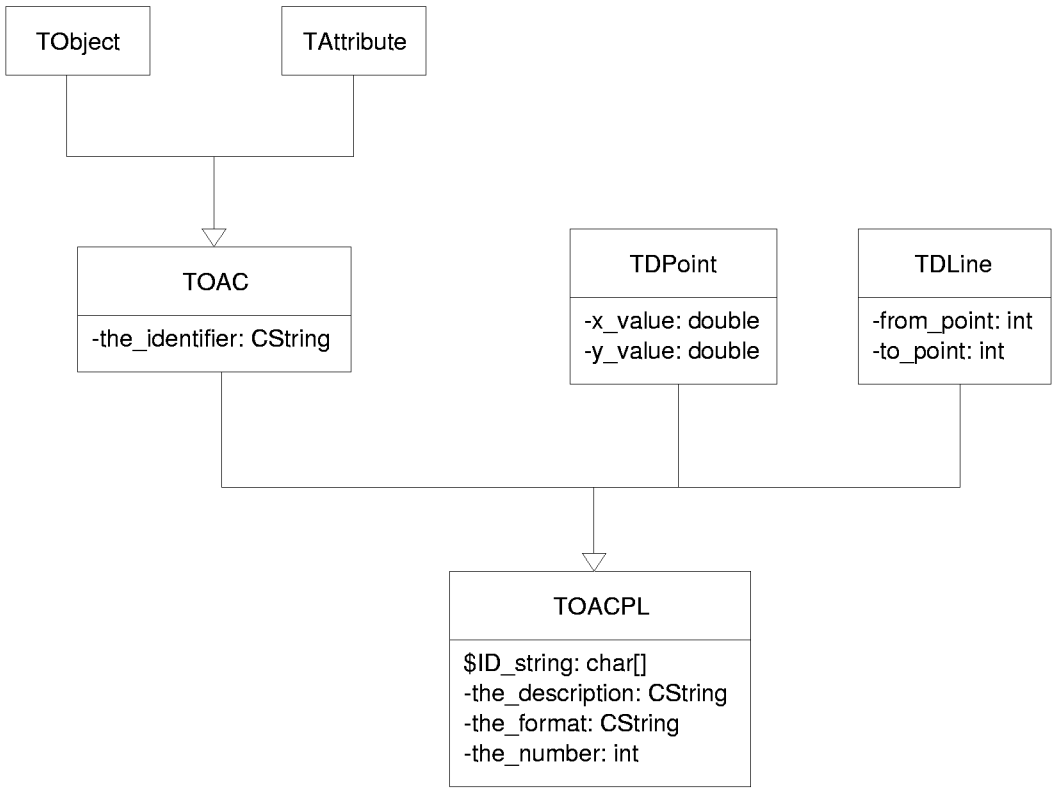


Abb. 5.17 Klassen zur Darstellung von formalen Gegenständen/ Merkmalen, Diagrammknoten und -linien

Die Klasse *TObject* speichert formale Gegenstände und die Klasse *TAttribute* formale Merkmale. Identifiziert werden die einzelnen Gegenstände und Merkmale über einen internen eindeutigen Bezeichner *the_identifier*. Ihre für den Benutzer sichtbare Bezeichnung – also in BASE die Namen von Anwendungsfällen und "Dingen" – werden in *the_description* abgelegt. Diese Zeichenkette wird zur Anzeige im Anwendungsfall-Editor und im Liniendiagramm benutzt. Für die Liniendiagrammdarstellung nimmt *the_format* Angaben zur Darstellung des Textes auf. Das Attribut *the_number* hält in einem formalen Kontext die zugehörige Zeilen- bzw. Spaltennummer fest, in einem Liniendiagramm die Nummer des Knotens zum zugehörigen Gegenstands- bzw. Merkmalsbegriff. Das Klassenattribut *ID_string* enthält lediglich das ConScript-Schlüsselwort für die entsprechende Klasse.

Der Name eines Anwendungsfalls und die Zuordnung von involvierten "Dingen" können also direkt in einem durch die Klasse *TFormalContext* gegebenen formalen Kontext gespeichert werden. Für die anderen Angaben ist das von der Oberklasse

TFCAObject geerbte Attribut *the_specials* zuständig. Es speichert ein Array von Zeichenketten. Diese werden auch in ConScript-Dateien festgehalten (siehe 5.4.4 auf Seite 216). Von *The Formal Concept Analysis Library* und ConScript ist gerade vorgesehen, in dieser Weise anwendungsspezifische Informationen zu speichern. Für die Anwendungsfallangaben verwendet BASE spezielle Formate für die entsprechenden Zeichenketten (Darstellung in EBNF ([ISO 96]), Metazeichen sind grau gedruckt):

- `Anwendungsfallbeschreibung = 'USE_CASE' Bezeichner ':' Beschreibungstext`
- `UsesAngabe = 'USES' Bezeichner Bezeichner`
- `ExtendsAngabe = 'EXTENDS' Bezeichner Bezeichner`
- `Zerlegungsschritt = 'STEP' Bezeichner Nummer Bezeichner`

Die auftretenden `Bezeichner` sind dabei immer die intern genutzten und in *the_identifier* gespeicherten. `Beschreibungstext` ist die als Anwendungsfallbeschreibung eingegebene Zeichenkette. Bei "*uses*"-Angaben steht zuerst der `Bezeichner` des benutzenden und dann der des benutzten Anwendungsfalls, analog bei "*extends*"-Angaben erst der `Bezeichner` des allgemeineren und dann der des spezielleren Anwendungsfalls. Zu jedem funktional zerlegten Anwendungsfall wird die oberste Ebene der Zerlegungsschritte aufgeführt. Dazu sind pro Schritt zunächst der `Bezeichner` des zerlegten Anwendungsfalls, dann die betreffende Stelle in der Schrittfolge auf der obersten Zerlegungsebene (beginnend mit 0) und schließlich der `Bezeichner` der in dem betreffenden Schritt ausgeführten Funktion angegeben.

In dieser Weise können alle Anwendungsfalldaten in einem formalen Kontext gespeichert werden. Die durch "*uses*"-/"*extends*"-Beziehungen oder funktionale Zerlegungen induzierten Merkmalsimplikationen werden in der gespeicherten Inzidenzrelation nicht durch unmittelbar eingefügte Kreuze berücksichtigt, um dem Benutzer die Möglichkeit zu geben, solche Beziehungen wieder zurückzunehmen. Vor der Berechnung des zugehörigen Liniendiagramms müssen sie in den Kontext eingearbeitet werden. Dies ist der Grund, warum die spätere Bearbeitung von Modellen – bis auf Änderungen der Anwendungsfalldaten – allein anhand des Liniendiagramms geschieht: der gespeicherte formale Kontext paßt ohne die erzwungenen Merkmalsimplikationen nicht zum Diagramm.

Während der Eingabe der Anwendungsfalldaten werden die im Endeffekt in *the_specials* gespeicherten Angaben in einer eigenen Datenstruktur verwaltet, um nicht immer die oben spezifizierten Zeichenketten parsen zu müssen. Die Klasse für diese Datenstruktur heißt *CUsecaseAngaben*. Ihre Deklaration ist in Anhang D wiedergegeben.

5.4.3 Verwaltung der Liniendiagramme

Liniendiagramme werden mit Hilfe der Klasse *TLineDiagram* verwaltet. Wie sie aus formalen Kontexten berechnet werden, erläutert Anhang B.2.

Die Attribute der Klasse *TLineDiagram* sind:

```
class _FCAEXPORT TLineDiagram: public TFCAObject
{
    static const char ID_string[];
       // ConScript-Schlüsselwort für Liniendiagramme
    double the_unitlength;
       // Zeicheneinheit, bzgl. der Positions- Längenangaben zu
       // verstehen sind
    CString the_unit;
       // Maßeinheit der Zeicheneinheit
    TQSDPointArray    the_points;
       // Diagrammknoten
    TQSDLineArray     the_lines;
       // Diagrammkanten
    TQSObjectArray    the_objects;
       // formale Gegenstände
    TQSAttributeArray the_attributes;
       // formale Merkmale
    TQSConceptArray   the_concepts;
       // Namen von formalen Begriffen
    TRelation the_transitive_closure;
       // reflexiv-transitive Hülle des Liniendiagramms
    int transitive_closure_ok;
       // Ist die gespeicherte reflexiv-transitive Hülle gültig?
    TintArray the_indices;
       // Speichert zu jeder Knotennummer die Position in
       // <the_points>.
    int indices_ok;
       // Sind die Angaben in <the_indices> gültig?
};
```

Die reflexiv-transitive Hülle des Liniendiagramms als gerichteter Graph ist gerade die Verbandsordnung, wenn das Liniendiagramm einen Verband darstellt. Aus ihr können die Knotennummern zu über oder unter einem Knoten gelegenen Knoten – d.h. dessen Hauptfilter und Hauptideal (Definition 5 auf Seite 66) – direkt abgelesen werden. Diese spielen für Verbandsoperationen eine herausragende Rolle (siehe etwa Anhang B.4.2). Deshalb wird die reflexiv-transitive Hülle im Attribut *the_transitive_closure* gespeichert, wenn sie einmal berechnet wurde. Mit der in B.2.2 auf Seite 286 erläuterten Erweiterung in der Diagrammerzeugung geschieht dies sofort mit der Diagrammberechnung.

Die Klasse *TLineDiagram* stellt aber auch Operationen wie das Zufügen und Löschen von Knoten oder Kanten bereit, die Einfluß auf die reflexiv-transitive Hülle haben. Weil die reflexiv-transitive Hülle nur bei einigen Operationen benötigt

wird, wird sie nicht immer sofort neu berechnet. In *transitive_closure_ok* kann man deshalb abfragen, ob *the_transitive_closure* gültig ist. Analog verhält es sich mit *the_indices* und *indices_ok*. Innerhalb von BASE werden keine dementsprechenden Änderungen vorgenommen und die Knotennummern entsprechen immer den Arraypositionen in *the_points*. Dadurch sind die beiden "ok-Attribute" nicht von Interesse. Nicht einmal *the_indices* wird benötigt.

Die Attribute der Klassen für Knoten und Kanten *TDPoint* und *TDLine* sind Abbildung 5.17 auf Seite 211 zu entnehmen. Ein Knoten speichert in *x_value* und *y_value* seine Koordinaten (im durch *the_unitlength* und *the_unit* spezifizierten Koordinatensystem), in *the_number* seine Nummer und in *the_format* Angaben zu seiner Formatierung. Eine Kante enthält in *from_point* und *to_point* die Nummern ihres Start- (oben) und Zielknotens (unten), in *the_number* ihre eigene Nummer und in *the_format* Angaben zu ihrer Formatierung. Für beide bleibt *the_description* in der Regel leer.[1]

Für die Verwendung in BASE wurde *The Formal Concept Analysis Library* um die Berechnung von Duquenne-Guigues-Implikationenbasen (Satz 14 auf Seite 109) ergänzt (siehe Anhang B.3). Um diese nicht immer neu berechnen zu müssen, werden sie mit dem Liniendiagramm gespeichert. Dies geschieht wieder über das von *TFCAObject* an *TLineDiagram* vererbte Attribut *the_specials*. Gespeichert werden nur die Prämissen der Implikationen, also die Pseudoumfänge bzw. Pseudoinhalte. In BASE werden die Pseudoumfänge sofort bei der Erstellung des Liniendiagramms mit berechnet, um Überprüfungsfragen zur Beurteilung des Detaillierungsgrads einer funktionalen Zerlegung sofort ohne vorherige Implikationenberechnung anzeigen zu können. (Die Berechnung der Pseudoumfänge geht mit der Berechnung der Begriffsumfänge einher, siehe B.3 auf Seite 287) Die Basis der Merkmalsimplikationen wird jedoch nur bei Bedarf berechnet, dann aber auch im Liniendiagramm gespeichert.

Die Speicherung der Pseudoumfänge und Pseudoinhalte in *the_specials* geschieht in der folgenden Form (Darstellung in EBNF, Metazeichen sind grau gedruckt):

- `Pseudoumfang = 'PSEUDO_EXTENT' { Bezeichner }`
- `Pseudoinhalt = 'PSEUDO_INTENT' { Bezeichner }`

Die `Bezeichner` sind wieder die intern verwendeten in *the_identifier* gespeicher-

1. Die Speicherung des Liniendiagramms in Knoten und Kanten ist in *The Formal Concept Analysis Library* an der graphischen Repräsentation orientiert. Insbesondere können so beiden Diagrammelementen graphische Attribute zugeordnet werden. Die Umsetzung der Verbandsoperationen (Infimums-/Supremumsbildung, Vergeleiche bzgl. der Ordnung, Ermittlung von unteren/oberen Nachbarn) erfolgt über die reflexiv-transitive Hülle. Alternative wäre eine Graphen-orientierte Speicherung – wie etwa in [Lin 99], 3.6. Peter Becker diskutiert in seiner Diplomarbeit [Bec 99] verschiedene Möglichkeiten in einem solchen Ansatz die Begriffsumfänge und -inhalte zu speichern. In [ABLN 89] werden Möglichkeiten dargestellt, Ordnungen in kompakter Weise zu kodieren und trotzdem Verbandsoperationen effizient zu implementieren.

ten (siehe Abbildung 5.17 auf Seite 211). Durch sie werden alle dem betreffenden Pseudoumfang angehörenden formalen Gegenstände bzw. alle dem Pseudoinhalt zugehörigen Merkmale identifiziert. Die leere Menge ist dementsprechend durch das Schlüsselwort `PSEUDO_EXTENT` oder `PSEUDO_INTENT` allein angegeben.

Zu einem Liniendiagramm berechnete Blockzerlegungen werden selber wieder als Liniendiagramm gespeichert (siehe Anhang B.4). Um die Querverweise zu erhalten, werden deren Namen ebenfalls in *the_specials* festgehalten. Die Form der entsprechenden Angaben in EBNF ist:

- `VerweisAufBlockzerlegung = 'BLOCKRELATION' Diagrammname`

Damit ist es immer wieder möglich, zu den schon berechneten Blockrelationsdiagrammen zu gelangen. Wird eines dieser Diagramme gelöscht, wird auch der Verweis im Originaldiagramm entfernt.

Zu jeder berechneten Blockrelation wird ein Liniendiagramm in der Projektdatei gespeichert. Seine Anordnung wird dadurch bestimmt, daß jeder Knoten in den Mittelpunkt des entsprechenden Blocks im Ausgangsdiagramm der Berechnung gelegt wird. Für diese Übersichtsdiagramme zu Blockzerlegungen wird in BASE zusätzlich gespeichert, daß es sich um ein Blockrelationsdiagramm handelt, wie das zugrunde liegende Diagramm heißt und welche Knoten im Originaldiagramm Infimum und Supremum eines Blocks bilden.

Um ein Diagramm als zu einer Blockrelation gehörig auszuzeichnen, wird das von *TFCAObject* an *TLineDiagram* vererbte Attribut *the_remark* mit der Zeichenkette "`BLOCKRELATION`" belegt. Zunächst wird dabei noch "not confirmed" angehängt, um anzuzeigen, daß der Benutzer die Speicherung dieser Blockzerlegung noch nicht bestätigt hat. Wird ein Diagramm mit diesem Zusatz angezeigt, wird nach einer solchen Bestätigung gefragt (siehe Abbildung 5.12 auf Seite 203). Bestätigt der Benutzer die Speicherung, wird der Zusatz entfernt und ein eventuell neu festgelegter Name eingetragen. Lehnt der Benutzer die Blockzerlegung ab, wird sie aus der Datei gelöscht.

Auf das einer Blockrelation zugrunde liegende Ausgangsdiagramm wird ein Verweis dadurch hergestellt, daß in *the_specials* folgendes eingetragen wird:

- `VerweisAufAusgangsdiagramm = 'ORIGINALDIAGRAM' Diagrammname`

Dieser Verweis wird ausgenutzt, wenn der Benutzer den Anzeigemodus für Blockrelationsdiagramme (respektive Komponentenzerlegungen) ausschaltet, um wieder zum Ausgangsdiagramm zurückzukehren, das der Zerlegung zugrunde liegt.

Abbildung 5.17 auf Seite 211 ist zu entnehmen, daß die Klasse für Diagrammknoten *TDPoint* ein Attribut *the_description* enthält. Hierin wird die Angabe von Block-Infimum und Block-Supremum gespeichert. (Im Übersichtsdiagramm sind die Blöcke durch einzelne Knoten gegeben.) Das Format ist (in EBNF):

- `BlockAngabe = 'BLOCK' Infimumsnummer Supremumsnummer`

Die Nummern sind dabei die Knotennummern des Ausgangsdiagramms.

Anhand der Nummern von Block-Infimum und -Supremum aller Blöcke und des Verweises auf das Ausgangsdiagramm kann auch das Detaildiagramm zu einer Blockzerlegung berechnet werden. Es wird genau wie Diagrammausschnitte (Hauptfilter/-ideale, einzelne Blöcke) nur als temporäres Objekt der Klasse *TLineDiagram* erzeugt und gleich wieder gelöscht, wenn ein anderes Diagramm angezeigt wird. In dem schon oben angesprochenen Attribut *the_remark* erhalten sie den Namen des Diagramms, aus dem sie berechnet wurden. So kann der Benutzer zu diesem zurückkehren.

5.4.4 Dateiformat

BASE benutzt *ConScript* als Dateiformat. Dieses wurde von der Arbeitsgruppe "Allgemeine Algebra und Diskrete Mathematik" des Mathematik-Fachbereichs der Technischen Hochschule Darmstadt um Rudolf Wille entwickelt. [Vog 96] (darin Anhang A) enthält eine Spezifikation. Durch die Verwendung von ConScript ist BASE mit den in Darmstadt entwickelten Werkzeugen wie *Anaconda* von Frank Vogt vollständig kompatibel. Allerdings stehen die BASE-spezifischen Informationen anderen Werkzeugen lediglich als Zeichenketten zur Verfügung.

ConScript-Dateien werden als einfacher ASCII-Text gespeichert. Sie sind also mit jedem Editor lesbar. Deshalb soll hier der für BASE interessante Teil von ConScript vorgestellt werden, damit man die Dateien auch in ihrer Textrepräsentation verstehen kann. Die folgenden Spezifikationen sind in EBNF abgefaßt, wobei Metazeichen zur besseren Lesbarkeit grau gedruckt sind. Innerhalb von ConScript mit Hilfe von Schlüsselwörtern spezifizierte Konstrukte sind dabei mit englischen Bezeichnungen belegt, um an die englischen ConScript-Schlüsselwörter zu erinnern. Grundlegend gelten die folgenden Festlegungen:

- Ein `Bezeichner` besteht aus ASCII-Buchstaben, Ziffern und dem Unterstrich, sein erstes Zeichen ist aber keine Ziffer.
- Eine `Zeichenkette` besteht aus beliebigen ASCII-Zeichen. Doppelte Anführungszeichen " müssen durch den Schrägstrich \ maskiert werden.
- `NatZahl` steht für eine natürliche Zahl (einschließlich 0).
- `GpZahl` bezeichnet Gleitpunktzahlen.

ConScript-Dateien

Eine ConScript-Datei hat das folgende Format:

```
ConScriptFile = {  RemarkList |
                   FormalContext_List |
                   LineDiagramList |
                   IncludeFile |
                   ConceptualFile_List |
                   ConceptualScheme_List |
                   DatabaseList |
                   AbstractScaleList |
                   ConcreteScaleList |
                   RealizedScaleList |
                   StringMapList |
                   IdentifierMapList |
                   QueryMapList
                }
```

Für BASE sind nur die ersten drei Alternativen von Bedeutung. Andere ConScript-Dateien als *Include_Files* einzubinden, ist nicht nötig. Alle restlichen in einer ConScript-Datei speicherbaren Strukturen dienen Werkzeugen wie *Toscana* ([KSVW 94]), die mehrwertige Merkmale relationaler Datenbanken mit Hilfe von begrifflichen Skalen behandeln und sind für BASE ohne Bedeutung.

Formale Kontexte

Formale Kontexte werden in ConScript in der folgenden Weise gespeichert:

```
FormalContextList = 'FORMAL_CONTEXT' { FormalContext }

FormalContext =
       Bezeichner '='
       [ Title ]
       [ Remark ]
       [ ContextSpecialList ]
       'OBJECTS'
         { ZeilenNr Bezeichner [ Beschreibung [ TextFormat ]] }
       'ATTRIBUTES'
         { SpaltenNr Bezeichner [ Beschreibung [ TextFormat ]] }
       'Relation'
         Höhe ',' Breite { '.' | '*' }
       ';'

ZeilenNr = NatZahl

SpaltenNr = NatZahl

Höhe = NatZahl

Breite = NatZahl
```

Die Speicherung eines formalen Kontext geschieht in Übereinstimmung mit der in 5.4.2 auf Seite 210 dargestellten Klasse *TFormalContext*. 'OBJECTS' leitet die Liste

der formalen Gegenstände ein, 'ATTRIBUTES' die der formalen Merkmale. Zur Speicherung der Relation wird erst die Zeilenanzahl, dann die Spaltenanzahl und schließlich zeilenweise die Belegung der Kreuztabelle angegeben. Dabei steht '.' für eine leere Tabellenzelle und '*' für ein Kreuz. Es müssen auf diese Weise genau HöheXBreite Tabellenzellen angegeben werden und alle zu formalen Gegenständen angegebenen Zeilennummern müssen zwischen 0 und Höhe-1 liegen, die Spaltennummern zu den Merkmalen zwischen 0 und Breite-1. Weiter müssen die Bezeichner von formalen Gegenständen und Merkmalen jeweils paarweise verschieden sein. In TextFormat stehen Formatierungsinformationen. Da diese in BASE innerhalb von formalen Kontexten bedeutungslos sind, wird TextFormat zusammen mit der ConScript-Definition für Liniendiagramme auf Seite 221 erklärt. Die anderen Bestandteile sind:

```
Title = 'TITLE' '"' Zeichenkette '"' TextFormat

Remark = 'REMARK' '"' Zeichenkette '"'

ContextSpecialList =
       'SPECIAL' {
                    '"'
                      ( Anwendungsfallbeschreibung |
                        UsesAngabe |
                        ExtendsAngabe |
                        Zerlegungsschritt |
                        Zeichenkette
                      )
                    '"'
                  }
```

Die Formate von Anwendungsfallbeschreibung, UsesAngabe, ExtendsAngabe, und Zerlegungsschritt wurden schon in 5.4.2 auf Seite 212 erklärt. Andere Anwendungen definieren eventuell noch andere Zeichenketten, um sie unter 'SPECIAL' in einem formalen Kontext zu speichern.

Damit ist dargestellt wie die Speicherung der Anwendungsfalldaten in ConScript-Dateien geschieht.

Liniendiagramme

Liniendiagramme werden in ConScript wie folgt festgehalten:

```
LineDiagramList = 'LINE_DIAGRAM' { LineDiagram }
```

```
LineDiagram =
       Bezeichner '='
       [ Title ]
       [ DiagramRemark ]
       [ DiagramSpecialList ]
       'UNITLENGTH' GpZahl MaßEinheit
       'POINTS'
         { KnotenNr xKoord yKoord [ KnotenBsbg ] [ PunktFormat ] }
       'LINES'
         { '(' Startknoten ',' Zielknoten ')' [ LinienFormat ] }
       'OBJECTS'
         { KnotenNr Bezeichner [ Beschreibung [ TextFormat ]] }
       'ATTRIBUTES'
         { KnotenNr Bezeichner [ Beschreibung [ TextFormat ]] }
       'CONCEPTS'
         { KnotenNr Bezeichner [ Beschreibung [ TextFormat ]] }
       ';'

MaßEinheit = 'mm' | 'cm' | 'in'

KnotenNr = NatZahl

xKoord = GpZahl

yKoord = GpZahl

Startknoten = NatZahl

Zielknoten = NatZahl
```

Die Koordinaten der Punkte beziehen sich auf das unter 'UNITLENGTH' angegebene Längenmaß. BASE unterstützt nur die beiden metrischen Maßeinheiten, nicht aber Inch. Bei der Auflistung der formalen Gegenstände und Merkmale ist der entscheidende Unterschied zum formalen Kontext, daß hier die Nummern der Knoten der betreffenden Gegenstands- bzw. Merkmalsbegriffe wiedergegeben werden. In dem durch 'CONCEPTS' eingeleiteten Teil können Namen für formale Begriffe (die durch Knoten dargestellt werden) festgelegt werden. BASE nutzt dies nicht.

Für die Konsistenz des Diagramms müssen die Knotennummer paarweise verschieden sein, formale Gegenstände/Merkmale und Begriffsbezeichnungen dürfen nur aufgeführte Knotennummern referenzieren. Weiter müssen die Bezeichner von formalen Gegenständen/Merkmalen und Begriffsbezeichnungen jeweils paarweise verschieden sein.

```
DiagramRemark =
       'REMARK' '"' (   'BLOCKRELATION' |
                        'BLOCKRELATION not confirmed' |
                        AusgangsDiagrammName |
                        Zeichenkette
                    ) '"'
```

Die ersten drei Alternativen sind für BASE von Bedeutung (siehe 5.4.3). Darüber hinaus wird ist eine gespeicherte Bemerkung zu einem Liniendiagramm einfach eine Zeichenkette.

```
DiagramSpecialList =
        'SPECIAL' {
                    '"'
                       ( Pseudoumfang |
                         Pseudoinhalt |
                         VerweisAufBlockzerlegung |
                         VerweisAufAusgangsdiagramm |
                         Zeichenkette
                       )
                    '"'
                  }
```

Die Formate von Pseudoumfang, Pseudoinhalt, VerweisAufBlockzerlegung, und VerweisAufAusgangsdiagramm wurden schon in 5.4.3 erklärt. Andere Anwendungen definieren eventuell noch andere Zeichenketten, um sie unter 'SPECIAL' in einem Liniendiagramm zu speichern.

```
KnotenBsbg = '"' ( BlockAngabe | Zeichenkette ) '"'
```

Der Aufbau von BlockAngabe ist schon in 5.4.3 dokumentiert.

Die Angaben zu Formatierungen werden wie folgt gemacht:

```
PunktFormat =
        '"'
          [ Radius [ Maßeinheit ]] ','
          [ LinienStil ] ','
          [ LinienStärke [ Maßeinheit ] ] ','
          [ LinienFarbe ] ','
          [ Füllstil ] ','
          [ Füllfarbe ]
        '"'
```

Diagrammknoten werden als Kreise gezeichnet. Die Art der Kreislinie und der Füllung innerhalb der Kreislinie können unabhängig voneinander gewählt werden.

```
Radius = GpZahl
```

```
LinienStil = 's' | 'd' | 't'
```

Dabei bedeutet 's' *durchgezogen*, 'd' *gestrichelt* und 't' *gepunktet*. Die beiden letzteren stehen aber in BASE nur zur Verfügung, wenn die Linienstärke 0.1 mm beträgt.

```
LinienStärke = GpZahl

LinienFarbe = Farbe

Füllfarbe = Farbe

FüllStil = 'e' | 'f' | 'l' | 'u'
```

Dabei bedeutet 'e' *unausgefüllt*, 'f' *komplett ausgefüllt*, 'l' *in der unteren Hälfte ausgefüllt* und 'u' *in der oberen Hälfte ausgefüllt*. Farbe wird weiter unten erklärt.

```
LinienFormat =
        '"'
          [ LinienStil ] ','
          [ LinienStärke [ Maßeinheit ] ] ','
          [ LinienFarbe ]
        '"'
```

Zur Formatierung und Ausrichtung von Beschriftungen dient:

```
TextFormat =
        '"'
          [ SchriftartName ] ','
          [ SchriftAttribut ] ','
          [ Farbe ] ','
          [ GpZahl SchriftgrößenEinheit ] ','
          [ '(' HoriPosition ',' VertPosition ')' ] ','
          [ HoriBezugspunkt ] [ VertBezugspunkt ] ','
          [ '(' BoxBreite ',' BoxHöhe ')' ]
        '"'

SchriftartName = Zeichenkette

SchriftgrößenEinheit = 'pt' | 'mm'

SchriftAttribut = 'b' | 'c' | 'o' | 's' | 'u'
```

Dabei bedeutet 'b' *fett*, 'c' *kursiv*, 'o' *durchgestrichen*, 's' *schattiert* und 'u' *unterstrichen*. Eine Textdarstellung mit Schatten wird aber von vorliegendem Werkzeug nicht unterstützt. Desweiteren schließen sich die Schriftattribute aufgrund der Definition von ConScript gegenseitig aus.

```
HoriPosition = GpZahl

VertPosition = GpZahl

HoriBezugspunkt = 'l' | 'c' | 'r'

VertBezugspunkt = 'b' | 'c' | 't'
```

Die Position einer Beschriftung wird relativ zum betreffenden Diagrammknoten in der unter 'UNITLENGTH' gegebenen Längeneinheit angegeben. Der Bezugspunkt dieser Angabe innerhalb der Beschriftung kann verschieden gewählt werden. Dabei bedeutet 'l' *links*, 'c' *zentriert*, 'r' *rechts*, 'b' *unten* und 't' *oben*.

```
BoxBreite = GpZahl

BoxHöhe = GpZahl
```

Die letzten beiden Angaben spezifizieren die Größe eines Bereichs, außerhalb dessen nichts dargestellt wird.

In ConScript ist lediglich festgelegt, daß Farben als Zeichenketten angegeben werden. BASE unterstützt die folgenden Angaben:

```
Farbe = 'White' | 'Gray' | 'Dark Gray' | 'Black' | 'Red' |
        'Green' | 'Blue' | 'Yellow' | 'Pink' | 'Cyan' | 'Brown' |
        'Dark Green' | 'Dark Blue' | 'Beige' | 'Magenta' |
        'Turquoise' |
        '#' rWert gWert bWert |
        Zeichenkette
```

Die Farben der üblichen Palette sind aus Gründen der Kompatibilität mit *Anaconda* benannt. Um aber Blöcke von Blockzerlegungen automatisch verschieden einfärben zu können, wurde zusätzlich die Farbspezifikation von HTML eingearbeitet. Dementsprechend sind rWert, gWert und bWert zweistellige Sedezimalzahlen (wobei Kleinbuchstaben verwendet werden). So ist die Speicherung von Farben in RGB-Format möglich.

Der Projektname

Der Projektname wird als Kommentar zur Datei gespeichert. Das geschieht mit Hilfe der Klasse *TRemark*. Sie ist eine Unterklasse von *TFCAObject*. Dementsprechend können Bemerkungen auch als eigene Strukturen (wie formale Kontexte und Liniendiagramme) in Dateien gespeichert werden. Es wird einfach nur das Attribut *the_remark* von *TFCAObject* mit einer Zeichenkette belegt. Zur eindeutigen Identifizierung wird dem Projektnamen noch "PROJECT" als Schlüsselwort vorangestellt.

In ConScript sieht das dann folgendermaßen aus:

```
RemarkList = 'REMARK'
          { '"'
                    ( 'Project' Projektname | Zeichenkette )
          '"' }
```

6 Verwandte Anwendungen von FBA

Neben BASE gibt es verschiedene andere Ansätze, die Formale Begriffsanalyse in der Software-Entwicklung einzusetzen. Einige sind in der folgenden Tabelle aufgeführt.

Autoren und Referenz	Zielrichtung	Formale Gegenstände	Formale Merkmale	Formale Begriffe	Semantik der Ordnung
BASE	Ermittlung von Klassenkandidaten	Relevante "Dinge"	Anwendungsfälle und Funktionen	Nuclei zur Klassenbildung	Benutzungs-hierarchie
R. Godin et al. [GMM+ 98]	Strukturierung von Klassenhierarchien	Klassen	Attribute und Operationen	(Abstrakte) Klassen	Vererbung
G. Snelting, F. Tip [S-T 99]	Strukturierung von Klassenhierarchien	Programm-variablen	Attribute und Operationen	Klassen	Vererbung
C. Lindig, G. Snelting [L-S 97]	Modularisierung von Altsystemen	Programm-variablen	Programm-prozeduren	Module (maximaler Kohäsion)	Schachtelung von Modulen
C. Lindig, G. Snelting [Sne 96], [Lin 98]	Konfigurations-analyse	Codesegmente	Kontrollierende Ausdrücke	Konfigurationen	Spezialisierung von Konfigurationen
C. Lindig [Lin 95]	Suche in Komponentenbibliotheken	SoftwareKomponenten	Schlüsselworte	Zustände in Suchvorgängen	Verfeinerung in Suchvorgängen
F. Vogt [Vog 97]	Projektverfolgung	Relevante "Dinge"	Entwicklungs-tätigkeiten in verschiedenen Projektphasen	Systemteile gleichen Entwicklungs-stands	Entwicklungsvor-sprung zwischen verschiedenen Projektteilen

Abb. 6.1 Einsatz von FBA im Software Engineering

Die aufgelisteten Ansätze sollen hier kurz – im Vergleich zu BASE – erläutert werden. Um eine erste Einordnung zu geben, ist oben vermerkt, welche Bedeutung die zentralen begriffsanalytischen Strukturen – formale Gegenstände, Merkmale und

Begriffe und die Verbandsordnung auf den Begriffen – im jeweiligen Ansatz besitzen und welcher Zweck jeweils verfolgt wird.

BASE ist in Kapitel 4 schon hinreichend dargestellt. Hier soll nur noch auf die drei Verkürzungen in der Formulierung von Abbildung 6.1 hingewiesen werden.

Als Schlagwort für das Ziel von BASE ist die Ermittlung von Klassenkandidaten genannt. Zu diesem Zweck sieht BASE auch einige Techniken vor (siehe 4.5 und 4.6). Vor allem aber werden funktionale und datenorientierte Sicht zusammenhängend dargestellt und so Datenabhängigkeiten zwischen den einzelnen Anwendungsfällen herausgestellt. Außerdem wird die gemeinsame Einarbeitung von Entwicklern und Anwendern in den Untersuchungsbereich gefördert. Liniendiagramme und Überprüfungsfragen bilden dabei eine Richtschnur für die Kommunikation.

Die Bezeichnung von formalen Begriffen als Nuclei zur Klassenbildung (vergl. 4.5.2) ist nicht wirklich korrekt. Die angesprochenen Nuclei werden durch "Dinge" und Funktionen gebildet, die in der reduzierten Beschriftung des Liniendiagramms den gleichen Knoten referenzieren, nicht aber aus den formalen Begriffen mit komplettem Umfang und Inhalt.

Die Kennzeichnung der Semantik der Begriffsverbandsordnung ist ebenfalls verkürzt angegeben. Innerhalb von BASE stellt das Diagramm von oben nach unten gelesen die funktionale Verfeinerung des Anwendungsfallmodells dar. Entsprechende Unterordnung von Anwendungsfällen oder Funktionen unter andere muß aber nicht zwingend eine vorliegende Benutzung bedeuten. Lediglich finden sich oben die allgemeineren und unten die spezielleren Anwendungsfälle bzw. Funktionen. In bezug auf die involvierten "Dinge" stellt sich der Übergang von allgemein wichtigen zu in speziellen Zusammenhängen interessanten "Dingen" gerade von unten nach oben dar.

Zuerst seien nun die zwei Ansätze dargestellt, die sich um die Strukturierung von Klassenmodellen bemühen.

6.1 Klassenhierarchie nach Godin et al.

Robert Godin, Hafedh Mili, Guy W. Mineau, Rokia Missaoui, Amina Arfi und Thuy-Tien Chau schlagen in [GMM+ 98] vor, Vererbungshierarchien von Klassen als Begriffsverbände zu organisieren. Dazu werden jeder (als formaler Gegenstand) betrachteten Klasse ihre Attribute und Operationen als formale Merkmale zugesprochen. Aus dem so erstellten formalen Kontext gewinnt man einen Be-

griffsverband, dessen Begriffe in ihrem Umfang solche Klassen zusammenfassen, die gemeinsame Attribute und Operationen besitzen. Insofern ist dieser Ansatz dem von BASE ähnlich, als bei BASE unter den formalen Gegenständen auch (noch nicht identifizierte) Klassen auftauchen und die Operationen von Klassen ebenfalls unter den formalen Merkmalen zu finden sind. Attribute zu Klassen werden aber anders behandelt. Wichtigster Unterschied ist, daß Godin et al. ihre Analyse von der statischen Struktur eines Klassenmodells her unternehmen, während für BASE die Informationen über den Ablauf der Anwendungsfälle den Ausgangspunkt der Analyse bilden.

Abbildung 6.2 zeigt einen Ausschnitt der Klassenhierarchie der ISE-Eiffel-Kernel-Bibliothek ([Mey 92]). Ausgehend von der Klasse *ARRAY2* für zweidimensionale Felder wurden alle Oberklassen aufgenommen. Die Verbandsordnung bestimmt einen Vorschlag für die Vererbungshierarchie der Klassen. Jeder formale Begriff entspricht einer Klasse dieser Hierarchie. Durch die reduzierte Beschriftung des Liniendiagramms (siehe 3.3.4 auf Seite 86) sind zu jeder Klasse am entsprechenden Diagrammknoten genau die selber eingeführten und nicht ererbten Attribute und Operationen zu sehen – die ererbten bekommt man durch die Betrachtung der kompletten Begriffsinhalte dazu. Der Klassenname wird durch den in der reduzierten Beschriftung zugeordneten formalen Gegenstand gegeben. (Was es bedeutet, wenn dieser dadurch nicht eindeutig bestimmt ist, ist weiter unten ausgeführt. Der Fall, daß ein Begriff kein Gegenstandsbegriff ist, wird in Zusammenhang mit Abbildung 6.3 auf Seite 227 diskutiert.) Im kompletten Begriffsumfang sind überdies die Namen aller Oberklassen enthalten.

Im Sinne der Vermeidung von Redundanz ist die vorgeschlagene Vererbungshierarchie optimal, weil alle Attribute und Operationen, die einigen Klassen gemeinsam zukommen, durch genau eine Klasse eingeführt werden. Allerdings sind in Abbildung 6.2 nur die Deklarationen und nicht die Implementierungen von Operationen berücksichtigt. (siehe unten)

In Abbildung 6.2 ist zu sehen, daß jeder Aspekt der Klasse *ARRAY*, der sich durch gemeinsame Attribute und Operationen einiger Oberklassen ausdrücken läßt, auch wirklich als eigene Klasse modelliert ist. Dabei berücksichtigen die einzelnen "Vererbungslinien" verschiedene Kriterien. (Bertrand Meyer unterscheidet in [Mey 90] die Kriterien *Elementzugriff*, *Traversierbarkeit* und *Speicherung*.) Die Klasse *TO_SPECIAL* ganz links regelt die Speicherverwaltung für Arrays und Strings. Rechts oben ist erst einmal festgehalten, daß *ARRAY* eine Container-Klasse ist, in deren Objekte mehrere andere Objekte gebündelt werden können. In dem mittleren Strang wird dann präzisiert, wie sich Arrays in bezug auf ihre Größe verhalten, rechts dagegen, wie der Zugriff auf Array-Elemente gestaltet ist. Diese beiden Vererbungslinien unter *CONTAINER* betreffen die von Meyer genannten Kri-

terien der Speicherung bzw. des Elementzugriffs. Da bei direktem Zugriff auf die Elemente Fragen der Traversierung keine Rolle spielen, gibt es für dieses Kriterium keinen gesonderten Strang. *TO_SPECIAL* läßt sich nicht den Kriterien zuordnen, weil die Klasse allein aus technischen Gründen in die Klassenbibliothek aufgenommen wurde.

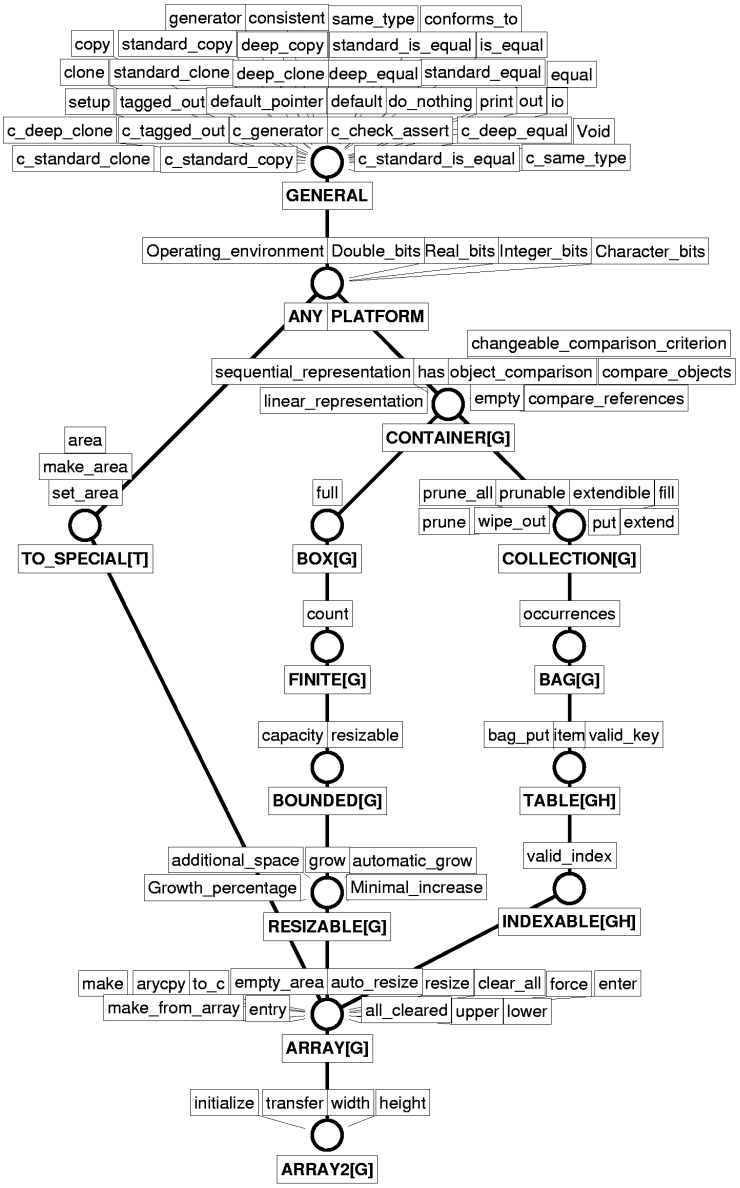


Abb. 6.2 Klassenhierarchie in Eiffel bis zu 2-dimensionalen Arrays

Alle Klassen der abgebildeten Hierarchie sind abstrakt – bis auf *ARRAY2*, *ARRAY, TO_SPECIAL*, *ANY, PLATFORM* und *GENERAL*. Die Klassen *ANY* und *PLATFORM* sind durch ihre Attribute und Operationsnamen nicht zu unterscheiden

(auch nicht durch die entsprechenden Operationsimplementierungen). Dieser kurios anmutende Umstand hat seine Begründung darin, daß *ANY* die Wurzelklasse der gesamten Klassenhierarchie von Eiffel-Programmen ist, *GENERAL* und *PLATFORM* aber nur für das Eiffelsystem selber wichtig und beim Programmieren in Eiffel unsichtbar sind. ANY kapselt deren Funktionalität, um eine Anpassung der Wurzelklasse an bestimmte Projektanforderungen zu ermöglichen.

An diesem Beispiel sieht man, wie die Untersuchung mit Formaler Begriffsanalyse hilft, Vererbungsstrukturen in Klassenbibliotheken oder Programmen darzustellen, um sie besser zu verstehen.

Robert Godin und Hafedh Mili sehen diesen Ansatz aber auch für den Aufbau einer Klassenhierarchie in den frühen Phasen eines Software-Entwicklungsprojekts vor. [GMl 93][1] trägt den Titel "Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices". In dieser Arbeit wird ein ähnliches Beispiel wie oben untersucht: Untersuchungsgegenstand sind einige Container-Klassen von Smalltalk.

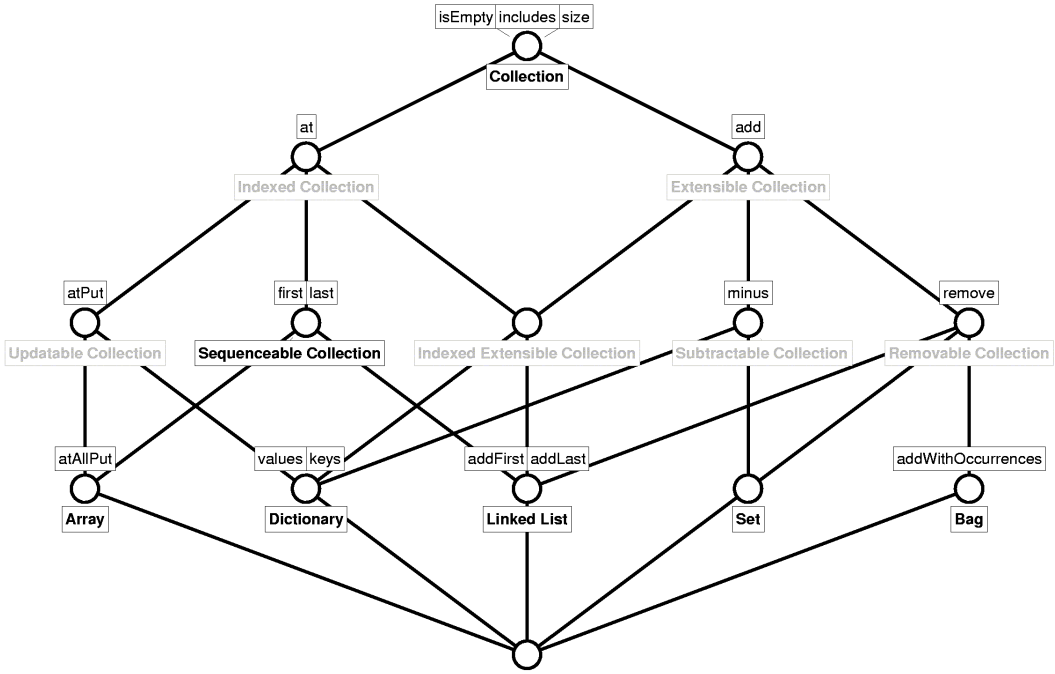


Abb. 6.3 Smalltalk-Container-Klassen

Die in Abbildung 6.3 grau angegebenen Klassen sind nicht in der Klassenbibliothek enthalten, sondern werden durch die Analyse als zusätzliche abstrakte Klassen vorgeschlagen, weil sie Gemeinsamkeit einiger vorhandener Klassen ausdrük-

1. Diese Arbeit bildet eine Vorläuferversion zu [GMM+ 98].

ken. Aus der Betrachtung gemeinsamer Attribute und Operationen erwächst also ein Vorschlag für eine Vererbungshierarchie. Ein solcher kann auch schon in frühen Projektphasen erstellt werden und Hilfe bei der Strukturierung des Klassenmodells bieten.

In Abbildung 6.3 fällt der mit *Indexed Extensible Collection* beschriftete Diagrammknoten besonders ins Auge. Ihm ist weder eine der vorhandenen Klassen noch ein Attribut oder eine Operation zugeordnet. Die vorgesehene abstrakte Klasse führt also keine eigenen Attribute und Operationen ein, sondern vereinigt nur diejenigen der ebenfalls neu eingeführten abstrakten Klassen *Indexed Collection* und *Extensible Collection*. Ihr Sinn für eine Klassenhierarchie erscheint deshalb fraglich. Die Autoren sehen deshalb eine alternative Repräsentation vor, in der nur die Merkmalsbegriffe des Begriffsverbands betrachtet werden – also ein Liniendiagramm der Merkmalsordnung (Definition A2 auf Seite 274). Dieses kann im übrigen sogar effizienter berechnet werden als der gesamte Begriffsverband.

Wenn auch die Operationsimplementierungen bei der Untersuchung bestehender Klassenbibliotheken berücksichtigt werden sollen, werden innerhalb der formalen Merkmale Operationsdeklarationen und -implementierungen jeweils als einzelne Merkmale behandelt. Der Zusammenhang zwischen ihnen kann durch Merkmalsimplikationen ausgedrückt werden (analog zur Behandlung von "*uses*"- und "*extends*"-Beziehungen und funktionalen Zerlegungen in BASE). Die Implementierung einer Operation impliziert als formales Merkmal dessen Deklaration. In der gleichen Weise können Spezialisierungsbeziehungen zwischen verschiedenen Implementierungen behandelt werden. Die letztgenannten sind aber schwierig aus dem Quellcode zu gewinnen. Alle anderen Angaben lassen sich leicht durch entsprechende Parser erstellen.

Während des Aufbaus eines Klassenmodells bietet es sich an, diesen Ansatz im Anschluß an die erste Analyse mit BASE anzuwenden.

6.2 Klassenhierarchie nach Snelting und Tip

Auch Gregor Snelting und Frank Tip untersuchen in [S-T 99] die Struktur von Klassenhierarchien mit Hilfe von Formaler Begriffsanalyse. Ihr Ansatzpunkt sind aber nicht wie bei Godin et al. Vererbungs- sondern Benutzungsbeziehungen. Ihr Verfahren erlaubt die Untersuchung einer Klassenhierarchie in bezug auf Programme, die deren Klassen benutzen. Als Spezialfall ist auch die Analyse einer Klassenbibliothek für sich allein vorgesehen. Im folgenden ist abkürzend immer von "Programmen" die Rede.

```
class String  { /*details omitted */ };
class Address  { /* details omitted */ };
enum Faculty  { Mathematics, ComputerScience };
class Professor;  /* forward declaration */
class Person
{ public:  String name;  Address address;  long socialSecurityNumber; };
class Student : public Person
{ public: Student (String sn, Address sa, int si)
            { name = sn;  address = sa;  studentId = si; };
    void setAdvisor (Professor *p)
            { avisor = p; };
    long studentId;
    Professor *advisor;
};
class Professor : public Person
{ public: Professor (String n, Faculty f, Address wa)
            { name = n; faculty = f; workAddress = wa; assistant = 0; };
    void hireAssistant (Student *s)
            { assistant = s; };
    Faculty faculty;
    Address workAddress;
    Student *assistant; /* either 0 or 1 assistant */
};
void main1()
{ String s1name, p1name;
  Address s1addr, p1addr;
  Student *s1 = new Student(s1name,s1addr,12345678);
  Professor *p1 = new Professor(p1name,Mathematics,p1addr);
  s1->setAdvisor(p1);
}
void main2()
{ String s2name,p2name;
  Address s2addr,p2addr;
  Student *s2 = new Student(s2name,s2addr,87654321);
  Professor *p2 = new Professor(p2name,ComputerScience,p2addr);
  p2->hireAssistant(s2);
}
```

Abb. 6.4 Beispiel für eine Klassenhierarchie und benutzende Programme

Als formale Gegenstände werden Variablen der betrachteten Programme behandelt, die als Typ eine Klasse aus der untersuchten Klassenhierarchie besitzen.[1] Die formalen Merkmale bilden die Attribute und Operationen der Klassen aus der untersuchten Klassenhierarchie. Die Inzidenzrelation des zur Analyse genutzten for-

1. Diese stehen quasi stellvertretend für die Objekte, die in ihnen gespeichert werden. Vergleiche die Diskussion einer "natürlichen" Klassenmodellierung mit formalen Begriffen in 1.6.2 auf Seite 16.

malen Kontexts bildet man anhand von Benutzungsbeziehungen (zwischen Variablen und Attributen/Operationen). Zu jeder Programmvariablen wird vermerkt, welche Attribute und Operationen von ihr aus benutzt werden. Dieser Ansatz des formalen Kontexts ist dem von BASE in der Beziehung ähnlich, daß die formalen Gegenstände auch Datenelemente repräsentieren und Funktionen als formale Merkmale auftauchen. Allerdings stehen die Attribute bei BASE "auf der anderen Seite". Der entscheidende Unterschied ist, daß BASE von den Funktionen eines Systems ausgeht und die Frage verfolgt, welche Datenelemente von einer Funktion benutzt werden, während für Snelting und Tip die Objekte eines Systems den Ansatzpunkt bilden und untersucht wird, welche Attribute und Operationen ihrer Klasse diese Objekte benutzen. Das angegebene Beispiel stammt aus [S-T 99].

In dem Beispiel sollen die Klassen *Person*, *Student* und *Professor* anhand der zwei Programme *main1* und *main2* untersucht werden. Es ergibt sich der folgende Begriffsverband:

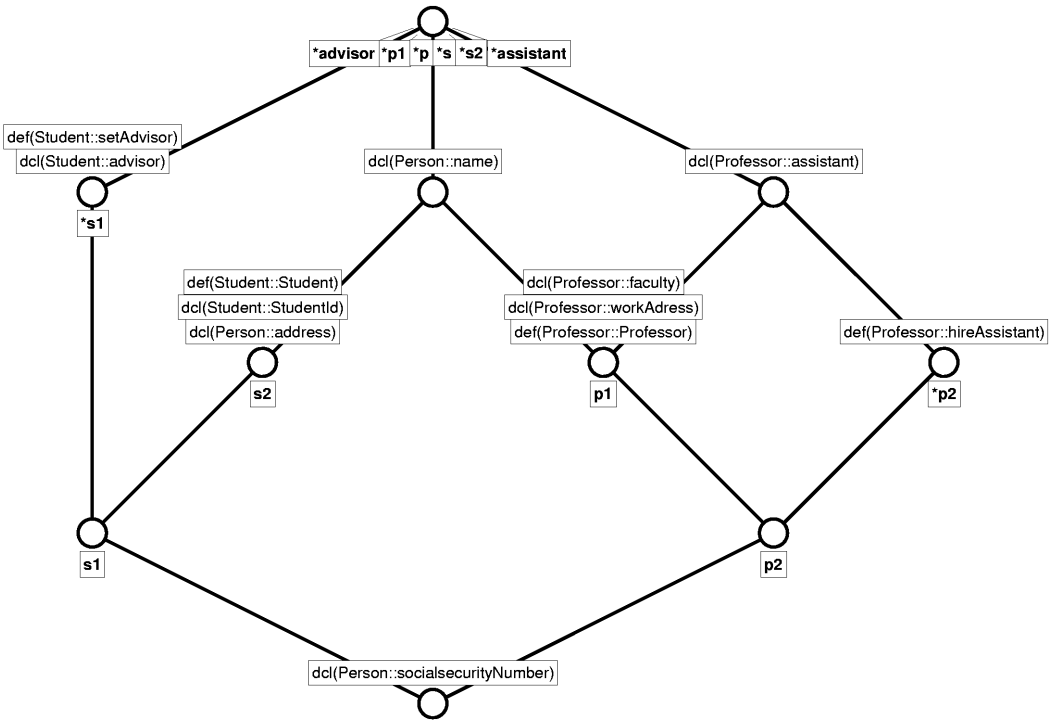


Abb. 6.5 Liniendiagramm zum Studenten/Professoren-Beispiel

Das Liniendiagramm stellt einen Restrukturierungsvorschlag für die untersuchte Klassenhierarchie dar. Der Begriffsinhalt eines jeden formalen Begriffs umfaßt die Attribute und Operationen einer der untersuchten Klassen, die von einigen Programmvariablen aus gemeinsam benutzt werden. Klassen der umstrukturierten Klassenhierarchie entsprechen den formalen Begriffen und erhalten wie bei Godin et al. die in der reduzierten Beschriftung zugeordneten formalen Merkmale als At-

tribute und Operationen. Diese werden ebenfalls wie bei Godin et al. gemäß der Verbandsordnung nach unten vererbt. In der reduzierten Beschriftung einem Diagrammknoten zugeordnete Variablen erhalten nach der Restrukturierung die entsprechende Klasse als Typ. Im vollständigen Begriffsinhalt sind darüber hinaus alle Variablen enthalten, die eine Unterklasse als Typ besitzen. Deshalb möchte man die Variablen als stellvertretend für Objekte ihres Typs ansehen, denn so bekommt man die in 1.6.2 als "natürlich" bezeichnete Modellierung von Klassen als formale Begriffe.

Die Klassen der umstrukturierten Hierarchie enthalten nur solche Attribute und Operationen, die auch wirklich zusammen genutzt werden. So können Klassen, deren Objekte in verschiedenen Rollen auftreten und dabei nur gewisse Teile der definierten Attribute und Operationen nutzen, identifiziert und danach aufgeteilt werden. Außerdem findet man Klassenteile, die gar nicht oder nur in bestimmten Unterklassen genutzt werden. Genauso stechen Variablen hervor, von denen aus keine Attribute oder Operationen benutzt werden.

In Abbildung 6.5 werden etwa Studenten mit (*s1*) und ohne zugeordnetem Betreuer (*s2*) unterschieden. Weiter ist ersichtlich, daß das Attribut *address* nur bei Studenten-Objekten genutzt wird. Alle am Supremum angegebenen Variablen benutzen kein Attribut und keine Operation ihrer Klassen. Und die Sozialversicherungsnummer spielt überhaupt keine Rolle. Zusammenhänge zwischen den Attributen und Operationen sind ebenfalls zu erkennen. Daß in Abbildung 6.5 der Merkmalsbegriff zum Attribut *advisor* kein Oberbegriff des Merkmalsbegriffs zum *student*-Konstruktor ist, zeigt etwa, daß vergessen wurde, dieses Attribut zu initialisieren.

Auf Basis des Liniendiagramms kann ein Entwickler entscheiden, ob er die Klassenhierarchie umstrukturieren will.

Obige Darstellung ist in bezug auf die Erstellung des Formalen Kontexts stark verkürzt, um schnell einen Eindruck geben zu können. Snelting und Tip beziehen sich im wesentlichen auf C++-Programme. Bei den formalen Gegenständen werden statische Variablen und Zeiger unterschieden. Insbesondere werden die *this*-Zeiger der Operationen auch berücksichtigt. Diesen werden "künstlich" die entsprechenden Operationen als inzident zugesprochen, damit jeder *this*-Zeiger in der umstrukturierten Klassenhierarchie auch wirklich die Klasse als Typ erhält, welche die Operation enthält. Attribute, deren Typ unter den untersuchten Klassen ist, treten sowohl als formale Gegenstände als auch als formale Merkmale auf. Bei Operationen wird zwischen Deklaration und Implementierung unterschieden (wie auch bei Godin et al. möglich). Dies hat besondere Bedeutung, um dynamisch gebundene Operationsaufrufe zutreffend nachbilden zu können. Für Zeiger ist es nötig zu ermitteln, auf welche Variablen sie im Ablauf der Programme zeigen können. Zuweisung werden durch Gegenstandsimplikationen modelliert. Sichtbarkeitseinschrän-

kungen zwischen Attributen und Operationen gleicher Signatur werden durch Merkmalsimplikationen nachgebildet.

Mit allen diesen Erwägungen ist die Methode allein für Re-Engineering-Aufgaben gedacht und findet in der Analysephase einer Neuentwicklung keine Anwendung.

6.3 Modularisierung

In 4.6 wurden schon die Arbeiten von Christian Lindig und Gregor Snelting zur Modularisierung von Altsystemen erwähnt. Ihr Ansatz, dabei Formale Begriffsanalyse einzusetzen, ist dem von Snelting/Tip und auch dem von BASE eng verwandt. Allerdings sind die Rollen von formalen Gegenständen und Merkmalen vertauscht.

Die Fragestellung ist, wie man im Quellcode von Altsystemen mögliche Komponentenzerlegungen entdecken kann, die bei der ursprünglichen Implementierung nicht explizit vorgenommen worden sind. Anhand einer solchen Zerlegung kann danach das System (besser als im vorliegenden Zustand) strukturiert werden, um spätere Erweiterungen und andere Pflegemaßnahmen leichter durchführen zu können. Sinnvolle Zerlegungen beachten das Prinzip der Datenabstraktion (siehe 2.2.2 auf Seite 24).

In ihrem Verfahren betrachten Lindig und Snelting – dual zu 6.2 und BASE – als formale Merkmale die Variablen und als formale Gegenstände die Prozeduren (inklusive Funktionen) eines Programms. Die Inzidenzrelation des formalen Kontexts gibt an, welche Variable in welcher Prozedur benutzt wird (ähnlich zu BASE). Das entstehende Liniendiagramm spiegelt wie in BASE die Datenabhängigkeiten der Prozeduren wieder. Dementsprechend dient es der Ableitung von Modularisierungsvorschlägen. Das Beispiel in Abbildung 6.6 stammt aus [L-S 97]. Es untersucht die Struktur eines Modula-2-Programms, das in 8 Modulen 33 Prozeduren und 16 Variablen enthält.

Zunächst einmal betrachten Lindig und Snelting formale Begriffe als Modulkandidaten. In den entsprechenden Modulen benutzen alle enthaltenen Prozeduren alle Variablen. Module dieser Art nennen die Autoren *Module maximaler Kohäsion*. In Abbildung 6.6 findet man solche Module durch die Knoten 2 bis 8. Sie sind vom restlichen System unabhängig. Weiter kann man Knoten 1 und 12 als eine Komponente auffassen, in der der Modul 12 lokal definiert ist.

Eine rigorose Modularisierung gemäß der formalen Begriffe würde eine Kopplung der Module 13 und 14 durch die ihnen globale Variable *namehashtab* bedeuten. Insbesondere wäre aber das Prinzip der Datenabstraktion verletzt, weil die Proze-

duren der Module 13, 14 und 15 Variablen verschiedener anderer Module benutzen und diese Abhängigkeiten auch nicht durch lokale Definition der betreffenden Module in anderen aufzuheben sind. Lindig und Snelting nennen diesen Umstand eine *Interferenz* der betreffenden Module. Sie sehen vor, solche Interferenzen aufzulösen, indem globale Variablen zu Parametern der entsprechenden Prozeduren gemacht werden.

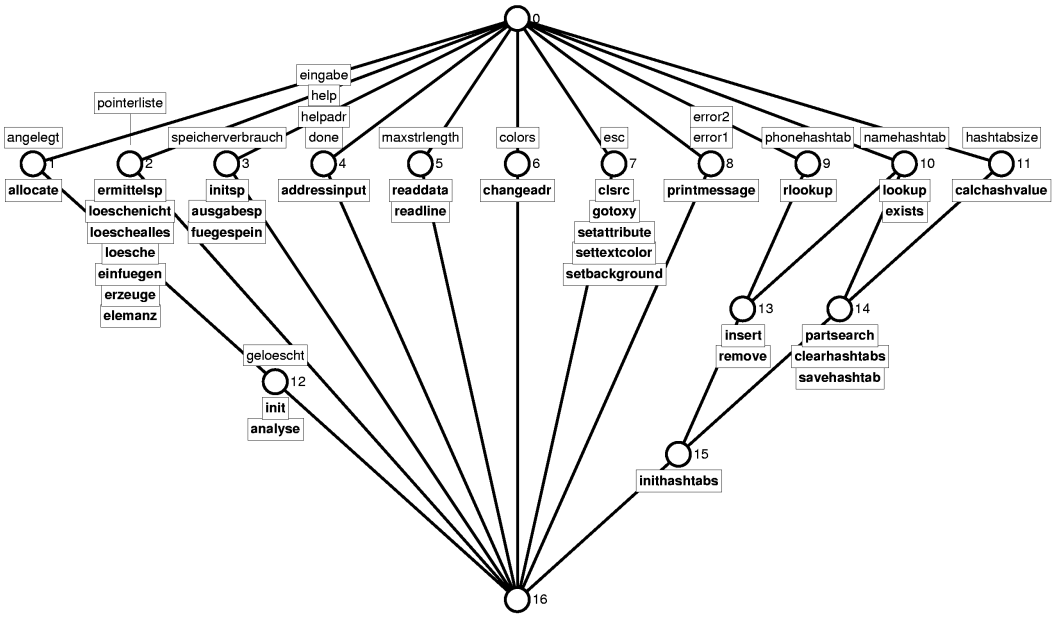


Abb. 6.6 Liniendiagramm zur Struktur eines Modula-2-Programms

Die Prozeduren und Variablen der Knoten 9, 10, 11, 13, 14 sind – wie im Diagramm zu sehen – schwer zu trennen. Also sollte man sie vielleicht in einer Komponente zusammenfassen. Dies bildet die Motivation, eine allgemeinere Art von Diagrammteilen als nur einzelne Knoten als Modulkandidaten zu betrachten. Lindig und Snelting betrachten vor allem horizontale Summanden: Ein Verband V heißt horizontale Summe seiner Unterverbände $V_1, \ldots, V_n$, wenn $V = \bigcup_{i=1}^{n} V_i$ und $V_i \cap V_j = \{\mathbf{0}_V, \mathbf{1}_V\}$ gilt. $V_1, \ldots, V_n$ heißen dann horizontale Summanden (vergl. [G-W 96], 1.3, S. 41). Im Liniendiagramm findet man horizontale Summanden dadurch, daß man das globale Supremum und Infimum entfernt. Die schwachen Zusammenhangskomponenten des verbleibenden Graphen bilden dann die horizontalen Summanden. Geht man davon aus, daß keine globale Variable von allen Prozeduren benutzt wird und keine Prozedur alle Variablen benutzt, so sind die als horizontale Summanden ermittelten Module voneinander unabhängig.

Wie in 4.6.1 erwähnt, werden auf der Suche nach Modulen auch Blockzerlegungen betrachtet. Mit der Annahme, daß das Supremum eines Blocks ein Merkmalsbegriff und sein Infimum ein Gegenstandsbegriff ist, erhält man auf diese Weise *Mo-*

dule mit regulärer Kohäsion ([L-S 97]), in denen zumindest eine Prozedur alle Variablen benutzt und dual mindesten eine Variable von allen Prozeduren benutzt wird. Die Autoren erwähnen diese Möglichkeit der Modularisierung, ohne ein konkretes Anwendungsbeispiel zu geben. Die Überlegung, daß innerhalb von BASE eher mit Klassenkandidaten mit regulärer Kohäsion zu rechnen ist, als bei beliebigen nach Datenabstraktion gebildeten Modulen, gab den Anstoß, Blockzerlegungen für BASE zu betrachten. (Die Klasse selbst spielt in allen ihren Operationen eine Rolle und ein eventuell modellierter Konstruktor initialisiert (hoffentlich) alle Attribute der Klasse.)

6.4 Konfigurationsanalyse

Lindig und Snelting bearbeiten eine weitere Aufgabenstellung aus dem Software-Re-Engineering mit Formaler Begriffsanalyse. Speziell Programme, die für verschiedene Plattformen geschrieben sind, weisen verschiedene Konfigurationen auf. Um solche Programme verstehen und pflegen zu können, ist es wichtig, ihre Konfigurationsstruktur zu erkennen.

Die Autoren betrachten durch Präprozessorsymbole in C++-Code erzeugte Konfigurationen. Abbildung 6.7 zeigt einen Teil des Quellcodes aus *rcsedit.c*.[1] Das Beispiel stammt aus [Sne 96]. Für die Behandlung mit Formaler Begriffsanalyse werden die verschiedenen Code-Stücke als formale Gegenstände und die sie regierenden Ausdrücke als formale Merkmale behandelt. Den entstehenden formalen Kontext bezeichnet Snelting als *Konfigurationstabelle*. Im Beispiel bekommt man so etwa den Begriffsverband in Abbildung 6.8 auf Seite 236.

Im Umfang eines formalen Begriffs sind die Code-Stücke zusammengefaßt, die gerade von den im Inhalt angegebenen Ausdrücken abhängig sind. Hat das Infimum zu zwei Merkmalsbegriffen einen nicht leeren Inhalt, existiert ein Code-Stück, das von beiden betreffenden Ausdrücken regiert wird. Im Fall der Unterordnung des einen Merkmalsbegriffs unter den anderen ist der Ausdruck zum Unterbegriff als Spezialisierung des zum Oberbegriff gehörigen zu verstehen. So ist in Abbildung 6.8 der Ausdruck *bad_unlink* der Alternative aus ihm und *has_NFS* untergeordnet.

1. *RCS* bedeutet *Revision Control System*. Es ist ein System zur Versions- und Konfigurationsverwaltung von Textdokumenten unter UNIX. (siehe [Tic 85])

```
180  #if has_NFS || bad_unlink
181     int
182  un_link(s)
183     char const *s;
184  /*
185   * Remove S, even if it is unwritable.
186   * Ignore unlink() ENOENT failures; NFS generates bogus ones.
187   */
188  {
189  #  if bad_unlink
190        int e;
191        if (unlink(s) == 0)
192           return 0;
193        e = errno;
194  #     if has_NFS
195           if (errno == ENOENT)
196              return 0;
197  #     endif
198        if (chmod(s, S_IWUSR) != 0) {
199           errno = e;
200           return -1;
201        }
202  #  endif
203  #  if has_NFS
204        return unlink(s)==0 || errno==ENOENT  ?  0  :  -1;
205  #  else
206        return unlink(s);
207  #  endif
208  }
209  #endif
210  #if !has_rename
211  #  if !has_NFS
212  #     define do_link(s,t) link(s,t)
213  #  else
214        static int do_link P((char const*,char const*));
```

Abb. 6.7 Code-Fragment aus einem RCS-Stream-Editor

Sind aber beide beteiligten Merkmalsbegriffe unvergleichbar, spricht Snelting von einer *Interferenz* (analog zu 6.3). In Abbildung 6.8 bildet etwa das Infimum der Merkmalsbegriffe zu *bad_unlink* und *has_NFS* eine Interferenz. Diese kann eine Schwachstelle in der Konfigurationsstruktur kennzeichnen. Dies ist der Fall, wenn die beteiligten Ausdrücke sich gegenseitig ausschließen und so die betreffenden Code-Stücke nie ausgeführt werden oder wenn die Ausdrücke *orthogonal* zueinander sind. Dabei heißt orthogonal, daß sie verschiedene Aspekte des Konfigurationsraums betreffen – etwa zum einen die Benutzeroberfläche und zum anderen das Betriebssystem. Im Sinne der Erweiterbarkeit und Wartbarkeit des Systems ist es vorteilhafter, solche Konstellationen zu vermeiden, um die verschiedenen Aspekte einer Konfiguration unabhängig voneinander behandeln zu können.

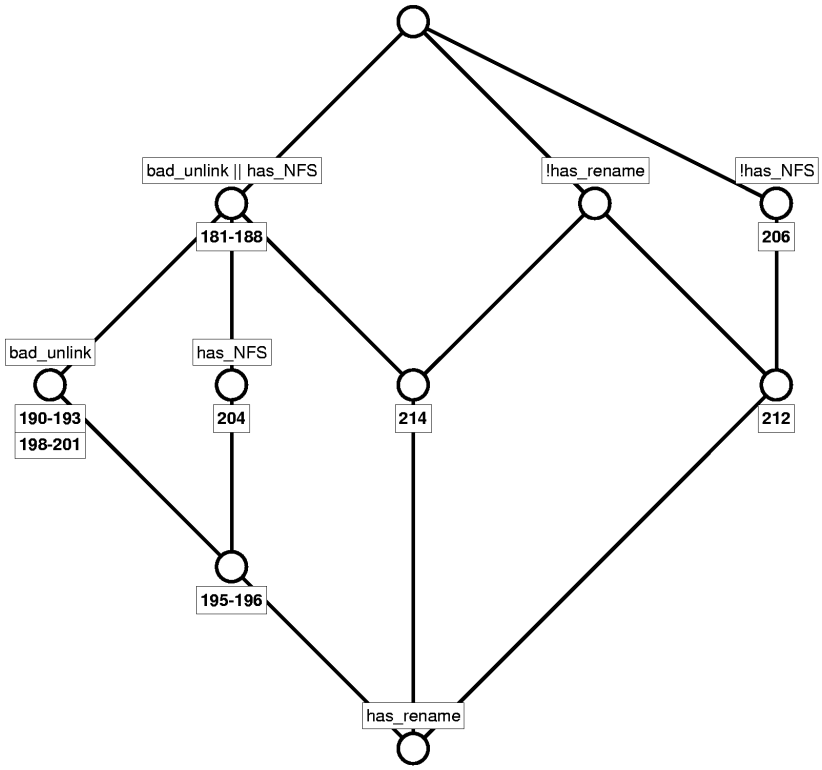


Abb. 6.8 Liniendiagramm zu einer Konfigurationstabelle

Um Interferenzen zu bestimmen und die Kopplung verschiedener Konfigurationen zu bewerten, sieht Snelting die in 6.3 schon vorgestellten horizontalen Verbandszerlegungen vor. Dabei erweitert er die Betrachtung auf Interferenzen k-fachen Zusammenhangs. Eine solche liegt vor, wenn das Liniendiagramm des Begriffsverbands ohne Supremum und Infimum k-fach (schwach) zusammenhängend ist, aber durch das Entfernen von k (einfachen) Interferenzen in zwei (schwache) Zusammenhangskomponenten zerfällt.

Weiter sieht Snelting vor, die Konfigurationstabelle auf solche Ausdrücke einzuschränken, die infimum-irreduzible Merkmalsbegriffe besitzen. Als Konsequenz aus Folgerung 5 (Seite 88) ergibt sich aus dem so verkleinerten formalen Kontext ein isomorpher Begriffsverband. Der untersuchte Quellcode kann in der gleichen Weise vereinfacht werden. Als Resultat benötigt man weniger Ausdrücke, um die gewünschten Konfigurationen zu erstellen.

In dem von Snelting betrachteten Begriffsverband sind in den Begriffsumfängen solche Code-Stücke zusammengefaßt, die von den gleichen Ausdrücken abhängen. Um die wirklich im Code enthaltenen – durch Kombinationen von Ausdrücken gegebenen – Varianten und zu ihnen alle Code-Teile zu bestimmen, die in ihnen aktiv sind, geht Lindig in [Lin 98] zum invertierten Kontext der Konfigurationstabelle über. So erhält er die vorkommenden Varianten als formale Begriffe, die in ihrem

Umfang gerade die aktiven Code-Stücke zusammenfassen.

6.5 Komponentensuche

Um die Wiederverwendung einmal erstellter Entwicklungsergebnisse zu ermöglichen, müssen diese in einem Bibliothekssystem verwaltet werden, das eine effektive Suche nach Software-Komponenten und anderen in Projekten erstellten Dokumenten (Spezifikationen, Entwurfsmuster, etc.) ermöglicht. Christian Lindig untersucht in [Lin 95], wie diese Suchaufgabe mit Hilfe von Formaler Begriffsanalyse unterstützt werden kann. Der von ihm genutzte grundlegende Ansatz findet sich in vielen Zusammenhängen (siehe z.B. [K-N 96], [C-E 96], [C-E 99], [Bec 99], [EKSW 00], [R-W 00]).

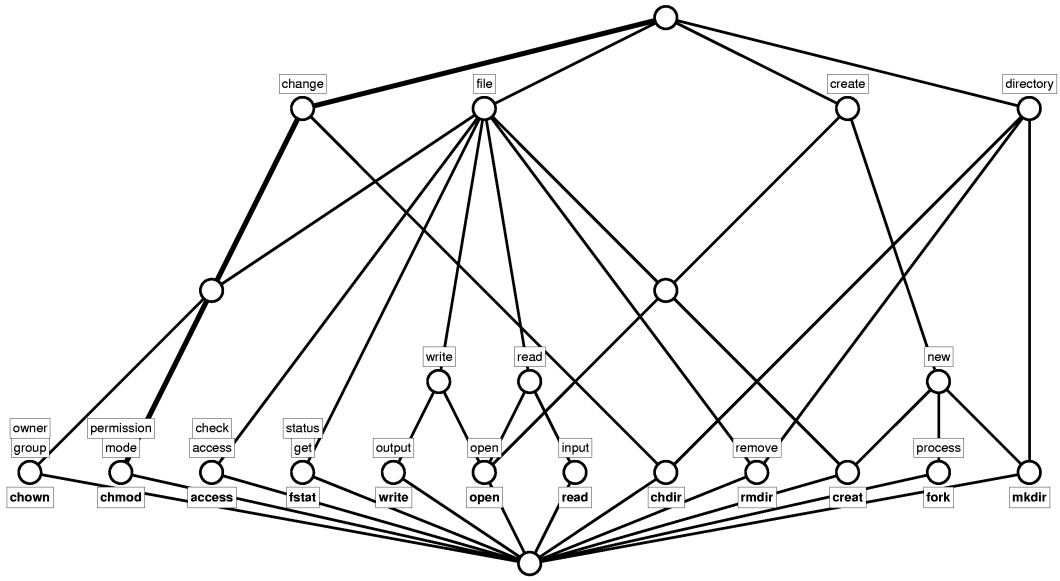


Abb. 6.9 UNIX-Betriebssystembefehle und zugehörige Schlagwörter

Als formale Gegenstände werden die zu suchenden Dokumente behandelt, als formale Gegenstände sie beschreibende Schlagwörter. Lindig behandelt als Beispiel UNIX-Betriebssystembefehle. Die entsprechenden Schlagwörter entnimmt er den zugehörigen Beschreibungen. Dabei kann er auf ein weitgehend standardisiertes Vokabular zurückgreifen. Den genannten Ansätzen ist gemein, daß sie auf die standardisierte Verwendung von Schlagwörtern angewiesen sind. Vorzugsweise nutzen sie vorgegebene Thesauri des behandelten Fachgebiets. Wie eine innerhalb eines Thesaurus spezifizierte Ordnung auf den enthaltenen Schlagwörtern berücksichtigt werden kann, wird in [GSW 98] erörtert.

Abbildung 6.9 gibt ein Beispiel aus [Lin 95] wieder. Die Suche nach den Dokumenten (hier in der niedrigsten Ebene aufgereiht) geschieht im Liniendiagramm vom Supremum aus nach unten. Jeder formale Begriff entspricht einem Zustand innerhalb eines Suchvorgangs, der durch die ausgewählten Schlagworte (Begriffsinhalt) und die selektierten Dokumente (Begriffsumfang) gekennzeichnet ist. Am Anfang der Suche sind im Supremum alle Dokumente selektiert und kein Schlagwort explizit ausgewählt. (Implizit sind die Schlagworte ausgewählt, die auf alle Dokumente zutreffen. Bei einer guten Indizierung sollte es solche nicht geben.)

Die meisten der nach dem geschilderten Prinzip arbeitenden Suchverfahren verzichten auf eine graphische Repräsentation des Begriffsverbands, weil diese bei großen Dokumentensammlungen sehr unübersichtlich werden kann. Sie präsentieren dem Benutzer lediglich die selektierten Dokumente, die explizit durch den Benutzer oder implizit aufgrund von Merkmalsimplikationen des Begriffsverbands ausgewählten Schlagwörter und eine Liste derjenigen Schlagwörter, die zusätzlich im nächsten Suchschritt wählbar sind, so daß echt weniger Dokumente selektiert werden, aber die Selektion nicht bei der leeren Dokumentenmenge landet. Die Menge der sinnvoll wählbaren Schlüsselworte berechnet man wie folgt. Zu einem formalen Begriff $(A, B) \in \underline{\mathfrak{B}}(G, M, I)$ mit $A \neq \emptyset$, der einen Zustand innerhalb des Suchvorgangs beschreibt, ermittelt man die Merkmalsmenge $\left(\bigcup_{g \in A} g' \right) \setminus B$, denn dies sind gerade die Merkmale, die – jeweils für sich allein genommen – auf eine echte, nicht-leere Teilmenge von A zutreffen. Die Möglichkeit, diese Schlagwörter dem Benutzer anzugeben, macht das Verfahren effektiv.

Der benutzte Begriffsverband muß nur bei Änderungen der Zuordnung der Schlagwörter zu Dokumenten neu berechnet werden. Die eigentliche Suche bedeutet viel weniger Aufwand und ist effizient implementierbar. Deshalb eignet sich der Ansatz besonders für Dokumentensammlungen, an denen nur selten Änderungen vorgenommen werden. Peter Becker berücksichtigt in [Bec 99] beim Verbandsaufbau den inkrementellen Algorithmus von Godin und Missaoui (siehe auch B.5.3 auf Seite 314) und macht das Verfahren für Sammlungen, in die regelmäßig neue Dokumente aufgenommen werden, noch interessanter.

In [Lin 99] sieht Lindig neben der Suche über Schlagwörter auch eine Suche mit Hilfe von Beispieldokumenten vor. In jedem einzelnen Suchschritt kann eine der beiden Möglichkeiten genutzt werden.

Die Autoren der Arbeiten [K-N 95], [C-E 96], [C-E 99], [EKSW 00] und [R-W 00] sehen eine Strukturierung innerhalb der Menge der Schlagwörter vor. Diese wird mit Hilfe einer *begrifflichen Skalierung* für die Begriffsanalyse aufbereitet (siehe [G-W 96], 1.3 und [GSW 98]). Anders ausgedrückt, wird so die Berücksichtigung mehrwertiger Merkmale möglich. Die benutzten Schlagwörter seien dafür ver-

schiedenen Themenbereichen oder verschiedenen Aspekten zu geordnet. Innerhalb eines Themenbereichs bestehen zwischen den Schlagwörtern Ober-/Unterbegriffsbeziehungen. Diese modelliert der Analytiker in einem Begriffsverband. In der Regel entsteht dieser nicht auf Grundlage der betrachteten Dokumentensammlung, sondern aufgrund des Begriffsverständnisses des Modellierers. Einen solchen Begriffsverband oder genauer einen ihm zugehörigen formalen Kontext nennt man eine Skala. Sie enthält alle denkbaren Merkmalskombinationen zu ihrem Themenbereich (wenn sie in zutreffender Weise erstellt wurde). Deshalb kann sie zur Suche in diesem Themenbereich herangezogen werden. Der Gesamtverband entsteht im wesentlichen als Produkt der Begriffsverbände zu den einzelnen Skalen ([G-W 96], 1.3, 2.2). Allerdings gibt es bei Skalen, die nicht aufgrund der vorliegenden Daten, sondern basierend auf der Erfahrung des Modellierers erstellt wurden, nicht zu jedem formalen Begriff Dokumente, die wirklich dessen Umfang auffüllen. Dies muß bei der Suche berücksichtigt werden. Vorteil für den Benutzer des Verfahren ist aber, daß er sich jeweils nur auf einen Themenbereich konzentrieren muß. Eine Visualisierung ist in gestuften Liniendiagrammen möglich. Dazu wird etwa bei zwei betrachteten Skalen in jedem Knoten des einen Liniendiagramms das andere angezeigt (siehe [G-W 96], 2.2, S. 75-79). *TOSCANA* ist ein kommerzielles Werkzeug zur Datenanalyse, das diese Technik nutzt ([KSVW 94]).

Im Bereich der Verwaltung von Projekt- und Software-Bibliotheken erscheint dieser Ansatz vielversprechend, wenn die Erstellung der Skalen auf einem bestehenden Klassifikationssystem – etwa einer Facetten-Klassifikation ([P-F 87]) – aufbauen kann.
Inwieweit der Ansatz der begrifflichen Skalierung in BASE zu tragen kommen könnte, wäre zu untersuchen. Vorstellbar ist die Sichten verschiedener Projektbeteiligter oder Anwendungsfallmodelle von Systemkomponenten einzeln zu skalieren.

6.6 Projektverfolgung

Frank Vogt sieht Liniendiagramme von Begriffsverbänden als geeignetes Mittel zur Kommunikationsunterstützung. In [Vog 97] wird als Beispiel ein Projekt zur Weiterentwicklung von *The Formal Concept Analysis Library* ([Vog 96]) untersucht. Formale Gegenstände sind bei diesem Ansatz die "Dinge" des Untersuchungsbereichs, die für die Systementwicklung relevant sind. In diesem Fall ist der Untersuchungsbereich die Formale Begriffsanalyse und dementsprechend sind die formalen Gegenstände Strukturen der Formalen Begriffsanalyse, die als Klassen modelliert sind oder werden sollen. Die formalen Merkmale werden nach den Ent-

wicklungstätigkeiten Analyse, Entwurf und Implementierung getrennt aufgestellt. Zunächst einmal sind es Eigenschaften der erfaßten Strukturen, die für die Systementwicklung interessant sind, und Aufgaben, die für sie durch das System zu erfüllen sind. Auf den Entwurf bezogen beschreiben die formalen Merkmale, welche Verantwortlichkeiten (siehe 2.4.3 auf Seite 43) für die modellierten Klassen zu spezifizieren sind. Die Inzidenzrelation gibt darüber Auskunft, welche davon bereits spezifiziert sind. Die Formulierung der Verantwortlichkeiten ist aus den in der Analyse festgelegten Aufgaben abgeleitet. Dies ermöglicht später einen Soll-Ist-Vergleich. Weiter wird festgehalten, welche Verantwortlichkeiten schon durch entsprechende Operationen und Attribute der Klassen implementiert sind.

Das Beispiel in Abbildung 6.10 stammt aus [Vog 97]. (Die Übersetzung ins Deutsche wurde ebenfalls von Vogt übernommen.) Ein vorgestelltes "A" (für "Analyse") kennzeichnet formale Merkmale, die der Problembeschreibung entnommene Eigenschaften und Aufgaben bezeichnen, "E" steht dagegen für "Entwurf" und "I" für "Implementierung". (Einige der im Entwurf ausgemachten Verantwortlichkeiten scheinen nach Vogt keiner Implementierung zu bedürfen.)

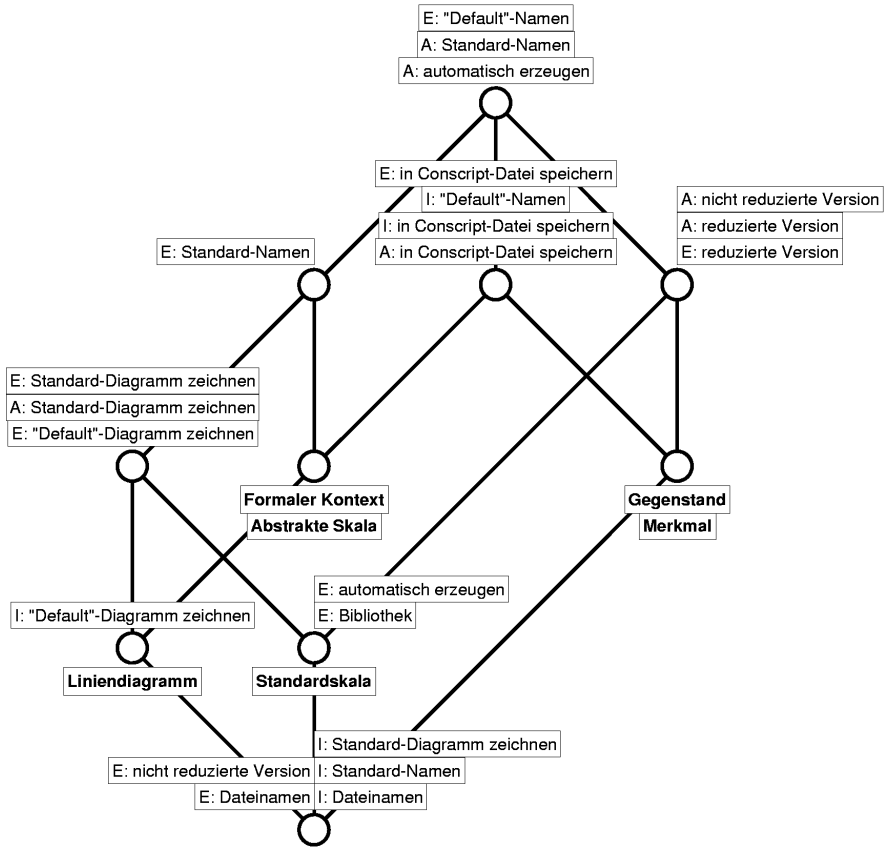


Abb. 6.10 Liniendiagramm zur Projektverfolgung

Der Projektfortschritt ist in diesem Liniendiagramm dokumentiert. Die Beschrif-

tung des Supremums zeigt etwa, daß für alle betrachteten Strukturen standardisierte Namen eingeführt werden sollen, dies ist aber im bisherigen Entwurf nur für formale Kontexte, abstrakte Skalen und Liniendiagramme berücksichtigt (linker unterer Nachbar des Supremums) und noch für keine Struktur implementiert (Infimum). Dagegen ist die Speicherung in ConScript-Dateien nur für alle Strukturen außer Standardskalen vorgesehen, schon entworfen und schon implementiert (mittlerer unterer Nachbar des Supremums).

Im Laufe des Projekts werden die zugrundeliegenden Kontexte modifiziert und die sich neu ergebenden Liniendiagramme spiegeln den jeweiligen Projektstand wieder. Bei Projektende sollten dann die sich entsprechenden formalen Merkmale aus der Analyse, dem Entwurf und der Implementierung den selben Knoten referenzieren.

Der Aufbau des "Analysekontexts" ist bezüglich der Rollen der beteiligten formalen Gegenstände und Merkmale dem Ansatz von BASE ähnlich. Entscheidender Unterschied ist aber, wie er erstellt wird. Während BASE von der funktionalen Beschreibung des zu entwickelnden Systems durch die Anwendungsfälle ausgeht, nimmt Vogt die relevanten "Dinge" als Ausgangspunkt. Problematisch scheint dabei, daß er per se davon ausgeht, mit der Identifikation dieser "Dinge" immer die für die weitere Entwicklung passenden Klassen gefunden zu haben. Nur diese Identifikation der Untersuchungsgegenstände in den verschiedenen Phasen ermöglicht den Soll-Ist-Vergleich über den Begriffsverband.

6.7 Vergleich mit BASE

Die aufgeführten Ansätze zeigen die breiten Anwendungsmöglichkeiten von Formaler Begriffsanalyse in der Software-Entwicklung. Im Gegensatz zu BASE konzentrieren sich die aus der Informatik stammenden Verfahren auf die späten Phasen des Entwicklungszyklus. Dies bietet sich an, weil die Extraktion der formalen Kontexte dann in der Regel automatisch erfolgen kann, wenn dies auch zum Teil – wie in 6.2 angedeutet – erheblichen Aufwand bedeutet.

Der Ansatz von Vogt bewegt sich wie BASE auf dem weniger gefestigten Boden der Analyse. Seine Behandlung von "Dingen" als formale Gegenstände erscheint verwandt zu BASE, führt aber gerade auf das Problem der geeigneten Klassenauswahl, das BASE zu lösen hilft. Ein anderer Unterschied ist, daß bei ihm die Unterscheidung von aktiven und passiven Modellelementen (Anwendungsfällen und "Dingen" in BASE) nicht zur Unterteilung in formale Gegenstände und Merkmale herangezogen wird.

Diese Unterscheidung trifft auf den Ansatz zur Modularisierung von Altsystemen zu. Deshalb wurde zur Analyse der entstehenden Begriffsverbände in BASE die Zerlegung durch Blockrelationen als Technik aus diesem Ansatz übernommen. (Die Untersuchung von Konfigurationen durch Lindig und Snelting verwendet ebenfalls entsprechende Techniken.) Die Erzwingung von Implikationen, wie sie in BASE zur Behandlung von "*uses*"-/"*extends*"-Beziehungen und funktionalen Zerlegungen verwendet wird, findet auch in mehreren der geschilderten Ansätze Anwendung.

Vorstellbar erscheint, nach einer Analyse mit BASE den Ansatz von Godin et al. zur Strukturierung der Klassenhierarchie zu nutzen. Im Fall, daß bei der Analyse schon auf Muster ([Fow 99]) zurückgegriffen werden kann, spielt auch die Suche in entsprechenden Bibliotheken früh eine Rolle. Offen ist noch, ob und wie BASE helfen könnte, Muster zu identifizieren oder Schlagworte für eine Mustersuche zu liefern.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung der wichtigsten Ergebnisse

Für die Entwicklung von Software hat sich die Objektorientierung als anerkanntes Paradigma für alle Entwicklungsphasen etabliert. Durch die durchgängige Nutzung objektorientierter Modelle von der Analyse bis zu Implementierung vermeidet man die den strukturierten Methoden anhängenden Phasenbrüche. Das Rückgrat einer objektorientierten Entwicklung bildet das Klassenmodell. Deshalb ist eine zentrale Frage in der objektorientierten Analyse:

Welches sind die geeigneten Klassen für ein Systemmodell?

Was zeichnet ein gutes Klassenmodell aus? Martin Fowler postuliert als Modellierungsprinzip: "Models are not right or wrong; they are more or less useful." ([Fow 97], S. 2). Geeignete Klassen sollten für den gesamten Entwicklungsprozeß die adäquate Bausteine für die Erarbeitung der jeweiligen Systemmodelle bieten. Die Beurteilung der Eignung von Klassenkandidaten kann deshalb nur aufgrund eines tiefgehenden Verständnisses des Untersuchungsbereichs geschehen.

In 2.6 wurde untersucht, wie bekannte Methoden diese Fragestellung angehen. Die bisher gebotenen Unterstützungen des Entwicklers erscheinen für diese Entwurfstätigkeit unzureichend (siehe 2.6.2). Entweder bleibt die Wahl der Klassen seiner Intuition überlassen, die gebotenen Verfahren vernachlässigen die fachliche Auseinandersetzung mit dem Untersuchungsbereich oder sie sind nicht praktikabel. BASE bietet dem Entwickler eine Richtschnur für die Identifikation der Klassen. Eine automatische Klassenextraktion aus den Anforderungen an das zu entwikkelnde System kann es aber nicht leisten. Die manchmal zur Motivation des objektorientierten Vorgehens ins Feld geführte "Natürlichkeit des objektorientierten An-

satzes" ist nicht wirklich gegeben, weil die reale Welt sich in aller Regel nicht als eine Ansammlung von Objekten darstellt. Auch ein Analysemodell fußt auf Entwurfsentscheidungen, die festlegen, wie der Untersuchungsbereich gesehen werden soll.[1] Deshalb ist eine vollautomatische Klassenextraktion nicht möglich.

Die von BASE geleistete Unterstützung besteht in vier Punkten:

1. Komplementäre Darstellung von Daten- und funktionaler Sicht auf das zu erstellende System
2. Vorschläge für Komponentenzerlegungen
3. Funktionale Zerlegung mit Kontrolle der Datenabhängigkeiten
4. Überprüfungsfragen zur Modellvalidierung

Ein weitgehendes Verständnis des Untersuchungsbereichs ist nach oben wesentlich für eine erfolgreiche Modellbildung. Da man davon ausgehen muß, daß dies den Entwicklern zumindest fehlt, wenn sie ein System für einen noch unbekannten Anwendungsbereich entwickeln, kann ein solches Verständnis nur in der Kommunikation mit Fachexperten hergestellt werden. Erfolgversprechende Analyseverfahren müssen also den Diskurs zwischen diesen beiden Personengruppen effektiv unterstützen. Dieses Motiv taucht in den folgenden Darlegungen zu BASE immer wieder auf.

7.1.1 Darstellung von Daten- und funktionaler Sicht

Ziel eines Software-Entwicklungsprojekts ist die Erfüllung der Anforderungen der Auftraggeber. Die wichtigsten Anforderungen betreffen die Systemfunktionalität.[2] Nicht zuletzt deshalb hat Jacobsons Ansatz der Anwendungsfallanalyse weite Verbreitung gefunden. Vor die Betrachtung von objektorientierten Modellen setzt er eine rein funktional geprägte Untersuchung. Anwendungsfälle beschreiben die Funktionalität eines Systems von der Warte der außenstehenden Benutzer aus. Ihre Abläufe sind die typischen Systemdurchläufe, die zu analysieren sind. Auf dieser

1. Peter Schefe unterscheidet in [Sche 99] explizit zwischen *Modellen von einem Untersuchungsbereich* und *Modelle für ein zu realisierendes System* und führt aus, daß diese zwei verschiedenen Bedeutungen Software-Spezifikationen häufig irreführender Weise gleichzeitig zugesprochen werden. Auch ein Analysemodell ist ein Modell für die weitere Entwicklung, denn Zielrichtung der Modellierung ist die Systementwicklung und nicht eine "Erklärung der Welt" (die ohnehin nicht zu leisten ist).
2. Andere Modellierungsmethoden stellen andere Systemaspekte heraus. DEMO (Dynamic Essential Modelling of Organisatons, [RMD 98]) konzentriert sich etwa auf die Kommunikationsflüsse in einer Organisation. Diese werden auf Basis der Sprechakt-Theorie ([W-F 86]) untersucht und modelliert. Die Autoren betrachten Organisationen auf drei Ebenen. Im "essential level" wird dabei ebenfalls die funktionale Sicht betont und die untersuchte Organisation als ein System von Aktoren betrachtet, die in Geschäftsprozessen zusammenarbeiten. Im Gegensatz zu den Autoren halte ich eine stärkere Fokussierung auf die funktionalen Anforderungen auch bei den Analysemodellen in der Software-Entwicklung für angebracht.

Grundlage wird dann ein Klassenmodell erstellt.

Die zunächst wichtigsten Klassen für ein Modell sind Entitätsklassen im Sinne von Jacobson (siehe 2.5 auf Seite 54). Unter ihnen befinden sich in erster Linie solche, die Daten über "Dinge" des Untersuchungsbereichs verwalten. Deshalb wird in BASE die funktionale Systemuntersuchung über die Anwendungsfälle mit einer Betrachtung solcher "Dinge" verbunden.

Die in BASE aus diesem Ansatz heraus entwickelten Liniendiagramme stellen komplementär eine funktional geprägte und eine datenorientierte Sicht auf das zu entwickelnde System dar. Das liefert ein globales Bild der Datenabhängigkeiten des Anwendungsfallmodells. Die Bedeutung der untersuchten Modellelemente für das Gesamtsystem ist den Diagrammen direkt zu entnehmen. Von oben nach unten gelesen wird die top-down-Entwicklung der Systemfunktionalität von allgemeinen zu spezialisierten Funktionen nachvollzogen. Dual nimmt von unten nach oben die Bedeutung der betreffenden "Dinge" ab. Quasi in der Diagrammitte treffen sich spezielle Funktionen mit den in ihrem Rahmen interessanten "Dingen".

In diesem Sinn liefert das Diagramm eine Richtschnur für die Identifikation von Klassenkandidaten. In der Diagrammitte zusammentreffende Funktionen und "Dinge" bilden zusammengehörige Nuclei für eine bottom-up-Entwicklung von Klassen. Für eine top-down-Entwicklung von Klassen sind die "Dinge" von unten nach oben zu betrachten, weil die unten angeordneten "Dinge" im System eine tragende Rolle spielen. Weitere Betrachtungen unterstützen die Zuordnung von Attributen und Operationen zu solchen Klassenkandidaten

7.1.2 Komponentenzerlegungen

Die formalen Begriffe der in BASE erstellten Liniendiagramme gruppieren zusammengehörige Anwendungsfälle und "Dinge". Auf Basis von Blockrelationen ist es möglich, allgemeinere Zusammenfassungen zu betrachten.[1] So werden Vorschläge für Bausteinzerlegungen des zu erstellenden Systems erstellt. Bausteine können Klassen oder Komponenten sein. Im ersten Fall ist ein weiterer Beitrag zur Beantwortung der Ausgangsfrage geleistet. Meistens sind die entstehenden Systemzerlegungen aber gröber. Eine entsprechende Komponentenaufteilung liefert eine wertvolle Strukturierung des entstehenden Systems, aber auch des Untersuchungsbereichs. Seine Analyse kann nämlich nur sukzessive erfolgen. Eine Zerlegung in möglichst unabhängige Komponenten ist also hilfreich.

1. Diese Technik wurde für diesen Zweck in [L-S 97] vorgeschlagen, findet aber erst in BASE wirkliche Anwendungsbeispiele.

7.1.3 Funktionale Zerlegung in BASE

Die bei der Analyse schrittweise erstellten Modelle müssen von den Fachexperten in Hinblick auf ihre Eignung eingeschätzt werden. Sie müssen ihnen also – bis in ihre Details – verständlich sein. Klassenmodelle scheinen in diesem Zusammenhang nur bedingt geeignet. Deshalb betont BASE mit dem gewählten Anwendungsfall-Ansatz die funktionale Sicht und sieht funktionale Zerlegungen der Anwendungsfälle zur schrittweisen Modellverfeinerung vor. Dies erleichtert insbesondere den Anwendern das Verstehen des sich entwickelnden Modells, weil die funktionale Zerlegung nicht nur eine allgemein praktizierte Problemlösungsstrategie darstellt, sondern ihnen von der Gestaltung der Geschäftsprozesse ihres Unternehmens wohlbekannt ist. Außerdem bieten die funktionalen Zerlegungen eine gute Grundlage für die Beschreibung der Anwendungsfälle in Sequenzdiagrammen.

Die Datenabhängigkeiten der sich ergebenden Funktionen sind durch die Liniendiagramme explizit und damit kontrollierbar.

7.1.4 Überprüfungsfragen

Im Laufe der Entwicklung von BASE gewann die Unterstützung der Kommunikation zwischen Anwendern und Entwicklern an Gewicht. Als zusätzliche Repräsentationsform für die Modelldaten wurde deshalb die Möglichkeit der Generierung von Überprüfungsfragen in das Vorgehen eingebaut. Dies konnte mit dem innerhalb der Formalen Begriffsanalyse üblichen Ansatz über die Betrachtung von Implikationenbasen geschehen.

Ein großes Gewicht wurde auf die Durchführbarkeit des Verfahrens in realen Projekten gelegt. Das in Kapitel 5 beschriebene Werkzeug für BASE belegt, daß BASE einsetzbar ist, ohne daß Entwickler oder Anwender sich in die mathematischen Grundlagen einarbeiten müssen. Weiter bedeutet seine Anwendung keinen entscheidenden Mehraufwand gegenüber dem üblichen Vorgehen bei der Anwendungsfall-getriebenen Analyse. Um den Zusatzaufwand zu minimieren, wurde ein Algorithmus entwickelt, ein Liniendiagramm anhand eines anderen anzuordnen. Dies ist für die Erstellung von Liniendiagrammen zu funktional verfeinerten Anwendungsfallmodellen innerhalb von BASE von Bedeutung. Andere Anwendungen von Formaler Begriffsanalyse können gegebenenfalls auch von diesem Algorithmus profitieren, weil er lediglich auf den mathematischen Strukturen beruht. (Beschrieben ist der Algorithmus in Anhang B.5.3 auf Seite 313, seinen Effekt sieht man in Abbildung 5.11 auf Seite 202.)

Die Antwort auf die Ausgangsfrage fällt erwartungsgemäß nicht einfach aus:

> BASE stellt in einem gemeinsamen Modell komplementär eine funktionale und eine datenorientierte Sicht auf das zu erstellende System dar. Eine funktionale Zerlegung ist durchführbar, weil diese Darstellung die Datenabhängigkeiten global wiedergibt. Die Festlegung von Klassen kann zu einem späteren Zeitpunkt aufgrund eines besseren Systemverständnisses geschehen. Das mit BASE erstellte Modell liefert die Basis für
> - für die Festlegung von Klassen,
> - für Komponentenzerlegungen und
> - für Überprüfungsfragen an die Fachexperten.

7.2 Ausblick auf weiterführende Arbeiten

Vordringlich für weitere Arbeiten an BASE sind Tests der Methode in realen Projekten oder in Lehrpraktika. (Abschnitt 5.3 enthält Hinweise zur Durchführung und Bewertung solcher Tests.) Nur Projekterfahrungen können die Validität des Ansatzes zeigen. Auch die Berücksichtigung verschiedener möglicher Erweiterungen hängt von solchen Projekterfahrungen ab. Im folgenden werden einige Erweiterungsmöglichkeiten des Verfahrens angedacht.

7.2.1 Integration verschiedener Sichten

Nach einem Vortrag zu BASE wurde einmal der Einwand geäußert, daß die Ergebnisse des Verfahrens nicht invariant gegenüber den analysierenden Entwicklern und befragten Fachexperten sind. Klarerweise sind sie von deren Auswahl von Anwendungsfällen und "Dingen" abhängig. Nachdem oben klar gestellt wurde, daß auch in der Analyse schon Entwurfsentscheidungen (durch Entwickler und Fachexperten) getroffen werden, ist dies für jedes Modellierungsverfahren ein selbstverständliches Faktum. Im Falle von BASE bietet es sich sogar an, diesen Umstand produktiv zu nutzen. Vorstellbar ist, daß von einer Systementwicklung verschiedene Anwendergruppen betroffen sind, die im Entwicklungsprojekt verschiedene Ziele verfolgen. Ein Ansatz könnte sein, für die Gruppen zunächst separate Modelle zu erstellen, um so die Komplexität der Einzeldiagramme gering zu halten. Da-

nach müßten die verschiedenen Anwendungsfallmodelle zusammengeführt werden. Dies sollte in einer Diskussion zwischen den Gruppen geschehen. Eine Unterstützung durch BASE könnte so aussehen, daß eines der Liniendiagramme als Referenzmodell ausgewählt wird und alle anderen in ihrer graphischen Anordnung nach diesem ausgerichtet werden. Dazu könnte man prinzipiell Algorithmus B13 von Seite 320 benutzen. Ergänzend muß eine Behandlung von Synonymen und Homonymen erfolgen. Dazu könnte man zu gleichlautenden Bezeichnern in verschiedenen Liniendiagrammen die entsprechenden Merkmals- bzw. Gegenstandsbegriffe betrachten. Sind deren Umfänge bzw. -inhalte disjunkt oder zumindest sehr verschieden, ist nachzufragen, ob es sich um Homonyme handelt. Die Identifikation von Synonymen kann schrittweise vorgenommen werden, indem nach der Identifikation von zwei synonymen Bezeichnungen, die folgende Suche nach weiteren Synonymen auf die Hauptideale und -filter der entsprechenden Merkmals- bzw. Gegenstandsbegriffe oder sogar nur auf die entsprechenden Begriffsumfänge bzw. -inhalte eingeschränkt wird.

Mit leicht vergleichbaren Liniendiagrammen wäre eine wertvolle Diskussionsgrundlage für die Gruppen gegeben.

7.2.2 Vervollständigung der Implikationenbehandlung

In 4.3.2 und 4.4.1 wurde dargestellt, wie aus Implikationen des untersuchten formalen Kontexts Überprüfungsfragen generiert werden können. Abbildung 5.8 auf Seite 198 zeigt, wie solche Fragen präsentiert werden können. Zu jeder ihnen vorgelegten Frage entscheiden Entwickler und Fachexperten, ob der in ihr beschriebene Sachverhalt zutreffend ist. Stoßen sie auf eine fachlich nicht zutreffende Implikation, sind sie aufgefordert, das Anwendungsfallmodell entsprechend zu korrigieren oder zu ergänzen. In der vorliegenden Implementierung des Werkzeugs ist die Information über schon als zutreffend erkannte Implikationen verloren, sobald die Überprüfung abgebrochen wird. Bei einer neuerlichen Überprüfung werden unter Umständen die schon als zutreffend gekennzeichneten Sachverhalte noch einmal abgefragt. Auch die theoretische Aufbereitung von BASE sieht bisher keine Berücksichtigung der positiv beschiedenen Fragen für die Generierung weiterer Fragen vor.

Implikationen, die aus den als zutreffend erkannten ableitbar sind[1], sollten bei späteren Überprüfungen möglichst nicht mehr zur Generierung neuer Überprüfungsfragen herangezogen werden, um keine redundanten Fragen zu präsentieren. Deshalb müßte ein entsprechender Inferenzmechanismus eingebaut werden. Grundla-

1. Definition 26 auf Seite 94

ge dessen könnte folgender Umstand sein:
Die Gesamtheit aller respektierenden Mengen zu einer beliebigen Implikationenfamilie bildet ein Hüllensystem.[1] Über dieses Hüllensystem kann der Inferenzmechanismus implementiert werden. Werden bei einer späteren Überprüfung Implikationen erzeugt, die in diesem System gelten, sind sie durch die vorhergehenden Überprüfungen schon als zutreffend anerkannt. Entsprechende Überprüfungsfragen brauchen nicht mehr gestellt werden. Zu speichern wären die als zutreffend erkannten Implikationen (mit ihrer Prämisse und Konklusion) und nicht das Hüllensystem der sie respektierenden Mengen, weil sich die Grundmenge zu diesem System – die Menge der formalen Gegenstände oder Merkmale – in BASE in Folge von funktionalen Zerlegungen und Modellkorrekturen ändern kann. Für die Überprüfung der Gültigkeit einer einzelnen Implikation wird überdies nur ein Element des Hüllensystems benötigt, nämlich die Hülle des betrachteten (neuen) Pseudoinhalts bzw. -umfangs. Ist der nächstgrößere (neue) Begriffsumfang bzw. -inhalt auch in ihr enthalten, ist die (neue) Implikation schon aus den alten als zutreffend anerkannten ableitbar, sonst nicht.

Die durch Änderungen entstehenden formalen Kontexte können auch im Widerspruch zu einer als zutreffend abgespeicherten Implikation stehen. Also sind die abgespeicherten Implikationen auf ihre Gültigkeit in neuen Anwendungsfallmodellen zu überprüfen. Für die Überprüfung einer einzelnen Implikation ist wiederum nur die Betrachtung des kleinsten die Prämisse einschließenden (neuen) Begriffsinhalts bzw. -umfangs nötig. Ist die Konklusion ebenfalls in ihm enthalten, gilt die (alte) Implikation auch im neuen formalen Kontext, sonst steht sie zu dem neuen Modell in Widerspruch. Gegebenenfalls müssen die Analytiker auf einen Widerspruch hingewiesen werden. Sie können dann entscheiden, bei welcher der beiden Indizierungen der Anwendungsfälle ihnen ein Fehler unterlaufen ist.

7.2.3 Analysemetriken

BASE erlaubt schon sehr früh in einem Entwicklungsprojekt, in Form von Liniendiagrammen formal definierte Modelle zu erstellen. Auf Basis solcher wäre es möglich, Metriken zu definieren und basierend darauf Aufwandsschätzungen durchzuführen. Entsprechende Metriken würden den Grad der im Anwendungsfallmodell enthaltenen Datenabhängigkeiten quantifizieren. Denkbar wäre es, das Verhältnis von Knoten zu Kanten oder die zyklomatische Zahl nach Thomas McCabe ([Mcc 76]) zu betrachten. Außerdem könnte man wie Gregor Snelting bei

1. Siehe Hilfssatz 11 und Satz 10 auf Seite 100, der zugehörige Hüllenoperator ist in [G-W 96], 2.3, S. 80 angegeben. Er ist prinzipiell aufgebaut wie derjenige zur Berechnung der Pseudoinhalte bzw. Umfänge in Satz B4 auf Seite 287.

der Untersuchung von Konfigurationsstrukturen (siehe 6.4) bestimmen, wie viele Knoten entfernt werden müssen, damit das Liniendiagramm horizontal zerlegbar ist. (Horizontale Summanden bedeuten unabhängige Systemteile.) Für die entsprechenden horizontalen Summanden wären eine bestimmte Anzahl oder eine gewisse Größe – absolut (z.B. für Summandengröße 7±2-Regel nach [Mil 56]) oder relativ zur Diagrammgröße – anzupeilen.

7.2.4 Verbindung mit anderen Analyseverfahren

BASE betrachtet keine Beziehungen der betrachteten "Dinge" untereinander. Dies würde aber eine wertvolle Bereicherung der Analyse im Hinblick auf die Frage bringen, was als Klasse und was als Attribut modelliert werden soll.

Ein möglicher Ansatz hierfür ist *KCPM* (*Klagenfurt Conceptual Predesign Model*, [K-M 98a], [K-M 98b]). Die Autoren von KCPM sehen die Eignung objektorientierter Modelle für die Kommunikation zwischen Entwicklern und Anwendern ebenfalls kritisch. Weiter dient ihr Ansatz wie BASE dazu, die Entscheidung, was als Klasse modelliert werden soll, in der Analysephase eines Software-Entwicklungsprojekts zeitlich nach hinten zu verschieben, um sie auf der Basis eines tieferen Systemverständnisses treffen zu können. Deshalb schalten sie zwischen die Aufnahme der Systemanforderungen in natürlicher Sprache und der Erstellung eines objektorientierten Analysemodells ihr *Predesign-Modell*.

Die Elemente dieses Modells sind

- *"Ding"-Typen*,
- *Beziehungstypen* (Beziehungen zwischen "Dingen"),
- *Operationstypen* (Operationen ausgeführt von "Dingen"),
- *Ereignistypen* (Zustandsübergänge durch Ausführung von Operationen)
- *Geschäftsprozeßtypen* (Geschäftsprozesse als Abfolgen von Ereignissen).

Bei den Dingen und Operationen findet sich so eine Übereinstimmung mit den innerhalb von BASE betrachteten Modellelementen.[1] Entscheidender Unterschied ist, daß in BASE gefragt wird, welche "Dinge" von einem Anwendungsfall betroffen sind, während sich das KPCM an den "Dingen" orientiert und nach von ihnen ausgeführten Operationen sucht. KCPM verfolgt also keine strenge top-down-Strategie in der Untersuchung der Systemfunktionalität. Die entsprechenden Modellelemente werden im Unterschied zu BASE weitgehend durch sprachliche Analyse von Anforderungsdokumenten gewonnen.

1. BASE führt auch eine Analyse auf Typebene durch, der kompakteren Formulierung wegen wurde in der vorliegenden Darstellung auf den Zusatz "Typ" aber immer verzichtet.

Wichtiges Darstellungsmittel in KCPM sind Glossare in Tabellenform. In ihnen werden die Daten zu den Modellelementen festgehalten. Argumente für diese Präsentation der Inhalte sind die größere Platzökonomie gegenüber Diagrammen und die bessere Verständlichkeit durch die Anwender, die das Lesen von Bilanzen in ähnlicher Form gewohnt sind. Die Kommunikation zwischen den beiden Gruppen wird durch die entsprechenden Tabellen in der Weise gesteuert, daß anfangs freie Tabellenzellen sukzessive aufgefüllt werden müssen.

Aus dem Predesign-Modell kann halbautomatisch ein objektorientiertes Modell in UML abgeleitet werden. Die Identifikation von Klassen (und Attributen) wird in KCPM dabei anhand der Glossare zu "Dingen" und Beziehungen vorgenommen. Die benutzten Regeln unterteilen sich in Gesetze, die eine gewisse Modellierungsform vorschreiben, und Empfehlungen für eine Modellierungsmöglichkeit. Die wesentliche Idee ist, daß man ein semantisches Netz (siehe [Rei 91]) aus den "Ding"-Typen und Beziehungstypen betrachtet. Innere Knoten dieses Netzes sind dann die Klassenkandidaten. Die äußeren Netzknoten, die nur zu einem anderen Knoten eine Verbindung besitzen, enthalten Kandidaten für Attribute des Klassenkandidatens, mit dem sie verbunden sind.

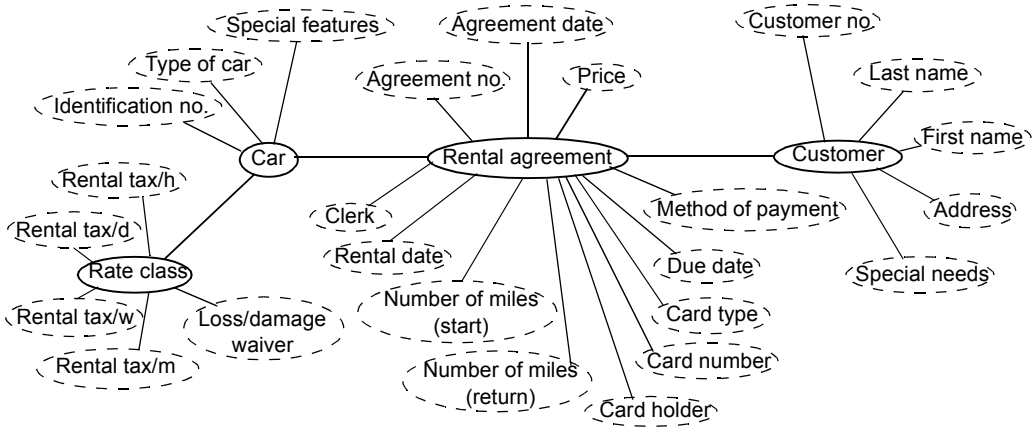


Abb. 7.1 Semantisches Netz aus "Dingen" und ihren Beziehungen

Das Beispiel in Abbildung 7.1 stammt von den Autoren von KCPM. Die "Dinge" *Rate class*, *Car*, *Rental agreement* und *Customer* sind die Klassenkandidaten. Ihre Attribute findet man in den mit ihnen verbundenen gestrichelt gezeichneten Knoten.

KCPM bietet so einfache Darstellungsmittel für die Kommunikation zwischen Anwendern und Entwicklern – insbesondere eine einfache Behandlung der datenorientierten Sicht auf ein System. Die Modellierung der funktionalen Sicht erscheint in BASE überzeugender. Deshalb wäre eine Kombination der beiden Ansätze er-

folgversprechend.

Als weitere Darstellungsform bieten sich John F. Sowas *Conceptual Graphs* an ([Sow 92]). Sie stellen Begriffe und deren Beziehungen untereinander dar. Möglich ist eine Übersetzung in die Prädikatenlogik erster Stufe. Ein Analyseansatz mit Conceptual Graphs würde an den Begriffen des Untersuchungsbereichs ansetzen, wäre also mit BASE nicht direkt kompatibel. Da aber Bemühungen bestehen, Formale Begriffsanalyse und Conceptual Graphs zu einer gemeinsamen begriffsbasierten Methode zu vereinigen (siehe [MSW 99]), soll dieser Ansatz hier nicht unerwähnt bleiben.

A Mathematische Details

Um die Darstellung im Kapitel 3 zur Formalen Begriffsanalyse zu straffen, sind im folgenden einige Beweise zu dort gemachten Aussagen zusammengefaßt, die einigermaßen lang oder aber nicht für das Verständnis der Argumentation in Kapitel 3, sondern aus technischen Gründen nötig sind. Nur für diese Beweise interessante Definitionen und Aussagen, sind in Kapitel 3 erst gar nicht wiedergegeben. Sie tragen hier Nummern beginnend mit einem 'A'. Dagegen sind die Nummern von referenzierten Aussagen in der Numerierung von Kapitel 3 aufgeführt – insbesondere ohne führendes 'A'.

Der Aufbau hier gleicht dem von Kapitel 3. In Abschnitt A.2 ist als Ergänzung zu Kapitel 3 gezeigt, daß die ursprüngliche Definition der Implikationenbasis durch Vincent Duquenne und J.L. Guigues mit der in Kapitel 3 benutzten Definition von Bernhard Ganter übereinstimmt. Abschnitt A.4 führt noch zwei Konzepte ein, die für BASE selber keine Rolle spielen, aber bei der Beschreibung des Ansatzes von Godin et al. in 6.1 und der Algorithmendarstellung in Anhang B.5.1 benutzt werden.

A.1 Ordnungstheoretische Grundlagen

A.1.1 Isomorphismen

Zwei in ihrer Ordnungsstruktur nicht unterscheidbare Mengen bezeichnet man als isomorph (Definition 13 auf Seite 74).

Wenn vollständige Verbände betrachtet werden, stehen drei Homomorphismen-Konzepte zur Auswahl, um Abbildungen strukturerhaltende Eigenschaften zuzusprechen: Ordnungshomomorphismus, Verbandshomomorphismus und vollständiger Verbandshomomorphismus (Definition 14 auf Seite 74).

Interessiert man sich jedoch nur für Isomorphismen, fallen alle Homomorphismen-Konzepte zusammen. Weiter ist die inverse Abbildung zu einem bijektiven (voll-

ständigen) Verbandshomomorphismus wieder ein (vollständiger) Verbandshomomorphismus. Deshalb sind in Definition 14 "Verbandsisomorphismen" gar nicht erst gesondert definiert.

Hilfssatz A1 (*Umkehrabbildung von Verbandshomomorphismen*)
Sind V und W zwei (vollständige) Verbände und $\varphi : V \to W$ ein bijektiver (vollständiger) $\wedge$-Morphismus bzw. $\vee$-Morphismus, so ist auch φ^{-1} ein (vollständiger) $\wedge$-Morphismus / $\vee$-Morphismus.
(Vergl. [Ern 82], 6.2 Lemma, S. 114)

Beweis: Betrachtet wird der Fall eines vollständigen $\wedge$-Morphismus. Die Behandlung von $\vee$-Morphismen ist analog und der Fall nicht-vollständiger Verbände ergibt sich durch Einschränkung auf zweielementige Mengen Y.
Sei $Y = \{w_t \mid t \in T\} \subseteq W$.
Dann ist

$$\varphi^{-1}\left(\wedge_W Y\right) = \varphi^{-1}\left(\wedge_W \{\varphi(\varphi^{-1}(w_t)) \mid t \in T\}\right)$$

$$= \varphi^{-1}\left(\varphi\left(\wedge_V \{\varphi^{-1}(w_t) \mid t \in T\}\right)\right) = \wedge_V \{\varphi^{-1}(w_t) \mid t \in T\}$$

$$= \wedge_V \varphi^{-1}(Y)$$

Also ist auch φ^{-1} ein vollständiger $\wedge$-Morphismus.

Satz 4 (*Verbandsisomorphismus ≅ Ordnungsisomorphismus*)
V und W seien zwei Verbände und $\varphi : V \to W$ eine Abbildung.

a) φ ist genau dann ein bijektiver Verbandshomomorphismus, wenn φ ein Ordnungsisomorphismus ist.
b) Sind V und W vollständige Verbände, so ist zusätzlich zu den beiden Eigenschaften aus a) äquivalent, daß φ ein bijektiver vollständiger Verbandshomomorphismus ist.

([Bir 67], II.3, Lemma 2, S. 24; [Ern 82], 6.5 Satz, S. 116)

Beweis: ([Grä 98], I.3, S. 20; vergl. [Ern 82], 6.4 Proposition, S. 116)
Zeige: Ist φ supremum-erhaltend, so auch ordnungserhaltend.
Seien also φ supremum-erhaltend und $x, y \in V$ mit $x \leq_V y$.
Dann ist nach Hilfssatz 1 auf Seite 67 $x \vee_V y = y$, also
$\varphi(x) \vee_W \varphi(y) = \varphi(x \vee_V y) = \varphi(y)$, d.h. $\varphi(x) \leq_W \varphi(y)$.
Insbesondere sind also (vollständige) Verbandshomomorphismen Ordnungshomomorphismen.
Nach Hilfssatz A1 sind damit bijektive (vollständige) Verbandshomomorphismen auch Ordnungsisomorphismen.
Seien nun V und W vollständig, φ ein Ordnungsisomorphismus und $X \subseteq V$.

Dann gilt $\wedge_V X \leq x$ für alle $x \in X$.

Also ist $\varphi(\wedge_V X) \leq \varphi(x) \;\; \forall x \in X$ und damit $\varphi(\wedge_V X) \leq \wedge_W \varphi(X)$.
Nach Voraussetzung ist auch φ^{-1} ordnungserhaltend.
Also gilt wegen $\wedge_W \varphi(X) \leq \varphi(x)$ für alle $x \in X$:
$\varphi^{-1}(\wedge_W \varphi(X)) \leq \varphi^{-1}(\varphi(x)) = x$
Daraus folgt $\varphi^{-1}(\wedge_W \varphi(X)) \leq \wedge_V X$.
Damit ist auch $\wedge_W \varphi(X) = \varphi(\varphi^{-1}(\wedge_W \varphi(X))) \leq \varphi(\wedge_V X)$.
Insgesamt gilt $\varphi(\wedge_V X) = \wedge_W \varphi(X)$.
D.h. φ ist ein vollständiger $\wedge$-Morphismus.
Durch die duale Betrachtung sieht man, daß φ auch ein vollständiger $\vee$-Morphismus ist, also ein vollständiger Verbandshomomorphismus.
Für nicht vollständige Verbände V und W schränke man die Betrachtung einfach auf zweielementige Mengen X ein.

A.1.2 Hüllensysteme

Die Inklusionsordnung auf Mengenfamilien spielt in der Formalen Begriffsanalyse eine herausragende Rolle. Die vollständigen Verbände mit dieser Ordnung sind gerade die Hüllensysteme (Definition 16 auf Seite 77). Diese können durch Hüllenoperatoren beschrieben werden (Definition 17 auf Seite 77). Das wird in Algorithmen zur Formalen Begriffsanalyse ausgenutzt (siehe Anhang B.1 auf Seite 276).

Hilfssatz 4 (*Hüllensystem ↔ Hüllenoperator*)

Sei X eine beliebige Menge.
Ist $\mathcal{U} \subseteq \mathfrak{P}(X)$ ein Hüllensystem auf X, so definiert
$\varphi_{\mathcal{U}}(V) := \bigcap\{A \in \mathcal{U} \mid V \subseteq A\}$ für $V \subseteq X$ einen Hüllenoperator auf X.
Ist umgekehrt $\varphi : \mathfrak{P}(X) \to \mathfrak{P}(X)$ ein Hüllenoperator auf X, so ist das System seiner Hüllen $\mathcal{U}_\varphi := \{\varphi(Z) \mid Z \subseteq X\}$ ein Hüllensystem auf X.
Weiter ist $\varphi_{\mathcal{U}_\varphi} = \varphi$ und $\mathcal{U}_{\varphi_{\mathcal{U}}} = \mathcal{U}$.

Beweis: ([G-W 96], 0.3, Satz 1, S. 9)

Seien also $\mathcal{U} \subseteq \mathfrak{P}(X)$ ein Hüllensystem auf X und $V, W \subseteq X$.
Gilt $V \subseteq W$, so ist für alle $A \in \mathcal{U}$ mit $W \subseteq A$ auch $V \subseteq A$ und so
$\varphi_{\mathcal{U}}(V) = \bigcap\{A \in \mathcal{U} \mid V \subseteq A\} \subseteq \bigcap\{A \in \mathcal{U} \mid W \subseteq A\} = \varphi_{\mathcal{U}}(W)$.
D.h. $\varphi_{\mathcal{U}}$ ist monoton.
Weiter ist $V \subseteq \bigcap\{A \in \mathcal{U} \mid V \subseteq A\} = \varphi_{\mathcal{U}}(V)$, also $\varphi_{\mathcal{U}}$ extensiv.
Damit ist für alle $A \in \mathcal{U}$ mit $\varphi_{\mathcal{U}}(V) \subseteq A$ auch $V \subseteq A$.
Andersherum gilt mit $V \subseteq A$ auch $\varphi_{\mathcal{U}}(V) = \bigcap\{A \in \mathcal{U} \mid V \subseteq A\} \subseteq A$.
Das bedeutet
$\varphi_{\mathcal{U}}(V) = \bigcap\{A \in \mathcal{U} \mid V \subseteq A\} = \bigcap\{A \in \mathcal{U} \mid \varphi_{\mathcal{U}}(V) \subseteq A\} = \varphi_{\mathcal{U}}(\varphi_{\mathcal{U}}(V))$

Also ist $\varphi_{\mathcal{U}}$ idempotent und insgesamt ein Hüllenoperator.
Sei nun umgekehrt $\varphi : \mathfrak{P}(X) \to \mathfrak{P}(X)$ ein Hüllenoperator auf X.
Dann gilt für $\mathfrak{X} \subseteq \mathcal{U}_\varphi$ wegen der Extensität von φ sicher $\bigcap \mathfrak{X} \subseteq \varphi(\bigcap \mathfrak{X})$.
Sei $V \in \mathfrak{X}$.
Weil $V \in \mathfrak{X} \subseteq \mathcal{U}_\varphi = \{\varphi(Z) \mid Z \subseteq X\}$, gibt es ein $W \subseteq X$ mit $V = \varphi(W)$,
also $\varphi(V) = \varphi(\varphi(W)) = \varphi(W) = V$.
Dann ist $\bigcap \mathfrak{X} \subseteq V$ und nach der Monotonie von φ damit
$\varphi(\bigcap \mathfrak{X}) \subseteq \varphi(V) = V$.
Insgesamt folgt also auch die andere Inklusion $\varphi(\bigcap \mathfrak{X}) \subseteq \bigcap \mathfrak{X}$.
Somit ist $\bigcap \mathfrak{X}$ eine Hülle von φ und $\mathcal{U}_\varphi$ ein Hüllensystem.
Weiter gilt für φ und $V \subseteq X$:

$$\begin{aligned}\varphi_{\mathcal{U}_\varphi}(V) &= \bigcap\{A \in \mathcal{U}_\varphi \mid V \subseteq A\}\\ &= \bigcap\{A \in \{\varphi(Z) \mid Z \subseteq X\} \mid V \subseteq A\}\\ &= \bigcap\{\varphi(Z) \mid Z \subseteq X \quad \text{und} \quad V \subseteq A\}\\ &\subseteq \varphi(V)\end{aligned}$$

Ist $A \in \mathcal{U}_\varphi$, so ist nach oben $\varphi(A) = A$.
Für $V \subseteq A$ gilt damit nach der Extensität von φ, daß $\varphi(V) \subseteq \varphi(A) = A$.
Also ist $\varphi(V) \subseteq \bigcap\{A \in \mathcal{U}_\varphi \mid V \subseteq A\} = \varphi_{\mathcal{U}_\varphi}(V)$.
Insgesamt folgt die behauptete Gleichheit $\varphi_{\mathcal{U}_\varphi} = \varphi$.
Für ein Hüllensystem $\mathcal{U}$ auf X und $V \in \mathcal{U}_{\varphi_{\mathcal{U}}}$ ist analog zu oben $\varphi_{\mathcal{U}}(V) = V$,
also $V = \bigcap\{A \in \mathcal{U} \mid V \subseteq A\} \in \mathcal{U}$.
D.h. $\mathcal{U}_{\varphi_{\mathcal{U}}} \subseteq \mathcal{U}$.
Ist umgekehrt $V \in \mathcal{U}$, so ist $\varphi_{\mathcal{U}}(V) = \bigcap\{A \in \mathcal{U} \mid V \subseteq A\} \subseteq V$ und nach der Extensität von $\varphi_{\mathcal{U}}$ damit $\varphi_{\mathcal{U}}(V) = V$, d.h. $V \in \mathcal{U}_{\varphi_{\mathcal{U}}}$.
Das bedeutet $\mathcal{U}_{\varphi_{\mathcal{U}}} \supseteq \mathcal{U}$.
Insgesamt gilt die behauptete Gleichung $\mathcal{U}_{\varphi_{\mathcal{U}}} = \mathcal{U}$.

A.2 Implikationen

Wie in 3.4.1 auf Seite 100 erwähnt, definieren Vincent Duquenne und J.L. Guigues saturierte Prämissen anders als Bernhard Ganter und Rudolf Wille. Im folgenden ist aufgezeigt, daß beide Vorgehensweisen zu der selben Implikationenbasis führen.

Saturierte Prämissen werden betrachtet, um Redundanzen aufgrund von Hilfssatz 12 auf Seite 99 zu vermeiden. D.h. man hat dafür zu sorgen, daß Prämisse und Konklusion einer Basisimplikation nicht als Vereinigung von Prämissen bzw.

Konklusionen anderer Basisimplikationen dargestellt werden können. Dazu wählt man nur solche informativen Implikationen mit maximaler Konklusion aus, bei denen echte Teilmengen der Prämisse nichts über die Gesamtprämisse hinausgehendes implizieren. Die Konstruktion dieser Prämissen läuft bei Duquenne und Guigues über mehrere Stufen.

Definition A1 (*DG-Saturierte Prämisse*)

Sei X eine Menge, $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen und $V \subseteq X$.
Dann setze $V^\circ := V \cup \bigcup \{k_\mathfrak{R}(U) \mid U \subseteq V, k_\mathfrak{R}(U) \neq k_\mathfrak{R}(V)\}$.
Weiter sei $p_\mathfrak{R}(V) := \bigcup_{n \in \mathbb{N}_0} V_n$ mit $V_0 := V$, $V_{n+1} := V_n^\circ$ für $n \in \mathbb{N}_0$.
Ist $V = p_\mathfrak{R}(V)$, so heißt V eine *DG-saturierte Prämisse*.
(Vergl. [G-D 86], S.12/13 und [Duq 87], S. 223)

Natürlich sind auch schon V° und V_n von $\mathfrak{R}$ abhängig. Dies wird der einfacheren Notation halber durch die Bezeichnungen nicht ausgedrückt, sie spielen aber nur als "Zwischenstationen" in Beweisen eine Rolle.
Die wichtigsten Eigenschaften der in Definition A1 definierten Abbildungen $^\circ$ und $p_\mathfrak{R}$ sind in folgendem Hilfssatz zusammengefaßt.

Hilfssatz A2 (*Eigenschaften von $^\circ$ und $p_\mathfrak{R}$*)

Sei X eine Menge, $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen und $V \subseteq X$.
Dann ist

a) $V \subseteq V^\circ \subseteq k_\mathfrak{R}(V)$

b) $k_\mathfrak{R}(V^\circ) = k_\mathfrak{R}(V)$

c) $V \subseteq p_\mathfrak{R}(V) \subseteq k_\mathfrak{R}(V)$

d) $k_\mathfrak{R}(p_\mathfrak{R}(V)) = k_\mathfrak{R}(V)$

e) Ist X endlich, so ist
$p_\mathfrak{R}(V) = V \cup \bigcup \{k_\mathfrak{R}(U) \mid U \subseteq p_\mathfrak{R}(V), k_\mathfrak{R}(U) \neq k_\mathfrak{R}(V)\}$.

f) Ist X endlich, so ist $p_\mathfrak{R}(p_\mathfrak{R}(V)) = p_\mathfrak{R}(V)$.

Beweis: (Vergl. [G-D 86], S. 13)

a) $V \subseteq V^\circ$ ist klar nach Konstruktion und, da $k_\mathfrak{R}(U) \subseteq k_\mathfrak{R}(V)$ für alle $U \subseteq V$, ist auch $V^\circ \subseteq k_\mathfrak{R}(V)$.

b) Nach a) und Definition von $k_\mathfrak{R}$ ist $k_\mathfrak{R}(V^\circ) \supseteq k_\mathfrak{R}(V)$.
Ist andererseits $V \subseteq R$ für ein $R \in \mathfrak{R}$, so ist nach a) und Definition von $k_\mathfrak{R}$ auch $V^\circ \subseteq k_\mathfrak{R}(V) \subseteq R$, d.h.
$k_\mathfrak{R}(V^\circ) = \bigcap \{R \in \mathfrak{R} \mid V^\circ \subseteq R\} \subseteq \bigcap \{R \in \mathfrak{R} \mid V \subseteq R\} = k_\mathfrak{R}(V)$.

Im folgenden betrachte $(V_n)_{n \in \mathbb{N}_0}$ wie in Definition A1.

c) $V \subseteq p_\mathfrak{R}(V)$ ist klar nach Konstruktion und $p_\mathfrak{R}(V) \subseteq k_\mathfrak{R}(V)$ gilt, weil nach a) und b) induktiv $V_n \subseteq k_\mathfrak{R}(V)$ für alle $n \in \mathbb{N}_0$ gilt.

d) Nach b), c) und Definition von $p_{\mathfrak{K}}$ ist
$k_{\mathfrak{K}}(V) = k_{\mathfrak{K}}(V^{\circ}) \subseteq k_{\mathfrak{K}}(p_{\mathfrak{K}}(V)) \subseteq (k_{\mathfrak{K}}(V))$.
Also ist $k_{\mathfrak{K}}(p_{\mathfrak{K}}(V)) = k_{\mathfrak{K}}(V)$.

e) Für jedes $n \in \mathbb{N}_0$ ist $V_n \subseteq p_{\mathfrak{K}}(V)$.
Zeige: $V_n \subseteq V \cup \bigcup\{k_{\mathfrak{K}}(U) \mid U \subseteq p_{\mathfrak{K}}(V), k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V)\}$ für alle $n \in \mathbb{N}_0$.
Für $V_0 = V$ ist das klar.
Ist aber für ein $n \in \mathbb{N}_0$ gezeigt, daß
$V_n \subseteq V \cup \bigcup\{k_{\mathfrak{K}}(U) \mid U \subseteq p_{\mathfrak{K}}(V), k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V)\}$, so folgt auch
$V_{n+1} \subseteq V \cup \bigcup\{k_{\mathfrak{K}}(U) \mid U \subseteq p_{\mathfrak{K}}(V), k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V)\}$, denn

$$\begin{aligned} V_{n+1} &= V_n^{\circ} \\ &= V_n \cup \bigcup\{k_{\mathfrak{K}}(U) \mid U \subseteq V_n, k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V_n)\} \\ &= V_n \cup \bigcup\{k_{\mathfrak{K}}(U) \mid U \subseteq V_n, k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V)\} \end{aligned}$$

(Letzte Gleichung folgt induktiv aus b)).
Damit gilt $V_n \subseteq V \cup \bigcup\{k_{\mathfrak{K}}(U) \mid U \subseteq p_{\mathfrak{K}}(V), k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V)\}$ für alle $n \in \mathbb{N}_0$.
Insgesamt ist $p_{\mathfrak{K}}(V) \subseteq V \cup \bigcup\{k_{\mathfrak{K}}(U) \mid U \subseteq p_{\mathfrak{K}}(V), k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V)\}$.
Jetzt ist noch die andere Inklusion zu zeigen.
Weil X endlich ist, gibt es ein $r \in \mathbb{N}_0$ mit $V_n = V_r$ für alle $n \geq r$.
Damit ist $p_{\mathfrak{K}}(V) = V_r$.
Ist $U \subseteq p_{\mathfrak{K}}(V)$, so daß $k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V)$, so gilt $U \subsetneq V_r$.
Damit ist $k_{\mathfrak{K}}(U) \subseteq V_{r+1} = V_r = p_{\mathfrak{K}}(V)$.
Somit ist $V \cup \bigcup\{k_{\mathfrak{K}}(U) \mid U \subseteq p_{\mathfrak{K}}(V), k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V)\} \subseteq p_{\mathfrak{K}}(V)$.

f) Nach e) ist

$$\begin{aligned} p_{\mathfrak{K}}(V)^{\circ} &= p_{\mathfrak{K}}(V) \cup \bigcup\{k_{\mathfrak{K}}(U) \mid U \subseteq p_{\mathfrak{K}}(V), k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V)\} \\ &= V \cup \bigcup\{k_{\mathfrak{K}}(U) \mid U \subseteq p_{\mathfrak{K}}(V), k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V)\} \\ &\cup (\bigcup\{k_{\mathfrak{K}}(U) \mid U \subseteq p_{\mathfrak{K}}(V), k_{\mathfrak{K}}(U) \neq k_{\mathfrak{K}}(V)\}) \\ &= p_{\mathfrak{K}}(V) \end{aligned}$$

Das bedeutet induktiv $(p_{\mathfrak{K}}(V))_n = p_{\mathfrak{K}}(V)$ für alle $n \in \mathbb{N}_0$.
Also ist $p_{\mathfrak{K}}(p_{\mathfrak{K}}(V)) = p_{\mathfrak{K}}(V)$.

Nimmt man die informativen Implikationen mit maximaler Konklusion und DG-saturierter Prämisse, wobei man jeweils von der minimalen Menge ausgeht, um diese Konklusion und Prämisse zu bilden, so bekommt man eine Basis der Implikationen auf einer endlichen Menge.

Satz A1 (*Duquenne-Guigues-Basis*)

Sei X eine **endliche** Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Dann ist

$$\mathfrak{DGO}_{\mathfrak{R}}(X) := \{p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V) \mid V \subseteq X, p_{\mathfrak{R}}(V) \neq k_{\mathfrak{R}}(V), \\ (U \subseteq p_{\mathfrak{R}}(V), k_{\mathfrak{R}}(U) = k_{\mathfrak{R}}(V)) \Rightarrow p_{\mathfrak{R}}(U) = p_{\mathfrak{R}}(V)\}$$

eine Basis der in $\mathfrak{R}$ gültigen Implikationen.

Beweis: (Vergl. [Duq 87], Theorem 1, S. 223)

Für alle $V \subseteq X$ ist nach Hilfssatz A2 $k_{\mathfrak{R}}(V) = k_{\mathfrak{R}}(p_{\mathfrak{R}}(V))$, also gilt nach Hilfssatz 10 auf Seite 95 die Implikation $p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)$ in $\mathfrak{R}$.
D.h. $\mathfrak{DGO}_{\mathfrak{R}}(X)$ gilt in $\mathfrak{R}$.
Zeige, daß jede informative Implikation mit in $\mathfrak{R}$ maximaler Konklusion aus $\mathfrak{DGO}_{\mathfrak{R}}(X)$ folgt.

Sei also $W \subseteq X$ mit $W \neq k_{\mathfrak{R}}(W)$ und betrachte $W \to k_{\mathfrak{R}}(W)$.
Nach Hilfssatz 8 auf Seite 95 folgt $W \to k_{\mathfrak{R}}(W)$ aus $W \to p_{\mathfrak{R}}(W)$ und $p_{\mathfrak{R}}(W) \to k_{\mathfrak{R}}(W)$.
Die zweite Implikation $p_{\mathfrak{R}}(W) \to k_{\mathfrak{R}}(W)$ ist von der in $\mathfrak{DGO}_{\mathfrak{R}}(X)$ beschriebenen Form, nur eventuell nicht informativ oder nicht minimal.

Ist $p_{\mathfrak{R}}(W) \to k_{\mathfrak{R}}(W)$ nicht informativ, folgt sie nach Hilfssatz 6 auf Seite 93 aus jeder Implikationenfamilie.
Für den Fall, daß $p_{\mathfrak{R}}(W) \to k_{\mathfrak{R}}(W)$ informativ ist ($p_{\mathfrak{R}}(W) \neq k_{\mathfrak{R}}(W)$), wird ein Element aus $\mathfrak{DGO}_{\mathfrak{R}}(X)$ konstruiert, aus dem $p_{\mathfrak{R}}(W) \to k_{\mathfrak{R}}(W)$ folgt.
Setze dazu $W_0 := p_{\mathfrak{R}}(W)$.
Bis kein solches mehr existiert, finde zu W_n mit $n \in \mathbb{N}_0$ ein $W_{n+1} \subsetneq W_n$, so daß $W_{n+1} = p_{\mathfrak{R}}(W_{n+1})$ und $k_{\mathfrak{R}}(W_{n+1}) = k_{\mathfrak{R}}(W_n)$.
Da X endlich ist, bricht diese Konstruktion nach endlich vielen Schritten ab.
Sei W_r mit $r \in \mathbb{N}_0$ die so zuletzt konstruierte Menge.
Für diese gilt nach Konstruktion $W_r \subseteq p_{\mathfrak{R}}(W)$ und $k_{\mathfrak{R}}(W_r) = k_{\mathfrak{R}}(p_{\mathfrak{R}}(W)) = k_{\mathfrak{R}}(W)$.
Nach Hilfssatz 9 auf Seite 95 folgt also $p_{\mathfrak{R}}(W) \to k_{\mathfrak{R}}(W)$ aus $W_r \to k_{\mathfrak{R}}(W_r)$.
Sei nun $Y \subseteq p_{\mathfrak{R}}(W_r) = W_r$ mit $k_{\mathfrak{R}}(Y) = k_{\mathfrak{R}}(W_r)$.
Dann ist

$$\begin{aligned} Y^\circ &= Y \cup \bigcup \{k_{\mathfrak{R}}(U) \mid U \subseteq Y, k_{\mathfrak{R}}(U) \neq k_{\mathfrak{R}}(Y)\} \\ &= Y \cup \bigcup \{k_{\mathfrak{R}}(U) \mid U \subseteq Y, k_{\mathfrak{R}}((U) \subsetneq k_{\mathfrak{R}}(Y))\} \\ &\subseteq W_r \cup \bigcup \{k_{\mathfrak{R}}(U) \mid U \subseteq W_r, k_{\mathfrak{R}}(U) \subsetneq k_{\mathfrak{R}}(W_r)\} \\ &= W_r^\circ = p_{\mathfrak{R}}(W_r) = W_r \end{aligned}$$

Also gilt induktiv $p_{\mathfrak{R}}(Y) \subseteq p_{\mathfrak{R}}(W_r) = W_r$.

Dann muß aber schon $p_{\mathfrak{R}}(Y) = p_{\mathfrak{R}}(W_r)$ gelten, weil sonst nach Hilfssatz A2.f) auf Seite 257 die Folge der W_n mit $W_{r+1} := p_{\mathfrak{R}}(Y)$ hätte fortgesetzt werden können.

Das bedeutet, daß $(W_r \to k_{\mathfrak{R}}(W_r)) \in \mathfrak{DQO}_{\mathfrak{R}}(X)$.

Nun zur Implikation $W \to p_{\mathfrak{R}}(W)$.

Hilfssatz A2.e) besagt:

$$p_{\mathfrak{R}}(W) = W \cup \bigcup \{k_{\mathfrak{R}}(U) \mid U \subseteq p_{\mathfrak{R}}(W), k_{\mathfrak{R}}(U) \neq k_{\mathfrak{R}}(W)\}.$$

Laut Hilfssatz 6 auf Seite 93 folgt $W \to W$ aus jeder Implikationenfamilie.

Damit folgt $W \to p_{\mathfrak{R}}(W)$ nach Hilfssatz 12 auf Seite 99 aus $\{U \to k_{\mathfrak{R}}(U) \mid U \subseteq p_{\mathfrak{R}}(W), k_{\mathfrak{R}}(U) \neq k_{\mathfrak{R}}(W)\}$.

Die nicht informativen unter diesen Implikationen $U \to k_{\mathfrak{R}}(U)$ folgen nach Hilfssatz 6 aus jeder Implikationenfamilie.

Nach Hilfssatz 7 auf Seite 94 folgt also $W \to p_{\mathfrak{R}}(W)$ aus $\{U \to k_{\mathfrak{R}}(U) \mid U \subseteq p_{\mathfrak{R}}(W), U \neq k_{\mathfrak{R}}(U) \neq k_{\mathfrak{R}}(W)\}$.

Wegen $U \subseteq p_{\mathfrak{R}}(W)$ ist $k_{\mathfrak{R}}(U) \subseteq k_{\mathfrak{R}}(p_{\mathfrak{R}}(W)) = k_{\mathfrak{R}}(W)$, also $k_{\mathfrak{R}}(U) \subsetneqq k_{\mathfrak{R}}(W)$.

Die Konklusionen der neu betrachteten Implikationen sind also echt kleiner als die ursprüngliche.

Dieser rekursive Abstieg in den Konklusionen findet in endlichen Schritten ein Ende, weil X endlich ist.

Gibt es am Ende für eine informative Implikation $U \to k_{\mathfrak{R}}(U)$ kein $Y \subseteq p_{\mathfrak{R}}(U)$ mit $k_{\mathfrak{R}}(Y) \neq k_{\mathfrak{R}}(U)$, so ist nach Hilfssatz A2.e) auf Seite 257 $p_{\mathfrak{R}}(U) = U$.

Also ist $U \to k_{\mathfrak{R}}(U)$ von der Form $p_{\mathfrak{R}}(U) \to k_{\mathfrak{R}}(U)$ und informativ.

Genau wie oben für $p_{\mathfrak{R}}(W) \to k_{\mathfrak{R}}(W)$ gezeigt, folgt also $U \to k_{\mathfrak{R}}(U)$ aus einer Implikation aus $\mathfrak{DQO}_{\mathfrak{R}}(X)$.

Der rekursive Abstieg von $W \to p_{\mathfrak{R}}(W)$ aus endet also ebenfalls bei Implikationen aus $\mathfrak{DQO}_{\mathfrak{R}}(X)$.

$W \to k_{\mathfrak{R}}(W)$ folgt also aus $\mathfrak{DQO}_{\mathfrak{R}}(X)$.

Jede informative Implikation mit in $\mathfrak{R}$ maximaler Konklusion folgt somit aus $\mathfrak{DQO}_{\mathfrak{R}}(X)$.

Nach Folgerung 9 auf Seite 98 und Hilfssatz 7 auf Seite 94 folgt damit jede in $\mathfrak{R}$ gültige Implikation aus $\mathfrak{DQO}_{\mathfrak{R}}(X)$, d.h. $\mathfrak{DQO}_{\mathfrak{R}}(X)$ ist für $\mathfrak{R}$ vollständig.

Zeige noch, daß $\mathfrak{DQO}_{\mathfrak{R}}(X)$ nicht redundant ist.

Sei also $(p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)) \in \mathfrak{DQO}_{\mathfrak{R}}(X)$.

Dann gilt $p_{\mathfrak{R}}(V) \subsetneqq k_{\mathfrak{R}}(V)$, also wird $p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)$ *NICHT* von $p_{\mathfrak{R}}(V)$

respektiert.
Sei $(p_{\mathfrak{R}}(U) \to k_{\mathfrak{R}}(U)) \in \mathfrak{DQO}_{\mathfrak{R}}(X)$ mit $p_{\mathfrak{R}}(U) \subsetneq p_{\mathfrak{R}}(V)$.
Dann ist $k_{\mathfrak{R}}(p_{\mathfrak{R}}(U)) \neq k_{\mathfrak{R}}(p_{\mathfrak{R}}(V)) = k_{\mathfrak{R}}(V)$, weil sonst nach der Minimalität von $p_{\mathfrak{R}}(V)$ auch $p_{\mathfrak{R}}(U) = p_{\mathfrak{R}}(V)$ wäre.
Nach Hilfssatz A2.e) auf Seite 257 folgt $k_{\mathfrak{R}}(U) = k_{\mathfrak{R}}(p_{\mathfrak{R}}(U)) \subseteq p_{\mathfrak{R}}(V)$.
D.h. $p_{\mathfrak{R}}(V)$ respektiert $p_{\mathfrak{R}}(U) \to k_{\mathfrak{R}}(U)$.
Insgesamt respektiert $p_{\mathfrak{R}}(V)$ ganz $\mathfrak{DQO}_{\mathfrak{R}}(X) \setminus \{p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)\}$.
Also folgt $p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)$ nicht aus $\mathfrak{DQO}_{\mathfrak{R}}(X) \setminus \{p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)\}$.
Da $p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)$ beliebig gewählt war, ist somit $\mathfrak{DQO}_{\mathfrak{R}}(X)$ nicht redundant.

Insgesamt ist gezeigt, daß $\mathfrak{DQO}_{\mathfrak{R}}(X)$ eine Basis der in $\mathfrak{R}$ gültigen Implikationen ist.

Vincent Duquenne bezeichnet die Duquenne-Guigues-Basis als kanonische saturierte Basis ([Duq 87], S. 225). In welchem Sinn sie kanonisch ist, erläutert der folgende Satz.

Hilfssatz A3 (*Kanonische saturierte Basis*)

Sei X eine endliche Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Ist eine Implikationenfamilie $\mathfrak{I} \subseteq \mathfrak{P}(X)^2$ für $\mathfrak{R}$ vollständig, so gibt es zu jeder Implikation $p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)$ aus $\mathfrak{DQO}_{\mathfrak{R}}(X)$ eine Implikation $(U \to W) \in \mathfrak{I}$ mit $U \subseteq p_{\mathfrak{R}}(V)$ und $k_{\mathfrak{R}}(U) = k_{\mathfrak{R}}(V)$.
Haben alle Implikationen aus $\mathfrak{I}$ eine DG-saturierte Prämisse, so gilt sogar $U = p_{\mathfrak{R}}(V)$.

Beweis: (Vergl. [G-W 96], 2.3, Hilfssatz 25, S. 84)
Sei also $(p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)) \in \mathfrak{DQO}_{\mathfrak{R}}(X)$ beliebig.
Nach Satz A1 gilt dann $p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)$ in $\mathfrak{R}$.
Nach Voraussetzung folgt also $p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)$ aus $\mathfrak{I}$.
Wegen $p_{\mathfrak{R}}(V) \neq k_{\mathfrak{R}}(V)$ respektiert $p_{\mathfrak{R}}(V)$ die Implikation $p_{\mathfrak{R}}(V) \to k_{\mathfrak{R}}(V)$ nicht.
Damit kann $p_{\mathfrak{R}}(V)$ auch nicht $\mathfrak{I}$ respektieren.
D.h. es gibt $(U \to W) \in \mathfrak{I}$ mit $U \subseteq p_{\mathfrak{R}}(V)$ und $W \not\subseteq p_{\mathfrak{R}}(V)$.
Wäre $k_{\mathfrak{R}}(U) \neq k_{\mathfrak{R}}(V)$, so wäre nach Hilfssatz 10 auf Seite 95 und Hilfssatz A2.e) auf Seite 257 $W \subseteq k_{\mathfrak{R}}(U) \subseteq p_{\mathfrak{R}}(V)$.
Also ist $k_{\mathfrak{R}}(U) = k_{\mathfrak{R}}(V)$.
Wegen der Minimalität von V folgt $p_{\mathfrak{R}}(U) = p_{\mathfrak{R}}(V)$.
Ist U DG-saturiert, d.h. $U = p_{\mathfrak{R}}(U)$, ist also $U = p_{\mathfrak{R}}(V)$.

In diesem Sinne ist also $\mathfrak{DQO}_{\mathfrak{R}}(X)$ genau wie $\mathfrak{DQ}_{\mathfrak{R}}(X)$ minimal (vergl. Satz 12 auf Seite 102). Nun folgt auch die Gleichheit der beiden Basen.

Satz A2 (*Duquenne-Guigues-Basis für formale Kontexte*)

Sei X eine endliche Menge und $\mathfrak{R} \subseteq \mathfrak{P}(X)$ ein System von Teilmengen.
Dann ist $\mathfrak{DOO}_{\mathfrak{R}}(X) = \mathfrak{DO}_{\mathfrak{R}}(X)$.
([G-W 96], 2.3, S. 83)

Beweis: Sei also $V \subseteq X$ eine saturierte Prämisse (nach Definition 29 auf Seite 99) und damit $(V \to k_{\mathfrak{R}}(V)) \in \mathfrak{DO}_{\mathfrak{R}}(X)$.
Weil $\mathfrak{DOO}_{\mathfrak{R}}(X)$ nach Satz A1 auf Seite 259 vollständig ist, gibt es nach Satz 12 auf Seite 102 eine Implikation $(p_{\mathfrak{R}}(U) \to k_{\mathfrak{R}}(U)) \in \mathfrak{DOO}_{\mathfrak{R}}(X)$ mit $p_{\mathfrak{R}}(U) \subseteq V$ und $k_{\mathfrak{R}}(U) = k_{\mathfrak{R}}(V)$.
Weil $\mathfrak{DO}_{\mathfrak{R}}(X)$ nach Satz 11 auf Seite 100 vollständig ist, gibt es nach Hilfssatz A3 ein $(W \to k_{\mathfrak{R}}(W)) \in \mathfrak{DO}_{\mathfrak{R}}(X)$ mit $W \subseteq p_{\mathfrak{R}}(U)$ und $k_{\mathfrak{R}}(W) = k_{\mathfrak{R}}(U) = k_{\mathfrak{R}}(V) \not\subseteq V$.
Weil V eine saturierte Prämisse ist, gilt damit $W = V$.
Es folgt $V = p_{\mathfrak{R}}(U)$.
Somit ist $\mathfrak{DO}_{\mathfrak{R}}(X) \subseteq \mathfrak{DOO}_{\mathfrak{R}}(X)$.
Weil aber $\mathfrak{DOO}_{\mathfrak{R}}(X)$ vollständig und $\mathfrak{DO}_{\mathfrak{R}}(X)$ nicht redundant sind, gilt schon die Gleichheit.

Als Folgerung aus Satz A2 und der Definition von saturierten Prämissen in Definition 29 auf Seite 99 kann man feststellen, daß es reicht, bei der Saturierung der Prämissen nur Begriffsinhalte (-umfänge) P'' einzuschließen, für die P ein Pseudoinhalt (-umfang) ist (vergl. auch Satz B4 auf Seite 287). D.h. Bei der Bildung von $V^\circ := V \cup \bigcup\{U'' | U \subseteq V \wedge U'' \neq V''\}$ sind nur Pseudoinhalte bzw. -umfänge U zu betrachten. Nicht ausreichend ist es aber – wie in [G-D 86] fälschlicherweise behauptet –, allein Teilmengen zu betrachten, bei denen nur ein Element aus der Obermenge entnommen ist.

Bemerkung A1 (*Fehler in Duquenne-Guigues-Arbeit*)

Sei (G, M, I) ein endlicher formaler Kontext.
Dann ist für $B \subseteq M$ im allgemeinen
$B^\circ \neq B \cup \bigcup\{(B\backslash\{x\})'' \mid x \in B, (B\backslash\{x\})'' \neq B''\}$
(Vergl. [G-D 86], (6'), S. 13)

Beweis: Im nebenstehendem formalen Kontext gilt

$\{a, b\}^\circ$
$= \{a, b\} \cup \bigcup\{U'' | U \subseteq \{a, b\}, U'' \neq \{a, b\}''\}$
$= \{a, b\} \cup \bigcup\{U'' | U \subseteq \{a, b\}, U'' \neq \{a, b, c\}\}$
$= \{a, b\} \cup \varnothing'' = \{a, b, c\}$

	a	b	c
1	×	×	×
2	×	×	×
3			×

Abb. A.1 Bsp.-Kontext

Aber $\{a, b\} \cup \bigcup\{(B\backslash\{x\})'' \mid x \in B, (B\backslash\{x\})'' \neq \{a, b\}''\} = \{a, b\}$, weil $a'' = b'' = \{a, b\}'' = \{a, b, c\}$.

A.3 Blockrelationen

Vollständige Toleranzrelationen (Definition 34 auf Seite 115) liefern Zerlegungen von vollständigen Verbänden in sogenannte Blöcke (Definition 35 auf Seite 117). Im Falle von Begriffsverbänden lassen sich die Toleranzrelationen durch Blockrelationen (Definition 33 auf Seite 112) des zugrunde liegenden formalen Kontexts beschreiben.

Isomorphe vollständige Verbande haben isomorphe Verbände von vollständigen Toleranzrelationen (Vergl. Bemerkung 9 auf Seite 115).

Hilfssatz A4 (*Toleranzrelationen isomorpher Verbände*)
Sind V und W isomorphe vollständige Verbände, so sind auch die Verbände der vollständigen Toleranzrelationen auf V und auf W isomorph.

Beweis: Sei $\varphi : V \to W$ ein Isomorphismus.
Dann definiere für eine vollständige Toleranzrelation Θ auf V die Relation
$\psi(\Theta) := \{(\varphi(a), \varphi(b)) \mid a, b \in V \text{ und } a\Theta b\}$
$\psi(\Theta)$ ist dann eine binäre Relation auf W.
Mit Θ ist auch $\psi(\Theta)$ reflexiv, weil φ surjektiv ist.
Durch die Symmetrie von Θ ist auch $\psi(\Theta)$ symmetrisch.
Für $x, y \in W$ mit $x\ \psi(\Theta)\ y$ gibt es nach Konstruktion $a, b \in V$ mit $a\Theta b$ und $\varphi(a) = x, \varphi(b) = y$.
Wegen der Bijektivität von φ sind $a = \varphi^{-1}(x), b = \varphi^{-1}(y)$.
D.h. aus $x\ \psi(\Theta)\ y$ folgt $\varphi^{-1}(x)\ \Theta\ \varphi^{-1}(y)$.
Ist T eine beliebige Indexmenge und sind $x_t, y_t \in W$ mit $x_t\ \psi(\Theta)\ y_t$ für alle $t \in T$, so gilt $\varphi^{-1}(x_t)\ \Theta\ \varphi^{-1}(y_t)$ für alle $t \in T$.
Wegen der Infimumsverträglichkeit von Θ folgt
$(\bigwedge_{t \in T} \varphi^{-1}(x_t))\ \Theta\ (\bigwedge_{t \in T} \varphi^{-1}(y_t))$.
Weil φ ein vollständiger $\wedge$-Morphismus ist, folgt

$$\bigwedge_{t \in T} x_t = \bigwedge_{t \in T} \varphi(\varphi^{-1}(x_t))$$
$$= (\varphi(\bigwedge_{t \in T} \varphi^{-1}(x_t)))\ \psi(\Theta)\ (\varphi(\bigwedge_{t \in T} \varphi^{-1}(y_t))) = \bigwedge_{t \in T} y_t$$

Also ist auch $\psi(\Theta)$ infimumsverträglich.
Für die Verträglichkeit mit Suprema analog.
D.h. $\psi(\Theta)$ ist eine vollständige Toleranzrelation auf W.

Die Abbildung ψ zwischen den Mengen der vollständigen Toleranzrelationen auf V und aus W ist nach Definition ordnungserhaltend (Inklusionsordnung).
Mit derselben Konstruktion erhält man aus φ^{-1} eine ordnungserhaltende Abbildung σ von den vollständigen Toleranzrelationen auf W in die vollständigen Toleranzrelationen auf V.
Für eine vollständige Toleranzrelation Θ auf V gilt

$$\begin{aligned}\sigma(\psi(\Theta)) &= \{(\varphi^{-1}(x), \varphi^{-1}(y)) \mid x\ \psi(\Theta)\ y\} \\ &= \{(\varphi^{-1}(x), \varphi^{-1}(y)) \mid (x, y) \in \{(\varphi(a), \varphi(b)) \mid a\Theta b\}\} \\ &= \{(\varphi^{-1}(\varphi(a)), \varphi^{-1}(\varphi(b))) \mid a\Theta b\} \\ &= \{(a, b) \mid a\Theta b\} \\ &= \Theta\end{aligned}$$

Und genauso $\sigma(\psi(\vartheta)) = \vartheta$ für jede vollständige Toleranzrelation ϑ auf W.
Damit ist ψ ein Isomorphismus zwischen den Verbänden der vollständigen Toleranzrelationen auf V und auf W.

Man stellt fest, daß der Verband der Toleranzrelationen und der der Blockrelationen (Satz 15 auf Seite 114) isomorph sind.

Satz 16 (*Toleranzrelationen* $\leftrightarrow$ *Blockrelationen*)
Sei (G, M, I) ein formaler Kontext.
Dann ist der Verband aller Blockrelationen von (G, M, I) isomorph zum Verband der vollständigen Toleranzrelationen auf $\underline{\mathfrak{B}}(G, M, I)$.
Die durch $g\ (\beta(\Theta))\ m \quad :\Leftrightarrow \quad \gamma g\ \Theta\ (\gamma g \wedge \mu m)$ für $g \in G, m \in M$ und eine vollständige Toleranzrelationen Θ auf $\underline{\mathfrak{B}}(G, M, I)$ definierte Abbildung β ist ein entsprechender Isomorphismus.
Die Umkehrabbildung ist gegeben durch
$(A, B)\ \beta^{-1}(J)\ (C, D) \quad :\Leftrightarrow \quad A \times D \cup C \times B \subseteq J$ für
$(A, B), (C, D) \in \underline{\mathfrak{B}}(G, M, I)$ und eine Blockrelation J von (G, M, I).

Beweis: ([G-W 96], 3.4, Satz 15, S. 123)
Sei Θ eine vollständige Toleranzrelation auf $\underline{\mathfrak{B}}(G, M, I)$.
Zeige: $\beta(\Theta)$ ist Blockrelation von (G, M, I).
Seien $g \in G, m \in M$ mit $g\, I\, m$.
Dann ist $m \in g'$, also $g'' \subseteq m'$.
Das bedeutet $\gamma g = (g'', g') \leq (m', m'') = \mu m$.
Wegen der Reflexivität von Θ folgt $\gamma g\ \Theta\ \gamma g = \gamma g \wedge \mu m$.
Also gilt $g\ (\beta(\Theta))\ m$ und insgesamt $I \subseteq \beta(\Theta)$.
Sei $g \in G$.

Zeige: $g^{\beta(\Theta)}$ ist Begriffsinhalt von (G, M, I).

Nach Folgerung 5 auf Seite 88 ist $((g^{\beta(\Theta)})^I, (g^{\beta(\Theta)})^{II}) = \bigwedge_{m \in g^{\beta(\Theta)}} \mu m$.

Sei $n \in (g^{\beta(\Theta)})^{II}$.

Dann gilt $h\, I\, n$ für alle $h \in (g^{\beta(\Theta)})^I$.

Damit ist $(g^{\beta(\Theta)})^I \subseteq n^I$ und so

$$\bigwedge_{m \in g^{\beta(\Theta)}} \mu m = ((g^{\beta(\Theta)})^I, (g^{\beta(\Theta)})^{II}) \leq (n^I, n^{II}) = \mu n$$

Somit gilt auch

$$\bigwedge_{m \in g^{\beta(\Theta)}} (\gamma g \wedge \mu m) = \bigwedge_{m \in g^{\beta(\Theta)}} \gamma g \wedge \bigwedge_{m \in g^{\beta(\Theta)}} \mu m = \gamma g \wedge \bigwedge_{m \in g^{\beta(\Theta)}} \mu m \leq \gamma g \wedge \mu n$$

Für $m \in g^{\beta(\Theta)}$ gilt nach Konstruktion $\gamma g\ \Theta\ (\gamma g \wedge \mu m)$.

Wegen der Infimumsverträglichkeit von Θ gilt dann auch

$$\gamma g = \bigwedge_{m \in g^{\beta(\Theta)}} \gamma g\ \Theta\ \bigwedge_{m \in g^{\beta(\Theta)}} (\gamma g \wedge \mu m)$$

Nach Hilfssatz 2.d) auf Seite 68, der Supremumsverträglichkeit von Θ und der Ungleichung von oben folgt

$$\gamma g = \gamma g \vee (\gamma g \wedge \mu n)\ \Theta\ (\bigwedge_{m \in g^{\beta(\Theta)}} (\gamma g \wedge \mu m)) \vee (\gamma g \wedge \mu n) = \gamma g \wedge \mu n$$

Das bedeutet $n \in g^{\beta(\Theta)}$.

Also ist $(g^{\beta(\Theta)})^{II} \subseteq g^{\beta(\Theta)}$ und so nach Folgerung 2.c) auf Seite 82 damit $(g^{\beta(\Theta)})^{II} = g^{\beta(\Theta)}$.

D.h. $g^{\beta(\Theta)}$ ist ein Begriffsinhalt von (G, M, I).

Dual zeigt man, daß $m^{\beta(\Theta)}$ mit $m \in M$ ein Begriffsumfang von (G, M, I) ist.

Insgesamt ist $\beta(\Theta)$ eine Blockrelation von (G, M, I).

Andersherum betrachte zu einer Blockrelation J von (G, M, I) die Relation $\tau(J) \in \underline{\mathfrak{B}}(G, M, I)^2$ mit $(A, B)\ \tau(J)\ (C, D) \quad :\Leftrightarrow \quad A \times D \cup C \times B \subseteq J$.

Für $(A, B) \in \underline{\mathfrak{B}}(G, M, I)$ ist $A \times B \cup A \times B = A \times B \subseteq I \subseteq J$.

D.h. $(A, B)\ \tau(J)\ (A, B)$ und $\tau(J)$ ist reflexiv.

Nach Konstruktion ist $\tau(J)$ symmetrisch.

Zeige: $\tau(J)$ ist verträglich mit Infima.

Sei also T eine beliebige Indexmenge und für alle $t \in T$ gelte $(A_t, B_t), (C_t, D_t) \in \underline{\mathfrak{B}}(G, M, I)$ und $(A_t, B_t)\ \tau(J)\ (C_t, D_t)$.

D.h. $A_t \times D_t \cup C_t \times B_t \subseteq J$.

Für alle $t \in T$ und $m \in D_t$ gilt damit $g\, J\, m\ \ \forall g \in A_t$.

Also ist $C_t^I = D_t \subseteq \{m \in M \mid gJm\} = g^J$ für alle $t \in T$ und $g \in A_t$.

Nach Folgerung 2.b) auf Seite 82 folgt für alle $t \in T$ und $g \in A_t$, daß $g^{JI} \subseteq C_t^{II} = C_t$.

Für $g \in \bigcap_{t \in T} A_t$ gilt somit $g^{JI} \subseteq \bigcap_{t \in T} C_t$ und damit wieder nach Folgerung 2.b) $\left(\bigcap_{t \in T} C_t\right)^I \subseteq g^{JII} = g^J$.

Also gilt für alle $g \in \bigcap_{t \in T} A_t$ und $m \in \left(\bigcap_{t \in T} C_t\right)^I$, daß $g\,J\,m$.

D.h. $\bigcap_{t \in T} A_t \times \left(\bigcap_{t \in T} C_t\right)^I \subseteq J$.

Analog gilt $\bigcap_{t \in T} C_t \times \left(\bigcap_{t \in T} A_t\right)^I \subseteq J$.

Damit ist $\bigcap_{t \in T} A_t \times \left(\bigcap_{t \in T} C_t\right)^I \cup \bigcap_{t \in T} C_t \times \left(\bigcap_{t \in T} A_t\right)^I \subseteq J$.

Nach Satz 7 auf Seite 84 ist $\bigwedge_{t \in T} (A_t, B_t) = \left(\bigcap_{t \in T} A_t, \left(\bigcap_{t \in T} A_t\right)^I\right)$ und

$\bigwedge_{t \in T} (C_t, D_t) = \left(\bigcap_{t \in T} C_t, \left(\bigcap_{t \in T} C_t\right)^I\right)$.

Deshalb bedeutet obige Inklusion, daß $\bigwedge_{t \in T} (A_t, B_t)\ \tau(J)\ \bigwedge_{t \in T} (C_t, D_t)$.

Dual zeigt man die Supremumsverträglichkeit von $\tau(J)$.

Insgesamt ist $\tau(J)$ eine vollständige Toleranzrelation auf $\underline{\mathfrak{B}}(G, M, I)$.

Sofort sieht man, daß die beiden Abbildungen β und τ ordnungserhaltend sind, denn:

Sind Θ_1, Θ_2 vollständige Toleranzrelationen auf $\underline{\mathfrak{B}}(G, M, I)$ mit $\Theta_1 \subseteq \Theta_2$, so gilt für $g \in G, m \in M$ mit $g\ (\beta(\Theta_1))\ m$, daß $(\gamma g, (\gamma g \wedge \mu m)) \in \Theta_1 \subseteq \Theta_2$ und also auch $g\ (\beta(\Theta_2))\ m$.

D.h. $\beta(\Theta_1) \subseteq \beta(\Theta_2)$.

Sind J_1, J_2 Blockrelationen von (G, M, I) mit $J_1 \subseteq J_2$, so gilt für $(A, B), (C, D) \in \underline{\mathfrak{B}}(G, M, I)$ mit $(A, B)\ \tau(J_1)\ (C, D)$, daß $A \times D \cup C \times B \subseteq J_1 \subseteq J_2$ und also auch $(A, B)\ \tau(J_2)\ (C, D)$.

D.h. $\tau(J_1) \subseteq \tau(J_2)$.

Zeige nun, daß die Abbildungen β und τ zueinander invers sind.

Sei also Θ eine vollständige Toleranzrelation auf $\underline{\mathfrak{B}}(G, M, I)$.

Seien weiter $(A, B), (C, D) \in \underline{\mathfrak{B}}(G, M, I)$ mit $(A, B)\ \Theta\ (C, D)$.

Betrachte $g \in A, m \in D$.

Es gilt nach Folgerung 5 auf Seite 88, daß $\gamma g \leq \bigvee_{h \in A} \gamma h = (A, B)$ und $\mu m \geq \bigwedge_{n \in D} \mu n = (C, D)$.

Aus der Reflexivität und Infimumsverträglichkeit von Θ folgt damit

$$\gamma g = (\gamma g \wedge (A, B))\ \Theta\ (\gamma g \wedge (C, D))$$

und weiter $(\gamma g \wedge \mu m)\ \Theta\ (\gamma g \wedge (C, D) \wedge \mu m) = \gamma g \wedge (C, D)$.

Durch die Supremumsverträglichkeit von Θ gilt auch

$\gamma g \wedge \mu m = (\gamma g \wedge \mu m) \vee (\gamma g \wedge (C, D))\ \Theta\ (\gamma g \wedge (C, D)) \vee \gamma g = \gamma g$.

D.h. $g\ \beta(\Theta)\ m$.

Also ist $A \times D \subseteq \beta(\Theta)$.

Analog zeigt man $C \times B \subseteq \beta(\Theta)$ und erhält $(A, B)\ \tau(\beta(\Theta))\ (C, D)$.

Damit ist $\Theta \subseteq \tau(\beta(\Theta))$.

Gilt umgekehrt $(A, B)\ \tau(\beta(\Theta))\ (C, D)$ für

$(A, B), (C, D) \in \underline{\mathfrak{B}}(G, M, I)$, so folgt nach Hilfssatz 15 auf Seite 115

$((A, B) \wedge (C, D))\ \tau(\beta(\Theta))\ ((A, B) \vee (C, D))$.

Also gilt $((A \cap C) \times (B \cap D)) \cup ((A \cup C)'' \times (B \cup D)'') \subseteq \beta(\Theta)$.

Für $g \in (A \cup C)''$, $m \in (B \cup D)''$ gilt also $(g, m) \in \beta(\Theta)$, d.h.

$\gamma g\ \Theta\ (\gamma g \wedge \mu m)$.

Mit der Supremumsverträglichkeit von Θ folgt

$(\gamma g \vee \mu m)\ \Theta\ ((\gamma g \wedge \mu m) \vee \mu m) = \mu m$.

Aus Folgerung 5 auf Seite 88 und der Infimumsverträglichkeit von Θ folgert man $(A, B) \wedge (C, D) = \bigwedge_{m \in B \cap D} \mu m\ \Theta \bigwedge_{m \in B \cap D} (\gamma g \vee \mu m)$ für alle $g \in (A \cup C)''$.

Weiter gilt

$$\begin{aligned}(A, B) \wedge (C, D) \wedge \gamma g\ \Theta\ ((&\bigwedge_{m \in B \cap D} (\gamma g \vee \mu m)) \wedge \gamma g)\\ &= (\bigwedge_{m \in B \cap D} (\gamma g \vee \mu m)) \wedge \bigwedge_{m \in B \cap D} \gamma g\\ &= \bigwedge_{m \in B \cap D} ((\gamma g \vee \mu m) \wedge \gamma g)\\ &= \bigwedge_{m \in B \cap D} \gamma g = \gamma g\end{aligned}$$

Mit Folgerung 5 und der Supremumsverträglichkeit von Θ folgt

$(A, B) \vee (C, D) = \bigvee_{g \in (A \cup C)''} \gamma g\ \Theta \bigvee_{g \in (A \cup C)''} ((A, B) \wedge (C, D) \wedge \gamma g)$

Damit gilt auch

$$\begin{aligned}(A, B) \vee (C, D) &= ((A, B) \vee (C, D)) \vee ((A, B) \wedge (C, D))\\ &\Theta \bigvee_{g \in (A \cup C)''} ((A, B) \wedge (C, D) \wedge \gamma g) \vee ((A, B) \wedge (C, D))\\ &= \bigvee_{g \in (A \cup C)''} (((A, B) \wedge (C, D) \wedge \gamma g) \vee ((A, B) \wedge (C, D)))\\ &= \bigvee_{g \in (A \cup C)''} ((A, B) \wedge (C, D))\\ &= (A, B) \wedge (C, D)\end{aligned}$$

Mit Hilfssatz 15 auf Seite 115 folgt $(A, B)\ \Theta\ (C, D)$.

Also ist auch $\tau(\beta(\Theta)) \subseteq \Theta$, d.h. $\tau(\beta(\Theta)) = \Theta$.

Sei nun J eine Blockrelation von (G, M, I) .

Dann gilt für $A \subseteq G$ wegen $I \subseteq J$, daß $A^I \subseteq A^J$.

Weiter ist nach Folgerung 2.c),a) auf Seite 82 $A \subseteq A^{JJ} = \bigcap_{m \in A^J} m^J$ als Schnitt von Begriffsumfängen von (G, M, I) selber ein Begriffsumfang von (G, M, I) .

Nach Folgerung 3 auf Seite 83 folgt $A^{II} \subseteq A^{JJ}$.
Dual gilt für $B \subseteq M$, daß $B^{I} \subseteq B^{J}$, $B^{II} \subseteq B^{JJ}$.
Seien jetzt $g \in G, m \in M$ mit $g\,J\,m$.
Dann ist $g \in m^{J}$ und so nach Folgerung 3 $g^{II} \subseteq m^{J}$, weil m^{J} ein Begriffsumfang von (G, M, I) ist.
Damit folgt nach oben $g^{II} \subseteq g^{JJ} \cap m^{J}$.
Weiter ist nach oben $(g^{II} \cap m^{I})^{I} \subseteq (g^{JJ} \cap m^{J})^{J}$.
Also gilt $g^{II} \times (g^{II} \cap m^{I})^{I} \subseteq (g^{JJ} \cap m^{J}) \times (g^{JJ} \cap m^{J})^{J} \subseteq J$.
Weiter ist $g^{II} \cap m^{I} \subseteq g^{II} \subseteq g^{JJ}$, also auch
$(g^{II} \cap m^{I}) \times g^{I} \subseteq g^{JJ} \times g^{J} \subseteq J$.
Weil nach Satz 7 auf Seite 84 $\gamma g \wedge \mu m = (g^{II} \cap m^{I}, (g^{II} \cap m^{I})^{I})$, bedeuten die hergeleiteten Inklusionen $\gamma g\ \tau(J)\ (\gamma g \wedge \mu m)$.
Nach der Definition der Abbildung β bedeutet dies $g\ (\beta(\tau(J)))\ m$.
D.h. $J \subseteq \beta(\tau(J))$.
Gelte jetzt umgekehrt $g\ (\beta(\tau(J)))\ m$ für $g \in G, m \in M$.
D.h. $(g^{II} \times (g^{II} \cap m^{I})^{I}) \cup ((g^{II} \cap m^{I}) \times g^{I}) \subseteq J$.
Also ist nach oben und Folgerung 2.b),a) auf Seite 82

$$g^{II} \subseteq (g^{II} \cap m^{I})^{IJ} \subseteq (g^{II} \cap m^{I})^{JJ} \subseteq (g^{JJ} \cap m^{J})^{JJ} = (g^{J} \cup m)^{JJJ} = (g^{J} \cup m)^{J} = g^{JJ} \cap m^{J}$$

Insbesondere gilt damit $g \in g^{II} \subseteq m^{J}$, d.h. $g\,J\,m$.
Es folgt $\beta(\tau(J)) \subseteq J$ und insgesamt $\beta(\tau(J)) = J$.
Also sind die Abbildungen β und τ zueinander invers und ordnungserhaltend und damit der Satz bewiesen.

Als Konsequenz sind damit die Verbände der Blockrelationen zweier formaler Kontexte mit isomorphen Begriffsverbänden auch isomorph.

Folgerung A1 (*Formale Kontexte mit isomorphen Begriffsverbänden*)
Sind (G, M, I) und (H, N, R) zwei formale Kontexte, so daß $\underline{\mathfrak{B}}(G, M, I) \cong \underline{\mathfrak{B}}(H, N, R)$, so sind auch die Verbände der Blockrelationen von (G, M, I) und (H, N, R) isomorph.

Beweis: Nach Hilfssatz A4 auf Seite 263 gibt es einen Isomorphismus ψ zwischen den Verbänden der vollständigen Toleranzrelationen auf $\underline{\mathfrak{B}}(G, M, I)$ und auf $\underline{\mathfrak{B}}(H, N, R)$.
Sind $\beta_{\underline{\mathfrak{B}}(G, M, I)}$ und $\beta_{\underline{\mathfrak{B}}(H, N, R)}$ wie in Satz 16 auf Seite 264 Isomorphismen zwischen den Verbänden der vollständigen Toleranzrelationen auf $\underline{\mathfrak{B}}(G, M, I)$ bzw. auf $\underline{\mathfrak{B}}(H, N, R)$ und denen der Blockrelationen von (G, M, I) bzw. (H, N, R), so ist $\beta_{\underline{\mathfrak{B}}(H, N, R)} \circ \psi \circ \beta^{-1}_{\underline{\mathfrak{B}}(G, M, I)}$ ein Isomorphismus zwischen den betrachteten Verbänden von Blockrelationen.

Der Verband der Blöcke einer vollständigen Toleranzrelation (vergl. Satz 17 auf Seite 120) ist isomorph zu dem Verband der Blöcke der zugehörigen Toleranzrelation auf einem isomorphen Verband.

Folgerung A2 (*Blöcke in isomorphen Verbänden*)

Seien V und W isomorphe vollständige Verbände, Θ eine vollständige Toleranzrelation auf V und ϑ die zugehörige Toleranzrelation auf W wie in Hilfssatz A4 auf Seite 263.

Dann sind die Verbände der Blöcke V/Θ und W/ϑ isomorph.

Beweis: Sei $\varphi : V \to W$ ein Isomorphismus.

Erkläre $\sigma : V/\Theta \to W/\vartheta,\ [a]_\Theta \mapsto [\varphi(a)]_\vartheta$.

Sei ψ der Isomorphismus zwischen den vollständigen Toleranzrelationen auf V und denen auf W aus Hilfssatz A4, also $\vartheta = \psi(\Theta)$.

Dann gilt nach dem Beweis von Hilfssatz A4, daß

$x \,\vartheta\, y \Leftrightarrow x\ \psi(\Theta)\ y \Leftrightarrow \varphi^{-1}(x)\ \Theta\ \varphi^{-1}(y)$ (Hinrichtung direkt im zitierten Beweis und Rückrichtung nach Konstruktion von $\psi(\Theta)$)

Also ist

$$(\varphi(a))_\vartheta = \wedge\{y \in W \mid y\ \vartheta\ \varphi(a)\} = \wedge\{y \in W \mid \varphi^{-1}(y)\ \Theta\ a\}$$

$$= \wedge\{\varphi(b) \mid b \in V, b\,\Theta\,a\} = \varphi(\wedge\{b \mid b \in V, b\,\Theta\,a\}) = \varphi(a_\Theta)$$

Denn φ ist ein vollständiger $\wedge$-Morphismus.

Danach ist die Abbildung σ ordnungserhaltend, weil φ ordnungserhaltend ist. (Ordnung der Blöcke über untere Intervallgrenzen erklärt)

Mit der gleichen Argumentation ist $\rho : W/\vartheta \to V/\Theta,\ [x]_\Theta \mapsto [\varphi^{-1}(x)]_\Theta$ ordnungserhaltend.

Nach Konstruktion sind σ und ρ zueinander invers.

Also ist σ der gesuchte Isomorphismus.

Nun bekommt man auch eine Isomorphie zwischen den formalen Begriffen eines aus einer Blockrelation neu gebildeten Kontexts und den Blöcken der entsprechenden Toleranzrelation.

Satz 18 (*Blockrelation ↔ Toleranzrelation*)

Seien (G, M, I) ein formaler Kontext, Θ eine vollständige Toleranzrelation auf $\underline{\mathfrak{B}}(G, M, I)$ und $J := \beta(\Theta)$ die zugehörige Blockrelation von (G, M, I) (mit β aus Satz 16 auf Seite 264).

Dann sind $\underline{\mathfrak{B}}(G, M, I)/\Theta$ und $\underline{\mathfrak{B}}(G, M, J)$ isomorph.

Genauer gilt:

a) $[(B^I, B), (A, A^I)]$ ist genau dann ein Block von Θ, wenn (A, B) ein formaler Begriff von (G, M, J) ist.

b) Die Abbildung $[(B^I, B), (A, A^I)] \mapsto (A, B)$ ist ein Isomorphismus zwischen $\underline{\mathfrak{B}}(G, M, I)/\Theta$ und $\underline{\mathfrak{B}}(G, M, J)$.

c) Für $(A, B) \in \underline{\mathfrak{B}}(G, M, J)$ gilt für den zugehörigen Block von Θ, daß $[(B^I, B), (A, A^I)] = \mathfrak{B}(A, B, I \cap (A \times B))$.

Beweis: ([G-W 96], 3.4, Korollar 57, S. 125)

a) "$\Rightarrow$" Sei $[(B^I, B), (A, A^I)]$ ein Block von Θ.

Dann gibt es einen formalen Begriff $(C, D) \in \underline{\mathfrak{B}}(G, M, I)$ mit $[(B^I, B), (A, A^I)] = [(C, D)]_\Theta$.

Also ist nach Hilfssatz 20 auf Seite 120

$(B^I, B) = (C, D)_\Theta = (C^{JI}, C^J)$ und

$(A, A^I) = ((C, D)_\Theta)^\Theta = (C^{JJ}, C^{JJI})$.

Damit ist $(A, B) = (C^{JJ}, C^J)$ ein formaler Begriff von (G, M, J).

"$\Leftarrow$" Sei (A, B) ein formaler Begriff von (G, M, J).

Dann ist nach Folgerung 2.c),a) auf Seite 82 $A = B^J = \bigcap_{m \in B} m^J$ als Schnitt von Begriffsumfängen von (G, M, I) selber ein Begriffsumfang von (G, M, I).

Damit ist $(A, A^I) \in \underline{\mathfrak{B}}(G, M, I)$.

Nach Hilfssatz 20 ist $(A, A^I)_\Theta = (A^{JI}, A^J) = (B^I, B)$ und

$((A, A^I)_\Theta)^\Theta = (A^{JJ}, A^{JJI}) = (A, A^I)$.

D.h. $[(B^I, B), (A, A^I)] = [(A, A^I)]_\Theta$ ist ein Block von Θ.

b) Nach a) ist die Abbildung wohldefiniert und surjektiv.

Nach Konstruktion ist sie injektiv.

Sind $[(B_1^I, B_1), (A_1, A_1^I)], [(B_2^I, B_2), (A_2, A_2^I)] \in \underline{\mathfrak{B}}(G, M, I)/\Theta$ mit $[(B_1^I, B_1), (A_1, A_1^I)] \leq [(B_2^I, B_2), (A_2, A_2^I)]$, so gilt nach Hilfssatz 18 auf Seite 118 $(A_1, A_1^I) \leq (A_2, A_2^I)$ in $\underline{\mathfrak{B}}(G, M, I)$.

D.h. $A_1 \subseteq A_2$.

Also ist $(A_1, B_1) \leq (A_2, B_2)$ in $\underline{\mathfrak{B}}(G, M, J)$.

Somit ist die angegebene Abbildung ordnungserhaltend.

Sind andersherum $(A_1, B_1), (A_2, B_2) \in \underline{\mathfrak{B}}(G, M, J)$ mit $(A_1, B_1) \leq (A_2, B_2)$, so gilt auch $(A_1, A_1^I) \leq (A_2, A_2^I)$ in $\underline{\mathfrak{B}}(G, M, I)$. und deshalb $[(B_1^I, B_1), (A_1, A_1^I)] \leq [(B_2^I, B_2), (A_2, A_2^I)]$ in $\underline{\mathfrak{B}}(G, M, I)/\Theta$.

Also ist die Abbildung ein Ordnungsisomorphismus.

c) Setze $K := I \cap (A \times B)$.

Betrachte jeweils X^K in $\mathfrak{B}(A, B, K)$ und X^I in $\underline{\mathfrak{B}}(G, M, I)$.

"$\subseteq$" Sei $(C, D) \in [(B^I, B), (A, A^I)]$.

Dann ist $C \subseteq A, D \subseteq B$.

Es gilt
$$C^K = \{m \in B \mid (g, m) \in K \ \forall g \in C\}$$
$$\subseteq \{m \in M \mid (g, m) \in I \ \forall g \in C\} = C^I$$

Sei $m \in C^I = D$.
Dann ist $m \in B$ und $(g, m) \in I \quad \forall g \in C$.
Wegen $C \subseteq A$ ist also $(g, m) \in I \cap (A \times B) = K \quad \forall g \in C$.
D.h. $m \in C^K$.
Also ist $C^K = C^I = D$.
Analog ist $D^K = C$, also $(C, D) \in \mathfrak{B}(A, B, I \cap (A \times B))$.
"$\supseteq$" Sei $(C, D) \in \mathfrak{B}(A, B, I \cap (A \times B))$.
Dann ist $C \subseteq A, D \subseteq B$.
Weil J Blockrelation von (G, M, I) und (A, B) ein formaler Begriff von (G, M, J) ist, gilt $A^I \subseteq A^J = B, B^I \subseteq B^J = A$.
Wie oben ist weiter $B^K \subseteq B^I$.
Für $g \in B^I$ gilt $g \in A$ und $(g, m) \in I \quad \forall m \in B$.
Es folgt $(g, m) \in I \cap (A \times B) = K \quad \forall m \in B$.
D.h. $g \in B^K$.
Also ist $B^I \subseteq B^K$ und insgesamt $B^K = B^I$.
Damit ist $(B^I, B) = (B^K, B) \in \mathfrak{B}(A, B, K)$.
Es folgt mit Folgerung 2.b) auf Seite 82 $B^I = B^K \subseteq D^K = C$ und so $C^I \subseteq B^{II} = B$.
Wie oben ist $C^K \subseteq C^I$.
Für $m \in C^I$ gilt $m \in B$ und $(g, m) \in I \quad \forall g \in C$.
Wegen $C \subseteq A$ ist also $(g, m) \in I \cap (A \times B) = K \quad \forall g \in C$.
D.h. $m \in C^K$.
Das bedeutet $C^I \subseteq C^K$, also insgesamt $C^I = C^K = D$.
Analog ist $D^I = C$.
Damit ist $(C, D) \in \underline{\mathfrak{B}}(G, M, I)$.
$C \subseteq A, D \subseteq B$ ergibt somit $(C, D) \in [(B^I, B), (A, A^I)]$.

Die Bildung von Blockrelationen eines formalen Kontexts und Toleranzrelationen des zugehörigen Begriffsverbands sind gleichbedeutend. Die formalen Begriffe des durch eine Blockrelation neu definierten Kontexts entsprechen bestimmten Intervallen des originalen Begriffsverbands, nämlich gerade den Blöcken der zugehörigen Toleranzrelation. Insbesondere haben formale Kontexte mit isomorphen Begriffsverbänden auch isomorphe Verbände von Blockrelationen.

Satz 19 (*Blockrelationen und isomorphe Begriffsverbände*)
Seien (G, M, I) und (H, N, R) zwei formale Kontexte, so daß $\underline{\mathfrak{B}}(G, M, I) \cong \underline{\mathfrak{B}}(H, N, R)$.
Dann sind (nach Folgerung A1 auf Seite 268) auch die Verbände der Blockrelationen von (G, M, I) und (H, N, R) isomorph.

Sei weiter J eine Blockrelation von (G, M, I) und S die nach dieser Isomorphie zugehörige Blockrelation von (H, N, R).
Dann sind die durch die Blockrelationen definierten Begriffsverbände $\underline{\mathfrak{B}}(G, M, J)$ und $\underline{\mathfrak{B}}(H, N, S)$ isomorph.

Beweis: Nach Folgerung A1 ist mit den Bezeichnungen von dort $\beta_{\underline{\mathfrak{B}}(H, N, R)} \circ \psi \circ \beta^{-1}_{\underline{\mathfrak{B}}(G, M, I)}$ ein Isomorphismus zwischen den Verbänden der Blockrelationen von $\underline{\mathfrak{B}}(G, M, J)$ und $\underline{\mathfrak{B}}(H, N, S)$.
Also ist $S = \beta_{\underline{\mathfrak{B}}(H, N, R)}(\psi(\beta^{-1}_{\underline{\mathfrak{B}}(G, M, I)}(J)))$.
Setze $\Theta := \beta^{-1}_{\underline{\mathfrak{B}}(G, M, I)}(J)$.
Nach Folgerung A2 auf Seite 269 gibt es einen Isomorphismus $\sigma : \underline{\mathfrak{B}}(G, M, J)/\Theta \to \underline{\mathfrak{B}}(H, N, S)/\psi(\Theta)$.
Weiter gibt es Isomorphismen $\zeta_J : \underline{\mathfrak{B}}(G, M, I)/\Theta \to \underline{\mathfrak{B}}(G, M, J)$ und $\zeta_S : \underline{\mathfrak{B}}(H, N, R)/\beta^{-1}_{\underline{\mathfrak{B}}(H, N, R)}(S) \to \underline{\mathfrak{B}}(H, N, S)$ (Satz 18 auf Seite 121).
In folgender Darstellung sind alle Abbildungen Isomorphismen:

$$\underline{\mathfrak{B}}(G, M, J) \xrightarrow{\zeta_J^{-1}} \underline{\mathfrak{B}}(G, M, J)/\Theta \xrightarrow{\sigma} \underline{\mathfrak{B}}(H, N, S)/\psi(\Theta)$$
$$= \underline{\mathfrak{B}}(H, N, S)/\psi(\beta^{-1}_{\underline{\mathfrak{B}}(G, M, I)}(J))$$
$$= \underline{\mathfrak{B}}(H, N, S)/\beta^{-1}_{\underline{\mathfrak{B}}(H, N, R)}\beta_{\underline{\mathfrak{B}}(H, N, R)}(\psi(\beta^{-1}_{\underline{\mathfrak{B}}(G, M, I)}(J)))$$
$$= \underline{\mathfrak{B}}(H, N, S)/\beta^{-1}_{\underline{\mathfrak{B}}(H, N, R)}(S) \xrightarrow{\zeta_S} \underline{\mathfrak{B}}(H, N, S)$$

Also ist $\zeta_S \circ \sigma \circ \zeta_J^{-1}$ ein Isomorphismus von $\underline{\mathfrak{B}}(G, M, J)$ auf $\underline{\mathfrak{B}}(H, N, S)$.

In Kapitel 3 war auf Seite 117 als Beschreibung der Blöcke einer vollständigen Toleranzrelation angegeben, daß diese gerade die maximalen Teilmengen sind, deren Elemente paarweise in Relation Θ stehen.

Bemerkung A2 (*Blöcke ≙ maximale "in Relation stehende" Teilmengen*)

Ist V ein vollständiger Verband und Θ eine vollständige Toleranzrelation auf V, so sind die Blöcke von Θ genau die maximalen Teilmengen U von V, für die gilt $x \,\Theta\, y \quad \forall x, y \in U$.

Beweis: ([G-W 96], 3.4, Hilfssatz 55, S. 121)
Sei $a \in V$.
Es gilt $[a]_\Theta = [a_\Theta, (a_\Theta)^\Theta] = [a_\Theta \wedge (a_\Theta)^\Theta, a_\Theta \vee (a_\Theta)^\Theta]$ und $a_\Theta \,\Theta\, (a_\Theta)^\Theta$ (Hilfssatz 16 auf Seite 117).
Für $x, y \in [a]_\Theta$ liefert Hilfssatz 15 auf Seite 115 also $x \Theta y$.

Sei $z \in V$ mit $z \Theta y \quad \forall y \in [a]_\Theta$.
Dann gilt nach Hilfssatz 16 insbesondere $z \Theta a$ und $z \Theta a_\Theta$.
Also ist $z \geq a_\Theta$ und $z \leq (a_\Theta)^\Theta$, d.h. $z \in [a]_\Theta$.

Die Blöcke von Θ sind also maximal bzgl. der Eigenschaft, daß alle ihre Elemente paarweise in Relation Θ stehen.
Sei nun $U \subseteq V$, so daß für alle $\forall x, y \in U \; x \Theta y$ gilt und U bzgl. dieser Eigenschaft maximal ist.

Dann gilt wegen der Verträglichkeit von Θ mit Infima und Suprema für jedes $y \in U$, daß $(\wedge U) \; \Theta \; (\underset{x \in U}{\wedge} y) = y$ und $(\vee U) \; \Theta \; (\underset{x \in U}{\vee} y) = y$.
Wegen der Maximalität von U gilt also $\wedge U, \vee U \in U$.
Für $x \in U \subseteq [\wedge U, \vee U]$ und
$y \in [\wedge U, \vee U] = [(\wedge U) \wedge (\vee U), (\wedge U) \vee (\vee U)]$ gilt nach Hilfssatz 15 auf Seite 115 $x \Theta y$.
Wegen der Maximalität von U ist also $U = [\wedge U, \vee U]$.
Also ist $(\wedge U) \; \Theta \; (\vee U)$ und so $(\vee U)_\Theta \leq \wedge U$.
Für $x \leq \wedge U$ mit $x \; \Theta \; (\wedge U)$ gilt wegen der Infimumsverträglichkeit von Θ für alle $y \in U$, daß $x = (x \wedge y) \; \Theta \; ((\vee U) \wedge y) = y$.
Also ist $y \in U$, d.h. $y = \wedge U$, wegen der Maximalität von U.
D.h. $(\vee U)_\Theta = \wedge \{x \in V \mid x \Theta (\vee U)\} = \wedge U$.
Analog zeigt man $(\wedge U)^\Theta = \vee U$.

Damit ist $U = [\wedge U, \vee U] = \left[(\vee U)_\Theta, ((\vee U)_\Theta)^\Theta \right]$ ein Block von Θ.

Die Blöcke einer Toleranzrelation brauchen nicht disjunkt sein und können quer zueinander liegen.

Bemerkung 11 (*Blöcke nicht disjunkt*)

Die Blöcke einer vollständigen Toleranzrelation Θ auf einem vollständigen Verband V brauchen nicht disjunkt zu sein.
Für $a, b \in V$ gilt $[a]_\Theta \cap [b]_\Theta \neq \emptyset \quad \Leftrightarrow \quad a_\Theta \leq (b_\Theta)^\Theta$ und $b_\Theta \geq (a_\Theta)^\Theta$.

Beweis: "$\Rightarrow$" Es gebe ein $x \in [a]_\Theta \cap [b]_\Theta$.
Nach Bemerkung A2 gilt damit $a_\Theta \; \Theta \; x$ und $x \; \Theta \; b_\Theta$.
Wegen der Infimumsverträglichkeit von Θ gilt dann
$a_\Theta = (a_\Theta \wedge x) \; \Theta \; (x \wedge b_\Theta) = b_\Theta$ und wegen der Symmetrie von Θ also auch $b_\Theta \; \Theta \; a_\Theta$.
Somit ist $a_\Theta \leq \vee \{x \in V \mid x \Theta b_\Theta\} = (b_\Theta)^\Theta$ und
$b_\Theta \leq \vee \{x \in V \mid x \Theta a_\Theta\} = (a_\Theta)^\Theta$.
"$\Leftarrow$" Es gelte $a_\Theta \leq (b_\Theta)^\Theta$ und $b_\Theta \geq (a_\Theta)^\Theta$.
Dann sind $a_\Theta, b_\Theta \leq (a_\Theta)^\Theta \wedge (b_\Theta)^\Theta$.
Also ist $((a_\Theta)^\Theta \wedge (b_\Theta)^\Theta) \in [a]_\Theta \cap [b]_\Theta$.

A.4 Gegenstands- und Merkmalsordnung

Die beiden in der Überschrift genannten Konzepte spielen in BASE selber keine Rolle, sondern werden nur bei der Erläuterung des Algorithmus des additiven Zeichnens (B.5.1 auf Seite 305) und der Vorstellung eines Ansatz zur Gestaltung von Vererbungshierarchien (6.1 auf Seite 224) erwähnt.

Definition A2 (*Gegenstands- und Merkmalsordnung*)
Ist (G, M, I) ein formaler Kontext, so heißt die durch $g \leq h \;:\Leftrightarrow\; \gamma g \leq \gamma h$ bestimmte Ordnung auf G die Gegenstandsordnung von (G, M, I) und dual die mit $m \leq n \;:\Leftrightarrow\; \mu m \leq \mu n$ gegebene Ordnung auf M die Merkmalsordnung von (G, M, I).
([Vog 96], Lemma 3.10, S. 59)

Gegenstands- und Merkmalsordnung findet man im formalen Kontext:

Bemerkung A3 (*Gegenstands- und Merkmalsordnung*)
Seien (G, M, I) ein formaler Kontext, $g, h \in G$ und $m, n \in M$.
Dann gilt $g \leq h \Leftrightarrow g' \supseteq h'$ und $m \leq n \Leftrightarrow m' \subseteq n'$ (bzgl. Gegenstands- bzw. Merkmalsordnung).
([Vog 96], Lemma 3.10, S. 59)

In der Kontexttabelle umfassen also Zeilen zu (in der Gegenstandsordnung) kleineren Gegenständen solche von größeren und Spalten zu kleineren Merkmale sind in solchen zu größeren enthalten.

B Algorithmen zur FBA

Im folgenden sind die wichtigsten Algorithmen dargestellt, die in BASE zur Behandlung von begriffsanalytischen Strukturen benutzt werden. Die grundlegenden berechnen aus einem formalen Kontext andere Strukturen. Dies kann im wesentlichen mit Hilfe eines einzigen Algorithmus geschehen, weil die entsprechenden Strukturen jeweils Hüllensysteme bilden. Deshalb wird in Abschnitt B.1 zunächst einmal die Erzeugung eines beliebigen Hüllensystems mit Hilfe von *Ganters Hüllenalgorithmus* erklärt. B.2 zeigt dann die Erstellung eines Begriffsverbands zu einem formalen Kontext. In B.3 werden Implikationenbasen[1] behandelt und in B.4 Blockrelationen. Im Abschnitt B.2 werden bereits Liniendiagramme behandelt, denn in dieser Form werden Begriffsverbände gespeichert. B.5 zeigt, wie zusätzlich zu der in B.2 erstellten Ordnungsinformation eine graphische Anordnung für ein Liniendiagramm berechnet werden kann.

Soweit möglich, wurde versucht, in der Darstellung der Algorithmen von konkreten Datenstrukturen zu abstrahieren, um jeweils den Kern des betreffenden Algorithmus herauszustellen. Deshalb wurde auch eine Pseudocode-Notation gewählt, die keine Variablenvereinbarungen enthält. Dabei sind die Anweisungen in Pascal-Stil angegeben (ergänzt durch Fortrans *ELSEIF*), Bedingungen und Anweisungsblöcke aber wie in der Sprache C geklammert und Kommentare ebenfalls wie in C eingeleitet. (Diese Mischung ergibt eine klare und – wo möglich – kompakte Syntax.) Eingaben der Algorithmen – also im wesentlichen formale Gegenstände, Merkmale, Kontexte und Begriffsverbände – sind durch ihre Schriftart von Schlüsselwörtern und Algorithmusvariablen abgesetzt. Dies hebt die wichtigsten Stellen hervor, die an konkrete Datenstrukturen angepaßt werden müssen.

Bei den Abschätzungen von Laufzeiten gehen immer die Kardinalitäten endlicher Mengen ein. Im folgenden bezeichne also $|X|$ die Elementanzahl einer endlichen Menge X. In der gleichen Weise werden Array-Längen notiert.

---

1. Die Erstellung von Implikationenbasen ist auch anhand von Liniendiagrammen möglich. In der Regel geht man in begriffsanalytischen Untersuchungen aber von einem formalen Kontext aus. Deshalb wurde er hier auch als Ausgangsposition gewählt. Bei dem in Kapitel 5 vorgestellten Werkzeug zu BASE muß vor der Berechnung einer Implikationenbasis oder der Blockzerlegungen erst aus dem gespeicherten Liniendiagramm ein formaler Kontext erstellt werden.

B.1 Berechnung von Hüllensystemen

Wichtige Strukturen innerhalb der Formalen Begriffsanalyse bilden Hüllensysteme (Definition 16 auf Seite 77), insbesondere:

- die Begriffsumfänge eines formalen Kontexts (Folgerung 3 auf Seite 83)
- die Begriffsinhalte eines formalen Kontexts (Folgerung 3)
- die Pseudoumfänge und Begriffsumfänge eines endlichen formalen Kontexts zusammen (Satz B3 auf Seite 287)
- die Pseudoinhalte und Begriffsinhalte eines endlichen formalen Kontexts zusammen (Satz B3)
- die Blockrelationen eines formalen Kontexts (Satz 15 auf Seite 114)

Deshalb wird zunächst erläutert, wie die Mengen eines Hüllensystems zu einem Hüllenoperator φ (Definition 17 auf Seite 77) auf einer endlichen Menge X berechnet werden können. Dies liefert der *Hüllenalgorithmus von Ganter* ([G-W 96], 2.1, S. 67). Die Ideen hinter diesem Algorithmus und der Algorithmus selber sind im folgenden wiedergegeben.

B.1.1 Potenzmengenaufzählung in lektischer Ordnung

Insbesondere ist auf jeder Menge X die zugehörige Potenzmenge $\mathfrak{P}(X)$ ein Hüllensystem. Daher ist damit zu rechnen, daß ein Hüllensystem zu einer n-elementigen Menge 2^n Elemente besitzt. Es ist in den meisten Fällen schwierig, die Größe des zu berechnenden Hüllensystems a priori abzuschätzen. Deshalb werden die Hüllen durch Ganters Algorithmus sukzessive berechnet. Dazu müssen sie linear aufgezählt werden können, damit zu jeder Hülle klar ist, welches die nächste ist. Kann man die Potenzmenge $\mathfrak{P}(X)$ in einer solchen Weise aufzählen, ergibt sich auch für jedes in ihr enthaltene Hüllensystem eine lineare Aufzählung. Aufgabe ist also, eine lineare Ordnung auf $\mathfrak{P}(X)$ bereitzustellen.

Definition B1 (*Lektische Ordnung*)

Seien $X = \{x_0, \ldots, x_{n-1}\}$ eine endliche Menge und $U, V \subseteq X$.
Dann definiere für $i \in \{0, \ldots, n-1\}$:

$$U <_i V \quad :\Leftrightarrow \quad x_i \in V \setminus U \quad \text{und} \quad U \cap \{x_0, \ldots, x_{i-1}\} = V \cap \{x_0, \ldots, x_{i-1}\}$$

Weiter setze $U \leq V \quad :\Leftrightarrow \quad U = V$ oder $\exists i \in \{0, \ldots, n-1\} : U <_i V$.
$\leq$ heißt lektische Ordnung auf $\mathfrak{P}(X)$.

Noch nachzuweisen ist, daß in Definition B1 wirklich eine Ordnung (Definition 2 auf Seite 64) definiert wurde.

Hilfssatz B1 (*Lektische Ordnung*)

Für eine endliche Menge $X = \{x_0, \ldots, x_{n-1}\}$ ist $\leq$ aus Definition B1 eine lineare Ordnung auf $\mathfrak{P}(X)$.

([G-W 96], 2.1, S. 67)

Beweis: Nach der Definition ist $\leq$ reflexiv.

Seien $U, V, W \subseteq X$ mit $U \leq V$ und $V \leq W$.

Gilt $U = V$ oder $V = W$, so natürlich auch $U \leq W$.

Seien also $i, j \in \{0, \ldots, n-1\}$ mit $U <_i V$ und $V <_j W$.

Setze $k := \min\{i, j\}$.

Dann ist
$$\begin{aligned} U \cap \{x_0, \ldots, x_{k-1}\} &= (U \cap \{x_0, \ldots, x_{i-1}\}) \cap \{x_0, \ldots, x_{k-1}\} \\ &= (V \cap \{x_0, \ldots, x_{i-1}\}) \cap \{x_0, \ldots, x_{k-1}\} \\ &= V \cap \{x_0, \ldots, x_{k-1}\} \\ &= (V \cap \{x_0, \ldots, x_{j-1}\}) \cap \{x_0, \ldots, x_{k-1}\} \\ &= (W \cap \{x_0, \ldots, x_{j-1}\}) \cap \{x_0, \ldots, x_{k-1}\} \\ &= W \cap \{x_0, \ldots, x_{k-1}\} \end{aligned}$$

Für $i = j$ wäre $x_i \in (V \setminus U) \cap (W \setminus V) = \emptyset$.

Dieser Fall ist also unmöglich.

Ist $i < j$, so ist $x_k = x_i \notin U$ und

$x_k \in (V \cap \{x_0, \ldots, x_{j-1}\} = W \cap \{x_0, \ldots, x_{j-1}\} \subseteq W)$.

Im anderen Fall ist $x_k \in (W \setminus V)$ und $x_k = x_j \in \{x_0, \ldots, x_{i-1}\}$.

Mit $x_k \in U$ wäre $x_k \in U \cap \{x_0, \ldots, x_{i-1}\} = V \cap \{x_0, \ldots, x_{i-1}\} \subseteq V$.

Also ist in jedem Fall $x_k \in W \setminus U$ und insgesamt $U <_k W$.

Somit ist $\leq$ transitiv.

Seien $U, V \in X$ mit $U \leq V$ und $V \leq U$.

Wäre $U \neq V$, so gäbe es $i, j \in \{0, \ldots, n-1\}$ mit $U <_i V$ und $V <_j U$.

Mit $k := \min\{i, j\}$ folgt daraus wie oben $x_k \in U \setminus U = \emptyset$.

Da dies unmöglich ist, muß also $U = V$ gelten.

Damit ist $\leq$ antisymmetrisch.

Also ist $\leq$ eine Ordnung auf $\mathfrak{P}(X)$.

Seien $U, V \in X$ mit $U \neq V$.

Setze $W := (V \setminus U) \cup (U \setminus V)$.

Dann gilt $\emptyset \neq W \subseteq \{x_0, \ldots, x_{n-1}\}$.

Also existiert $k := \min\{i \mid x_i \in W\}$.

Nach Konstruktion ist dann $U \cap \{x_0, \ldots, x_{k-1}\} = V \cap \{x_0, \ldots, x_{k-1}\}$.

Ist $x_k \in V \setminus U$, so folgt $U <_k V$.

Andernfalls ist $x_k \in U \setminus V$ und so $V <_k U$.

Somit sind alle Elemente von $\mathfrak{P}(X)$ in $\leq$ vergleichbar.

Schreibt man die Teilmengen von X als Bitvektoren der Länge n, wobei eine 1 an der Position i ($i \in \{0, \ldots, n-1\}$ und die Positionen von vorn nach hinten numeriert) das Vorhandensein von x_i in der betreffenden Teilmenge codiert, so ist die lektische Ordnung gerade die übliche lexikographische Ordnung bzw. die natürliche Ordnung auf den als Binärzahlen interpretierten Bitvektoren.

Die lektische Ordnung bildet eine Erweiterung der Inklusionsordnung. Dies kann in Algorithmen zu effizienten Minimums- und Maximumsbestimmungen (bzgl. der Inklusionsordnung) ausgenutzt werden.

Hilfssatz B2 (*Lektische Ordnung* $\leftrightarrow$ *Inklusionsordnung*)
Seien $X = \{x_0, \ldots, x_{n-1}\}$ eine endliche Menge und $U, V \subseteq X$.
Dann folgt aus $U \subsetneqq V$, daß $U < V$ (bzgl. der lektischen Ordnung).

Beweis: ([Lin 99], 3.1.2, Theorem 7)
Wegen $U \subsetneqq V$ gilt $\varnothing \neq V \setminus U \subseteq \{x_0, \ldots, x_{n-1}\}$.
Also existiert $k := \min\{i \mid x_i \in V \setminus U\}$.
Nach Konstruktion von k gilt $U <_k V$.

B.1.2 Hüllenbestimmung

Zu einem Hüllenoperator φ auf einer endlichen Menge X sollen die Hüllen in lektischer Reihenfolge bestimmt werden.

Satz B1 (*Lektisch nächste Hülle*)
Seien $X = \{x_0, \ldots, x_{n-1}\}$ eine endliche Menge, φ ein Hüllenoperator, $\mathcal{U}_\varphi$ das zugehörige Hüllensystem wie in Hilfssatz 4 auf Seite 77 und $V \subsetneqq X$.
Für $i \in \{0, \ldots, n-1\}$ setze $V \oplus_\varphi x_i := \varphi((V \cap \{x_0, \ldots, x_{i-1}\}) \cup \{x_i\})$.
Weiter sei $k := \max\{i \mid V <_i (V \oplus_\varphi x_i)\}$.
Dann ist $V \oplus_\varphi x_k$ die lektisch kleinste Hülle von φ, die lektisch echt größer als V ist.

Beweis: ([G-W 96], 2.3, Satz 5, S. 67)
Es ist $X = \bigcap\varnothing \in \mathcal{U}_\varphi$.
Nach Hilfssatz B2 folgt $V < X$ aus $V \subsetneqq X$.
Damit existiert $W := \min\{U \in \mathcal{U}_\varphi \mid V < U\}$ (Minimum bzgl. der lektischen Ordnung, $\mathcal{U}_\varphi$ ist nach Voraussetzung endlich).
Wegen $V < W$ gibt es ein $j \in \{0, \ldots, n-1\}$ mit $V <_j W$.
D.h. $x_j \in W \setminus V$ und $V \cap \{x_0, \ldots, x_{j-1}\} = W \cap \{x_0, \ldots, x_{j-1}\}$.
Weil φ bzgl. der Inklusion monoton und W eine Hülle von φ ist, gilt
$V \oplus_\varphi x_j = \varphi((V \cap \{x_0, \ldots, x_{j-1}\}) \cup \{x_j\}) \subseteq \varphi(W) = W$.
Nach Hilfssatz B2 folgt $(V \oplus_\varphi x_j) \leq W$.

Weiter ist $x_j \in (V \oplus_\varphi x_j) \setminus V$ und

$$\begin{aligned} V \cap \{x_0, \ldots, x_{j-1}\} &\subseteq (V \oplus_\varphi x_j) \cap \{x_0, \ldots, x_{j-1}\} \\ &\subseteq W \cap \{x_0, \ldots, x_{j-1}\} \\ &= V \cap \{x_0, \ldots, x_{j-1}\} \end{aligned}$$

D.h. $V <_j (V \oplus_\varphi x_j)$.

Nach Wahl von W ist damit $W \leq (V \oplus_\varphi x_j)$, also $W = V \oplus_\varphi x_j$.

Insbesondere ist somit k in der Behauptung wohldefiniert.

Annahme: $k > j$.

Dann wäre $x_j \in \{x_0, \ldots, x_{k-1}\} \setminus V$.

Wegen $V \cap \{x_0, \ldots, x_{k-1}\} = (V \oplus_\varphi x_k) \cap \{x_0, \ldots, x_{k-1}\}$, wäre also $x_j \notin (V \oplus_\varphi x_k)$ und

$$\begin{aligned} &(V \oplus_\varphi x_k) \cap \{x_0, \ldots, x_{j-1}\} \\ &\quad = ((V \oplus_\varphi x_k) \cap \{x_0, \ldots, x_{k-1}\}) \cap \{x_0, \ldots, x_{j-1}\} \\ &\quad = (V \cap \{x_0, \ldots, x_{k-1}\}) \cap \{x_0, \ldots, x_{j-1}\} \\ &\quad = V \cap \{x_0, \ldots, x_{j-1}\} \\ &\quad = W \cap \{x_0, \ldots, x_{j-1}\} \end{aligned}$$

Das bedeutet $(V \oplus_\varphi x_k) <_j W$ im Widerspruch zur Wahl von W, weil $V \oplus_\varphi x_k$ als Bild unter dem Hüllenoperator φ_{ll} ein Element von $\mathcal{U}$ ist.

Also ist $k = j$ und die Behauptung bewiesen.

B.1.3 Ganters Hüllenalgorithmus

Nach den Vorbereitungen nun der Algorithmus zur Bestimmung der Hüllen:

Algorithmus B1 (*Bestimmung eines Hüllensystems*)

Gegeben sei eine endliche Menge $X = \{x_0, \ldots, x_{n-1}\}$ und ein Hüllenoperator φ auf X.

Der Algorithmus berechnet in der Variablen V sukzessive alle Hüllen von φ in lektischer Reihenfolge.

```
(B1.1)    // die lektisch erste Hülle
          V := φ(∅)
(B1.2)    // Die letzte Hülle ist die Gesamtmenge.
          WHILE (V ≠ X) DO
          {
(B1.3)      // Ab hier: Konstruktion des Nachfolgers
            k := n
(B1.4)      REPEAT
(B1.5)        REPEAT
```

```
(B1.6)            k := k - 1
(B1.7)          UNTIL (x_k ∉ V)
(B1.8)          W := φ((V ∩ {x_0, ..., x_{k-1}}) ∪ {x_k})
(B1.9)        UNTIL (V ∩ {x_0, ..., x_{k-1}} = W ∩ {x_0, ..., x_{k-1}})
(B1.10)       // W ist der lektische Nachfolger
              V := W
            } // Schleife über die Hüllen
```

([G-W 96], 2.1, S. 67)

Die Korrektheit des Algorithmus folgt aus den Vorüberlegungen.

Um eine lektisch nächste Hülle zu berechnen, ermittelt der Algorithmus erst einmal ein Maximum aus (höchstens) n Elementen, probiert dann im schlimmsten Fall n Möglichkeiten für k durch (n sei die Kardinalität der Grundmenge). Für den Test jeder dieser Möglichkeiten muß einmal der Hüllenoperator φ berechnet werden. Die gesamte Schleife wird so oft durchlaufen, wie das berechnete Hüllensystem Elemente besitzt. Neben dem Aufwand $ho(n)$ (hier vereinfachend als von n abhängig darstellbar angenommen) für die Berechnung des Hüllenoperators, ist es wichtig, die Mengenoperationen effizient umzusetzen. Insbesondere ist die Schnittbildung entscheidend, denn der Algorithmus benutzt Vereinigungen nur für den Sonderfall, daß eine der beteiligten Mengen einelementig ist, und die Abfrage auf das Enthaltensein eines Elements kann auch über eine Schnittbildung umgesetzt werden. Implementiert man (wie in *The Formal Concept Analysis Library*) Mengen als Bitvektoren, können die Mengenoperationen effizient als bitweise logische Operatoren implementiert werden. Sie haben dann logarithmischen Aufwand. (Die Basis des Logarithmus ist die Stellenanzahl des zur Speicherung verwendeten Ganzzahltyps, in *The Formal Concept Analysis Library* 32.)

Als Abschätzung für die Komplexität der Berechnung einer Hülle erhält man $O(n \cdot (\log n + ho(n)))$ und $O(n \cdot (\log n + ho(n)) \cdot |\mathcal{U}_\varphi|)$ für das gesamte Hüllensystem. An dieser Stelle sind die Größen noch wenig aussagekräftig, weil $ho(n)$ vollkommen unbekannt ist. Es kann sogar sein, daß die Schreibweise $ho(n)$ grob verkürzend ist, weil der Aufwand zur Berechnung des Hüllenoperators wesentlich von anderen Einflußgrößen abhängt (siehe z.B. B.3 auf Seite 292). Abgesehen von dem uninteressanten Hüllenoperator zu Potenzmenge ist aber zu erwarten, daß der Aufwand der Hüllenberechnung $\log n$ übersteigt, so daß der erste Summand entfällt.

Die auf dem Weg zur lektisch nächsten Hülle verworfenen W waren eventuell nicht die gesuchte nächste Hülle (können es aber gewesen sein, wenn sich für ein kleineres k die selbe Menge ergibt), aber auf jeden Fall waren sie Hüllen, die irgendwann berechnet werden müssen. Christian Lindig schlägt deshalb in [Lin 99] vor, solche Zwischenergebnisse für die weitere Auswertung in einem Cache zu speichern.

Die Abfrage in (B1.9) überprüft, ob der Ausgangsmenge durch den Hüllenoperator ein Element mit einem Index kleiner als k zugeschlagen wurde. Bei aufwendig zu berechnenden Hüllenoperatoren kann diese Bedingung auch schon während ihrer Berechnung überprüft werden, um eventuell die Berechnung vorzeitig abzubrechen, weil nicht sicher die lektisch nächste Hülle bestimmt wird (siehe Algorithmus B8 auf Seite 290 und Abschnitt B.4.1 auf Seite 296).

Für die konkreten Anwendungen sind im folgenden (prinzipiell) nur noch die betreffenden Hüllenoperatoren einzusetzen.

B.2 Berechnung eines Begriffsverbands

Nach Folgerung 3 auf Seite 83 bilden die Begriffsumfänge bzw. -inhalte zu einem formalen Kontext jeweils ein Hüllensystem. Diese Systeme für endliche Kontexte berechnen zu können, ist essentiell für Anwendungen der Formalen Begriffsanalyse. Mit den Begriffsumfängen oder -inhalten sind nach Definition 20 auf Seite 82 auch die formalen Begriffe eines Kontexts vollständig bestimmt. Implizit ist damit auch die Verbandsstruktur über die Ordnung der Begriffsumfänge gegeben, aber noch nicht explizit als Liniendiagramm gespeichert (siehe B.2.2 und B.2.3).

B.2.1 Berechnung der formalen Begriffe

In Folgerung 3 auf Seite 83 sind schon die nötigen Hüllenoperatoren angegeben: $''\colon \mathfrak{P}(G) \to \mathfrak{P}(G)$ für das System der Begriffsumfänge und $''\colon \mathfrak{P}(M) \to \mathfrak{P}(M)$ für die Menge aller Begriffsinhalte.

Um aus Algorithmus B1 einen Algorithmus für die Bestimmung aller Begriffsumfänge oder -inhalte zu erhalten, ist lediglich die Berechnung der $''$-Abbildungen zu erklären. Diese sind wiederum über die $'$-Abbildungen erklärt. Nach Folgerung 2.a) auf Seite 82 gilt $A' = \bigcap_{g \in A} g'$ für $A \subseteq G$ und $B' = \bigcap_{m \in B} m'$ für $B \subseteq M$.

Algorithmus B2 (*Inhalts-/Umfangsberechnung*)

Gegeben sei ein endlicher formaler Kontext (G, M, I) mit $G = \{g_0, \ldots, g_{|G|-1}\}$ und eine Gegenstandsmenge $A \subseteq G$.
Der Algorithmus berechnet A'.

```
(B2.1)    A' := M
(B2.2)    FOR (i:=0) TO (|G| - 1) DO
(B2.3)       IF (g_i ∈ A) THEN
(B2.4)          A' := A' ∩ g_i'
```

Dabei kann g_i' direkt als die in I zu g_i gehörige Tabellenzeile ausgelesen werden.
Für eine Merkmalsmenge $B \subseteq M$ berechnet sich B' analog, nur daß anfangs B' auf G gesetzt wird, die Schleife über $M = \{m_0, \ldots, m_{|M|-1}\}$ läuft und m_i' als Tabellenspalte gelesen wird.
([Gan 87], 4., S. 252)

Mit entscheidend für eine effiziente Implementierung ist die Gestaltung des Zugriffs auf Tabellenzeilen und -spalten der Inzidenzrelation. Deshalb werden – wie schon in 5.4.2 auf Seite 210 erwähnt – Zeilen und Spalten vorteilhafterweise redundant als Bit-Arrays gespeichert. Damit ist der Zugriff auf sie in konstanter Zeit möglich.

Mit der schon oben getroffenen Annahme, daß die Mengenoperationen logarithmischen Aufwand verursachen, hat Algorithmus B2 einen Aufwand von $O(|G| \cdot \log|G|)$ (bzw. $O(|M| \cdot \log|M|)$ bei der Bestimmung eines Umfangs zu einer Merkmalsmenge).

Die Berechnung des Hüllenoperators erfolgt durch zweimalige Anwendung von Algorithmus B2 und hat also den Aufwand $O(n \cdot \log n)$ mit $n = \max\{|G|, |M|\}$.

Algorithmus B3 (*Lektisch nächster Begriffsumfang-/inhalt*)

Gegeben sei ein endlicher formaler Kontext (G, M, I) mit
$G = \{g_0, \ldots, g_{|G|-1}\}$ und $A \subsetneq G$.
Der Algorithmus berechnet in der Variablen `C` den nach A lektisch nächsten Begriffsumfang.

```
          nextExtent(A)
(B3.1)    k := |G|
(B3.2)    REPEAT
(B3.3)       REPEAT
(B3.4)          k := k - 1
(B3.5)       UNTIL (g_k ∉ A)
(B3.6)       C := ((A ∩ {g_0, ..., g_{k-1}}) ∪ {g_k})''
(B3.7)    UNTIL (A ∩ {g_0, ..., g_{k-1}} = C ∩ {g_0, ..., g_{k-1}})
```

Für die Begriffsinhalte ist der Algorithmus `nextIntent` analog, nur daß die Schleife über $M = \{m_0, \ldots, m_{|M|-1}\}$ läuft.
([G-W 96], 2.1, S. 67)

Nach den Vorüberlegungen ist die Komplexität des Algorithmus $O(n^2 \cdot \log n)$ mit $n = \max\{|G|, |M|\}$. Frank Vogt zitiert in [Vog 96], 2.2 Martin Skorsky mit einer Abschätzung von $O(n^3)$. Auch Christian Lindig kommt in [Lin 99], 3.2 auf diese Abschätzung. Diese Autoren scheinen von einer ineffizienteren Mengenimplemen-

tierung auszugehen.

Im Vergleich zu Algorithmus B1 ist für den Fall der Bestimmung aller Begriffsumfänge/-inhalte am Anfang eine kleine Verkürzung möglich, weil $\varnothing' = M$ (bzw. $\varnothing' = G$) ohne Berechnung bekannt sind.

Algorithmus B4 (*Berechnung aller Begriffsumfänge/-inhalte*)

Gegeben sei ein endlicher formaler Kontext (G, M, I) mit $G = \{g_0, \ldots, g_{|G|-1}\}$.

Der Algorithmus berechnet in der Variablen `A` alle Begriffsumfänge.

```
(B4.1)    // der lektisch erste Begriffsumfang
          A := M'
(B4.2)    // Der letzte Umfang ist die Gesamtmenge.
          WHILE (A ≠ G) DO
(B4.3)      A := nextExtent(A)
```

Für die Begriffsinhalte ist der Algorithmus analog, nur daß man mit G' startet und `nextIntent` aufruft.

([G-W 96], 2.1, S. 67)

Nach den Vorüberlegungen ist die Komplexität des Algorithmus $O(n^2 \cdot \log n \cdot |\mathfrak{B}(G, M, I)|)$ mit $n = \max\{|G|, |M|\}$.

Die vollständigen formalen Begriffe erhält man dadurch daß man zu jedem Begriffsumfang den entsprechenden Inhalt berechnet (bzw. zu jedem Begriffsinhalt den entsprechenden Umfang). Dies ist nach oben in $O(n \cdot \log n \cdot |\mathfrak{B}(G, M, I)|)$ möglich, erhöht den Aufwand also asymptotisch nicht.

Im schlechtesten Fall hat Algorithmus B4 ein exponentielles Laufzeitverhalten, weil $\mathfrak{B}(G, M, I)$ $2^{\min\{|G|, |M|\}}$ formale Begriffe enthalten kann.

Satz B2 (*Exponentielle Begriffsanzahl*)

Sind $n \in \mathbb{N}$, $G := M := \{1, \ldots, n\}$ und $I := \{(i, j) \in G \times M \mid i \neq j\}$, so umfaßt $\mathfrak{B}(G, M, I)$ 2^n formale Begriffe.

Beweis: ([Lin 99], 3.1)

Für jede Teilmenge $U \subseteq \{1, \ldots, n\}$ ist $(U, \{1, \ldots, n\} \setminus U)$ ein formaler Begriff.

Erfreulicherweise ist eine exponentielle Begriffsanzahl eine pathologische Ausnahme. Jedoch ist eine bessere Abschätzung anhand des zugrunde liegenden formalen Kontexts schwer. (Bernhard Ganter und Rudolf Wille zitieren in [G-W 96] ein Ergebnis von Dieter Schütt, nach dem $|\mathfrak{B}(G, M, I)| \leq \frac{3}{2} \cdot 2^{\sqrt{|I|+1}} - 1$ gilt, wenn die Inzidenzrelation mehr als 2 Einträge besitzt.) Christian Lindig untersucht in

[Lin 99] (Kapitel 6) die Größe von Begriffsverbänden anhand von Experimenten. Er schließt danach auf eine Approximation des Erwartungswerts für die Begriffsanzahl eines formalen Kontext (G, M, I) in $O(|I|^2)$. Eine interessante Abschätzung stammt von Robert Godin: Wenn für alle betrachteten formalen Kontexte und alle beteiligten formalen Gegenstände die Anzahl der inzidenten Merkmale durch eine Konstante beschränkt ist, dann ist $|\mathfrak{B}(G, M, I)| = O(|G|)$ (siehe [GMs 94]).

B.2.2 Berechnung der Verbandsstruktur

Mit den formalen Begriffen ist auch die Ordnung des Begriffsverbands als Inklusionsordnung auf den Begriffsumfängen festgelegt. Praktisch wird aber ein Begriffsverband in einem Liniendiagramm gespeichert, dessen Kanten zu berechnen sind.

Dies kann als Erweiterung in Algorithmus B4 eingefügt werden. In der folgenden Darstellung wird der Verband anhand der Begriffsumfänge aufgebaut, weil dies in BASE so vorgenommen wird, um die Pseudoumfänge für Gegenstandsimplikationen sofort mit zu berechnen (siehe 5.4.3 auf Seite 214 und B.3 auf Seite 287). In der Originalversion von *The Formal Concept Analysis Library* wird dagegen die lektische Ordnung der Begriffsinhalte betrachtet. Der Algorithmus ist jedoch vollkommen analog.

Mit einem neu gefundenen Begriffsumfang ist auch ein neuer formaler Begriff, d.h. ein Knoten des Liniendiagramms des Begriffsverbands gefunden. Die Knoten werden deshalb in der lektischen Reihenfolge ihrer Begriffsumfänge erzeugt und numeriert. Nach Hilfssatz B2 auf Seite 278 sind damit vor jedem Knoten schon alle unter ihm liegenden berechnet. Aus diesen müssen die unteren Nachbarn (Definition 9 auf Seite 70) bestimmt und mit dem gerade neu erzeugten Knoten durch eine Kante verbunden werden. Um dies effizient zu ermöglichen, speichert der folgende Algorithmus zu jedem Knoten dessen Hauptideal – d.h. die unter ihm liegenden Knoten (siehe Definition 5 auf Seite 66). Da die größten schon erzeugten Knoten nach Hilfssatz B2 die höchsten Nummern tragen, werden die unteren Nachbarn zu einem neu erzeugten Knoten von hinten nach vorn gesucht. Wird ein unterer Nachbar gefunden, so scheiden alle unter ihm liegenden Knoten (im entsprechenden Hauptideal gespeichert) als untere Nachbarn des neu erzeugten aus.

Algorithmus B5 (*Erzeugung der Verbandsstruktur*)

Gegeben sei ein endlicher formaler Kontext (G, M, I) mit
$G = \{g_0, \ldots, g_{|G|-1}\}$.

Der Algorithmus berechnet im Array `umf` alle Begriffsumfänge und in `kanten` die Kanten des Liniendiagramms von $\underline{\mathfrak{B}}(G, M, I)$ als Paare von Knotennummern. Dabei ist zuerst der höhere Knoten angegeben. Die Knoten sind wie ihre Begriffsumfänge in `umf` numeriert.

```
(B5.1)    // der lektisch erste Begriffsumfang/Knoten
          umf[0] := M'
(B5.2)    // bei nur einem Knoten keine Kante
          kanten := ∅
(B5.3)    // Numerierung der Knoten
          knr := 1
(B5.4)    ideale[0] := {0}
(B5.5)    // Der letzte Umfang ist die Gesamtmenge.
          WHILE (𝔸 ≠ G) DO
          {
(B5.6)      // der Begriffsumfang
            umf[knr] = nextExtent(umf[knr-1])
(B5.7)      // Kandidaten für untere Nachbarn
            nbKand := {0,...,knr-1}
(B5.8)      // kleinere Begriffsumfänge/Knoten
            ideale[knr] := {knr}
(B5.9)      i:=knr-1
(B5.10)     WHILE (nbKand ≠ ∅  AND  i ≥ 0)
            {
(B5.11)       // Kommt i noch als unterer Nachbar in Frage und
              // repräsentiert i wirklich einen Unterbegriff?
              IF (i ∈ nbKand  AND  umf[knr] ⊂ umf[i]) THEN
              {
(B5.12)         // Kante gefunden
                kanten := kanten ∪ {(knr,i)}
(B5.13)         // noch kleinere Knoten ausschließen
                nbKand := nbKand \ ideale[i]
(B5.14)         // Ideal ergänzen
                ideale[knr] := ideale[knr] ∪ ideale[i]
              } // Kante gefunden
(B5.15)       i := i - 1
            }
(B5.16)     knr := knr + 1
          } // Schleife über die Begriffsumfänge
```

Für die Begriffsinhalte ist der Algorithmus analog, nur daß man in `(B5.1)` mit G' startet, den Vergleich der Umfänge in `(B5.11)` mit der umgedrehten Inklusion durchführt und in `(B5.12)` die Kanten in der Reihenfolge `(i,knr)` einfügt. Daß Hauptfilter statt Hauptideale betrachtet werden müssen, ändert an der Algorithmusformulierung nichts.
([Vog 96], Quellcode)

Der Aufwand ist $O((n^2 + |\mathfrak{B}(G, M, I)|) \cdot \log n \cdot |\mathfrak{B}(G, M, I)|)$ mit $n = \max\{|G|, |M|\}$, weil zur Bestimmung der Kanten die Diagrammknoten

durchlaufen werden und die dabei vorgenommenen Mengenoperationen einen Aufwand von $O(\log n)$ verursachen. Die Inklusionsbeziehung zwischen zwei Mengen kann über die Schnittbildung getestet werden. Für die Behandlung der Hauptideale (bzw. -filter) ist hier aber auch die Vereinigungsbildung interessant.

In [Lin 99] wird auch ein Algorithmus untersucht, der weniger Knoten darauf testet, ob sie als Nachbarn mit dem aktuell erzeugten verbunden werden müssen. Mehrere verschiedene Algorithmen zur Verbandserzeugung werden in [GMA 95] verglichen.

Nach Durchführung von Algorithmus B5 liegt insbesondere in der Gesamtheit der ermittelten Hauptideale (Hauptfilter bei der Orientierung an den Begriffsinhalten) der formalen Begriffe die Ordnung des Begriffsverbands vor. *The Formal Concept Analysis Library* sieht für diese eine Speicherung im Liniendiagramm vor, um Verbandsoperationen effizient unterstützen zu können (Siehe 5.4.3 auf Seite 213). Im Rahmen der Arbeiten zu BASE wurde aus diesem Grund die Liniendiagrammberechnung in *The Formal Concept Analysis Library* so erweitert, daß die schon (implizit) ermittelte reflexiv-transitive Hülle der Nachbarschaftsrelation auch wirklich im Diagramm gespeichert wird.

B.2.3 Zuordnung der Diagrammbeschriftung

Sind die Begriffsumfänge und -inhalte alle berechnet, können zu allen formalen Gegenständen und Merkmalen die Nummern ihrer Gegenstands- bzw. Merkmalsbegriffe (Definition 23 auf Seite 87) leicht gesucht werden.

Algorithmus B6 (*Zuordnung der Diagrammbeschriftung*)

Gegeben sei ein endlicher formaler Kontext (G, M, I) mit $G = \{g_0, \ldots, g_{|G|-1}\}$.

Weiter seien die Begriffsinhalte bestimmt und nach den zugehörigen Knotennummern numeriert im Array `inh` gegeben.

Der Algorithmus berechnet im Array `ggstBeg` die Nummern der Gegenstandsbegriffe und in `mml` die Nummern der Merkmalsbegriffe.

```
(B6.1)    FOR (i:=0) TO (|G| − 1 ) DO
          {
(B6.2)      knr := 0
(B6.3)      WHILE (inh[knr] ≠ g_i')
(B6.4)        knr := knr + 1
(B6.5)      ggstBeg[i] := knr
          } // Schleife über die Gegenstände
```

Für die Merkmalsbegriffe zu $M = \{m_0, \ldots, m_{|M|-1}\}$ dual über die Begriffsumfänge. ([Vog 96], Quellcode)

Die Laufzeit von Algorithmus B6 ist $O(|G| \cdot \log|G| \cdot |\mathfrak{B}(G, M, I)|)$ für die formalen Gegenstände und $O(|M| \cdot \log|M| \cdot |\mathfrak{B}(G, M, I)|)$ für die formalen Merkmale. (Der logarithmische Faktor ergibt sich aus der Überprüfung der Mengengleichheit.) Der Gesamtaufwand zur Diagrammerstellung wird also asymptotisch nicht erhöht.

B.3 Berechnung der Duquenne-Guigues-Basis

Die auftretenden Implikationenbasen (Satz 14 auf Seite 109) werden ebenfalls mit Ganters Hüllenalgorithmus (Algorithmus B1) berechnet. Das betreffende Hüllensystem ist das folgende:

Satz B3 (*Hüllensystem der Pseudo- und Begriffsinhalte/-umfänge*)
Sei (G, M, I) ein endlicher formaler Kontext.
Dann bildet die Menge aller Pseudo- und Begriffsinhalte von (G, M, I) ein Hüllensystem auf M und dual die Menge aller Pseudo- und Begriffsumfänge von (G, M, I) ein Hüllensystem auf G.
(Pseudoinhalte und -umfänge wurden in Definition 32 auf Seite 109 eingeführt.)
([G-W 96], 2.3, Hilfssatz 26, S. 85)

Beweis: Nach der Bemerkung nach Definition 32 auf Seite 109, Hilfssatz 14 auf Seite 105 und Hilfssatz 13 auf Seite 102 ist der Schnitt von zwei Pseudo- oder Begriffsinhalten ein Pseudo- oder Begriffsinhalt.
Weiter ist $M = \bigcap\varnothing$ ein Begriffsinhalt.
Weil mit M auch $\mathfrak{P}(M)$ endlich ist, sind beliebige Schnitte von Pseudo- oder Begriffsinhalten selber Begriffsinhalte.
Für Pseudo- und Begriffsumfänge analog.

Zu diesem Hüllensystem muß nun der Hüllenoperator gefunden werden, um den Algorithmus zu seiner Erzeugung umsetzen zu können.

Satz B4 (*Hüllenoperator für Begriffs- und Pseudoinhalte/-umfänge*)
Seien (G, M, I) ein endlicher formaler Kontext, $\mathfrak{PI}$ die Menge seiner Pseudoinhalte und $\mathfrak{PU}$ die Menge seiner Pseudoumfänge.
Für $B \subseteq M$ definiere

a) $B^\bullet := B \cup \bigcup\{P'' \mid P \in \mathfrak{PI}, P \subsetneq B\}$.

b) $B_0 := B$ und $B_{k+1} := B_k^\bullet$ für $k \in \mathbb{N}_0$

c) $p_i(B) := \bigcup_{k \in \mathbb{N}_0} B_k$

Für $A \subseteq G$ definiere analog

d) $A^\bullet := A \cup \bigcup \{P'' \mid P \in \mathfrak{PU}, P \subsetneq A\}$.

e) $A_0 := A$ und $A_{k+1} := A_k^\bullet$ für $k \in \mathbb{N}_0$

f) $p_u(A) := \bigcup_{k \in \mathbb{N}_0} A_k$

Dann ist $p_i : \mathfrak{P}(M) \to \mathfrak{PI} \cup \{B'' \mid B \subseteq M\}$ der Hüllenoperator zu $\mathfrak{PI} \cup \{B'' \mid B \subseteq M\}$ und $p_u : \mathfrak{P}(G) \to \mathfrak{PU} \cup \{A'' \mid A \subseteq G\}$ der Hüllenoperator zu $\mathfrak{PU} \cup \{A'' \mid A \subseteq G\}$.
([G-W 96], 2.3, S. 85)

Beweis: Sei $B \subseteq M$ beliebig.

Mit M ist auch $\mathfrak{PI} \subseteq \mathfrak{P}(M)$ endlich.
Deshalb gibt es ein $n \in \mathbb{N}_0$ mit $B_n^\bullet = B_n$.
Mit diesem n gilt also $p_i(B) = B_n$ und es ist $p_i(B)^\bullet = p_i(B)$.
Ist $p_i(B)'' = p_i(B)$, so ist $p_i(B) \in \mathfrak{PI} \cup \{B'' \mid B \subseteq M\}$.
Andernfalls ist $p_i(B)'' \neq p_i(B)$ und für jeden Pseudoinhalt $P \subseteq M$ mit $P \subsetneq p_i(B)$ ist $P'' \subseteq p_i(B)^\bullet = p_i(B)$.
Damit ist $p_i(B) \in \mathfrak{PI}$.

D.h. $p_i(\mathfrak{P}(M)) \subseteq \mathfrak{PI} \cup \{B'' \mid B \subseteq M\}$.

Sei andersherum $B \subseteq M$ ein Begriffsinhalt.
Ist $P \in \mathfrak{PI}$ mit $P \subsetneq B$, so ist nach Folgerung 3 auf Seite 83 $P'' \subseteq B'' = B$.
Also ist $B^\bullet = B$ und so auch $p_i(B) = B$.

Begriffsinhalte lassen sich also als Bilder unter p_i darstellen.

Sei nun noch $Q \in \mathfrak{PI}$ ein Pseudoinhalt.
Dann gilt nach Definition für jeden Pseudoinhalt $P \subseteq Q$ mit $P \subsetneq Q$, daß $P'' \subseteq Q$.
Das bedeutet $Q^\bullet = Q$ und so auch $p_i(Q) = Q$.

Damit ist $\mathfrak{PI} \subseteq p_i(\mathfrak{P}(M))$.
Insgesamt folgt $p_i(\mathfrak{P}(M)) = \mathfrak{PI} \cup \{B'' \mid B \subseteq M\}$.
Nach Konstruktion ist p_i monoton und extensiv.
p_i ist wegen $p_i(B)^\bullet = p_i(B)$ für alle $B \subseteq M$ auch idempotent.
Damit ist p_i der gesuchte Hüllenoperator.
Analog ist p_u der Hüllenoperator zu $\mathfrak{PU} \cup \{A'' \mid A \subseteq G\}$.

Um zu sehen, daß die Charakterisierung der Hüllenoperatoren in Satz B4 deren Berechnung ermöglicht, muß noch einmal genau hingeschaut werden. Zur Berechnung von $p_i(B)$ $(p_u(A))$ sind nur echt kleinere Pseudoinhalte (-umfänge) nötig. Die Berechnung eines Hüllensystems nach Algorithmus B1 verläuft nach der lektischen Ordnung der Hüllen. Bezüglich Inklusion kleinere Hüllen sind nach Hilfssatz B2 auf Seite 278 auch lektisch kleiner. Die nötigen Pseudoinhalte (-umfänge)

sind also schon vorher bestimmt. Zuerst wird ermittelt, ob die leere Menge ein Pseudo- oder Begriffsinhalt (-umfang) ist (siehe Bemerkung 6 auf Seite 101), und dann ausgehend von der leeren Menge jeweils die lektisch nächste Hülle (Pseudo- oder Begriffsinhalt bzw. -umfang) bestimmt. Die Berechnung dazu geschieht folgendermaßen:

Algorithmus B7 (*Lektisch nächster Pseudo-/Begriffsinhalt / -umfang*)

Gegeben sei ein endlicher formaler Kontext (G, M, I) mit $M = \{m_0, \ldots, m_{|M|-1}\}$.

Weiter sei $B \subsetneq M$ und `pInh` speichere die in B enthaltenen Pseudoinhalte.

Der Algorithmus berechnet in `D` den lektisch nächsten Pseudo- oder Begriffsinhalt.

```
          nextIntentPseudointent(B,pInh)
(B7.1)    k := |M|
(B7.2)    REPEAT
(B7.3)      REPEAT
(B7.4)        k := k - 1
(B7.5)      UNTIL (m_k ∉ B)
(B7.6)      D := (B ∩ {m_0, ..., m_k-1}) ∪ {m_k}
(B7.7)      // ab hier: Berechnung des Hüllenoperators
            REPEAT
(B7.8)        D• := D
(B7.9)        // Schleife über die Pseudoinhalte
              FOR (i:=0) TO (|pInh|-1) DO
(B7.10)         IF (pInh[i] ⊊ D) THEN
(B7.11)           D• := D• ∪ {pInh[i]''}
(B7.12)     UNTIL (D = D•)
(B7.13)   UNTIL (D ∩ {m_0, ..., m_k-1}  =  B ∩ {m_0, ..., m_k-1})
```

In `(B7.7)` bis `(B7.12)` wird der Hüllenoperator aus Satz B4 ausgewertet. Der Beweis von Satz B4 garantiert, daß die Repeat-Schleife auch wirklich abbricht.

Für die Pseudoumfänge ist der Algorithmus `nextExtentPseudoExtent` über $G = \{g_0, \ldots, g_{|G|-1}\}$ analog.

([G-W 96], 2.3, S. 85)

Entscheidend für eine Laufzeitabschätzung für Algorithmus B7 ist, wie oft die Repeat-Schleife von `(B7.7)` bis `(B7.12)` durchlaufen wird. Dies kann höchstens $(|M| + 1)$-mal geschehen, denn öfter als $|M|$-mal kann `D`$^\bullet$ nicht innerhalb von M vergrößert werden. In ihrem Inneren werden jedesmal alle Pseudoinhalte getestet und dabei eventuell der minimal umfassende Begriffsinhalt bestimmt (Komplexität

$O(n \cdot \log n)$ nach B.2.1 auf Seite 282). Also verursacht die Hüllenberechnung einen Aufwand von $O(n^2 \cdot \log n \cdot p)$ mit $n = \max\{|G|, |M|\}$ und $p = |\mathfrak{PI}|$ bzw. $p = |\mathfrak{PU}|$. Dabei sind die Kardinalitäten der Pseudoinhaltsmenge $\mathfrak{PI}$ und Pseudoumfangsmenge $\mathfrak{PU}$ schwer durch den zugrunde liegenden Kontext abzuschätzen (siehe auch B.2.1 auf Seite 283). Insgesamt ergibt sich so für Algorithmus B7 eine Komplexität von $O(n^3 \cdot \log n \cdot p)$.

Wegen des hohen Aufwands für die Berechnung des Hüllenoperators scheint es sinnvoll, die Berechnung der Hülle abzubrechen, wenn ein Merkmal (Gegenstand) mit einem Index kleiner als das gerade betrachtete `k` der Hülle zugefügt wurde. In diesem Fall ist nämlich die Bedingung in `(B7.13)` sicher verletzt und es muß sowieso ein kleineres `k` getestet werden. Diese Modifikation ist leicht in Algorithmus B7 einzubauen:

Algorithmus B8 (*Vorzeitiger Abbruch der Hüllenberechnung*)

Gegeben sei ein endlicher formaler Kontext (G, M, I) mit $M = \{m_0, \ldots, m_{|M|-1}\}$.
Weiter sei $B \subsetneqq M$ und `pInh` speichere die in B enthaltenen Pseudoinhalte.
Der Algorithmus berechnet in `D` den lektisch nächsten Pseudo- oder Begriffsinhalt.

`nextIntentPseudointent(B,pInh)`

```
(B8.1)    k := |M|
(B8.2)    REPEAT
(B8.3)      REPEAT
(B8.4)        k := k - 1
(B8.5)      UNTIL (m_k ∉ B)
(B8.6)      D := (B ∩ {m_0, ..., m_{k-1}}) ∪ {m_k}
(B8.7)      // ab hier: Berechnung des Hüllenoperators
            REPEAT
(B8.8)        D• := D
(B8.9)        i:=0
(B8.10)       weiter := TRUE
(B8.11)       // Schleife über die Pseudoinhalte
              WHILE (i < |pInh|  AND  weiter) DO
              {
(B8.12)         IF (pInh[i] ⊊ D) THEN
                {
(B8.13)           D• := D• ∪ {pInh[i]''}
(B8.14)           IF ((D•\B) ∩ {m_0, ..., m_{k-1}}) ≠ ∅) THEN
(B8.15)             weiter := FALSE
                }
(B8.16)         i := i + 1
              }
(B8.17)     UNTIL (NOT weiter  OR  D = D•)
(B8.18)   UNTIL (D ∩ {m_0, ..., m_{k-1}}  =  B ∩ {m_0, ..., m_{k-1}})
```

Der Algorithmus gleicht Algorithmus B7, nur daß durch `(B8.15)` die Hüllenberechnung abgebrochen wird, wenn in `(B8.14)` festgestellt wurde, daß nicht sicher die lektisch nächste Hülle berechnet wird.
Für die Pseudoumfänge ist der Algorithmus `nextExtentPseudoExtent` über $G = \{g_0, \ldots, g_{|G|-1}\}$ analog.

An der asymptotischen Komplexität ändert der Abbruch natürlich nichts.

Algorithmus B9 (*Bestimmung der Duquenne-Guigues-Basis*)

Gegeben sei ein endlicher formaler Kontext (G, M, I) mit $M = \{m_0, \ldots, m_{|M|-1}\}$.
Der Algorithmus berechnet in `pInh` die Pseudoinhalte.

```
(B9.1)    // Anzahl der Pseudoinhalte
          pAnz := 0
(B9.2)    // Ist die leere Menge Pseudoinhalt?
          IF (G' ≠ ∅) THEN
          {
(B9.3)      pInh[pAnz] := ∅
(B9.4)      pAnz := pAnz + 1
          }
(B9.5)    // die lektisch erste Hülle ist die leere Menge
          B := ∅
(B9.6)    // Die letzte Hülle ist die Gesamtmenge.
          WHILE (B ≠ M) DO
          {
(B9.7)      B := nextIntentPseudointent(B,pInh)
(B9.8)      // Wurde ein Pseudoinhalt gefunden?
            IF (B'' ≠ B)
            {
(B9.9)        pInh[pAnz] := B
(B9.10)       pAnz := pAnz + 1
            }
          } // Schleife über Begriffs- und Pseudoinhalte
```

Nach der Überprüfung der leeren Menge in `(B9.2)` bis `(B9.4)` wird in `(B9.5)` bis `(B9.10)` Ganters Hüllenalgorithmus abgearbeitet. Ist die lektisch nächste Hülle berechnet, muß in `(B9.8)` noch entschieden werden, ob es sich um einen Begriffs- oder Pseudoinhalt handelt.
Für die Pseudoumfänge ist der Algorithmus über $G = \{g_0, \ldots, g_{|G|-1}\}$ analog, nur für die Überprüfung, ob die leere Menge ein Pseudoumfang ist, muß dann in `(B9.2)` M' berechnet und zur Berechnung der nächsten Hülle in `(B9.7)` `nextExtentPseudoExtent` aufgerufen werden.
([G-W 96], 2.3, S. 85)

Bezeichnet $\mathfrak{PI}$ die Menge der Pseudoinhalte und $\mathfrak{PU}$ die der Pseudoumfänge und sind $n = \max\{|G|, |M|\}$ und $p = |\mathfrak{PI}|$ bzw. $p = |\mathfrak{PU}|$, ergibt für sich Algorithmus B9 eine Komplexität von $O(n^3 \cdot \log n \cdot p^2)$.

Die Duquenne-Guigues-Implikationenbasis erhält man aus dem System der Pseudoinhalte, indem man die Implikationen mit einem Pseudoinhalt als Prämisse und dem nächstgrößeren Begriffsinhalt als Konklusion bildet (Satz 14 auf Seite 109, für Pseudoumfänge sinngemäß).

Im besonderen berechnet Algorithmus B9 neben den Pseudoinhalten (-umfängen) auch die Begriffsinhalte (-umfänge). Will man also zu einem formalen Kontext nicht nur ein Liniendiagramm erzeugen, sondern auch eine Implikationenbasis, so kann Algorithmus B9 den Algorithmus B4 zur Erzeugung aller Begriffsinhalte bzw. -umfänge ersetzen. In dem in Kapitel 5 vorgestellten Werkzeug werden so die Pseudoumfänge zu formalen Kontexten immer gleich mit erzeugt (siehe 5.4.3 auf Seite 214).

Der angegebene Algorithmus wurde innerhalb der Arbeiten zu BASE für Pseudoinhalte und -umfänge als Erweiterung in *The Formal Concept Analysis Library* integriert. Einmal wurden dazu in der Klasse *TFormalContext* (siehe 5.4.2) zwei neue Operationen zur Berechnung der Pseudoinhalte bzw. -umfänge eingefügt, zum anderen wurde bei der Liniendiagrammberechnung die Möglichkeit ergänzt, über einen Parameter die Erzeugung der Duquenne-Guigues-Basis über Pseudoinhalte oder -umfänge zu aktivieren. (Zu weiteren Funktionen des schon vorher vorhandenen Parameters siehe Algorithmus B12 auf Seite 313.) Wählt man die Erzeugung der Pseudoumfänge, so werden die formalen Begriffe in der lektischen Reihenfolge ihrer Umfänge und nicht – wie sonst in *The Formal Concept Analysis Library* – in der lektischen Reihenfolge ihrer Inhalte berechnet. Um die Konsistenz mit anderen Teilen von *The Formal Concept Analysis Library* herzustellen, wird diese Reihenfolge am Ende nur umgedreht, so daß höhere Diagrammknoten immer kleinere Nummern haben. Die entstehende Numerierung entspricht in der Regel nicht der lektischen Ordnung der Begriffsinhalte. Dennoch ist sie für andere Algorithmen genauso hilfreich, weil sie genauso im Sinne von Hilfssatz B2 auf Seite 278 mit der Inklusionsordnung verträglich ist:

Hilfssatz B3 (*Umgedrehte lektische Ordnung* ↔ *Inklusionsordnung*)

Sei (G, M, I) ein endlicher formaler Kontext mit $G = \{g_0, \ldots, g_{|G|-1}\}$. Weiter sei $\underline{\mathfrak{B}}(G, M, I) = \{(A_0, B_0), \ldots, (A_{n-1}, B_{n-1})\}$ und die formalen Begriffe seien in der umgekehrten lektischen Reihenfolge ihrer Umfänge numeriert.

Dann folgt aus $(A_i, B_i) < (A_j, B_j)$, daß $i > j$.

Beweis: Es seien $i, j \in \{0, \ldots, n-1\}$ mit $(A_i, B_i) < (A_j, B_j)$.

Das bedeutet $A_i \subsetneq A_j$.
Nach Hilfssatz B2 auf Seite 278 folgt $A_i < A_j$ bzgl. der lektischen Ordnung der Begriffsumfänge.
Nach Voraussetzung gilt also $n - 1 - i < n - 1 - j$ und so $i > j$.

B.4 Berechnung von Blockzerlegungen

Die Berechnung der Blockrelationen (Definition 33 auf Seite 112) eines gegebenen formalen Kontexts ist eine weitere Anwendung von Ganters Hüllenalgorithmus (Algorithmus B1). Sie ist in B.4.1 erklärt. Wie man in BASE daraus die in 4.6 und 5.1.2 dargestellten Liniendiagramme berechnet, ist B.4.2 zu entnehmen. Abschnitt B.4.3 untersucht (empirisch) die Existenz interessanter Blockrelationen.

B.4.1 Erzeugung der Blockrelationen

Wieder muß zu dem Hüllensystem der Blockrelationen ein Hüllenoperator gefunden werden.

Satz B5 (*Hüllenoperator für Blockrelationen*)
Seien (G, M, I) ein endlicher formaler Kontext und $\mathfrak{BR}$ die Menge seiner Blockrelationen.
Für $J \subseteq G \times M$ definiere
a) $J^b := I \cup J \cup \bigcup_{g \in G} (\{g\} \times g^{JII}) \cup \bigcup_{m \in M} (m^{JII} \times \{m\})$.
b) $J_0 := J$ und $J_{k+1} := J_k^{\,b}$ für $k \in \mathbb{N}_0$
c) $b(J) := \bigcup_{k \in \mathbb{N}_0} J_k$
Dann ist $b : \mathfrak{P}(G \times M) \to \mathfrak{BR}$ der Hüllenoperator zu $\mathfrak{BR}$.
(Vergl. [Lin 99], 3.9)

Beweis: Sei für das folgende $J \subseteq G \times M$ beliebig.
Mit G und M ist auch $\mathfrak{P}(G \times M)$ endlich.
Deshalb gibt es ein $n \in \mathbb{N}_0$ mit $J_n^{\,b} = J_n$
Mit diesem n gilt also $b(J) = J_n$ und es ist $b(J)^b = b(J)$.
Aus $b(J)^b = b(J)$ folgt $b(b(J)) = b(J)$.
Also ist die Abbildung b idempotent.
Nach Definition ist b extensiv.
Sei $R \subseteq G \times M$ mit $J \subseteq R$.
Dann gilt für $g \in G$, daß

$$g^J = \{m \in M \mid (g, m) \in J \ \ \forall g \in A\}$$
$$\subseteq \{m \in M \mid (g, m) \in R \ \ \forall g \in A\} = g^R$$

Nach Folgerung 3 auf Seite 83 folgt daraus $g^{JII} \subseteq g^{RII}$.
Analog erhält man $m^{JII} \subseteq m^{RII}$ für alle $m \in M$.
Also ist $J^b \subseteq R^b$ und so induktiv auch $b(J) \subseteq b(R)$.

Somit ist b monoton und insgesamt ein Hüllenoperator auf $G \times M$.
Nach Konstruktion ist $I \subseteq J^b \subseteq b(J)$.
Nach oben ist

$$b(J)^b = I \cup b(J) \cup \bigcup_{g \in G} (\{g\} \times g^{b(J)II}) \cup \bigcup_{m \in M} (m^{b(J)II} \times \{m\}) = b(J)$$

Für alle $g \in G$ gilt demnach $g^{b(J)II} \subseteq g^{b(J)}$, also nach Folgerung 2.c) auf Seite 82 $g^{b(J)II} = g^{b(J)}$.

Also ist $g^{b(J)}$ ein Begriffsinhalt von (G, M, I).
Analog ist $m^{b(J)}$ für jedes $m \in M$ ein Begriffsumfang von (G, M, I).
Damit ist $b(J)$ eine Blockrelation von (G, M, I) und $b(\mathfrak{P}(G \times M)) \subseteq \mathfrak{BR}$.
Sei nun andersherum J eine Blockrelation von (G, M, I).

Dann gilt $I \subseteq J$ und für jedes $g \in G$ ist g^J ein Begriffsinhalt von (G, M, I), d.h. nach Hilfssatz 5 auf Seite 83 $g^{JII} = g^J$, und ebenso gilt $m^{JII} = m^J$ für alle $m \in M$.

Es folgt $b(J) = I \cup J \cup \bigcup_{g \in G} (\{g\} \times g^{JII}) \cup \bigcup_{m \in M} (m^{JII} \times \{m\}) = J$.
D.h. insgesamt $b(\mathfrak{P}(G \times M)) = \mathfrak{BR}$.

Aus Satz B5 kann man eine Implementierung für den Hüllenoperator zum System der Blockrelationen ablesen:
Sei (G, M, I) der betrachtete formale Kontext und J eine beliebige Relation zwischen G und M. Um die lektisch kleinste Blockrelation von (G, M, I) zu berechnen, in der J enthalten ist, trage man in der Kreuztabelle zu J zusätzlich alle Kreuze von I ein und fülle dann die Tabellenzeilen und -spalten bis zum kleinsten sie umfassenden Begriffsinhalt bzw. -umfang des Ausgangskontexts I auf. Zur Berechnung aller Blockrelationen startet man mit der kleinsten – also der Inzidenzrelation des Ausgangskontexts – und betrachtet weiter nur größere Relationen. Deshalb braucht man dabei die Vereinigung mit der Inzidenzrelation nicht zu berücksichtigen.

Zu klären ist jetzt noch, wie die Numerierung der Elemente von $G \times M$ vorgenommen werden soll. In der vorgenommenen Implementierung werden sie "zeilenweise" durchnumeriert. Sind also $G = \{g_0, \ldots, g_{|G|-1}\}$ und $M = \{m_0, \ldots, m_{|M|-1}\}$, so erhält (g_i, m_j) die Nummer $i \cdot |G| + j$. Damit kann der Hüllenalgorithmus für die Blockrelationen formuliert werden:

Algorithmus B10 (*Berechnung der Blockrelationen*)

Gegeben sei ein endlicher formaler Kontext (G, M, I) mit $G = \{g_0, \ldots, g_{|G|-1}\}$ und $M = \{m_0, \ldots, m_{|M|-1}\}$.

Der Algorithmus berechnet in J die Blockrelationen von (G, M, I).

```
(B10.1)   // die lektisch erste Blockrelation
          J := I
(B10.2)   // Die letzte Blockrelation ist die volle Relation.
          WHILE (J ≠ G×M) DO
          {
(B10.3)     // Ab hier: Konstruktion des Nachfolgers
            k := |G| · |M|
(B10.4)     REPEAT
(B10.5)       REPEAT
(B10.6)         k := k - 1
(B10.7)         z := k DIV |G|
(B10.8)         s := k MOD |G|
(B10.9)       UNTIL ((g_z, m_s) ∉ J)
(B10.10)      // vor (g_z, m_s) gelegene Elemente
              VB :=
          ({g_0, ..., g_{z-1}} × {m_0, ..., m_{|M|-1}} ∪ {g_z} × {m_0, ..., m_{s-1}})
(B10.11)      R := (J ∩ VB) ∪ {(g_z, m_s)} ∪ I
(B10.12)      // ab hier: Berechnung des Hüllenoperators
              REPEAT
(B10.13)        R^b := R
(B10.14)        // Schleife über die Zeilen
                FOR (i:=0) TO (|G| - 1) DO
(B10.15)          R^b := R^b ∪ {g} × g^JII
(B10.16)        // Schleife über die Spalten
                FOR (i:=0) TO (|M| - 1) DO
(B10.17)          R^b := R^b ∪ m^JII × {m}
(B10.18)      UNTIL (R = R^b)
(B10.19)    UNTIL (R ∩ VB =  J ∩ VB)
(B10.20)    // R ist der lektische Nachfolger
            J := R
          } // Schleife über die Blockrelationen
```

Die Anweisung (B10.10) führt VB als Abkürzung für die Menge der vor der Nummer k liegenden Elemente ein. Diese lassen sich in der zweidimensionalen Grundmenge schlechter charakterisieren als in den zuvor betrachteten Hüllenalgorithmen.

Beim Verkürzen von R auf vor k liegende Elemente in (B10.11), dürfen in der Kreuztabelle keine Kreuze gelöscht werden, die schon in der originalen Inzidenzrelation enthalten waren. Deshalb wird I noch einmal R zugeschlagen.

In (B10.12) bis (B10.18) werden Kreuze bis zur nächsten umfassenden

Blockrelation ergänzt.
(Vergl. [Lin 99], 3.9)

Für eine Laufzeitabschätzung ist wichtig, wie oft die Schleife von `(B10.12)` bis `(B10.18)` durchlaufen wird. In ihr werden Kreuze in der Relationstabelle ergänzt, die noch nicht in der Inzidenzrelation des Ausgangskontext enthalten waren. Solche Ergänzungen können höchstens $(|G| \cdot |M| - |I|)$-mal vorgenommen werden. So erhält man eine Laufzeitabschätzung von $O(n^6 \cdot \log n \cdot |\mathfrak{BR}|)$, wenn $n = \max\{|G|, |M|\}$ und $\mathfrak{BR}$ die Menge der Blockrelationen von (G, M, I) ist.

Werden in `(B10.15)` oder `(B10.17)` Kreuze vor der Stelle `(z,s)` ergänzt, kann die Hüllenberechnung abgebrochen werden. Dies erscheint lohnend, weil die Hüllenberechnung sehr aufwendig ist. Ein solcher Abbruch kann analog zu Algorithmus B8 vorgenommen werden. Deshalb ist diese Lösung hier nicht noch einmal dargestellt.

In der vorgenommenen Implementierung ist der Abbruch vorgesehen und auch die Ergänzung von Kreuzen anders vorgenommen. Es reicht nämlich, solche Zeilen und Spalten auf eine nötige Ergänzung zu überprüfen, die in `R` gegenüber `J` verändert wurden. (Dies trifft in der Regel auch auf die über `(z,s)` liegenden zu, weil dort Kreuze entfernt wurden.) Diese Handhabung ermöglicht auch einen leichten Test, ob die Hüllenbildung schon abgeschlossen ist: Das ist dann der Fall, wenn keine Zeilen und Spalten mehr zur Überprüfung anstehen.

Algorithmus B10 wurde in *The Formal Concept Analysis Library* eingebracht, indem die Klasse *TFormalContext* eine zusätzliche Operation erhalten hat, die die Blockrelationen als Array von formalen Kontexten berechnet. Dazu benutzt sie eine neue Operation von *TRelation*, die den eigentlichen Algorithmus umsetzt und ein Array von Relationen erstellt. (Zu *TFormalContext* und *TRelation* siehe 5.4.2.)

An dieser Stelle sei Bernhard Ganter und Christian Lindig gedankt. Anhand ihrer Anmerkungen ist die Implementierung von Algorithmus B10 entstanden.

B.4.2 Erzeugung der Blockzerlegungsdiagramme

Sind die Blockrelationen zu einem formalen Kontext (G, M, I) mit Hilfe von Algorithmus B10 bestimmt, sind durch sie neue formale Kontexte gegeben. (Die beiden trivialen Blockrelationen I und $G \times M$ sind jedoch nicht von Interesse, siehe Bemerkung 8 auf Seite 113.) Zu den neuen Kontexten werden in BASE wieder Liniendiagramme erzeugt. Um diese sofort in einer befriedigenden Anordnung bereitzustellen, werden die Knotenkoordinaten nicht wie in B.5 erläutert bestimmt, sondern ein Knoten des neuen Diagramms – der ja einem ganzen Block (Satz 18 auf Seite 121) im alten Liniendiagramm entspricht – wird in den Mittelpunkt die-

ses Blocks gelegt. Nach Hilfssatz 18 auf Seite 120 und Satz 18 auf Seite 122 ist die so entstehende vertikale Anordnung der Knoten des Blockrelationsdiagramms korrekt. Zu einem Blockrelationskontext ist also nach der Berechnung seines Begriffsverbands zu jedem formalen Begriff der entsprechende Block im ursprünglichen Diagramm zu ermitteln. Dies ermöglicht dann auch, die Blöcke innerhalb des Originaldiagramms darzustellen wie in Abbildung 5.5 auf Seite 196 gezeigt.

Wie zu einem formalen Begriff eines Blockrelationskontexts der zugehörige Block ermittelt werden kann, zeigen Satz 18 auf Seite 121 und Folgerung 5 auf Seite 88:

- Ist J eine Blockrelation zum formalen Kontext (G, M, I), so ist zu $(A, B) \in \underline{\mathfrak{B}}(G, M, J)$ das Intervall $[(B^I, B), (A, A^I)]$ der zugehörige Block in $\underline{\mathfrak{B}}(G, M, I)$.
- Dabei ist $(B^I, B) = \bigwedge_{m \in B} \mu m = \bigwedge_{m \in B} (m^I, m^{II})$ und $(A, A^I) = \bigvee_{g \in A} \gamma g = \bigvee_{g \in A} (g^{II}, g^I)$

Damit sind zu einem formalen Begriff des Blockrelationsverbands Infimum und Supremum bestimmt. Wie die Umsetzung dieser Vorschrift konkret aussieht, hängt stark von der benutzten Datenstruktur für Liniendiagramme ab. Im Falle von einer Datenstruktur wie in *The Formal Concept Analysis Library*, die

- zu Liniendiagrammen Knoten und Kanten speichert,
- zu formalen Gegenständen jeweils den Knoten des entsprechenden Gegenstandsbegriffs und zu formalen Merkmalen jeweils den Knoten des entsprechenden Merkmalsbegriffs vermerkt,
- die Verbandsordnung als reflexiv-transitive Hülle des Liniendiagramms enthält,

sieht der Algorithmus folgendermaßen aus:

Algorithmus B11 (*Blockinfimum und -supremum*)

Gegeben sei ein endlicher formaler Kontext (G, M, I) mit $G = \{g_0, \ldots, g_{|G|-1}\}$ und $M = \{m_0, \ldots, m_{|M|-1}\}$ und außerdem sein Begriffsverband $\underline{\mathfrak{B}}(G, M, I) = \{k_0, \ldots, k_{n_I-1}\}$ als Liniendiagramm.
Weiter sei J eine Blockrelation von (G, M, I) und $\underline{\mathfrak{B}}(G, M, J) = \{b_0, \ldots, b_{n_J-1}\}$ ebenfalls als Liniendiagramm gegeben.
(Vereinfachend sind in der folgenden Formulierung Begriffsverbände und Liniendiagramme bzw. formale Begriffe und Diagrammknoten gleichgesetzt.)
Der Algorithmus berechnet in `inf` und `sup` Infima und Suprema der nach Satz 18 auf Seite 121 zu J zugehörigen Blöcke in $\underline{\mathfrak{B}}(G, M, I)$.
Zu $x \in G \cup M$ und einem Begriffsverband V ermittele `kn(`x`,` V`)` den Gegenstands- bzw. Merkmalsbegriff von x in V (nach Annahme mit Gegen-

stand/Merkmal gespeichert).
Für einen Begriffsverband V und $v \in V$ werde durch `ideal(`v`,`V`)` das Hauptideal und durch `filter(`v`,`V`)` der Hauptfilter von v in V geliefert (nach Annahme in reflexiv-transitiver Hülle gespeichert). (Ebenso ist die Überprüfung der Ordnung $\leq_V$ mit Hilfe der reflexiv-transitiven Hülle möglich.)

```
(B11.1)   // alle über den Infima liegenden Merkmalsbegriffe
          FOR (r:=0) TO (n_J − 1) DO
(B11.2)     mmlKn[r] := ∅
(B11.3)   // für Merkmalsbegriffe in 𝔅(G, M, J)
          FOR (i:=0) TO (|M| − 1) DO
(B11.4)     mmlKn[kn(m_i, 𝔅(G, M, J))] :=
              mmlKn[kn(m_i, 𝔅(G, M, J))] ∪ {kn(m_i, 𝔅(G, M, I))}
(B11.5)   // gesamte Begriffsinhalte in 𝔅(G, M, J)
          FOR (r:=0) TO (n_J − 1) DO
(B11.6)     FOR (s:=0) TO (n_J − 1) DO
(B11.7)       IF (b_r ≤_𝔅(G,M,J) b_s) THEN
(B11.8)         mmlKn[r] := mmlKn[r] ∪ mmlKn[s]
(B11.9)   // für untere Schranken
          FOR (r:=0) TO (n_J − 1) DO
(B11.10)    us[r] := {k_0, ..., k_n−1}
(B11.11)  FOR (r:=0) TO (n_J − 1) DO
          {
(B11.12)    // untere Schranken
            FOR (s:=0) TO (n_I − 1) DO
(B11.13)      IF (k_s ∈ mmlKn[r]) THEN
(B11.14)        us[r] := us[r] ∩ ideal(k_s, 𝔅(G, M, I))
(B11.15)    // größte untere Schranke
            FOR (s:=0) TO (n_I − 1) DO
(B11.16)      IF (k_s ∈ us[r]) THEN
(B11.17)        us[r] := us[r] ∩ filter(k_s, 𝔅(G, M, I))
(B11.18)    // us[r] enthält nur noch das Infimum
            s := 0
(B11.19)    WHILE (k_s ∉ us[r]) DO
(B11.20)      s := s + 1
(B11.21)    inf[r] := k_s
          }
(B11.22)  // dasselbe zur Supremaermittlung über die Ggst.
          // alle Gegenstandsbegriffe unter den Suprema
          FOR (r:=0) TO (n_J − 1) DO
(B11.23)    ggstKn[r] := ∅
```

```
(B11.24)  // für Gegenstandsbegriffe in 𝔅(G, M, J)
          FOR (i:=0) TO (|G| − 1) DO
(B11.25)    ggstKn[kn(g_i, 𝔅(G, M, J))] :=
              ggstKn[kn(g_i, 𝔅(G, M, J))] ∪ kn(g_i, 𝔅(G, M, I))
(B11.26)  // gesamte Begriffsumfänge in 𝔅(G, M, J)
          FOR (r:=0) TO (n_J − 1) DO
(B11.27)    FOR (s:=0) TO (n_J − 1) DO
(B11.28)      IF (b_s ≤_𝔅(G,M,J) b_r) THEN
(B11.29)        ggstKn[r] := ggstKn[r] ∪ ggstKn[s]
(B11.30)  // für obere Schranken
          FOR (r:=0) TO (n_J − 1) DO
(B11.31)    os[r] := {k_0, ..., k_{n−1}}
(B11.32)  FOR (r:=0) TO (n_J − 1) DO
          {
(B11.33)    // obere Schranken
            FOR (s:=0) TO (n_I − 1) DO
(B11.34)      IF (k_s ∈ mmlKn[r]) THEN
(B11.35)        os[r] := os[r] ∩ filter(k_s, 𝔅(G, M, I))
(B11.36)    // kleinste obere Schranke
            FOR (s:=0) TO (n_I − 1) DO
(B11.37)      IF (k_s ∈ os[r]) THEN
(B11.38)        os[r] := os[r] ∩ ideal(k_s, 𝔅(G, M, I))
(B11.39)    // os[r] enthält nur noch das Supremum
            s := n
(B11.40)    WHILE (k_s ∉ os[r]) DO
(B11.41)      s := s - 1
(B11.42)    sup[r] := k_s
          }
```

In (B11.1) bis (B11.8) werden zu jedem formalen Begriff $b \in \underline{\mathfrak{B}}(G, M, J)$ zunächst alle Merkmalsbegriffe aus $\underline{\mathfrak{B}}(G, M, I)$ ermittelt, deren zugehörigen formale Merkmale im Inhalt von b enthalten sind. In (B11.3) und (B11.4) werden dazu erst einmal nur Merkmale erfaßt, deren Merkmalsbegriff in $\underline{\mathfrak{B}}(G, M, J)$ gerade b ist. In (B11.5) bis (B11.8) werden dann noch die von Oberbegriffen von b zugeschlagen.
Von diesen Begriffen ist das Infimum zu bilden. Dazu werden in (B11.9) bis (B11.14) die unteren Schranken durch den Schnitt der Hauptideale der Begriffe bestimmt. Danach wird in (B11.15) bis (B11.17) deren Maximum bestimmt. Dieses muß dann in (B11.18) bis (B11.21) nur noch aus der Mengennotation bestimmt werden.
Für das Supremum ist das Vorgehen dual.

Für die Laufzeitabschätzung sei angenommen, daß die Operationen `kn`, `ideal`, `filter` und $\leq_V$ konstanten Aufwand verursachen. (Dies ist der Fall, wenn die reflexiv-transitive Hülle schon vorberechnet ist (bei der Erstellung des Liniendiagramms geleistet, siehe B.2.2) und sie wie zu Algorithmus B2 erläutert zeilen- und spaltenweise redundant als Bit-Arrays gespeichert wird.) Weiter werden die Mengenoperationen wieder als in logarithmischer Zeit durchführbar angenommen. Dann ist der Gesamtaufwand für Algorithmus B11 $O(n_J \cdot n_I \cdot \log n_I + |G| + |M|)$, wenn $n_I = |\underline{\mathfrak{B}}(G, M, I)|$ und $n_J = |\underline{\mathfrak{B}}(G, M, J)|$. Die entscheidenden Teile sind die Infimums- und Supremumsbestimmung in `(B11.11)` bis `(B11.21)` bzw. `(B11.32)` bis `(B11.42)`. Ihr Aufwand ist durch den ersten Summanden $n_J \cdot n_I \cdot \log n_I$ beschrieben. Weil $n_J \leq n_I$ gilt, wird durch ihn auch der Aufwand zu Bestimmung der Begriffsinhalte bzw. -umfänge majorisiert (`(B11.5)` bis `(B11.8)` bzw. `(B11.26)` bis `(B11.29)`). Die anderen beiden Summanden entstehen durch die Zuordnung von Merkmals- bzw. Gegenstandsbegriffen von $\underline{\mathfrak{B}}(G, M, I)$ zu solchen von $\underline{\mathfrak{B}}(G, M, I)$ (`(B11.3)`, `(B11.4)`, `(B11.24)` und `(B11.25)`).

Wenn man weiß, daß Knoten und Kanten des Blockrelationsdiagramms in der lektischen Ordnung der Begriffsinhalte vorliegen, kann die Doppelschleife in `(B11.5)` bis `(B11.8)` durch eine einzelne Schleife über die Kanten (in aufsteigender Reihenfolge) und die Doppelschleife in `(B11.26)` bis `(B11.29)` durch eine einzelne Schleife über die Kanten (in absteigender Reihenfolge) ersetzt werden, was auch die if-Abfrage überflüssig macht. Weiß man weiter, daß auch die Knoten von $\underline{\mathfrak{B}}(G, M, I)$ in lektischer Reihenfolge der Begriffsinhalte oder umgedrehter lektischer Reihenfolge der Begriffsumfänge (siehe Hilfssatz B3 auf Seite 292) gespeichert sind, so können die Maximumsbildung in `(B11.15)` bis `(B11.17)` und die Minimumsbildung in `(B11.36)` bis `(B11.38)` entfallen, weil das Maximum die kleinste und das Minimum die größte Nummer tragen.

Sind zu einem Block dessen Infimum und Supremum bekannt, können alle dem Block zugehörigen Knoten durch den Schnitt über den Hauptfilter des Infimums und das Hauptideal des Supremums leicht ermittelt werden. Deshalb reicht es, jeweils nur Infimum und Supremum zu speichern.

In BASE wird zur Erstellung eines Blockrelationsdiagramms wie folgt vorgegangen:

- Aus dem mit der Blockrelation erstellten Kontext wird ein Liniendiagramm ohne Koordinateninformationen erzeugt. (Alle Knoten tragen die Koordinaten (0,0). Eine entsprechende Operation wurde in die Klasse *TFormalContext* (siehe 5.4.2) von *The Formal Concept Analysis Library* eingefügt.)
- Zu jedem Knoten des entstandenen Liniendiagramms werden mit Algorithmus B11 Infimum und Supremum des zugehörigen Blocks berechnet und wie in 5.4.3

auf Seite 215 beschrieben gespeichert.
- Jedem Knoten wird als Position der Mittelpunkt zwischen dem Supremum und Infimum seines Blocks zugeordnet.

(Für diese beiden letzten Schritte wurde der Klasse *TLineDiagram* (siehe 5.4.3) aus *The Formal Concept Analysis Library* eine neue Operation zugefügt.)

B.4.3 Experimente zur Anzahl interessanter Blockrelationen

Wie schon in 4.6 auf Seite 176 angedeutet, besitzt nicht jeder formale Kontext interessante Blockrelationen (also andere als nur die Inzidenzrelation selber und ganz $G \times M$). Es ist Kontexten auch nicht leicht anzusehen, ob zu ihnen nicht-triviale Blockrelationen existieren. Aufgrund von Satz 19 auf Seite 121 ist es naheliegend, Liniendiagramme von Begriffsverbänden – und nicht die zugrundeliegenden formalen Kontexte – danach zu beurteilen, ob sie interessante Blockzerlegungen zulassen. Nach [G-W 96] (3.4, Satz 16, S. 125) ist eine Familie paarweise verschiedener Intervalle auf einem vollständigen Verband genau dann das System der Blöcke zu einer vollständigen Toleranzrelation auf einem vollständigen Unterverband (gebildet als Vereinigung der betrachteten Intervalle), wenn die oberen Intervallgrenzen $\bigwedge$-abgeschlossen, die unteren Intervallgrenzen $\bigvee$-abgeschlossen und untere und obere Intervallgrenzen gleich geordnet sind. (Die letzte Bedingung bedeutet für zwei Intervalle $[u_1, o_1], [u_2, o_2]$, daß $u_1 \leq u_2 \Leftrightarrow o_1 \leq o_2$.) Dadurch, daß Kandidaten für Blockzerlegungen einzeln getestet werden müssen, ist das Kriterium leider sehr unhandlich.

Aus oben angeführten Überlegungen heraus wurden einige Experimente durchgeführt, um einen Eindruck zu bekommen, ob Benutzer von BASE erwarten können, daß ihnen interessante Modularisierungen vorgeschlagen werden. Für solche Tests wäre es optimal, die Isomorphieklassen von Begriffsverbänden aufzuzählen und für jede Klasse einmal die Anzahl der nicht-trivialen Blockrelationen zu bestimmen. Einfacher als Begriffsverbände aufzuzählen ist aber, sich an formalen Kontexten zu orientieren. Spätestens die Ermittlung der Isomorphieklassen ist sehr aufwendig. Deshalb wurden für die Tests formale Kontexte betrachtet.

In einer Testreihe wurden alle quadratische Relationen aufgezählt, deren Hautdiagonale besetzt ist, die keine Vollzeilen -oder Spalten aufweisen und die keine doppelten Zeilen oder Spalten enthalten. Dies war bis zur Größe 5×5 problemlos möglich. (Bei einer Tabellengröße von 5×5 gibt es bereits $2^{5 \cdot 5} = 33554432$ Kontexte.) Sinnvoll wäre auch die alleinige Betrachtung von reduzierten Kontexten (zu denen alle Gegenstandsbegriffe $\bigvee$-irreduzibel und alle Merkmalsbegriffe $\bigwedge$-irreduzibel sind) zu betrachten. Dann hätte man aber die bequeme Einschränkung auf quadratische Kontexte fallen lassen müssen, um nicht nur Begriffsver-

bände mit gleicher Anzahl von $\bigvee$-irreduziblen und $\bigwedge$-irreduziblen Begriffen zu betrachten.

Bei einer Tabellengröße von 5×5 existieren 389310 formale Kontexte, die oben genannte Kriterien erfüllen. Von diesen weisen 94,89% interessante Blockrelationen auf. Die Verteilung der Anzahl interessanter Blockrelationen ist folgendem Diagramm zu entnehmen.

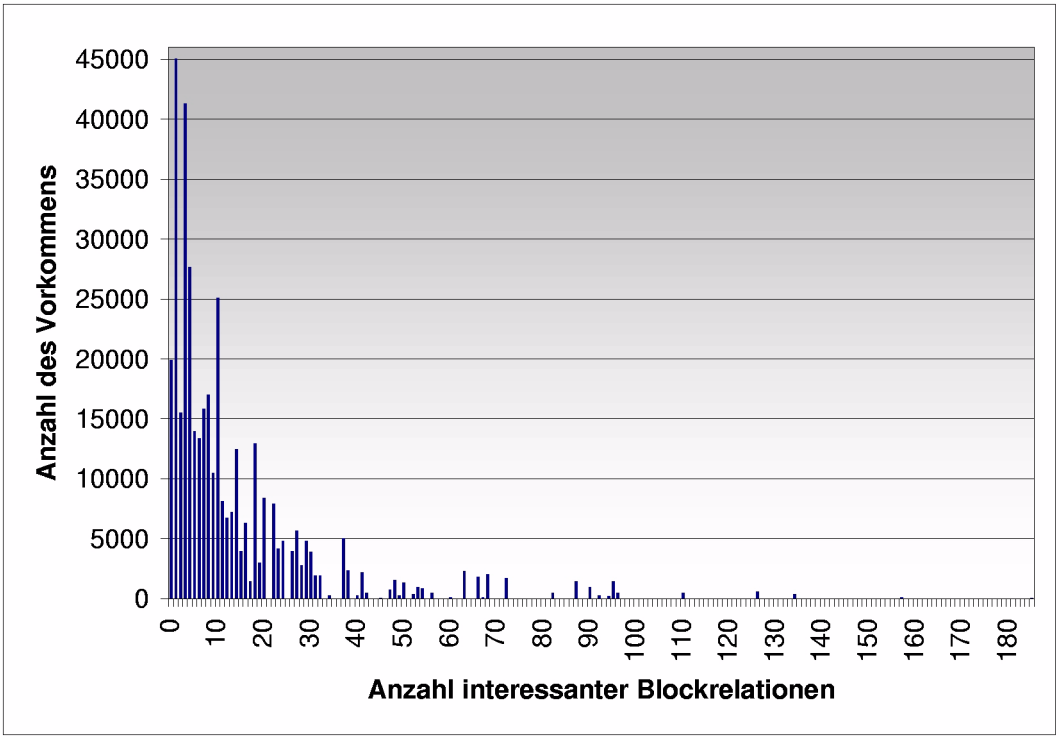


Abb. B.1 Verteilung von Blockrelationsanzahlen

Bei so kleinen Kontexten ist nach den ermittelten Ergebnissen damit zu rechnen, daß interessante Blockrelationen existieren, daß davon aber übermäßig viele auftreten, ist die Ausnahme.

Mit steigender Kontextgröße scheint die Anzahl der formalen Kontexte mit nichttrivialen Blockrelationen abzunehmen. Das sieht man in den folgenden Grafiken. Die erste gibt die Ergebnisse eines Experiments wieder, in dem 65532 zufällig erzeugte formale Kontexte der Größen 5×5 bis 20×20 (nicht notwendigerweise quadratisch) und einer Tabellenbelegung zwischen 10% und 50% untersucht wurden, in denen jeder formale Gegenstand mindestens ein formales Merkmal trug und dual jedes Merkmal auf mindesten einen Gegenstand zutraf. Unter diesen hatten nur 12,78% nicht-triviale Blockrelationen. In Abbildung B.2 ist abzulesen, welchen Anteil die formalen Kontexte mit nicht-trivialen Blockrelationen unter den untersuchten mit einer gewissen Tabellengröße hatten. Dieser Anteil sinkt mit der Tabellengröße. Weil die Tabellengrößen im Experiment nicht gleichverteilt

auftraten, kann man die entsprechenden Werte jedoch nicht direkt als Wahrscheinlichkeit interpretieren, bei einem Kontext einer bestimmten Größe auf interessante Blockrelationen zu stoßen. Die Verteilung der Kontextgrößen ist in Abbildung B.3 wiedergegeben.

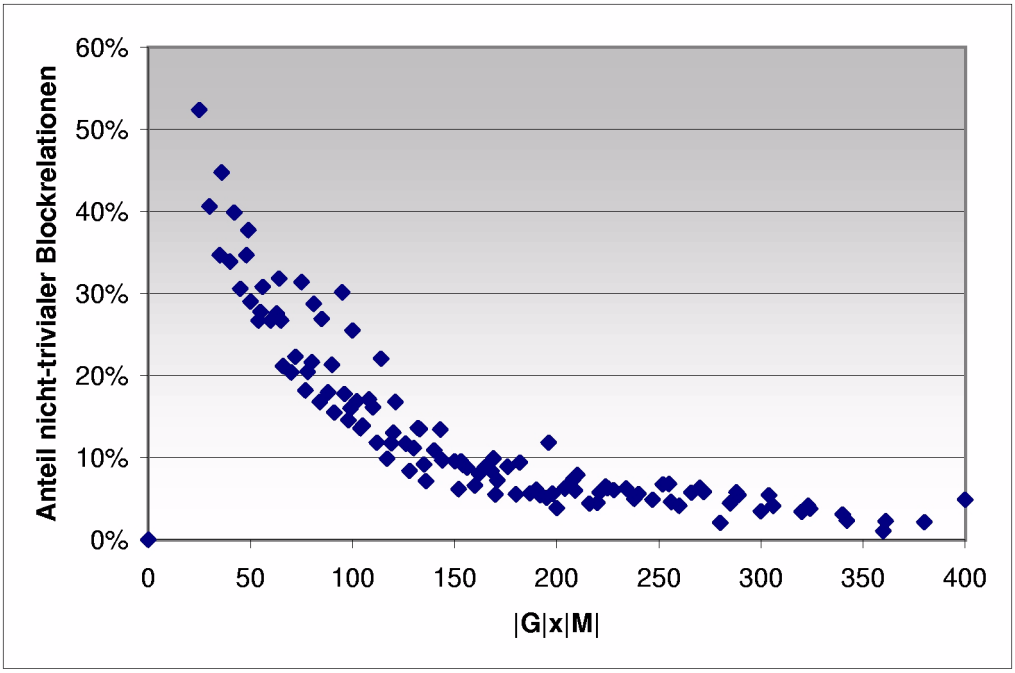


Abb. B.2 Anteil interessanter Blockrelationen

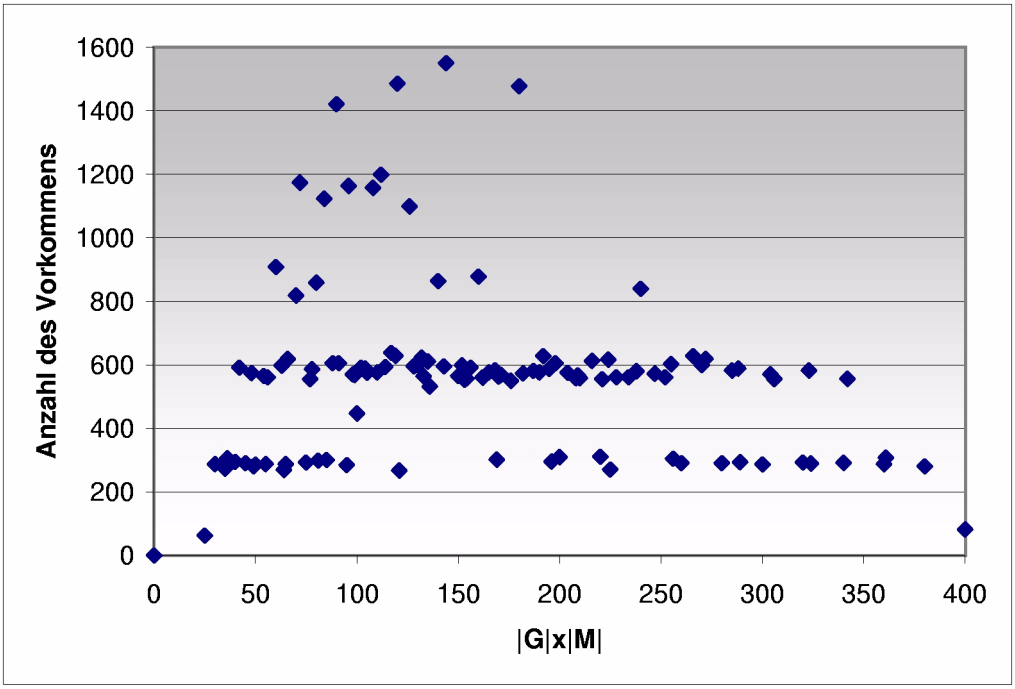


Abb. B.3 Verteilung der Kontextgrößen zu Abbildung B.2

Um die Ergebnisse zu überprüfen, wurden noch Experimente mit vollkommen zufällig erzeugten Kontexten durchgeführt. Abbildung B.4 spiegelt die Ergebnisse eines Tests mit 4936 formalen Kontexten der Größen 5×5 bis 30×30 wider.

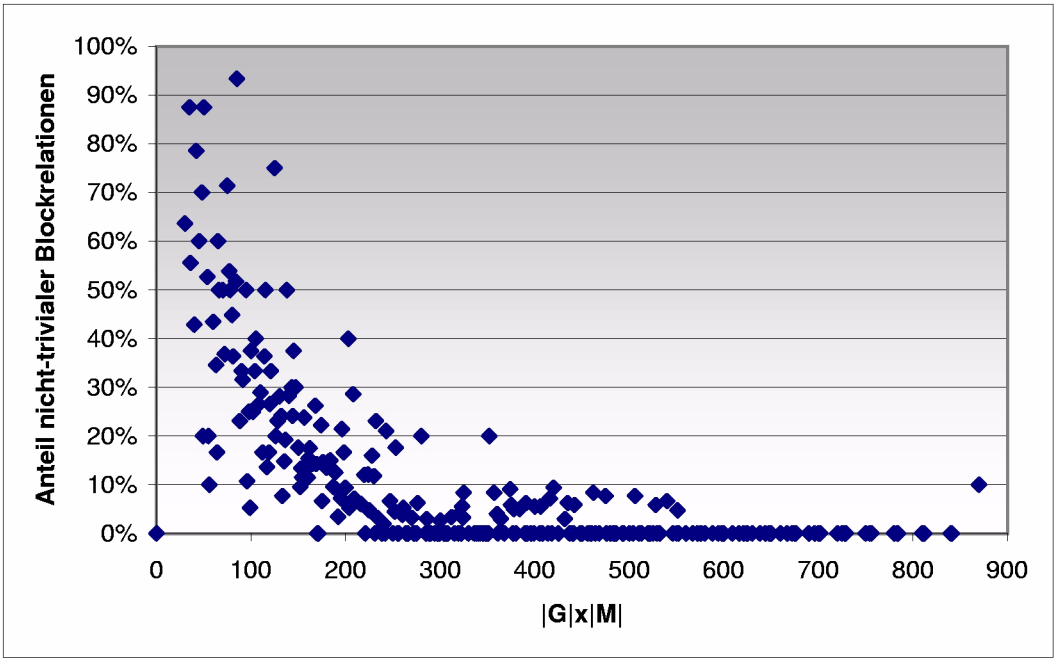


Abb. B.4 Anteil interessanter Blockrelationen bei völlig zufällig erzeugten Kontexten

Die Verteilung der Kontextgrößen war der des vorher beschriebenen Experiments ähnlich.

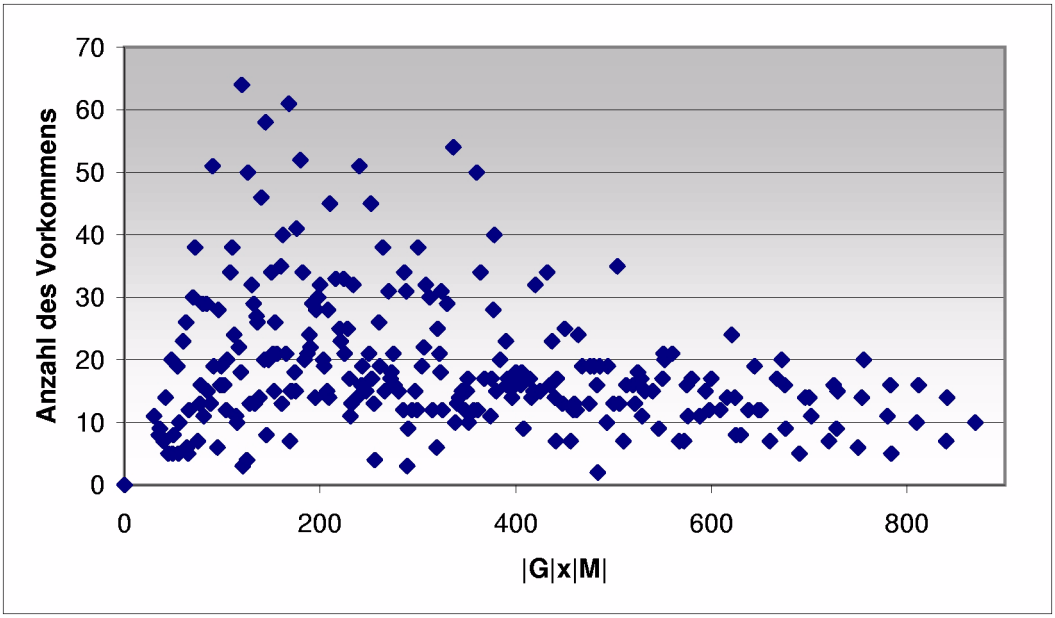


Abb. B.5 Verteilung der Kontextgrößen zu Abbildung B.4

Die Experimente lassen darauf schließen, daß es sinnvoll ist, in BASE zu früh erstellten kleinen Liniendiagrammen Blockzerlegungen zu betrachten, um das betrachtete System zu modularisieren. Mit welcher Wahrscheinlichkeit damit zu rechnen ist, nicht-triviale Blockrelationen zu finden, ist aus den gemachten Experimente nicht zu schließen, da die Ergebnisse sehr streuen. Dies müßten Praxistests mit echten Anwendungsfallmodellen zeigen. Für kleine Kontexte scheinen aber akzeptable Werte erreicht zu werden.

B.5 Automatisches Zeichnen von Diagrammen

Nach der Berechnung eines Begriffsverbands in den in B.2 dargestellten Schritten sind für ein vollständiges Liniendiagramm nur noch die Koordinaten der Knoten zu bestimmen. Dafür hat sich die *Methode des additiven Zeichnens* etabliert. Diese Methode soll in B.5.1 kurz dargestellt werden, ohne einen vollständigen Algorithmus anzugeben. Es wird nur das Prinzip erläutert, um eine innerhalb der Arbeiten zu BASE vorgenommene Modifikation im Abschnitt B.5.2 erklären zu können.[1] Diese Modifikation soll verhindern, daß der oberste bzw. unterste Diagrammknoten in überproportional großen Abstand zum restlichen Diagramm gezeichnet wird, wie es bei der Methode des additiven Zeichnens häufig der Fall ist. Für BASE wurde noch eine andere Erweiterung des automatischen Zeichnens entwikkelt, die es erlaubt, neue Diagramme anhand von schon vorliegenden zu zeichnen und so eine einmal erstellte Anordnung zu berücksichtigen. Diese zweite Erweiterung ist in B.5.3 erklärt.

B.5.1 Additives Zeichnen

Nach Folgerung 5 auf Seite 88 ist jeder formale Begriff als Supremum von Gegenstands- und als Infimum von Merkmalsbegriffen darstellbar. Darauf baut die *Methode des additiven Zeichnens* auf.

Je nachdem ob man sich an den Gegenstands- oder Merkmalsbegriffen orientiert, spricht man vom *Zeichnen nach Gegenständen* oder *Zeichnen nach Merkmalen*. Beim Zeichnen nach Gegenständen bestimmt man zu jedem $\bigvee$-irreduziblen Gegenstandsbegriff einen zweidimensionalen Vektor (also zu Gegenstandsbegriffen, die höchstens einen unteren Nachbarn besitzen, siehe Definition 11 auf Seite 72 und 5.4.2 auf Seite 210). Die $\bigvee$-reduziblen Gegenstandsbegriffe werden nicht berücksichtigt, weil sie selber Suprema anderer Gegenstandsbegriffe sind. Die Koor-

1. Eine ausführliche Darstellung der gesamten Zeichenmethode findet sich in [Vog 96], 3.3.

dinaten eines beliebigen Knotens errechnet man dann als Summe der Vektoren zu den in seinem Umfang enthaltenen formalen Gegenständen. (Damit man bei der Berechnung der Summe nicht immer überprüfen muß, ob ein Gegenstandsbegriff $\vee$-irreduzibel ist, ist es zweckmäßig, den formalen Gegenständen mit $\vee$-reduziblen Gegenstandsbegriffen den Nullvektor zuzuordnen.) Dual geht man beim Zeichnen nach Merkmalen anhand der $\wedge$-irreduziblen Merkmalsbegriffe vor. Nach Gegenständen und nach Merkmalen gezeichnete Diagramme eines formalen Kontexts sind in der Regel verschieden. Wie man geeignete Vektoren bekommt, entnehme man [Vog 96], 3.3. Es werden dazu Kettenzerlegungen der Gegenstands- bzw. Merkmalsordnung untersucht.[1] Frank Vogt gibt für die Vektorenzuordnung eine Komplexität von $O(|G|^3)$ bzw. $O(|M|^3)$ an.

Die Methode des additiven Zeichnens garantiert – bei sinnvoller Wahl der vertikalen Koordinaten der Vektoren (d.h. für Merkmale < 0, für Gegenstände > 0) – ein in dem Sinne korrektes Liniendiagramm, daß seine vertikale Anordnung die Verbandsordnung korrekt wiedergibt, also Knoten zu kleineren Begriffen unter denen zu größeren liegen. Weiter kann man die Vektoren so wählen, daß bei (über die entsprechenden Begriffe) unvergleichbaren formalen Gegenständen bzw. Merkmalen, alle Knoten des Diagramms verschiedene Positionen erhalten ([Vog 96], 3.3, S. 61/62). Im allgemeinen ist dies jedoch nicht garantiert. Ebensowenig kann die Methode ausschließen, daß Kanten zu ihnen nicht inzidente Knoten schneiden. Deshalb müssen Liniendiagramm-Editoren diesen Fall in der Darstellung der Diagramme hervorheben – etwa wie in Abbildung 5.3 auf Seite 194. Der Fall gleicher Knotenkoordinaten erscheint dermaßen pathologisch, daß beim Zeichnen der Diagramme im Werkzeug zu BASE (Kapitel 5) auf seine Behandlung verzichtet wurde. Auch bei späteren manuellen Korrekturen ist es wichtiger, unzutreffende Kanten-Knoten-Überschneidungen klar zu machen, weil der Benutzer in der Regel selber leicht überschauen kann, ob er einen Knoten exakt auf einem anderen positioniert.

Die Gestaltung des Diagramms erfolgt beim Zeichnen nach Gegenständen von unten nach oben (von den kleinen zu den großen Begriffsumfängen), beim Zeichnen nach Merkmalen von oben nach unten (von den kleinen zu den großen Begriffsinhalten). Die Methode der Koordinatenzuordnung ist aber unabhängig von der Art der Begriffsberechnung (anhand der Umfänge oder Inhalte). Im Werkzeug zu BASE (Kapitel 5) werden Diagramme nach Merkmalen gezeichnet, weil die Diagrammgestaltung von oben nach unten der üblichen Betrachtungsweise entspricht und so die Anordnung des Diagramms anhand der Anwendungsfälle vorgenommen wird.

1. Gegenstands- und Merkmalsordnung entstehen durch die Einschränkung der Begriffsverbandsordnung auf Gegenstands- bzw. Merkmalsbegriffe (siehe Definition A2 auf Seite 274).

B.5.2 Nachbehandlung von Infimum oder Supremum

Durch das additive Zeichnen werden Knoten mit vielen oberen Nachbarn (beim Zeichnen nach Merkmalen) bzw. vielen unteren Nachbarn (beim Zeichnen nach Gegenständen) vertikal weit von ihren Nachbarn abgerückt.

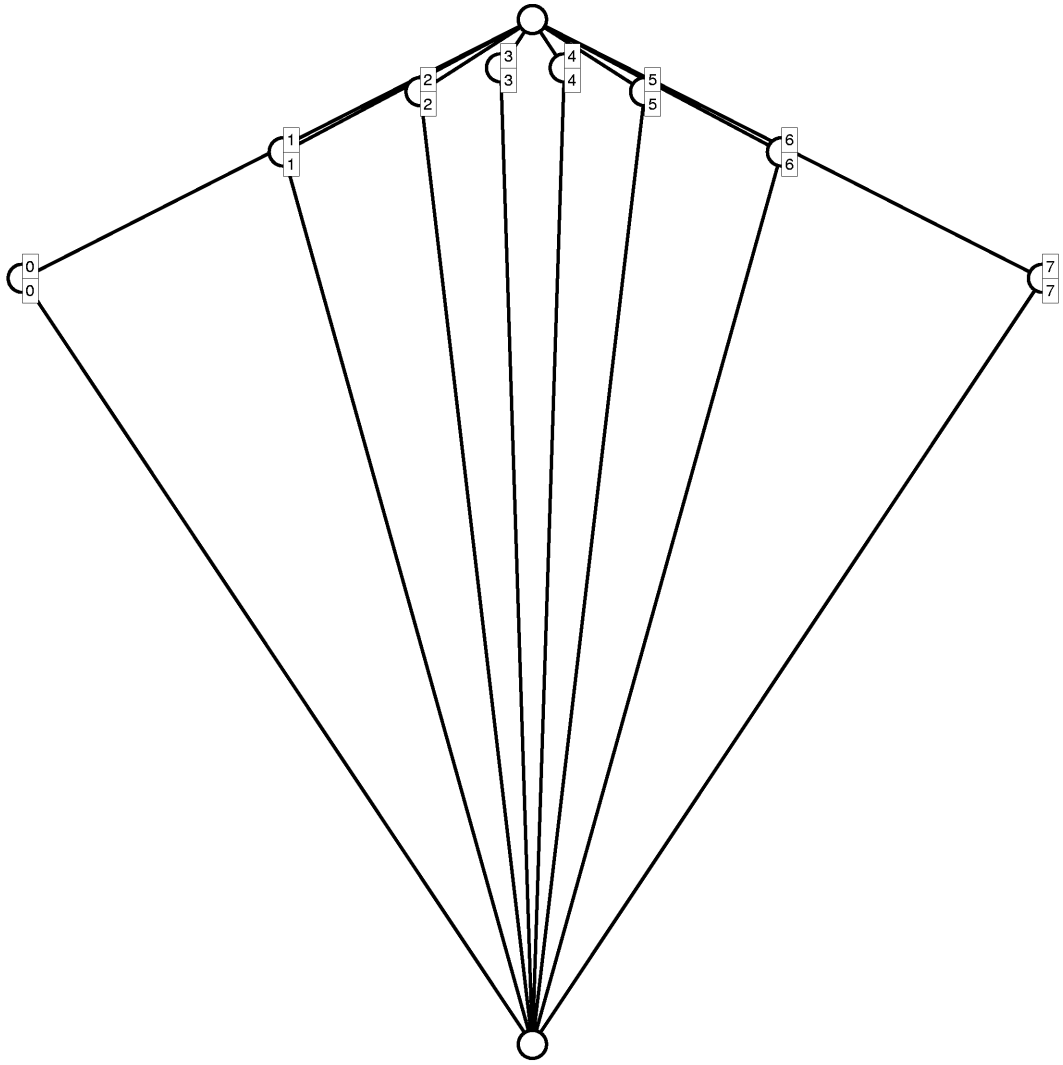


Abb. B.6 Automatisch gezeichnetes Diagramm zu $\underline{\mathfrak{B}}$({0,...,7},{0,...,7},=)

Im besonderen trifft dies auf das Infimum eines Verbands mit vielen Atomen bzw. das Supremum eines Verbands mit vielen Coatomen zu (Atome sind die oberen Nachbarn des Infimums, Coatome die unteren Nachbarn des Supremums, Definition 12 auf Seite 73). Dies sieht man gut am Beispiel des Liniendiagramms zu $\underline{\mathfrak{B}}(\{0, \ldots, 7\}, \{0, \ldots, 7\}, =)$.

Dieses weite Abrücken ist häufig zum Zeichnen nicht nötig, für den Betrachter des Diagramms aber sehr störend, zumal wenn das entstandene Diagramm nicht auf

den Bildschirm paßt. Deshalb wurde für BASE eine Heuristik entwickelt, beim Zeichnen nach Gegenständen den Supremumsknoten und beim Zeichnen nach Merkmalen der Infimumsknoten näher an das restliche Diagramm heranzurücken.

Stellvertretend wird nun das Vorgehen beim Zeichnen nach Merkmalen dargestellt. Das Diagramm wird erst in normaler Weise additiv gezeichnet und danach die Position des Infimums korrigiert. Dazu wird der Mittelwert der vertikalen Koordinaten der zum Zeichnen benutzten Vektoren (zu den Merkmalen) gebildet. Um seinen Betrag wird das Infimum gegenüber dem niedrigsten der Atome nach unten versetzt. Dieser Wert wurde gewählt, um das Diagramm möglichst harmonisch aussehen zu lassen. Diese Entscheidung ist aber vollkommen willkürlich. Für das weitere Vorgehen könnte man auch einen anderen Wert nehmen.

Die horizontale Koordinate soll nun so festgelegt werden, daß das Infimum möglichst zentriert unter den Atomen liegt. Die Kanten zu den Atomen sollen sich aber nicht überschneiden oder (noch schlimmer) durch andere Atome verlaufen. Um zu einer solchen Position zu gelangen, wird der Bereich für die x-Koordinate des Infimums sukzessive eingeschränkt. Zunächst einmal bilden die x-Koordinaten des am weitesten links und des am weitesten rechts gelegenen Atoms die linke und rechte Schranke. Um die angesprochenen Überschneidungen der Kanten zu vermeiden, werden die Atome nach aufsteigender x-Koordinate sortiert. Damit können Paare horizontal aufeinanderfolgender Knoten unter ihnen einfach betrachtet werden. Bei dieser Betrachtung sind vier Fälle zu unterscheiden:

1. Der linke und rechte Knoten haben verschiedene x-Koordinaten und der linke Knoten liegt auf gleicher Höhe mit dem rechten.
2. Der linke und rechte Knoten haben verschiedene x-Koordinaten und der linke Knoten liegt oberhalb des rechten.
3. Der linke und rechte Knoten haben verschiedene x-Koordinaten und der linke Knoten liegt unterhalb des rechten.
4. Der linke und rechte Knoten haben identische x-Koordinaten.

Diese Fälle sind in Abbildung B.7 dargestellt. Im ersten Fall, daß die beiden betrachteten Knoten nebeneinander liegen, erhält man keine Einschränkung an die horizontale Position des Infimums. Egal, wo es unter den beiden Knoten liegt, werden sich die Verbindungskanten nicht schneiden.

Im zweiten Fall, daß der (echt) linke Knoten oberhalb des rechten liegt, legt man eine Gerade durch die beiden Knoten. Damit sich ihre Kanten zum Infimum nicht schneiden, muß das Infimum links vom Schnittpunkt dieser Geraden mit der Parallelen zur x-Achse auf Höhe der y-Koordinate des Infimums liegen.

Analog erhält man im dritten Fall, in dem der (echt) linke Knoten unterhalb des rechten liegt eine untere Schranke für die x-Koordinate des Infimums.

Liegen die beiden Knoten übereinander, so darf die x-Koordinate des Infimums nur nicht gleich der x-Koordinate dieser beiden Knoten gewählt werden.

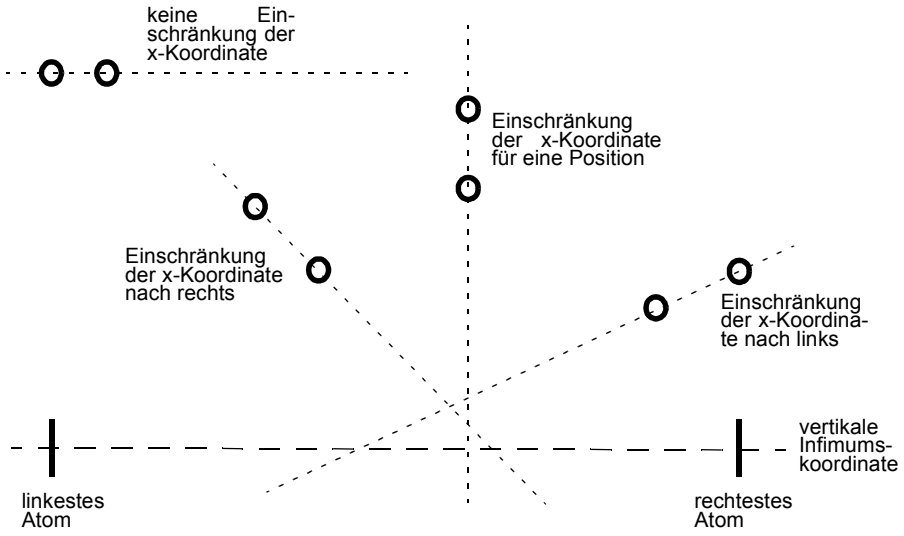


Abb. B.7 Paare oberer Nachbarn des Infimums

Um einen sinnvollen Bereich für die x-Koordinate des Infimums ausgehend von der unteren Schranke durch die x-Koordinate des am weitesten links liegenden Atoms und der oberen Schranke durch die x-Koordinate des am weitesten rechts liegenden Atoms zu bestimmen, geht man die Paare der horizontal aufeinanderfolgenden Atome durch. Trifft man den ersten Fall an, geschieht gar nichts. Im zweiten und dritten Fall ist zu prüfen, ob sich gegenüber dem aktuellen Bereich eine weitere Einschränkung ergibt. Diese wird dann im folgenden als sinnvoller Bereich betrachtet. Einschränkungen nach dem vierten Fall werden zunächst (für jede dabei auftretende x-Koordinate einmal) als "Separatoren" gespeichert und abschließend behandelt.

Reduziert sich der Bereich auf ein leeres Intervall, wird das Verfahren abgebrochen und die Position des Infimums nicht geändert. Erhält man dagegen auf diese Weise ein Intervall von sinnvollen x-Koordinaten für das Infimum, werden dessen untere und obere Schranke und die gespeicherten Separatoren, die in das Intervall fallen weiter betrachtet. (Die Separatoren sind paarweise verschieden, weil gleiche nur einmal gespeichert wurden.) Von diesen Werten werden die beiden horizontal aufeinanderfolgenden als endgültiges Intervall betrachtet, die die Mitte des ursprünglichen Intervalls einschließen. Die Mitte des so endgültig festgelegten Intervalls ist dann die x-Koordinate des Infimums. Die Mitte wird gewählt, um von den beiden einschränkenden Grenzen möglichst weit weg zu bleiben. Mit dieser Heuristik automatisch gezeichnet sieht das Liniendiagramm aus Abbildung B.6 wie folgt aus:

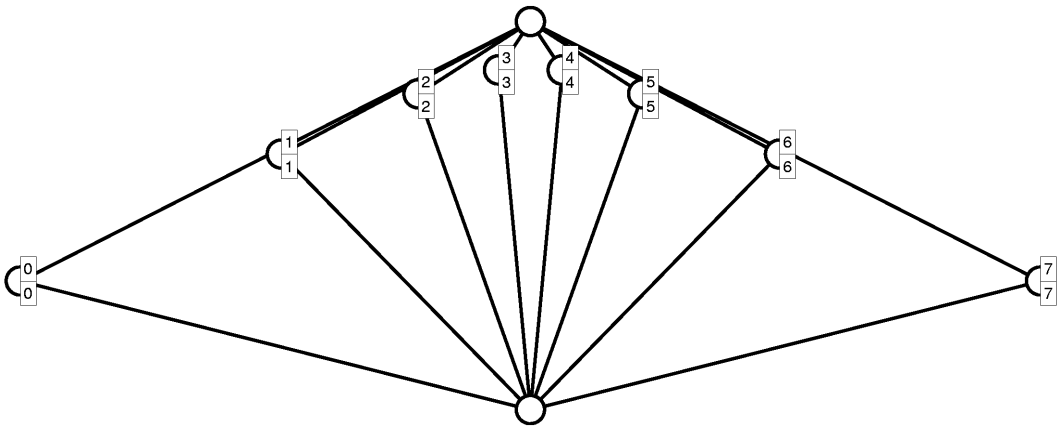


Abb. B.8 Verbessertes Liniendiagramm zu $\mathfrak{B}$({0,...,7},{0,...,7},=)

Bei weniger als 5 Knoten können durch das Zeichnen nach Merkmalen keine überdimensional großen vertikalen Abstände zum Infimum entstehen, deshalb wird für den Algorithmus angenommen, daß mindesten 5 Knoten vorhanden sind. (Ausgenutzt wird aber nur, daß mindestens zwei Knoten und damit mindestens ein formales Merkmal mit $\wedge$-irreduziblem Merkmalsbegriff existieren.)

Algorithmus B12 (*Koordinatenberechnung für das Infimum*)

Gegeben sei ein endlicher formaler Kontext (G, M, I) und ein zugehöriges additiv nach Merkmalen gezeichnetes Liniendiagramm mit mehr als 4 Knoten.

In `vectY` seien die y-Koordinaten der Vektoren zu den Merkmalen gespeichert und in `atomX` und `atomY` die Koordinaten und `aAnz` die Anzahl der Atome. Dabei seien die Positionen der Atome nach steigender x-Koordinate sortiert gespeichert.

Der Algorithmus berechnet in x und y die verbesserte Position des Infimums oder bricht ab, wenn die oben dargestellte Heuristik nicht zum Ziel führt.

```
(B12.1)   // linkestes Atom
          links := atomX[0]
(B12.2)   // rechtestes Atom
          rechts := atomX[aAnz-1]
(B12.3)   // Anzahl der Separatoren
          sInd := 0
(B12.4)   // Anzahl der "echten" Merkmalsvektoren
          nichtNull := 0
(B12.5)   // Summe der y-Koordinaten der Vektoren
          vDist := 0
```

```
(B12.6)   // für die y-Koordinate des Infimums
          FOR (i:=0) TO (|M|) DO
(B12.7)     // "echte Vektoren weisen "nach unten"
            IF (vectY[i] < 0) THEN
            {
(B12.8)       vDist := vDist + vectY[i]
(B12.9)       nichtNull := nichtNull + 1
            }
(B12.10)  y := min{atomY[0],...,atomY[aAnz-1]}+vDist/nichtNull
(B12.11)  // für Paare horizontal aufeinanderfolgender Atome
          FOR (i:=0) TO (aAnz-2) DO
          {
(B12.12)    // Die ersten drei Fälle
            IF (atomX[i+1] > atomX[i])
            {
(B12.13)      // zweiter Fall
              IF (atomY[i]>atomY[i+1] AND atomX[i+1]<rechts)
              THEN
              {
(B12.14)        // Gerade durch rechte Grenze & rechtes Atom
                x := (rechts-atomX[i+1])/(y-atomY[i+1])
                     *(atomY[i]-atomY[i+1]) + atomX[i+1]
(B12.15)        // neue rechte Grenze?
                IF (x < atomX[i])
(B12.16)          rechts := (atomX[i+1]-atomX[i])
                            /(atomY[i+1]-atomY[i])
                            *(y-atomY[i+1]) + atomX[i]
              } // zweiter Fall
(B12.17)      // dritter Fall
              ELSEIF (atomY[i]<atomY[i+1] AND atomX[i]>links)
              {
(B12.18)        // Gerade durch linke Grenze & linkes Atom
                x := (atomX[i]-links)/(atomY[i]-y)
                     *(atomY[i+1]-y) + links
(B12.19)        // neue linke Grenze?
                IF (x > atomX[i+1])
(B12.20)          links := (atomX[i+1]-atomX[i])
                           /(atomY[i+1]-atomY[i])
                           *(y-atomY[i]) + atomX[i]
              } // dritter Fall
(B12.21)      // Abbruch bei leerem Intervall
              IF (links > rechts) THEN
(B12.22)        ABORT
            } // erste drei Fälle
```

```
(B12.23)    // vierter Fall
            ELSEIF (sInd = 0  OR  atomX[i] > sep[sInd-1]) THEN
            {
(B12.24)      sep[sInd] := atomX[i]
(B12.25)      sInd := sInd + 1
            }
          } // Schleife über die Paare von Atomen
(B12.26)  // Intervallmitte
          x := (rechts + links) / 2
(B12.27)  i:=0
(B12.28)  // letzter Separator vor Intervallmitte
          WHILE (i < sInd  AND  sep[i] ≤ x) DO
          {
(B12.29)    IF (sep[i] > links)
(B12.30)      links := sep[i]
(B12.31)    i := i + 1
          }
(B12.32)  // erster Separator nach Intervallmitte
          IF (i < sInd) THEN
(B12.33)    rechts := sep[i]
(B12.34)  // endgültige x-Koordinate
          x := (rechts + links) / 2
```

In (B12.13) und (B12.17) wird zusätzlich mit jeweils einer Intervallgrenze verglichen, um nicht unnötig Geradenschnittpunkte zu berechnen.
Da die Steigung der Geraden durch zwei Atome im zweiten und dritten Fall (betragsmäßig) sehr gering sein kann, wird nicht gleich ihr Schnittpunkt mit der x-Achsenparallelen in Höhe der y-Koordinaten des Infimums berechnet. Dies könnte zu einer Überschreitung des Zahlbereichs führen. Sicherer erscheint es, in (B12.14) die Gerade durch die aktuelle rechte Grenze und das rechte der beiden Atome zu betrachten. Schneidet diese die x-Achsenparallele in Höhe der y-Koordinate des linken Atoms links von diesem Atom, so ergibt sich eine neue rechte Grenze. Die neue Grenze wird in (B12.16) mit Hilfe der Geraden durch die beiden Atome bestimmt. Analog geht der Algorithmus in (B12.18) bis (B12.20) für den dritten Fall vor.
Die Separatoren werden aufsteigend sortiert gespeichert. Deshalb findet man den ersten Separator nach der Intervallmitte in (B12.32) und (B12.33) sehr einfach.
Für das Zeichnen nach Gegenständen kann analog die Position des Supremums korrigiert werden.

Die Anzahl der Atome eines Begriffsverbands ist durch die Anzahl der formalen

Gegenstände beschränkt, weil jeder Begriffsumfang eines Atoms einen Gegenstand enthalten muß, der in keinem anderen Atom enthalten ist. Denn nach Folgerung 6 auf Seite 88 sind die Atome Gegenstandsbegriffe. Der entsprechende formale Gegenstand zu einem Atom kommt nur größeren Begriffen zu, also nicht den anderen Atomen. Dual ist die Anzahl der Coatome eines Begriffsverbands durch die Anzahl der formalen Merkmale beschränkt.

Als Laufzeitabschätzung für Algorithmus B12 ergibt sich damit $O(n)$ mit $n = \max\{|G|, |M|\}$. Die vorher zu erfolgende Bestimmung der Atome bzw. Coatome verursacht asymptotisch keinen zusätzlichen Aufwand, weil sie bei der Berechnung der Diagrammkanten mit abfällt. Allerdings müssen die Atome (Coatome) zusätzlich sortiert werden, was mit einem einfachen Sortierverfahren einen Aufwand von $O(n^2)$ verursacht. Alternativ könnte man sie in einem binären Suchbaum speichern oder ein schnelleres Sortierverfahren anwenden und bekäme eine Komplexität von $O(n \cdot \log n)$. In jedem Fall ist dies gegenüber dem Aufwand zur Berechnung der Vektoren vernachlässigbar. Asymptotisch erhöht also die Nachbehandlung von Infimum bzw. Supremum den Aufwand zur Liniendiagrammerstellung nicht.

Algorithmus B12 wurde innerhalb der Arbeiten zu BASE für formale Gegenstände und Merkmale als Erweiterung der Liniendiagrammberechnung in *The Formal Concept Analysis Library* integriert. Dazu wurde in der entsprechenden Operation der Klasse *TFormalContext* (siehe 5.4.2) die Möglichkeit ergänzt, über einen Parameter die Nachbehandlung des Infimums/Supremums zu aktivieren. (Andere Bits des schon vorher vorhandenen Parameters steuern, ob nach Gegenständen oder Merkmalen gezeichnet werden soll, die Art der Erzeugung der Zeichenvektoren und als weitere Ergänzung, ob eine Implikationenbasis mit erzeugt werden soll, siehe B.3.)

B.5.3 Ausrichtung nach einem bestehenden Diagramm

In BASE spielt die funktionale Verfeinerung von Anwendungsfallmodellen eine zentrale Rolle. Ein bei einer solchen Verfeinerung entstehendes Liniendiagramm weist in der Regel eine ähnliche Struktur auf wie das Ausgangsdiagramm des unverfeinerten Anwendungsfallmodells. Wie in 4.3 dargestellt, konzentrieren sich die Änderungen auf das Hauptideal zum Merkmalsbegriffs des zerlegten Anwendungsfalls. Wird das verfeinerte Diagramm wieder – wie in B.5.1 (und B.5.2) besprochen – von Grund auf neu gezeichnet, sind alle schon geleisteten Arbeiten zur manuellen Ausrichtung des Ausgangsdiagramms für das neue Diagramm verloren. Ein solches Vorgehen würde also die Akzeptanz von BASE bei Anwendern stark beeinträchtigen. Deshalb wurde eine Methode entwickelt, bei der Erstellung eines

neuen Diagramms dessen Ausrichtung an einem schon bestehenden zu orientieren.

Verfahren zur inkrementellen Begriffsverbandserstellung

Ein natürlicher und eleganter Ansatz bestünde darin, schon den entsprechenden Begriffsverband inkrementell auf Basis des vorliegenden Verbands zu berechnen. Dafür gibt es auch zwei Ansätze.

Robert Godin und Rokia Missaoui beschreiben in [GMs 94] und [GMA 95] einen Algorithmus zur inkrementellen Erzeugung eines Begriffsverbands und einer Menge zugehöriger Merkmalsimplikationen. Allerdings ist das "Inkrement" von besonderer Form. Der Algorithmus führt einen neuen formalen Gegenstand und eventuell diesen betreffende neue formale Merkmale in einen bestehenden Begriffsverband ein. Dazu wird der Gegenstandsbegriff des neuen Gegenstands vom globalen Infimum ausgehend anhand seines Begriffsinhalts gesucht. Besitzt kein formaler Begriff diesen Inhalt, hält die Suche beim unteren Nachbarn des Gegenstandsbegriffs im erweiterten Begriffsverband an. An dieser Stelle wird dann der neue Gegenstandsbegriff eingefügt. Danach muß der Verband noch gegen Suprema abgeschlossen werden. Damit dieses Vorgehen zum Ziel führt, müssen die Gegenstandsinhalte der schon vorher vorhandenen formalen Gegenstände unverändert bleiben. Insbesondere darf keines der neuen formalen Merkmale auf einen der vorhandenen Gegenstände zutreffen. Dual könnte man den Algorithmus aber auch anhand eines einzelnen neuen Merkmals, das dann auch auf schon vorhandene formale Gegenstände zutreffen darf, betrachten. Die inkrementelle Erzeugung von Begriffsverbänden (ohne Berücksichtigung von Implikationen) durch den Algorithmus von Godin und Missaoui ist in [Bec 99] noch eingehender untersucht und dargestellt. Insbesondere stellt Peter Becker einen analogen Algorithmus zum Entfernen von formalen Gegenständen aus dem Begriffsverband vor. (Ein Algorithmus zum Löschen von formalen Gegenständen wird auch in [GMA 95] erwähnt, aber nicht beschrieben.)

In BASE werden innerhalb der funktionalen Zerlegung von Anwendungsfällen neue Funktionen als formale Merkmale und neue "Dinge" als formale Gegenstände eingeführt. Dabei involvieren in der Regel die neuen Funktionen auch schon erfaßte "Dinge" und die neu eingeführten "Dinge" werden auch dem zerlegten Anwendungsfall zugesprochen. Mit dem Algorithmus von Godin/Missaoui könnte man zunächst die neuen Funktionen ohne neue "Dinge" einfügen und danach erst die neuen "Dinge" einzeln oder umgekehrt. Für beliebige nachträgliche Korrekturen der eingegebenen Indizierung müßte dann noch Beckers Algorithmus zum Entfernen von formalen Gegenständen und Merkmalen eingesetzt werden.

Bernhard Ganter und Sergei Kuznetsov stellen in [GKu 98] einen Algorithmus vor, mit dem ein Paar aus einem formalen Gegenstand und einem formalen Merkmal,

die zueinander inzident sind, in einen Begriffsverband eingefügt werden kann (siehe auch [Lin 99], 3.1.3). Dabei kann das neue Merkmal auch schon vorher vorhandenen formalen Gegenständen zugesprochen und der neue Gegenstand auch vorher vorhandene Merkmale besitzen. Es wird aber die Einschränkung gemacht, daß dem neuen Gegenstand nur solche vorhandene Merkmale zugeordnet werden können, die auch auf alle vorhandenen Gegenstände zutreffen, denen das neue Merkmal zugeordnet werden soll. Umgekehrt dürfen auch nur solche vorhandene Gegenstände das neue Merkmal erhalten, die schon alle dem neuen Gegenstand zugeordneten vorhandenen Merkmale tragen. (Diese Bedingung beschreiben die Autoren mit Hilfe von "Vorbegriffen".)

Vor allem die Einschränkung auf Paare aus formalen Gegenstand und formalem Merkmal im Algorithmus von Ganter/Kuznetsov erscheint für BASE zu restriktiv. Wie diese Einschränkung aufzulösen wäre, ist nicht zu sehen.

Für die Anwendung in BASE müßten beide Algorithmen noch um die Zuordnung von graphischen Koordinaten zu den neuen Liniendiagrammknoten erweitert werden, weil in beiden Fällen die graphische Repräsentation nicht behandelt wird.

Graphische Anpassung von Liniendiagrammen

Aus den angeführten Gründen wurde in BASE ein anderer Ansatz gewählt: Das neue Diagramm wird erst wie in B.2, B.5.1 und B.5.2 beschrieben komplett neu erstellt. Danach werden über die Knoteninhalte die Knoten des Ausgangsdiagramms mit Knoten des neuen Diagramms identifiziert. Die entsprechenden Knoten des neuen Diagramms erhalten die Koordinaten der Knoten des alten. Alle dabei noch nicht erfaßten Knoten des neuen Diagramms werden gemäß ihrer neu berechneten Koordinaten ins Diagramm eingepaßt. Dies gibt nur die Idee wieder, in der Umsetzung müssen noch wichtige Anpassungen vorgenommen werden, damit ein in seiner vertikalen Anordnung korrektes Liniendiagramm erzeugt wird. Im folgenden ist dargestellt, wie dies genau vor sich geht. Dual kann man auch anhand der Begriffsumfänge vorgehen, die entstehenden Anordnungen sind im allgemeinen verschieden.

"Bekannte" und neue Knoten

Die Zuordnung von Knoten des Ausgangsdiagramms zu solchen des neuen Diagramms geschieht anhand der Begriffsinhalte. Sei (G, M, I) der formale Kontext zum Ausgangsdiagramm und (H, N, J) der zum neuen Diagramm. Einem formalen Begriff $(A, B) \in \underline{\mathfrak{B}}(G, M, I)$ ordne $(B^J, B^{JJ}) \in \underline{\mathfrak{B}}(H, N, J)$ zu. Dabei läßt sich nach Folgerung 5 auf Seite 88 (B^J, B^{JJ}) als $\bigwedge_{m \in B}(m^J, m^{JJ})$ berechnen. Diese Berechnung ist schon in Algorithmus B11 ausgeführt worden.

Wenn B in $\underline{\mathfrak{B}}(H, N, J)$ noch als Inhalt existiert, haben alter und neuer Begriff den gleichen Inhalt. Sonst ist $B \subsetneqq B^{JJ}$. Nach Folgerung 2.b) auf Seite 82 definiert die Zuordnungsvorschrift $\underline{\mathfrak{B}}(G, M, I) \to \underline{\mathfrak{B}}(H, N, J), (A, B) \mapsto (B^J, B^{JJ})$ eine ordnungserhaltende Abbildung. Existieren alle alten Begriffsinhalte im neuen Begriffsverband weiter, so ist sie injektiv und die Ordnung der entsprechenden Begriffe im neuen Verband ist die selbe wie die im alten. Anhand dieses Falls soll zunächst einmal die weitere Vorgehensweise erklärt werden. Wie andere Fälle behandelt werden, wird danach aufgezeigt.

Sind allen Knoten des alten Diagramms über die Begriffsinhalte Knoten des neuen Diagramms zugeordnet, müssen dann nur noch die eventuell dazugekommenen angeordnet werden. Dazu sucht man zu einem neuen Begriff im neuen Begriffsverband den niedrigsten Knoten eines größeren Begriffs und den höchsten Knoten eines kleineren, die schon angeordnet sind. Die Knoten zum gesamten Intervall zwischen diesen beiden Begriffen müssen zwischen den beiden schon festgelegten Knoten angeordnet werden.

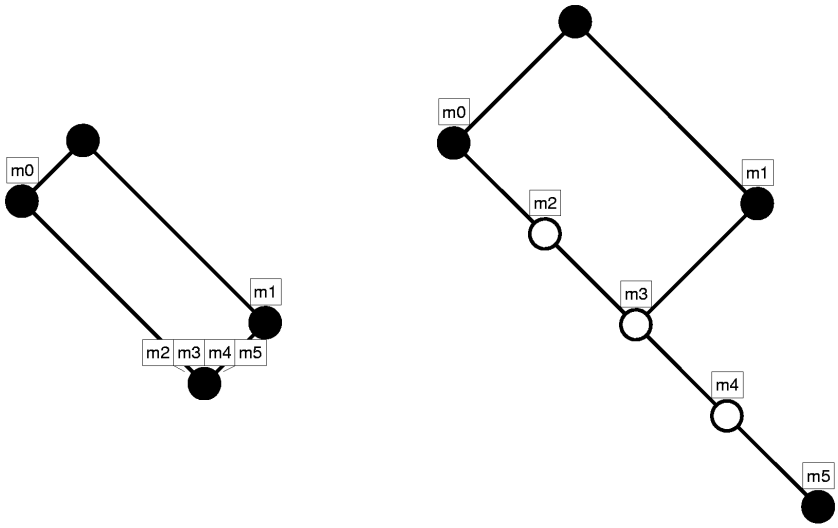


Abb. B.9 Altes und neues anzuordnendes Diagramm

In Abbildung B.9 solle das rechte Diagramm nach dem linken ausgerichtet werden. Alle vier Knoten des alten Diagramms können eindeutig solchen des neuen zugeordnet werden (ausgefüllt gezeichnete Knoten). Der (neue) Merkmalsbegriff zu *m2* ist noch anzuordnen. Das zu betrachtende Intervall ist $[\mu m_0, \mu m_5]$. Nach einer Skalierung mit dem Faktor 1/2 können die Knoten in das linke Diagramm eingepaßt

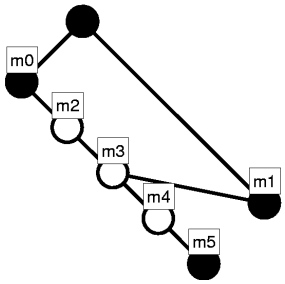


Abb. B.10 Eingepaßte Knoten

werden. Es ergibt sich das nebenstehende (inkorrekte) Liniendiagramm. Der Fehler ist entstanden, weil bei Positionierung des Knoten zum Merkmalsbegriff von *m3* nicht der Merkmalsbegriff von *m0*, sondern auch der von *m1* als niedrigster oberer Nachbar beachtet werden muß. Um dies zu leisten, werden die Knoten zu Oberbegriffen vor denen ihrer Unterbegriffe behandelt (etwa nach der lektischen Ordnung der Begriffsinhalte oder der umgedrehten lektischen Ordnung der Begriffsumfänge, siehe Hilfssatz B2 auf Seite 278/Hilfssatz B3 auf Seite 292). Die Positionsbestimmung wird dann in der skizzierten Weise nur für den Knoten zu einem größten formalen Begriff (im Beispiel dem Merkmalsbegriff zu *m2*) vorgenommen. Der betrachtete Punkt wird danach als angeordnet angesehen. So erhält man sukzessive eine korrekte Anordnung.

Zu erörtern bleibt noch, was mit Knoten geschieht, über oder unter denen noch keine der neuen Knoten angeordnet sind. Der Durchlauf der Knoten des neuen Diagramms beginnt bei seinem Supremum. Ist dieses einmal angeordnet, gibt es über jedem anderen Knoten mindestens einen schon in seiner Position bestimmten Knoten. Der Supremumsknoten ist also der einzige, über dem eventuell kein angeordneter Knoten liegt. Sicher liegt aber unter ihn ein angeordneter Knoten, weil ja jedem Knoten des Ausgangsdiagramms ein neuer Knoten zugeordnet wird. Man bestimme also den höchsten schon angeordneten Knoten. Das Intervall zwischen diesem und dem Supremum kann man einfach ohne Skalierung an das Diagramm so oben anhängen, wie es im neuen Diagramm durch das additive Zeichnen enthalten ist. Alle Knoten aus dem Intervall können danach als angeordnet markiert werden. Im Fall eines Knotens, unter dem kein angeordneter liegt, wird dagegen nur dieser eine Knoten in analoger Weise unter den niedrigsten angeordneten Knoten eines größeren Begriffs gehängt. (Wie oben kann es sein, daß sich andere Knoten aus seinem Hauptideal noch an weiteren schon angeordneten Knoten orientieren müssen.)

Probleme bei der Knotenzuordnung

Damit ist das grundsätzliche Vorgehen erklärt. Dies bleibt auch gleich, wenn nicht alle alten Begriffsinhalte auch im neuen Verband Inhalte sind. Dann sind zwei Fälle zu behandeln:

1. Mehreren alten Begriffen wird derselbe neue zugeordnet.
2. Die Ordnung auf den zugeordneten neuen Begriffe ist anders als die auf den originalen.

Im ersten Fall wird die Position des am niedrigsten positionierten der betroffenen alten Begriffe zugeordnet und alte Begriffe in den Hauptidealen der anderen entsprechend nach unten verschoben. Im Beispiel von Abbildung B.11 soll etwa das untere linke Diagramm als neues anhand des oberen linken ausgerichtet werden.

Die Merkmalsbegriffe zu *m1* und *m2* des alten Diagramms werden dem selben Begriff im neuen Diagramm zugeordnet. Für den Knoten zu diesem neuen Begriff ergeben sich also nicht eindeutig passende Koordinaten. Da die Diagrammknoten von oben nach unten abgearbeitet werden, wird für die Zuordnung der niedrigere der beiden Knoten gewählt. (Daß dies eine dem Vorgehen angepaßte Wahl ist, sieht man im folgenden.) Im Beispiel ist das der zu *m2*. In Folge dieser Entscheidung sind eventuell Knoten nach unten zu schieben, die

- kleiner als der höhere der beiden Knoten, aber nicht kleiner als der niedrigere sind, oder
- kleiner als ein unterer Nachbar des höheren der beiden Knoten, der nicht größer oder gleich dem niedrigeren der beiden Knoten ist.

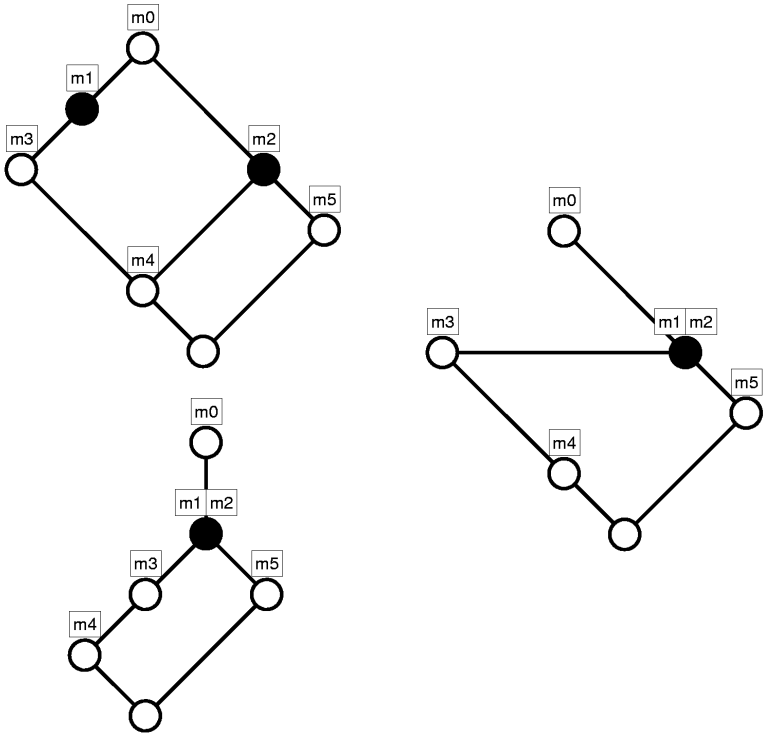


Abb. B.11 Mehrdeutige Zuordnung

Anders ausgedrückt bestimmt sich die Menge der möglicherweise zu verschiebenden Knoten als das Hauptideal des höheren Knotens ohne das Hauptideal des niedrigeren Knotens, aber vereinigt mit den Hauptidealen der unteren Nachbarn des höheren Knotens, die mit dem niedrigeren Knoten unvergleichbar sind. Im Beispiel sind das die Knoten zu *m3* und *m4* und der Infimumsknoten. Wird eine Korrektur der vertikalen Positionen dieser Knoten unterlassen, entstehen solche Diagramme wie in Abbildung B.11 rechts. Im neuen Diagramm ist – im Unterschied zum alten – der Merkmalsbegriff zu *m3* ein Unterbegriff des Merkmalsbegriffs von

m2. Die entsprechende Diagrammkante ist aber nicht absteigend, sondern verläuft waagerecht. Deshalb ist das Diagramm nicht korrekt.

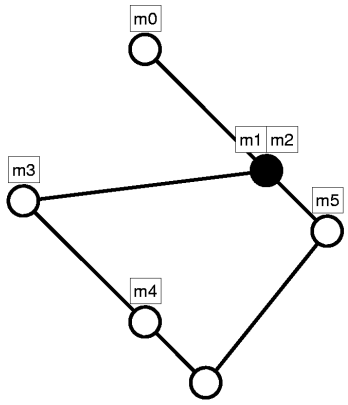


Abb. B.12 Korrigierte Anordnung

Indem man die erwähnten Knoten nach unten versetzt, erhält man ein korrektes Diagramm. Hätte man bei der Auswahl der Zuordnung eine höhere Position gewählt als die des niedrigeren der beiden beteiligten Knoten, müßte man mit der gleichen Argumentation Knoten von Oberbegriffen behandeln, indem man sie nach oben oder die Konten unter ihnen nach unten verschiebt. Um die bereits vorher abgehandelten höheren Knoten nicht mehr betrachten zu müssen, wurde als zugeordnete Position die des niedrigeren Knotens gewählt. So kann man einmal behandelte Knoten als endgültig festgelegt betrachten.

Oft werden Liniendiagramme per Hand von ihrem Supremumsknoten aus ausgerichtet. Um diesen Bezugspunkt fest zu lassen, wird für den Supremumsknoten eine Ausnahme gemacht. Wird einem Knoten des neuen Diagramms das Supremum des alten zugeordnet, so bleibt es dabei, auch wenn sich für kleinere Begriffe des Ausgangsverbands die gleiche Zuordnung ergibt. Dadurch können im neuen Diagramm keine aufsteigenden Kanten entstehen, wie man wie folgt sieht:

Sei (G, M, I) der formale Kontext zum Ausgangsdiagramm und (H, N, J) der zum neuen Diagramm. Ist $(A, B) \in \underline{\mathfrak{B}}(G, M, I)$ mit $B^{JJ} = G^{IJJ}$, gilt nach Folgerung 2.c) auf Seite 82 für alle $D \subseteq M$ mit $G^{I} \subseteq D \subseteq B$, daß $D^{JJ} = G^{IJJ}$. Das heißt, daß alle Oberbegriffe von (A, B) in $\underline{\mathfrak{B}}(G, M, I)$ ebenfalls dem selben neuen Knoten zugeordnet werden, wie das Supremum von $\underline{\mathfrak{B}}(G, M, I)$. Das bedeutet, daß in $\underline{\mathfrak{B}}(H, M, J)$ keine Oberbegriffe des (A, B) zugeordneten Knotens existieren, die einem formalen Begriff aus $\underline{\mathfrak{B}}(G, M, I)$ zugeordnet wären. Deshalb werden keine Kanten "verbogen".

Wie das Verschieben von Kanten genau vor sich geht, wird unten erklärt, nachdem der oben aufgeführte zweite Fall erörtert ist, in dem ebenfalls Verschiebungen nötig werden.

Auch wenn die Zuordnung der alten zu den neuen Knoten eindeutig ist, aber nicht alle Begriffsinhalte erhalten sind, kann es nötig sein, die vertikale Position einiger Knoten zu verändern. Durch die Veränderung des zugrunde liegenden formalen Kontexts kann die Ordnungsbeziehung zwischen den Begriffen des Ausgangsverbands und denen des neuen Begriffsverbands geändert sein.

Abbildung B.13 zeigt ein Beispiel dafür. Wird das rechts abgebildete Diagramm in der beschriebenen Weise nach dem linken ausgerichtet, muß der Knoten des Merkmalsbegriffs zu *m0* nach unten verschoben werden, damit die Verbandsordnung korrekt wiedergegeben wird.

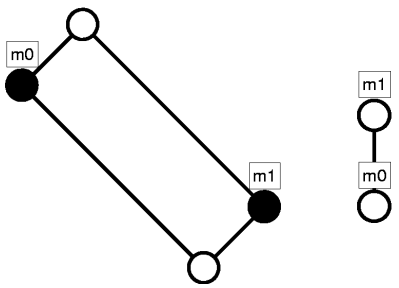


Abb. B.13 Veränderte Ordnung

Verschieben von Knoten

Nach der Zuordnung der neuen zu alten Knoten kann – wie oben erklärt – unter Umständen die Verbandsordnung inkorrekt abgebildet sein. Deshalb werden einige Knoten verschoben.

Das Verschieben von Knoten wird in einem Durchlauf über die Diagrammkanten des neuen Diagramms erledigt. Dabei werden – wie vorher bei den Knoten – höher gelegene Kanten vor niedrigeren erfaßt. Betrachtet werden Kanten, die zwei Knoten verbinden, die beide gewissen der alten Knoten zugeordnet sind. Weil jedem Knoten des Ausgangsdiagramms genau ein neuer Knoten zugeordnet wird, beziehen sich keine zwei verschiedenen Knoten des neuen Diagramms auf den selben Knoten des alten.

Ist eine Kante von einem Ober- zu einem Unterbegriffsknoten aufsteigend oder waagerecht, wird der Knoten des Unterbegriffs unter den des Oberbegriffs verschoben. Um die Form des ursprünglichen Diagramms zu erhalten, wird sein ganzes Hauptideal mit verschoben – allerdings nur solche Knoten, die nicht schon durch frühere Verschiebevorgänge auf noch niedrigeren Positionen gelandet sind.

Der Algorithmus im Ganzen

Algorithmus B13 (*Berücksichtigung der alten Diagrammanordnung*)

Gegeben seien zwei endliche formale Kontexte (G, M, I) und (H, N, J) mit $M \cap N \neq \emptyset$, $\underline{\mathfrak{B}}(G, M, I) = \{(A_0, B_0), \ldots, (A_{n-1}, B_{n-1})\}$ und $\underline{\mathfrak{B}}(H, N, J) = \{(C_0, D_0), \ldots, (C_{k-1}, D_{k-1})\}$.

Weiter seien `orig` ein Liniendiagramm zu $\underline{\mathfrak{B}}(G, M, I)$ und `neu` eins zu $\underline{\mathfrak{B}}(H, N, J)$. Die enthaltenen Knoten seien analog zu den repräsentierten Begriffen von 0 bis $n-1$ bzw. von 0 bis $k-1$ in der oben angegebenen Reihenfolge numeriert.

In beiden Liniendiagrammen sei die Knotennumerierung mit der jeweiligen Verbandsordnung absteigend. (Knoten zu größeren formalen Begriffen haben kleinere Nummern als die zu kleineren.)

`neu` habe die Kanten $\{(s_0, z_0), \ldots, (s_{r-1}, z_{r-1})\}$ (Paare von oberem Start- und unterem Zielknoten).

Der Algorithmus ordnet die Knoten von `neu` nach dem Vorbild `orig` an.
Für einen Begriffsverband V und $v \in V$ werde durch `ideal(`v`,` V`)` das Hauptideal und durch `filter(`v`,` V`)` der Hauptfilter von v in V geliefert (nach Annahme in reflexiv-transitiver Hülle gespeichert). (Ebenso ist die Überprüfung der Ordnung $<_V$ mit Hilfe der reflexiv-transitiven Hülle möglich.)
Sei V einer der beiden betrachteten Begriffsverbände und $(A, B) \in V$. Dann liefere `nr(`(A, B)`,` V`)` die Nummer von (A, B) in der Aufzählung der formalen Begriffe von V.
`x(i,d)` und `y(i,d)` seien die x- bzw. y-Koordinaten des `i`'ten Knotens im Liniendiagramm `d` für `d` $\in$ {`orig,neu`}.
`minDY` sei der minimale vertikale Abstand von benacharten Knoten.

```
(B13.1)   // Die originalen y-Koordinaten werden noch
          // verändert und deshalb in neuem Array gespeichert.
          FOR (i:=0) TO (n-1) DO
(B13.2)     origY[i] := y(i,orig)
(B13.3)   // Originalknoten zu neuen Knoten
          FOR (j:=0) TO (k-1) DO
(B13.4)     origKn[j] := -1
(B13.5)   FOR (i:=0) TO (n-1) DO
          { // Schleife über Originalknoten
(B13.6)     // entsprechender neuer Knoten
            knr := nr( ⋀_{m ∈ B_i} (m^J, m^JJ) , 𝔅(H,N,J) )
                                   // m^J = ∅ für m ∉ N
(B13.7)     IF (origKn[knr] = -1) THEN
              // erste Zuordnung
(B13.8)       origKn[knr] := i
(B13.9)     // Schon anderer Originalknoten zugeordnet
            ELSEIF (origKn[knr] > 0  AND
                      origY[origKn[knr]] > origY[i])
                   // alte Zuordnung nicht globales Supremum
                   // und höherer Knoten
(B13.10)      // neue Zuordnung
              origKn[knr] := i
          }
(B13.11)  FOR (i:=0) TO (r) DO
          { // Schleife über neue Kanten (Ordnungsüberprüfung)
(B13.12)    os := origKn[nr(s_i , 𝔅(H,N,J))]
(B13.13)    oz := origKn[nr(z_i , 𝔅(H,N,J))]
```

```
(B13.14)    IF ( os > -1  AND  oz > -1  AND
                 origY[os] ≤ orig[oz] )
            { // Start- und Zielknoten alten zugeordnet
              // aber neue Verbandsordnung nicht respektiert
(B13.15)      // nötige Verschiebung
              dY = origY[os]-minDY-y(j,oz)
(B13.16)      origY[oz] := origY[os]-minDY
(B13.17)      FOR (j:=0) TO (n − 1) DO
(B13.18)        IF ((A_j, B_j) <_{𝔅(G,M,I)} (A_oz, B_oz)  AND
                     y(j,orig)+dY < origY[j])
                     // zu verschiebende Knoten
(B13.19)          origY[j] := y(j,orig)+dY
              }
            } // Zuordnung der alten Knoten beendet
(B13.20)    // Retten der Koordinaten für Skalieren/Verschieben
            FOR (i:=0) TO (k − 1) DO
            {
(B13.21)      altX[i] := x(i,neu)
(B13.22)      altY[i] := y(i,neu)
            }
(B13.23)    // Koordinaten für die zugeordneten Knoten
            FOR (i:=0) TO (k − 1) DO
(B13.24)      IF (origKn[i] > -1)
              {
(B13.25)        x(i,neu) := x(origKn[i],orig)
(B13.26)        y(i,neu) := origY[origKn[i]]
              }
(B13.27)    IF (origKn[0] = -1)
            { // Supremum nicht zugeordnet
(B13.28)      id := 𝔅(H, N, J)
(B13.29)      // höchster zugeordneter Knoten
              huz := -1
(B13.30)      FOR (j:=1) TO (k − 1) DO
(B13.31)        IF (origKn[j] > -1) THEN
                {
(B13.32)          id := id \ ideal((C_j, D_j), 𝔅(H, N, J))
(B13.33)          IF (huz = -1  OR  y(huz,neu) < y(j,neu)
(B13.34)            huz := j
                }
(B13.35)      dX := x(huz,neu) - altX[huz]
(B13.36)      dY := y(huz,neu) - altY[huz]
```

```
(B13.37)    FOR (j:=0) TO (huz-1) DO
(B13.38)      IF ((C_j, D_j) ∈ filter((C_huz, D_huz), 𝔅(H, N, J)))
              THEN
              { // Knoten über dem höchsten zugeordneten
(B13.39)        x(j,neu) := x(j,neu)+dX
(B13.40)        y(j,neu) := y(j,neu)+dY
(B13.41)        // Knoten fertig
                origKn[j] := 0
              }
            }
(B13.42)  // Koordinaten für die nicht zugeordneten Knoten
          FOR (i:=1) TO (k − 1) DO
(B13.43)    IF (origKn[i] = -1)
            {   // nicht zugeordnet
(B13.44)      ft := filter((C_i, D_i), 𝔅(H, N, J))
(B13.45)      // niedrigster oberer angeordneter Nachbar
              noa := 0
(B13.46)      FOR (j:=i-1) DOWNTO (1) DO
(B13.47)        IF (origKn[j] > -1  AND  (C_j, D_j) ∈ ft) THEN
                {
(B13.48)          ft := ft \ filter((C_j, D_j), 𝔅(H, N, J))
(B13.49)          IF (y(noa,neu) > y(j,neu))
(B13.50)            noa := j
                }
(B13.51)      id := ideal((C_i, D_i), 𝔅(H, N, J))
(B13.52)      // höchste untere zugeordnete Schranke
              huz := -1
(B13.53)      FOR (j:=i+1) TO (k − 1) DO
(B13.54)        IF (origKn[j] > -1  AND  (C_j, D_j) ∈ id) THEN
                {
(B13.55)          id := id \ ideal((C_j, D_j), 𝔅(H, N, J))
(B13.56)          IF (huz = -1  OR  y(huz,neu) < y(j,neu))
(B13.57)            huz := j
                }
(B13.58)      IF (huz = -1) THEN
              {   // unterhalb keine zugeordneten Knoten
(B13.59)        x(i,neu) := x(i,neu)+x(noa,neu)-altX[noa]
(B13.60)        y(i,neu) := y(i,neu)+y(noa,neu)-altY[noa]
              }
(B13.61)      ELSE
              {   // huz gefunden -> Höhenanpassung
```

```
(B13.62)          faktor := (y(noa,neu)-y(huz,neu))
                              / (altY[noa] - altY[huz])
(B13.63)          x(i,neu) := x(noa,neu) +
                              (altX[i]-altX[noa])*faktor
(B13.64)          y(i,neu) := y(noa,neu) +
                              (altY[i]-altY[noa])*faktor
                  }
(B13.65)          // Knoten fertig
                  origKn[i] := 0
                }
```

In (B13.5) bis (B13.10) werden zu den Knoten des Ausgangsdiagramms die entsprechenden neuen Knoten bestimmt.
Die Schleife von (B13.11) bis (B13.19) korrigiert eventuell erzeugte der Verbandsordnung widersprechende Kanten.
In (B13.23) bis (B13.26) findet die Koordinatenfestschreibung für neuen Knoten statt, die gewissen alten Knoten zugeordnet sind.
Danach wird in (B13.27) bis (B13.41) ein noch nicht zugeordnetes Supremum behandelt. Dazu werden alle über dem höchsten zugeordneten Knoten hängenden Knoten in unveränderter Anordnung oben an das Diagramm angehängt. Sie werden dazu so verschoben, daß der höchste zugeordnete Knoten gerade auf seiner zugeordneten Position landet.
Da die Knotennummern der zugeordneten Knoten des Ausgangsdiagramms nicht mehr gebraucht werden, kann in `origKn` für schon bearbeitete Knoten eine `0` gespeichert werden, um die abgeschlossene Koordinatenfestlegung für den entsprechenden Knoten zu dokumentieren.
Für die anderen nicht zugeordneten Knoten geschieht die Koordinatenzuschreibung in (B13.42) bis (B13.65).
Dazu wird in (B13.44) bis (B13.50) zuerst der niedrigste Oberbegriffsknoten bestimmt, der schon zugeordnet ist. Dies ist ein oberer Nachbar, weil die Knotennumerierung in der Verbandsordnung absteigend ist und die nicht zugeordneten Knoten in der Reihenfolge dieser Numerierung untersucht werden. Der in (B13.51) bis (B13.57) ermittelte höchste schon zugeordnete Unterbegriffsknoten muß dagegen kein unterer Nachbar sein.
Zu unterscheiden sind die beiden Fälle, daß ein zugeordneter Unterbegriffsknoten gefunden wurde ((B13.61) bis (B13.64)) oder daß kein solcher existiert ((B13.58) bis (B13.60)). In jedem Fall kann der bearbeitete Knoten in (B13.65) als angeordnet markiert werden.
Dual kann die Identifikation der alten und neuen Begriffsknoten auch über die Begriffsumfänge hergestellt werden. Diese Zuordnung wird dann analog zu (B13.6) durch eine Anweisung `knr := nr(` $\bigvee_{g \in A_i} (g^J, g^{JJ})$, $\underline{\mathfrak{B}}(H, N, J)$ `)`

umgesetzt. Bei mehreren Zuordnungen wird der niedrigere neue Knoten gewählt und die Überprüfung der Verbandsordnung verläuft von unten nach oben. Das bedeutet, daß die Kanten und Knoten in gegenüber oben umgekehrter Reihenfolge durchlaufen werden und sich die Rollen von Filtern und Idealen vertauschen.

Für die Laufzeitabschätzung sei – wie in Algorithmus B11 – angenommen, daß die Operationen `ideal`, `filter` und $\leq_V$ konstanten Aufwand verursachen. (Dies ist der Fall, wenn die reflexiv-transitive Hülle schon vorberechnet ist (bei der Erstellung des Liniendiagramms geleistet, siehe B.2.2) und sie wie zu Algorithmus B2 erläutert zeilen- und spaltenweise redundant als Bit-Arrays gespeichert wird.) Weiter werden die Mengenoperationen wieder als in logarithmischer Zeit durchführbar angenommen.

Sei wie oben $|\underline{\mathfrak{B}}(G, M, I)| = n$, $|\underline{\mathfrak{B}}(H, N, J)| = k$ und r die Anzahl der Kanten des Liniendiagramms zu $\underline{\mathfrak{B}}(H, N, J)$.

Für eine Laufzeitabschätzung ist die Komplexität der Infimumsbestimmung in `(B13.6)` zu betrachten. Anhand von Algorithmus B11 auf Seite 297 ist diese im einzelnen nachzuvollzeihen. Zuerst sind zu allen Merkmalsbegriffen von $\underline{\mathfrak{B}}(G, M, I)$ die entsprechenden Merkmalsbegriffe aus $\underline{\mathfrak{B}}(H, N, J)$ zu bestimmen (`(B11.3)-(B11.4)`). Dies verursacht nach Algorithmus B11 für jeden einzelnen Merkmalsbegriff von $\underline{\mathfrak{B}}(G, M, I)$ einen Aufwand von $O(|M| \cdot \log k)$, also für alle zusammen $O(|M| \cdot |N| \cdot \log k)$. Daraus kann die Zusammenstellung der Merkmalsbegriffe für alle Begriffsinhalte von $\underline{\mathfrak{B}}(G, M, I)$ (auch derjenigen Begriffe, die keine Merkmalsbegriffe sind) in $O(n^2 \cdot \log k)$ möglich (`(B11.5)-(B11.8)`). Nach diesen Vorbereitungen geschieht die Infimumsbestimmung (`(B11.9)-(B11.21)`) für alle Knoten des Ausgangsdiagramms in $O(n \cdot k \cdot \log k)$. Die Schleife von `(B13.5)` bis `(B13.10)` verursacht damit einen Aufwand von $O(n \cdot (n + k) \cdot \log k)$.

Die Überprüfung der Ordnung in `(B13.11)` bis `(B13.19)` trägt einen Aufwand von $O(r \cdot n)$ bei.

Die Positionierung der nicht zugeordneten Knoten findet in den Anweisungen `(B13.42)` bis `(B13.65)` in einer Laufzeit von $O(k^2 \cdot \log k)$ statt.

Die Anzahl der Kanten im Liniendiagramm zu $\underline{\mathfrak{B}}(H, N, J)$ ist durch k^2 beschränkt. (Vermutlich gilt die bessere Schranke $k \cdot \log k$. Dies ist die Kantenanzahl im Liniendiagramm zum Potenzmengenverband. Wenn man zeigen kann, daß der Potenzmengenverband unter allen Verbänden mit 2^s Elementen im zugehörigen Liniendiagramm die maximale Kantenanzahl besitzt, ist der Nachweis für die bessere Schranke geführt.)

Mit $b := \max\{n, k\}$ ist also die Gesamtkomplexität $O(|M| \cdot |N| \cdot \log b + b^3)$. Mit der vermuteten besseren Schranke für r ergäbe sich $O((|M| \cdot |N| + b^2) \cdot \log b)$.

Der Algorithmus wurde durch eine neue Operation der Klasse *TLineDiagram* (siehe 5.4.3) in *The Formal Concept Analysis Library* integriert. Diese Operation ermöglicht die Diagrammanpassung bzgl. der Begriffsinhalte oder der Begriffsumfänge.

C Beispielsystem für BASE – JWI Inc.

Die Beschreibung des Geschäfts von JWI ist dem FRISCO-Report ([FHL+ 97]) entnommen. Sie besteht aus 28 umgangssprachlich formulierten Aussagen, die im Original in Englisch gegeben sind. Für Lehrveranstaltungen der Philipps-Universität Marburg – unter anderem das durchgeführte Studentenpraktikum ([FGT 98]), dessen Modellierung der Darstellung in Kapitel 4 zugrunde liegt – wurden sie ins Deutsche übersetzt. Diese Übersetzung ist im folgenden angegeben.

S1: Japan Wines Inc. (kurz: JWI) ist ein Wein-Vertriebszentrum.

S2: Das Geschäft von JWI besteht darin, den Lagerbestand zu verwalten und Produkte an Einzelhändler zu liefern - entsprechend deren Bestellungen.

S3: Alle Bestellungen, die an einem Tag einlaufen, werden am nächsten Tag bearbeitet.

S4: Das Lager wird täglich überprüft und aufgrund des festgestellten Bedarfs werden Bestellungen an Weinproduzenten erteilt.

S5: Das Zentrum ist nicht zuständig für Abrechnungen mit den Einzelhändlern oder Weinproduzenten.

S6: Es folgt eine genauere Beschreibung der Geschäftsabläufe:

S7: Das Zentrum empfängt Bestellungen der Einzelhändler telephonisch von 9-17 Uhr.

S8: Die folgende Abbildung zeigt ein Bestellformular, das für Bestellungen von Einzelhändlern verwendet wird:

Bestellformular
(eines Einzelhändlers)

Datum: _____
Bestell-Nr: R _____
Einzelhändler: _____
Adresse: _____
Telefon: _____

Lfd-Nr.	Posten	Menge
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

S9: Eine Bestellung kann aus mehreren Bestellposten bestehen. Diese beziehen sich auf einzelne Produkte.

S10: Jeder Bestellposten wird auf dem Formular in einer Zeile notiert.

S11: Wenn das Zentrum eine Bestellung erhält, wird umgehend der Lagerbestand für jeden Bestellposten geprüft.

S12: Das Zentrum nimmt anerkannte Reklamationen von den Einzelhändlern zurück.

S13: Wenn das Zentrum eine Bestellung entgegennimmt, wird jeder Bestellposten in eine der beiden folgenden Karteien eingeordnet:
(a) Liefer-Kartei, (b) Warte-Kartei.

S14: Falls das Zentrum für einen Bestellposten genügend Ware auf Lager hat, dann wird dieser in die Liefer-Kartei eingetragen und der Lagerbestand wird entsprechend korrigiert.

S15: Andernfalls wird der Bestellposten in die Warte-Kartei eingetragen.

S16: Täglich nach 17 Uhr werden die notwendigen Bestellungen an die Weinproduzenten sowie die Auslieferungs-Anweisungen wie folgt erstellt.

S17: Bestellungen an Weinproduzenten werden für solche Produkte erteilt, deren Menge unter den Minimal-Lagerbestand gesunken ist. Dabei wird so viel bestellt, daß der entsprechende Maximal-Lagerbestand erreicht wird.

S18: Die folgende Abbildung zeigt ein Formular für Bestellungen an Weinproduzenten:

Bestellformular
(an Weinproduzenten)

Datum: _____
Bestell-Nr: W _____
Produzent: _____
Adresse: _____
Telefon: _____

Lfd-Nr.	Posten	Menge
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

S19: Minimal- und Maximal-Lagerbestände werden für jedes Produkt definiert.

S20: Das Zentrum erstellt Lieferscheine für jeden Lieferwagen. Dabei werden den Lieferwagen die Bestellposten aus der Liefer-Kartei entsprechend ihrer Fahrtziele zugeordnet.

S21: Die folgende Abbildung zeigt einen Lieferschein:

Lieferschein

Datum: _____
Einzelhändler: _____
Adresse: _____
Telefon: _____

Lieferwagen-Nr: ____
Liefer-Nr: D _____
Bestell-Nr: _____

a= ausgeführt
r = Retoure

Lfd-Nr.	Posten	Menge	a/r
_____	_____	_____	___
_____	_____	_____	___
_____	_____	_____	___
_____	_____	_____	___

S22: Am nächsten Morgen lädt jeder Lieferwagen Produkte aus dem Lagerhaus gemäß dem Lieferschein und liefert diese aus.

S23: Nach erfolgter Auslieferung werden die Lieferscheine zurückgegeben mit einem Vermerk "ausgeliefert" oder "Retoure".

S24: Produkte mit dem Vermerk "Retoure" werden ins Lagerhaus zurückgebracht.

S25: Das Zentrum löscht alle (gemäß dem Vermerk auf dem Lieferschein) ausgelieferten Bestellposten aus der Liefer-Kartei.

S26: Bestellposten mit dem Vermerk "Retoure" verbleiben in der Liefer-Kartei und werden am Folgetag in die Lieferscheine aufgenommen.

S27: Frische Produkte von den Weinproduzenten treffen zwischen 10 und 16 Uhr ein.

S28: Nach Eintreffen werden die frischen Produkte (nach dem Grundsatz "zuerst bestellt - zuerst beliefert") den Bestellposten der Warte-Kartei zugeordnet. Alle nunmehr lieferbaren Posten werden von dort in die Liefer-Kartei übertragen. Der Rest der frischen Ware wird dem Lagerbestand zugeführt.

D Klassendeklarationen

Als Ergänzung zu 5.4 sind hier die Deklarationen folgender Klassen des in Kapitel 5 vorgestellten Werkzeugs angegeben:

- *CUsecaseAngaben* – Speichert die nicht direkt im formalen Kontext codierten Informationen zu Anwendungsfällen.
- *CBASEDoc* – Verwaltet die Gesamtheit der Daten zu einem Projekt.
- *CBASEView* – Stellt Liniendiagramme dar.

In der folgenden Darstellung sind Operationen, die allein zur (De-)Aktivierung von Menüeinträgen dienen.

D.1 CUsecaseAngaben

```
class CUsecaseAngaben
// Speichert während der Eingabe von Anwendungsfällen und Dingen
// zu allen Anwendungsfällen die Angaben, die im Kontext in den
// Specials als Strings abgelegt werden, d.h. Beschreibung, uses-
// und extends-Angaben.
// So müssen nicht immer wieder zusammengesetzte Strings geparst
// werden.
// Speicherung geschieht in einer Liste von <UsecaseDaten>.
// Die Anwendungsfälle sind durch ihre Identifier aus dem
// entsprechenden formalen Kontext gegeben.
{
public:
   CUsecaseAngaben();
        // Konstruiert eine leere Liste von Angaben.
   ~CUsecaseAngaben();
        // Löscht die dynamisch erzeugte Liste.
   void parseSpecials(const TstringArray& specials);
        // Parst die Angaben aus <specials> und stellt sie als Liste
        // von <UsecaseDaten> zur Verfügung.
   void entferneUsecase(const CString& uc);
        // Löscht alle Angaben, in denen der Identifier <uc>
        // vorkommt, d.h. den eigenen Listeneintrag und die
        // Einträge in uses-/extends-Listen und Zerlegungen
        // anderer Anwendungsfälle.
```

```
void setBeschreibung(const CString& b,const CString& uc);
     // Setzt <b> als Beschreibung des Anwendungsfalls mit
     // Identifier <uc>.
const CString* getBeschreibung(const CString& uc);
     // Gibt einen Zeiger auf die Beschreibung des
     // Anwendungsfalls mit Identifier <uc>.
void addUsesAngabe(CString usedUc,const CString& usingUc);
     // Ergänzt den Anwendungsfall mit Identifier <usedUc> in
     // der uses-Liste des Anwendungsfalls mit Identifier
     // <usingUc>.
void entferneUsesAngabe(CString usedUc,const CString& usingUc);
     // Löscht den Anwendungsfall mit Identifier <usedUc> aus
     // der uses-Liste des Anwendungsfalls mit Identifier
     // <usingUc>.
int uses(const CString& usingUc,const CString& usedUc);
     // Ermittelt, ob der Anwendungsfall mit Identifier <usedUc>
     // in der uses-Liste des Anwendungsfalls mit Identifier
     // <usingUc> vorkommt.
int usesTransitiv(const CString& usingUc,const CString& usedUc);
     // Ermittelt, ob der Anwendungsfall mit Identifier
     // <usedUc> vom Anwendungsfall mit Identifier <usingUc>
     // benutzt wird.
CStringArray* getUsesListe(const CString& uc);
     // Gibt einen Zeiger auf die uses-Liste des Anwendungsfalls
     // mit Identifier <uc>.
void addExtendsAngabe(CString extendedUc,
     const CString& extendingUc);
     // Ergänzt den Anwendungsfall mit Identifier <extendedUc>
     // in der extends-Liste des  Anwendungsfalls mit
     // Identifier <extendingUc>.
void entferneExtendsAngabe(CString extendedUc,
     const CString& extendingUc);
     // Löscht den Anwendungsfall mit Identifier <extendedUc>
     // aus der extends-Liste des Anwendungsfalls mit
     // Identifier <extendingUc>.
int extends(const CString& extendingUc,
     const CString& extendedUc);
     // Ermittelt, ob der Anwendungsfall mit Identifier
     // <extendedUc> in der extends-Liste des Anwendungsfalls
     // mit Identifier <extendingUc> vorkommt.
int extendsTransitiv(const CString& extendingUc,
     const CString& extendedUc);
     // Ermittelt, ob der Anwendungsfall mit Identifier
     // <extendedUc> vom Anwendungsfall mit Identifier
     // <extendingUc> spezialisiert wird.
CStringArray* getExtendsListe(const CString& uc);
     // Gibt einen Zeiger auf die extends-Liste des
     // Anwendungsfalls mit Identifier <uc>.
void insertTeilschritt(CString usedUc,int pos,
     const CString& usingUc);
     // Ergänzt den Anwendungsfall mit Identifier <usedUc> in
     // der Zerlegung des Anwendungsfalls mit Identifier
```

```
        // <usingUc> an der Stelle <pos>.
   void addTeilschritt(CString usedUc,const CString& usingUc);
        // Ergänzt den Anwendungsfall mit Identifier <usedUc> in
        // der Zerlegung des Anwendungsfalls mit Identifier
        // <usingUc> am Ende.
   void entferneTeilschritt(int pos,const CString& usingUc);
        // Löscht den <pos>'ten Teilschritt aus der Zerlegung des
        // Anwendungsfalls mit Identifier <usingUc>.
   int benutztInZerlegung(const CString& usingUc,
        const CString& usedUc);
        // Ermittelt, ob der Anwendungsfall mit Identifier <usedUc>
        // in der Zerlegung des Anwendungsfalls mit Identifier
        // <usingUc> auf der höchsten Ebene vorkommt.
   int benutztInZerlegungTransitiv(const CString& usingUc,
        const CString& usedUc);
        // Ermittelt, ob der Anwendungsfall mit Identifier
        // <usedUc> in der Zerlegung des Anwendungsfalls mit
        // Identifier <usingUc> auf irgendeiner Ebene vorkommt.
   BOOL usesNichtInZerlegung(CString& fehlenderSchritt,
        CString& falschZerlegterUc);
        // Gibt es in einer uses-Liste zu einem Anwendungsfall,
        // Funktionen die nicht in dessen funktionaler Zerlegung
        // (wenn schon vorgenommen) vorkommen?
        // Sind <fehlenderSchritt> und <falschZerlegterUc> beim
        // Aufruf leere Strings, so wird vom Anfang aller Angaben an
        // gesucht, sonst nur in der internen Reihefolge hinter
        // <fehlenderSchritt> und <falschZerlegterUc>.
        // Der in der internen Reihenfolge nächste fehlende
        // Zerlegungsschritt ergibt die Rückgabewerte von
        // <fehlenderSchritt> und <falschZerlegterUc>.
        // Sind alle uses-Angaben in den jeweiligen Zerlegungen
        // berücksichtigt, wird in <falschZerlegterUc> der leere
        // String zurückgegeben.
   CStringArray* getTeilschritte(const CString& uc);
        // Gibt einen Zeiger auf die Zerlegungsliste des
        // Anwendungsfalls mit Identifier <uc> (nur höchste Ebene).
   void addToSpecials(TstringArray& specials);
        // Überträgt die in der Liste enthaltenen Angaben als
        // entsprechende Strings in <specials>. Enthaltene alte
        // Angaben in <specials> werden nicht gelöscht.
   void kontextVervollstaendigen(TFormalContext& kontext);
        // Arbeitet die aus den uses-/extends-Listen und
        // funktionalen Zerlegungen resultierenden
        // Merkmalsimplikationen in <kontext> ein.
private:
   CUsecaseDaten* liste;
        // Die Liste der Angaben.
   CUsecaseDaten* aktuellePosition;
        // Aufsetzpunkt für Suche in der Liste. Verbleibt immer an
        // der zuletzt gefundenen Stelle.
   CUsecaseDaten* find(const CString& uc);
        // Liefert einen Zeiger auf die Angaben zum Anwendungsfall
```

```
        // mit Identifier <uc>.
        // Ist <uc> nicht enthalten, wird NULL zurückgeliefert.
// Hilfsfunktionen für <usesTransitiv>, <extendsTransitiv>,
// <benutztInZerlegungTransitiv>
   int RekUsesTransitiv(const CString& usingUc,
        const CString& usedUc,TstringArray& schonAbgehakt);
   int RekExtendsTransitiv(const CString& extendingUc,
        const CString& extendedUc,TstringArray& schonAbgehakt);
   int RekBenutztInZerlegungTransitiv(const CString& usingUc,
        const CString& usedUc,TstringArray& schonAbgehakt);
};      // end CUsecaseAngaben


class CUsecaseDaten
// Listenelement zur Speicherung der Angaben, die im Kontext in
// den Specials als Strings abgelegt werden, d.h. Beschreibung,
// uses- und extends-Angaben zu einem Anwendungsfall.
// Die Anwendungsfälle sind durch ihre Identifier aus dem
// entsprechenden formalen Kontext gegeben.
{
public:
   CUsecaseDaten(CString uc,CUsecaseDaten* n);
        // Konstruktor, legt ein Listenelement für den
        // Anwendungsfall <uc> an und hängt <n> dahinter.
   const CString& getUsecaseIdentifier();
        // Liefert den Anwendungsfallbezeichner.
   void setBeschreibung(CString b);
        // Setzt die Anwendungsfallbeschreibung.
   const CString& getBeschreibung();
        // Liefert die Anwendungsfallbeschreibung.
   void addUsesAngabe(CString usedUc);
        // Ergänzt den Anwendungsfall mit Identifier <usedUc> in
        // der uses-Liste.
   void entferneUsesAngabe(const CString& usedUc);
        // Löscht den Anwendungsfall mit Identifier <usedUc>
        // aus der uses-Liste.
   int uses(const CString& usedUc);
        // Ermittelt, ob der Anwendungsfall mit Identifier <usedUc>
        // in der uses-Liste vorkommt.
   CStringArray* getUsesListe();
        // Gibt einen Zeiger auf die uses-Liste.
   void addExtendsAngabe(CString extendedUc);
        // Ergänzt den Anwendungsfall mit Identifier <extendedUc>
        // in der extends-Liste.
   void entferneExtendsAngabe(const CString& extendedUc);
        // Löscht den Anwendungsfall mit Identifier
        // <extendedUc> aus der extends-Liste.
   int extends(const CString& extendedUc);
        // Ermittelt, ob der Anwendungsfall mit Identifier
        // <extendedUc> in der extends-Liste vorkommt.
   CStringArray* getExtendsListe();
        // Gibt einen Zeiger auf die extends-Liste.
```

```
  void insertTeilschritt(CString usedUc, int pos);
       // Ergänzt den Anwendungsfall mit Identifier <usedUc> in
       // der Zerlegung an der Stelle <pos>.
  void addTeilschritt(CString usedUc);
       // Ergänzt den Anwendungsfall mit Identifier <usedUc> in
       // der Zerlegung am Ende.
  void entferneTeilschritt(int pos);
       // Löscht den <pos>'ten Teilschritt aus der Zerlegung.
  void bereinigeZerlegung();
       // Löscht Einträge ohne angegebenem Identifier aus
       // der Zerlegung.
  int benutztInZerlegung(const CString& usedUc);
       // Ermittelt, ob der Anwendungsfall mit Identifier <usedUc>
       // in der Zerlegung auf der höchsten Ebene vorkommt.
  CStringArray* getTeilschritte();
       // Gibt einen Zeiger auf die Zerlegungsliste
       // (nur höchste Ebene).
  void entferneUsecase(const CString& uc);
       // Entfernt uses-, extends- und Zerlegungseinträge, die
       // den Anwendungsfall <uc> referenzieren.
  void setNext(CUsecaseDaten* n);
       // Setzt nächstes Listenelement.
  CUsecaseDaten* getNext();
       // Gibt nächstes Listenelement.
  void addToSpecials(TstringArray& specials);
       // Überträgt gespeicherten Angaben als entsprechende
       // Strings in <specials>. Enthaltene alte Angaben in
       // <specials> werden nicht gelöscht.
  void kontextVervollstaendigen(TFormalContext& kontext);
       // Arbeitet die aus der uses-/extends-Liste und
       // funktionalen Zerlegung resultierenden
       // Merkmalsimplikationen in <kontext> ein.
private:
  CString usecaseIdentifier;
       // Bezeichner des beschriebenen Anwendungsfall
  CString beschreibung;
       // Anwendungsfallbeschreibung
  CStringArray teilschritte;
       // funktionale Zerlegung (nur oberste Ebene)
  CStringArray usesListe;
       // benutzte Anwendungsfälle
  CStringArray extendsListe;
       // spezialisierte Anwendungsfälle
  CUsecaseDaten* next;
       // nächstes Listenelement
  int find(const CString& uc,const CStringArray& liste);
       // Gibt die Position des Eintrags zum Anwendungsfall <uc>
       // im Array <liste>.
};     // end CUsecaseDaten
```

D.2 CBASEDoc

```
class CBASEDoc : public CDocument
// Verwaltet innerhalb von BASE die Daten zu einem Projekt.
{
friend class CBASEView;  // Arbeiten eng zusammen
protected:
   CBASEDoc(); // Konstruktur, besetzt die projektbezogenen
               // Attribute mit Standardwerten.
// Attribute
private:
   TWFCAFile* cscDatei; // aktuelle Projektdatei
   CString projektName; // Name des aktuellen Projekts
   TLineDiagram* aktuellesDiagramm;
        // Liniendiagramm des aktuell bearbeiteten Modells
   TILineDiagramArray* blockrels;
        // Liniendiagramme zu eventuell vorhandenen Blockrelationen
   CImplikationenDlg* implDlg;
        // Dialog zur Anzeige von Prüfungsfragen
   BOOL blockrelationsModus;
        // Wird gerade ein Diagramm zu einer Blockrelation
        // angezeigt?
   BOOL uebersichtsModus;
        // Wird gerade ein in der Datei gespeichertes Diagramm
        // angezeigt?
// Operationen
public:
   void resetImplDlg();
        // Löscht den Verweis auf einen Überprüfungsfragendialog.
   TLineDiagram* getAktuellesDiagramm();
        // Liefert das Liniendiagramm des aktuell bearbeiteten
        // Modells.
   virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
        // Liest die Projektdatei mit dem Pfad <lpszPathName>
        // ein und fordert den Benutzer zur Auswahl eines Modells
        // auf. Scheitert das, ist der Rückgabewert FALSE, sonst
        // TRUE.
   virtual BOOL OnSaveDocument(LPCTSTR lpszPathName);
        // Schreibt die Daten in die Projektdatei mit dem Pfad
        // <lpszPathName>. Scheitert das, ist der Rückgabewert
        // FALSE, sonst TRUE.
   virtual void DeleteContents();
        // Löscht die projektbezogenen Attribute.
protected:
   virtual BOOL SaveModified();
        // Gibt die Möglichkeit, veränderte Daten vor dem Schließen
        // der Anwendung zu speichern.
        // Bleiben danach noch nicht gespeicherte Änderungen, ist
        // der Rückgabewert FALSE, sonst TRUE.
   afx_msg void OnFileNew();
        // Legt ein neues Projekt an.
   afx_msg void OnFileSaveAs();
```

```
        // Ermöglicht das Speichern des Projekts unter neuem
        // Dateinamen.
   afx_msg void OnFileSave();
        // Speichert das Projekt unter einem schon vorhandenen
        // Dateinamen.
        // Ist kein Dateiname zugeordnet, wie <OnFileSaveAs>.
   afx_msg void OnDiagrammLoeschen();
        // Läßt den Benutzer ein Diagramm aus der Projektdatei
        // auswählen und löschen.
   afx_msg void OnBearbeitenUsecases();
        // Öffnet den Anwendungsfall-Editor zur Eingabe und
        // Veränderung der Anwendungsfalldaten.
   afx_msg void OnDiagrammAuswaehlen();
        // Läßt den Benutzer ein Diagramm aus der Projektdatei
        // auswählen.
   afx_msg void OnBearbeitenUeberpruefen();
        // Stößt die Anzeige von Überprüfungsfragen an.
        // Der Benutzer entscheidet, ob anhand von "Dingen" oder
        // Funktionen untersucht werden soll.
   afx_msg void OnBearbeitenKomponenten();
        // Startet oder beendet die Anzeige von
        // Blockrelationsdiagrammen.
   afx_msg void OnKomponentenmodellWaehlen();
        // Läßt den Benutzer ein Blockrelationsdiagramm auswählen.
        // Zur Auswahl stehen die Blockrelationen zu dem zuletzt
        // betrachteten Nicht-Blockrelations-Diagramm.
   afx_msg void OnKomponentenmodellLoeschen();
        // Läßt den Benutzer das aktuelle Blockrelationsdiagramm
        // löschen.
// Interne Hilfsfunktionen
private:
   void eingabenUebernehmen(TFormalContext* eingabenInKontext,
        TFormalContext* zielKontext);
        // Kopiert die für BASE relevanten Daten vom formalen
        // Kontext <eingabenInKontext> in <zielKontext>.
   void diagrammBerechnenUndZufuegen(TFormalContext* kontext,
        CUsecaseAngaben* usecases);
        // Berechnet in <aktuellesDiagramm> ein Liniendiagramm zum
        // Kontext <kontext>, nachdem vorher die Implikationen zu
        // "uses"-/"extends"-Beziehungen und funktionalen
        // Zerlegungen aus <usecases> berücksichtigt wurden.
   CString strukturNamenWaehlen(const CString& namensVorschlag,
        const CString meldung);
        // Öffnet einen Dialog zu Eingabe eines Bezeichners für ein
        // Diagramm.
   void blockrelationenEinlesen(TLineDiagram* origDiag);
        // Liest schon berechnete Blockrelationen zu <origDiag> ein
        // und belegt <blockrels>.
   int anhaengendeBlockrelationenLoeschen(TLineDiagram* origDiag);
        // Ermöglicht das Löschen der Blockrelationsdiagramme zu
        // <origDiag>. Rückgabewert ist IDYES, wenn der Benutzer
        // die Blockrelationen löschen wollte, IDNO, wenn er es
```

```
        // ablehnte, und IDCANCEL, wenn er auf "Abbruch" geklickt
        // hat, um auch das Löschen von <origDiag> zu stoppen.
   TLineDiagram* getDiagramm(CString name);
        // Liest das Diagramm <name> aus der Projektdatei.
   TLineDiagram* getOriginalDiagrammZuBlockrelation
        (TLineDiagram* blockrel);
        // Ermittelt das Diagramm, zu dem das
        // Blockrelationsdiagramm <blockrel> berechnet wurde.
};      // end CBASEDoc
```

D.3 CBASEView

```
struct TextZeichnungsDaten
// Daten für die Ausgabe einer Beschriftung
{
   TOAC* element;
        // Referenzierter formaler Gegenstand oder
        // referenziertes formales Merkmal.
   CFont* font;
        // Schriftart
   COLORREF farbWert;
        // Zeichenfarbe
   CSize boxGr;
        // Textausdehnung in Zeichenbereichskoordinaten
   int posX;
   int posY;
        // Textposition in Zeichenbereichskoordinaten
};// end TextZeichnungsDaten


class CBASEView : public CScrollView
// Zur Darstellung von Liniendiagrammen.
{
protected:
   CBASEView();
        // Konstruktor, setzt die Attribute auf Standardwerte.
public:
   virtual ~CBASEView();
        // Destruktor, löscht die Zeichnungsdaten.
// Attribute
private:
   int randX,randY;
        // Randangaben in Zeichenbereichskoordinaten
   BOOL zentrieren;
        // Soll das anzuzeigende Diagramm zentriert werden?
   TLineDiagram* aktuellesDiagramm;
        // angezeigtes Diagramm, des schnelleren Zugriffs und
        // besser lesbaren Codes halber gegenüber <CBASEDoc>
        // zusätzlicher Zeiger
   double koordFaktor;
```

```
        // Umrechnungsfaktor:
        // Diagrammkoordinaten -> Zeichenbereichskoordinaten
   int ursprungX,ursprungY;
        // (0,0) bzgl. Diagrammkoordinaten in
        // Zeichenbereichskoordinaten
   int xMin,xMax,yMin,yMax;
        // Extreme Zeichenbereichskoordinaten des Diagramms, nur
        // bei Änderungen der Zeichenbereichsgröße aktuell.
   BOOL zeichnungsGroesseOk;
        // Paßt das Diagramm in den Zeichenbereich?
   BOOL ggsteZeichnen, mmleZeichnen, beschriftungLoeschen;
        // Soll die Beschriftung mit formalen Gegenständen bzw.
        // Merkmalen angezeigt oder die komplette Beschriftung
        // während des Verschiebens gelöscht werden?
   int klickX,klickY;
        // Zeichenbereichskoordinaten eines Mausklicks
   double verschXum,verschYum;
        // vorgenommene Verschiebung in Diagrammkoordinaten
   char verschiebenVon;
        // einzelnen 'P'unkten, Haupt'F'iltern, Haupt'I'dealen
   TBitArray verschobeneKnoten;
        // Beim Verschieben von Filtern und Idealen sind
        // gleichzeitig mehrere Knoten betroffen.
   TBitArray verschobeneKanten;
        // Beim jedem Knotenverschieben sind in der Regel mehrere
        // Kanten betroffen.
   int obereVerschGrenze,untereVerschGrenze;
        // vertikale Verschiebegrenzen in Zeichenbereichs-
        // koordinaten (bezogen auf den Punkt des Mausklicks)
   TObject* verschobenerGgst;
        // formaler Gegenstand, dessen Bezeichnung verschoben wird
   TAttribute* verschobenesMml;
        // formales Merkmal, dessen Bezeichnung verschoben wird
   int kontextmenueVonKnoten;
        // Nummer des Knotens, zu dem ein Kontextmenü angezeigt
        // wird.
// Operationen
public:
   virtual void OnDraw(CDC* pDC);
        // Gibt Diagramm auf Bildschirm oder Drucker aus.
protected:
   virtual void OnUpdate(CView* pSender, LPARAM lHint,
        CObject* pHint);
        // Bereitet das Zeichen vor, wenn eine Änderung im zugrunde
        // liegenden CBASEDoc-Objekt aufgetreten ist.
   afx_msg void OnPaint();
        // Zeichnet das Diagramm auf dem Bildschirm, vergrößert den
        // Zeichenbereich, wenn sich das Diagramm als zu groß
        // herausstellt, zentriert es durch Scrollen, wenn
        // <zentrieren> gesetzt ist.
   afx_msg void OnAnsichtGegenstaende();
        // Aktiviert/Deaktiviert die Anzeige von formalen
```

```
     // Gegenständen.
afx_msg void OnAnsichtMerkmale();
     // Aktiviert/Deaktiviert die Anzeige von formalen
     // Merkmalen.
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
     // Behandelt Klicks mit der linken Maustaste.
afx_msg void OnMouseMove(UINT nFlags, CPoint point);
     // Verschiebt Diagrammteile gemäß der Bewegung der Maus mit
     // gedrückter linker Maustaste.
afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
     // Beendet das Verschieben.
afx_msg void OnBearbeitenVerschFilter();
     // Setzt <verschiebenVon> auf 'F' zum Verschieben von
     // ganzen Hauptfiltern.
afx_msg void OnBearbeitenVerschIdeale();
     // Setzt <verschiebenVon> auf 'I' zum Verschieben von
     // ganzen Hauptidealen.
afx_msg void OnBearbeitenVerschPunkte();
     // Setzt <verschiebenVon> auf 'P' zum Verschieben von
     // einzelnen Punkten.
afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
     // Blendet auf Klick der rechten Maustaste ein Kontextmenü
     // ein, dessen Inhalt abhängig vom angezeigten Diagramm und
     // der Mausposition ist. Hauptsächlich enthält es
     // Menüeinträge zur Einblendung eines Diagrammsuschnitts.
afx_msg void OnZoom();
     // Ermöglicht die Skalierung der Diagrammgröße..
afx_msg void OnDinA4();
     // Skaliert das Diagramm auf DIN A4-Größe. Wählt die
     // größere Darstellung aus Hoch- und Querformat.
afx_msg void OnPunktgroesse();
     // Ermöglicht die Skalierung der Knotenradien.
afx_msg void OnSchriftart();
     // Ermöglicht die Festlegung der Schriftart für die
     // Beschriftung mit formalen Gegenständen bzw. Merkmalen.
afx_msg void OnFilterAusschnitt();
     // Beschränkt die Darstellung des Diagramms auf den
     // Hauptfilter zum Knoten <kontextmenueVonKnoten>.
afx_msg void OnIdealAusschnitt();
     // Beschränkt die Darstellung des Diagramms auf das
     // Hauptideal zum Knoten <kontextmenueVonKnoten>.
afx_msg void OnAnsichtUebersicht();
     // Schaltet von einem Ausschnittsdiagramm oder
     // Detaildiagramm zu einer Blockzerlegung auf
     // das übergeordnete Diagramm zurück.
afx_msg void OnAnsichtKomponenteninterna();
     // Aktiviert/Deaktiviert die Detaildarstellung einer
     // Blockzerlegung.
afx_msg void OnBlockAusschnitt();
     // Zeigt ausgehend vom Übersichtsdiagramm einer
     // Blockzerlegung den Block zum Knoten
     // <kontextmenueVonKnoten> in Detailsicht an.
```

```
  afx_msg void OnBlockfarbe();
       // Ermöglicht die Festlegung einer Farbe für einen
       // eingefärbten Block.
// Interne Hilfsfunktionen
private:
  void diagrammZeichnen(CDC* pDC);
       // Zeichnet das Diagramm.
  void kanteZeichnen(CDC* pDC, TDLine* kante,
       BOOL loeschen=FALSE);
       // Zeichnet/Löscht eine Kante.
  void knotenZeichnen(CDC* pDC, TDPoint* knoten, int& x, int& y,
       int& radius, BOOL loeschen=FALSE);
       // Zeichnet/Löscht einen Knoten. In <x>,<y> und <radius>
       // werden die ermittelte Position und der Punktradius in
       // Zeichenbereichskoordianten zurückgegeben.
  TextZeichnungsDaten ggstZeichnenVorbereiten
       (CDC* pDC, int knotenX, int knotenY, int knotenRadius,
       TObject* ggst, int ggstZuKnotenNr, BOOL loeschen=FALSE);
       // Berechnet Position, Ausdehnung und Schriftart einer
       // Beschriftung mit dem Gegenstand <ggst>. <knotenX> und
       // <knotenY> geben die Koordinaten, <knotenRadius> den
       // Radius des zugehörigen Knotens in
       // Zeichenbereichskoordinaten an. <ggstZuKnotenNr> gibt an,
       // wieviele Gegenstandsbeschriftungen zu dem Knoten schon
       // vorher berechnet wurden.
  TextZeichnungsDaten mmlZeichnenVorbereiten
       (CDC* pDC, int knotenX, int knotenY, int knotenRadius,
       TAttribute* mml, int mmlZuKnotenNr, BOOL loeschen=FALSE);
       // Berechnet Position, Ausdehnung und Schriftart einer
       // Beschriftung mit dem Merkmal <mml>. <knotenX> und
       // <knotenY> geben die Koordinaten, <knotenRadius> den
       // Radius des zugehörigen Knotens in
       // Zeichenbereichskoordinaten an. <ggstZuKnotenNr> gibt an,
       // wieviele Merkmalsbeschriftungen zu dem Knoten schon
       // vorher berechnet wurden.
  void textZeichnen(CDC* pDC, TextZeichnungsDaten& zd,
       BOOL loeschen=FALSE);
       // Zeichnet/Löscht einen Beschriftungstext nach den in <zd>
       // enthaltenen Daten.
  int selektierterKnoten(int xPos, int yPos,
       int& knotenX, int& knotenY);
       // Ermittelt die Nummer und Zeichenbereichskoordinaten
       // <knotenX> und <knotenY> eines an (<xPos>,<yPos>)
       // angeklickten Knotens.
  TObject* selektierterGgst(int xPos, int yPos, CDC* pDC);
       // Ermittelt den an (<xPos>,<yPos>) angeklickten
       // Gegenstand.
  TAttribute* selektiertesMml(int xPos, int yPos, CDC* pDC);
       // Ermittelt das an (<xPos>,<yPos>) angeklickte Merkmal.
  void diagrammGroesseAnpassen(double faktor);
       // Skaliert das Diagramm um den Faktor <faktor>. Die
       // Beschriftungsgröße bleibt unverändert.
```

```
  void beschriftungsAbstandAnpassen(double faktor);
       // Multipliziert die Positionsangaben aller Beschriftungen
       // mit <faktor>.
  int setzeBlockfarbe(TDPoint* knoten, CString& farbString);
       // Setzt die Farbe des durch <knoten> gegebenen Blocks.
  double laengenFaktor(TLineDiagram* diag);
       // Rechnet die Längeneinheit von <diag> in
       // Zeichenbereichskoordinaten um.
  double einheitenFaktor(const CString& einheit);
       // Rechnet die Maßeinheit von <diag> in
       // Zeichenbereichskoordinaten um.
  double linienStaerke(const TPointFormat& format);
       // Berechnet die Linienstärke einer Knotenumrandung in
       // Zeichenbereichskoordinaten.
  double linienStaerke(const TLineFormat& format);
       // Berechnet die Linienstärke einer Kante in
       // Zeichenbereichskoordinaten.
  int linienStil(char name);
       // Wandelt die ConScript-Angabe zum Linienstil in einen
       // Wert für die Festlegung des Zeichnestifts um.
  double knotenRadius(const TPointFormat& format);
       // Berechnet einen Knotenradius in
       // Zeichenbereichskoordinaten.
  double schriftGroesse(const TStringFormat& format);
       // Berechnet eine Schriftgröße in
       // Zeichenbereichskoordinaten.
  CString schriftArtName(const TStringFormat& format);
       // Ermittelt den Font-Namen, setzt Standardwert, soweit
       // in <format> nicht angegeben.
  LOGFONT schriftArt(const TStringFormat& format);
       // Ermittelt den in <format> angegebenen Font.
  CSize boxGroesse(CDC* pDC, const CString& text,
       const TStringFormat& format);
       // Berechnet die Ausdehnung der Beschriftung mit <text> in
       // der Formatierung aus <format> in
       // Zeichenbereichskoordinaten.
  int horOffset(TStringFormat& format);
       // Berechnet die horzontale Position einer Beschriftung
       // relativ zum zugehörigen Knoten in
       // Zeichenbereichskoordinaten.
  int verOffsetGgst(TStringFormat& format,int schonGezeichnet);
       // Berechnet die vertikale Position einer Beschriftung mit
       // einem Gegenstand relativ zum zugehörigen Knoten in
       // Zeichenbereichskoordinaten.
  int verOffsetMml(TStringFormat& format,int schonGezeichnet);
       // Berechnet die vertikale Position einer Beschriftung mit
       // einem Merkmal relativ zum zugehörigen Knoten in
       // Zeichenbereichskoordinaten.
};// end CBASEView
```

Abbildungsverzeichnis

Nachfolgend die Liste der enthaltenen Abbildungen.

Diese Arbeit wurde gesetzt mit Adobe FrameMaker von Adobe Systems Incorporated. Zu allen Grafiken, die nicht innerhalb von Framemaker oder dem Werkzeug zu BASE erzeugt wurden, ist angegeben, welche Software zu ihrer Erstellung benutzt wurde. Diesem Zweck dienen die Indices an den Titeln der Abbildungen. Sie sind in der anschließend an die Liste angegebenen Legende aufgeschlüsselt.

Legende

Rose	erstellt mit Rational Rose/C++ Demo 4.0.3, Rational Software Corporation, [Rat 00]
Together	erstellt mit Together/J 3.0 Whiteboard Edition, TogetherSoft LLC, [Tog 00]
Innovator	erstellt mit Innovator 6.1 - Demo-Version, MID GmbH, [Mid 99]
Anaconda	erstellt mit Anaconda 1.0, [Vog 99]
FGT	übernommen aus [FGT 98]
ObjectiF	erstellt mit ObjetiF 4.0 Demo, microTOOL GmbH, [Mic 99]
Paradigm	erstellt mit Paradigm Plus 3.51, Learning Edition, Platinum Technology, [C-A 00]
Excel	erstellt mit MS Excel 97, Microsoft Corporation, [Per 97]

Abkürzungsverzeichnis

BASE	*Begriffsbasiertes Analyseverfahren für die Software-Entwicklung*, Name der vorgestellten Methode
BON	*Business Object Notation*, Methode von Kim Waldén & Jean-Marc Nerson ([W-N 95])
EBNF	*Erweiterte Bauckus-Naur-Form*, Sprache zur Syntaxbeschreibung ([ISO 96])
EOS	Evolutionäre objektorientierte Software-Entwicklung, Vorgehensmodell von Wolfgang Hesse ([H-W 94], [Hes 96])
E/R	*Entity-Relationship*, Notation für Datenmodelle ([Che 76])
FBA	*Formale Begriffsanalyse*
FCA-L	*The Formal Concept Analysis Library*, C++-Klassenbibliothek für Anwendungen der Formalen Begriffsanalyse ([Vog 96])
FRISCO	*Framework of Information System Concepts*, Arbeitskreis zum Thema: Konzeptuelle Grundlagen von Informationssystemen (siehe [FHL+ 98])
GUI	*Graphical User Interface*, graphische Benutzerschnittstelle
HTML	*Hypertext Markup Language* (siehe [M-N 98])
JWI	*Japan Wines Incorporated*, fiktive Weingroßhandlung als Anwendungsbeispiel für BASE
LRDM	*Logical Relational Design Methodology*, Erweiterung des E/R-Modells ([TYF 86])
MFC	*Microsoft Foundation Classes*, Klassenbibliothek für die Entwicklung von Windows-Programmen unter MS Visual C++ ([Krg 96])
OMT	*Object Modeling Technique*, Methode von James Rumbaugh et al. [RBP+ 91]
OOA	*Objektorientierte Analyse*

OOAD	*Object-Oriented Analysis and Design*, Methode von Grady Booch ([Boo 94])
OOA/D	*Object-Oriented Analysis / Design*, Methode von Peter Coad & Edward Yourdon ([C-Y 90], [C-Y 91])
OOA&D	*Object-Oriented Analysis and Design*, Methode von James Martin & James Odell [M-O 92]
OOD	*Objektorientierter Entwurf* ("D" für "design")
OOP	*Objektorientierte Programmierung*
OOSA	*Object-Oriented Systems Analysis*, Methode von Sally Shlaer & Stephen J. Mellor ([S-M 88], [S-M 92])
OOSE	*Object-Oriented Software Engineering*, Methode von Ivar Jacobcon et al. ([JCJÖ 93])
OSA	*Object-Oriented Systems Analysis*, Methode von David W. Embley & Barry D. Kurtz ([EKW 92])
RDD	*Resposibility driven Design*, Methode von Rebecca Wirfs-Brock et al. ([WWW 90])
RGB	*Red Green Blue*, Farbmodell mit additiver Farbmischung aus den Grundfarben Rot, Grün, Blau
SADT	*Structured Analysis and Design Technique*, Notation und Methode für den Systementwurf ([Ros 77])
SDI	*Single Document Interface*, MFC-Framework zur Programmierung von Windows-Programmen, die immer nur eine Datei ("Dokument") bearbeiten (siehe [Krg 96])
UML	*Unified Modeling Language*, Notation für objektorientierte Modelle, Standard der *Object Management Group* ([BJR 97])
WWW	*World Wide Web*, weltumspannender Mythos der 90’er Jahre, siehe "*http://www.w3.org/*"

Index

C

D

E

F

G

H

I

J

K

L

Literatur

Im folgenden angegebene Links datieren vom 14.05.2001.

[Abb 83] R. Abbott: *Program Design by Informal English Descriptions*, Communications of the ACM, vol 26, no. 11, pp. 882-894, November 1983

[ABLN 89] H. Aït-Kaci, R. Boyer, P. Lincoln, R. Nasr: *Efficient Implementation of Lattice Operations*, ACM Transactions on Programming Languages and Systems, vol. 11, no. 1, pp. 115-146, January 1989

[A-G 96] K. Arnold, J. Gosling: *Die Programmiersprache Java*, Dt. Übersetzung von B. Krehl u.a., Addsion Wesley Publishing Company, 1996

[A-H 87] S. Abiteboul, R. Hall: *IFO - A formal semantic model*, ACM Transactons on Database Systems, vol. 12, no. 4, pp. 525-565, December 1987

[Alb 79] A.J. Albrecht: *Measuring application development productivity*, Proceedings of the IBM Applications Development Symposium, Monterey, pp. 83-92, 1979

[A-N 94] A. Arnauld, P. Nicole: *Die Logik oder die Kunst des Denkens*, aus dem Französischen übersetzt, 2. Auflage, Bibliothek klassischer Texte, Wissenschaftliche Buchgesellschaft, Darmstadt, 1994

[And 96] U. Andelfinger: *Diskursive Anforderungsanalyse*, Europäische Hochschulschriften, Reihe Informatik, vol. 25, Peter Lang, Frankfurt am Main, 1996

[Aon 99] Aonix: *Software through Pictures*, "*http://www.aonix.com/content/products/stp/stp.html*", 1999

[Ara 89] G. Arango: *Domain Analysis - From Art Form To Engineering Discipline*, Proceedings of the Fifth International Workshop on Software Specification and Design, Pittsburgh, ACM SIGSOFT Engineering Notes, vol. 14, no. 3, pp. 152-159, May 1989

[B-C 89] K. Beck, W.A. Cunningham: *A laboratory for teaching object-oriented thinking*, in: Norman K. Meyrowitz (Ed.): Conference on Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA 89), New Orleans, Proceedings, ACM SIGPLAN Notices 24(10), October 1989

[BDFS 84] C. Beeri, M. Dowd, R. Fagin, R. Stratman: *On the structure of Armstrong relations for functional dependencies*, Journal of the ACM, vol. 31, pp. 30-46, 1984

[Bec 99] P. Becker: *Einsatz der Formalen Begriffsanalyse zur Dokumentnavigation*, Diplomarbeit, Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, 1999

[Bir 40] G. Birkhoff: *Lattice Theory*, American Mathematical Society Colloquium Publications, vol. XXV, New York, 1940

[Bir 67] G. Birkhoff: *Lattice Theory*, Third edition, American Mathematical Society Colloquium Publications, vol. XXV, Providence, Rhode Island, 1967

[BJR 97] G. Booch, I. Jacobson, J. Rumbaugh: *UML 1.1 Notation Guide*, Rational Software Corporation, "*http://www.rational.com/uml/resources/documentation/formats.jsp"*, 1997

[BJR 99a] G. Booch, I. Jacobson, J. Rumbaugh: *The Unified Modeling Language Reference Manual*, Addison Wesley Object Technology Series, 1999

[BJR 99b] G. Booch, I. Jacobson, J. Rumbaugh: *The Unified Software Development Process*, Addison Wesley Object Technology Series, 1999

[BKKZ 92] R. Budde, K. Kautz, K. Kuhlenkamp, H. Züllighoven: *Prototyping : An Approach to Evolutionary System Development*, Springer-Verlag, Berlin Heidelberg, 1992

[B-M 70] M. Barbut, B. Monjardet: *Ordre et classification: Algèbre et combinatoire*, Série Méthodes mathématiques des sciences de l'homme, Hachette, Paris, 1970

[Boe 76] B.W. Boehm: *Software Engineering*, IEEE Transactions on Computers, C-25.12, pp. 1216-1241, 1976

[Boo 91] G. Booch: *Object-Oriented Design with Applications*, The Benjamin/Cummings Publishing Company, 1991

[Boo 94] G. Booch: *Object-Oriented Analysis and Design with Applications*, Second edition, The Benjamin/Cummings Publishing Company, 1994

[BPR 88] M. Blaha, W. Premerlani, J. Rumbaugh: *Relational Database Design using an Object-Oriented Methodology*, Communications of the ACM, vol. 31, no. 4, April 1988

[Bur 91] P. Burmeister: *Merkmalexploration bei unvollständigem Wissen*, S. 15-46 in: W. Lex (Hrsg.): Arbeitstagung Begriffsanalyse und Künstliche Intelligenz, Informatik-Bericht 89/3, TU Clausthal, 1991

[C-A 00] Computer Associates International: *Paradigm Plus - Enterprise Component Modeling*, "*http://www.cai.com/products/alm/paradigm_plus.htm*", 2000

[CAB+ 94] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremaes: *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Englewood Cliffs, New Jersey, 1994

[C-E 96] R.J. Cole, P.W. Eklund: *Application of Formal Concept Analysis to Information Retrieval using a Hierarchically Structured Thesaurus*, Proceedings of the 4th International Conference on Conceptual Structures ICCS '96, Sydney, pp. 1-12, University of New South Wales, LNAI, Springer-Verlag, 1996

[C-E 99] R.J. Cole, P.W. Eklund: *Analysing an Email Collection using Formal Concept Analysis*, 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases PKDD'99, LNCS 1704, Springer-Verlag, 1999

[CEG 97] R.J. Cole, P.W. Eklund, B. Groh: *Dealing with Large Contexts in Formal Context Analysis: A Case Study Using Medical Texts*, International Symposium on Knowledge Retrieval, Use, and Storage for Efficiency, KRUSE-97, 1997

[C-F 92] D. de Champeaux, P. Faure: *A comparative study of object-oriented analysis methods*, Journal of Object-Oriented Programming, vol. 5, no. 1, pp. 21-33, March/April 1992

[Che 76] P.P. Chen: *The entity-relationship model, Toward an unified view of data*, ACM Transactions on Database Systems, vol. 1, no. 1, pp. 9-36, March 1976

[C-Y 90] P. Coad, E. Yourdon: *Object-oriented analysis*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, New Jersey, 1990

[C-Y 91] P. Coad, E. Yourdon: *Object-oriented design*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, New Jersey, 1991

[DeM 78] T. DeMarco: *Structured Analysis and System Specification*, Prentice Hall, Englewood Cliffs New Jersey, 1978

[D-H 98a] S. Düwel, W. Hesse: *Bestimmung von Objektkandidaten mit Hilfe von Formaler Begriffsanalyse*, in: K. Pohl, A. Schürr, G. Vossen (Hrsg.): Proceedings Modellierung '98, Bericht Nr. 6/98-I, Angewandte Mathematik und Informatik, Universität Münster, auch unter "*http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-9*", 1998

[D-H 98b] S. Düwel, W. Hesse: *Identifying Candidate Objects During System Analysis*, Proceedings CAiSE'98/IFIP 8.1 3rd International Workshop on Evaluation of Modeling Methods in System Analysis and Design (EMMSAD'98), Pisa, 1998

[DIN 79] Deutsches Institut für Normung, Normenausschuß Terminologie (NAT): *DIN 2330 - Begriffe und Benennungen, Allgemeine Grundsätze*, Beuth Verlag, Berlin Köln, März 1979

[Duf 89] C.A. Dufour: *Die Lehre der Proprietates Terminorum, Sinn und Referenz in mittelalterlicher Logik*, Philosophia Verlag, München Hamden Wien, 1989

[Duq 87] V. Duquenne: *Contextual implications between attributes and some representation properties for finite lattices*, in [GWW 87], 1987

[EKSW 00] D. Eschenfelder, W. Kollewe, M. Skorsky, R. Wille: *Ein Erkundungssystem zum Baurecht: Methoden der Entwicklung eines TOSCANA-Systems*, in: [S-W 00], 2000

[EKW 92] D.W. Embley, B.D. Kurtz, S.N. Woodfield: *Object-Oriented Systems Analysis, A Model-Driven Approach*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, New Jersey, 1992

[Erd 98] M. Erdmann: *Formal Concept Analysis to Learn from the Sisyphus-III Material*, In: Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'98), Banff, Canada, 1998

[Ern 82] M. Erné: *Einführung in die Ordnungstheorie*, B.I. Wissenschaftsverlag, Zürich, 1982

[FGT 98] P. Franke, T. Graf, M. Trouvain: *Einarbeitung in UML 1.1 anhand eines Fallbeispiels*, Fortgeschrittenenpraktikum Informatik im WS 1997/98, "*http://www.mathematik.uni-marburg.de/~hesse/uml/Welcome.html*", 1998

[FHL+ 98] E. Falkenberg, W. Hesse, P. Lindgreen, B.E. Nilsson, J.L.H. Oei, C. Rolland, R.K. Stamper, F.J.M. Van Assche, A.A. Verrijn-Stuart, K. Voss: *FRISCO - A Framework of Information System Concepts - The FRISCO Report*, "*http://www.liacs.nl/~verrynst/frisco.html*", 1998

[FKM+ 96] G. Fliedl, C. Kop, W. Mayerthaler, H.C. Mayr, C. Winkler: *NTS-Based Derivation of KCPM Cardinalities: From Natural Language to Conceptual Predesign*, in R.P. van de Riet, J.F.M. Burg, A.J. van der Vos (eds): Applications of Natural Language to Information Systems, Proceedings of the Second International Workshop, Amsterdam, June 26-28, 1996

[FLS 95] P. Funk, A. Lewien, G. Snelting: *Algorithms for Concept Lattice Decomposition and their Applications*, Computer Science Report 95-09, Technical University Braunschweig, December 1995

[Fow 97] M. Fowler: *Analysis patterns : Reusable object models*, Addison-Wesley, Series in Object-Oriented software engineering, 1997

[F-S 97] M. Fowler, K. Scott: *UML Distilled, Applying the Standard Object Modeling Language*, Addison-Wesley Object Technology Series, 1997

[Gan 87] B. Ganter: *Algorithmen zur Formalen Begriffsanalyse*, in [GWW 87], 1987

[Gan 99] B. Ganter: *Attribute exploration with background knowledge*, Theoretical Computer Science 217, S. 215-233, 1999

[Gan 00] B. Ganter: *Begriffe und Implikationen*, in [S-W 00], 2000

[G-D 86] J.L. Guigues, V. Duquenne: *Familles minimales d'implications informatives résultant d'un tableau de données binaires*, Mathématiques et Sciences Humaines, no. 95, Editions de l'Ecole des Hautes Etudes en Sciences Sociales, Paris, 1986

[GHJV 95] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesley, Professional computing series, 1995

[GKr 99] B. Ganter, R. Krauße: *Pseudo Models and Propositional Horn Inference*, Preprint TU Dresden, MATH-AL-15-1999, 1999

[GKu 98] B. Ganter, S. O. Kuznetsov: *Stepwise construction of the Dedekind-MacNeille completion*, Proceedings ot the 6th International Conference on Conceptual Structures ICCS '98, Montpellier, August 1998

[GMl 93] R. Godin, H. Mili: *Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices*, in: A. Paepcke (Ed.): Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 93), ACM Press, Washington, 1993

[GMs 94] R. Godin, R. Missaoui: *An incremental concept formation approach for learning from databases*, Theoretical Computer Science 133, S. 387-419, 1994

[GMA 95] R. Godin, R. Missaoui, H. Alaoui: *Incremental concept formation algorithms based on Galois (concept) lattices*, Computational Intelligence, 11(2), pp. 246-267, 1995

[GMM+ 98] R. Godin, H. Mili, G.W. Mineau, R. Missaoui, A. Arfi, T.-T. Chau: *Design of Class Hierarchies based on Concept (Galois) Lattices*, Theory and Application of Object Systems (TAPOS), 4(2), pp. 117-134, 1998

[G-R 83] A. Goldberg, D. Robson: *Smalltalk-80, The Language and its Implementation*, corr. Reprint, Xerox Corporation, Series in Computer Science, Addison-Wesley Publishing Company, 1983

[Grä 98] G. Grätzer: *General Lattice Theory*, Second edition, Birkhäuser Verlag, Basel, 1998

[GSW 98] B. Groh, S. Strahringer, R. Wille: *TOSCANA-Systems based on thesauri*, in: M.L. Mugnier, M. Chein (eds.): Conceptual structures: theory, tools and applications, LNAI 1453, Springer-Verlag, pp. 127-138, 1998

[G-W 96] B. Ganter, R. Wille: *Formale Begriffsanalyse, Mathematische Grundlagen*, Springer-Verlag, Berlin Heidelberg New York, 1996

[GWW 87] B. Ganter, R. Wille, K.E. Wolff (Hrsg.): *Beiträge zur Formalen Begriffsanalyse, Vorträge der Arbeitstagung Begriffsanalyse*, Darmstadt 1986, B.I. Wissenschaftsverlag Zürich, 1987

[Hal 71] R. Haller: *Begriff*, in: J. Ritter: *Historisches Wörterbuch der Philosophie*, Band 1: A-C, Wissenschaftliche Buchgesellschaft Darmstadt, 1971

[Har 87] D. Harel: *Statecharts: A visual formalism for complex systems*, Science of Computer Programming, vol. 8, pp. 231-274, 1987

[Hes 96] W. Hesse: *Theory and Practice of the Software Process - a Field Study and its Implications for Project Management*, in: C. Montangero (Ed.): Software Process Technology, 5th European Workshop, EWSPT 96, LNCS 1149, Springer-Verlag, pp. 241-256, 1996

[Hes 97] W. Hesse: *Life Cycle Models for Object-Oriented Software Development*, in: [ZGHK 97], 1997

[Hes 98] W. Hesse: *Baustein-orientiert statt phasen-zentriert: Neue Entwicklungsmethoden erfordern neuartige Vorgehensmodelle*, Proceedings GI-Workshop "Änderung von objektorientierten Entwicklungsstrategien und deren Unterstützung durch Vorgehensmodelle", Frankfurt am Main, Juni 1998

[H-F 85] Y. Hanatani, R. Fagin: *A Simple Characterization of Database Dependency Implication*, IBM Research Report RJ 4777, San Jose, 1985

[HMF 92] W. Hesse, G. Merbeth, R. Frölich: *Software-Entwicklung - Vorgehensmodelle, Projektführung und Produktverwaltung*, Handbuch der Informatik, Band 5.3, Oldenbourg Verlag, München, 1992

[H-W 94] W. Hesse, F. Weltz, F.: *Projektmanagement für evolutionäre Software-Entwicklung*, Information Management, Ausg. 9, Nr. 3, S. 20-33, März 1994

[ISO 96] ISO/IEC 14977: *Information technology - Syntactic metalanguage - Extended BNF*, 1996

[Jac 99] I. Jacobson: *Applying UML in the Unified Process*, Keynote in UML World Conference, "*http://www.rational.com/products/reading/reading_rup.jsp*", March 1999

[J-C 95] I. Jacobson, M. Christerson: *Modeling with Use Cases, A growing consensus on use cases*, Journal of Object-Oriented Programming, vol. 8, no. 1, pp. 15-19, March/April 1995

[JCJÖ 93] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard: *Object-Oriented Software Engineering - A Use Case Driven Approach*, Revised Printing, ACM-Press, Addison-Wesley, 1993

[Kle 56] C.C. Kleene: *Representation of events in nerve nets and finite automata*, in: C. Shannon, J. McCarthy: Automata Studies, Princeton University Press, pp. 3-41, 1956

[K-M 98a] C. Kop, H.C. Mayr: *Conceptual Predesign - Bridging the Gap between Requirements and Conceptual Design*, Proceedings of the 3rd International Conference on Requirements Engineering (ICRE 98), Colorado Springs, April 1998

[K-M 98b] C. Kop, H.C. Mayr: *Conceptual Predesign as a Stopover for Mapping Natural Language Reqirements Sentences to State Chart Patterns*, ECOOP 98, Workshop: Automating the Software Development Process, Brüssel, June 1998

[K-N 95] R. E. Kent, C. Neuss: *Conceptual Analysis of Resource Meta-Information*, Computer Networks and ISDN Systems, 27, pp. 973-984, 1995

[K-N 96] R. E. Kent, C. Neuss: *Web Conceptual Space*, Proceedings of WebNet '96, San Francisco, October 1996

[KNS 92] G. Keller, M. Nüttgens, A.-W. Scheer: *Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK)*, Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Universität des Saarlandes, Heft 89, Januar 1992

[Krg 96] D. Kruglinski: *Inside Visual C++ : Version 4*, Third Edition, Microsoft Press, 1996

[Krc 99] P. Kruchten: *The Rational Unified Process: An Introduction*, Addison Wesley Object Technology Series, 1999

[KSVW 94] W. Kollewe, M. Skorsky, F. Vogt, R. Wille: *TOSCANA - ein Werkzeug zur begrifflichen Analyse und Erkundung von Daten*, in: R. Wille, M. Zickwolff: Begriffliche Wissensverarbeitung, Grundfragen und Aufgaben, B.I. Wissenschaftsverlag, Mannheim, 1994

[Kun 91] E. Kunz: *Algebra*, Friedrich Vieweg & Sohn Verlagsgesellschaft, Braunschweig, 1991

[Lin 95] C. Lindig: *Komponentensuche mit Begriffen*, Proceedings Softwaretechnik '95, Braunschweig, S. 67-75, Oktober 1995

[Lin 98] C. Lindig: *Analyse von Softwarevarianten*, Informatik-Bericht 98-04, Technische Universität Braunschweig, Januar 1998

[Lin 99] C. Lindig: *Algorithmen zur Begriffsanalyse und ihre Anwendung bei Softwarebibliotheken*, Dissertation, Technische Universität Braunschweig, "*http://www.gaertner.de/~lindig/diss*", 1999

[L-S 97] C. Lindig, G. Snelting: *Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis*, Proceedings of the International Conference on Software Engineering (ICSE 97), Boston, USA, pp. 349-359; 1997

[L-W 00] D. Leffingwell, D. Widrig: *Managing Software Requirements: A Unified Approach*, Addison Wesley Object Technology Series, 2000

[L-Z 74] B.H. Liskov, S. Zilles: *Programming with Abstract Data Types*, Sigplan Notices, vol. 9, no. 4, pp. 50-59, 1974

[Mcc 76] T.J. McCabe: *A Complexity Measure*, IEEE Transactions on Software Engineering, vol. 2, no. 4, pp. 308-320, December 1976

[Mey 88] B. Meyer: *Object oriented Software Construction*, Prentice Hall International, 1988

[Mey 90] B. Meyer: *Lessons from the Design of the Eiffel Libraries*, Communications of the ACM, vol. 33, no. 9, pp. 68-88, September 1990

[Mey 92] B. Meyer: *Eiffel, The language*, Prentice Hall, Object-oriented Series, 1992

[Mic 99] microTOOL GmbH: *objectiF*, "*http://www.microtool.de/objectiF*", 1999

[Mid 99] MID GmbH: *Innovator*, "*http://www.mid.de*", 1999

[Mil 56] G.A. Miller: *The magical number seven, plus or minus two: Some limtis on our capacity for processing information*, Psychological Review, vol. 63, pp. 81-97, 1956

[M-N 98] S. Münz, W. Nefzger: *HTML 4.0 Handbuch*, Franzis-Verlag, oder "*http://www.teamone.de/selfaktuell*", 1998

[M-O 92] J. Martin, J. Odell: *Object-Oriented Analysis and Design*, Prentice Hall, Englewood Cliffs, New Jersey, 1992

[M-O 95] J. Martin, J. Odell: *Object-Oriented Methods, A Foundation*, Prentice Hall, Englewood Cliffs, New Jersey, 1995

[M-P 92] D.E. Monarchi, G.I. Puhr: *A Research Typology for Object-Oriented Analysis and Design*, Communications of the ACM, Vol. 35, No. 9, pp. 35-47, September 1992

[MSW 99] G. Mineau, G. Stumme, R. Wille: *Conceptual Structures Represented by Conceptual Graphs and Formal Concept Analysis*, Preprint Nr. 2034, Technische Universität Darmstadt, Fachbereich Mathematik, April 1999

[Ore 44] O. Ore: *Galois Connexions*, Transactions of the American Mathematical Society, Vol. 55, 1944

[Ort 95] E. Ortner: *Elemente einer methodenneutralen Konstruktionssprache für Informationssysteme*, Informatik - Forschung und Entwicklung, Band 10, Ausgabe 3, S. 148-160, 1995

[Ort 97] E. Ortner: *Methodenneutraler Fachentwurf*, Reihe Wirtschaftsinformatik, Teubner, Leipzip, 1997

[Per 97] Perspection, Inc.: *Microsoft® Excel 97 At a Glance*, Microsoft Press, 1997

[P-F 87] R. Pieto-Díaz, P. Freeman: *Classifying Software for Reusability*, IEEE Software, vol. 4, no. 1, January 1987

[Rat 00] Rational Software Corporation: *Rational Rose*, "*http://www.rational.com/products/rose/index.jtmpl*", 2000

[RBP+ 91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen: *Object Oriented Modelling and Design*, Prentice Hall, Englewood Cliffs, New Jersey, 1991

[Rei 91] U. Reimer: *Einführung in die Wissensrepräsentation: netzartige und schema-basierte Repräsentationsformate*, Teubner, Leitfäden der angewandten Informatik, Stuttgart, 1991

[R-G 92] K.S. Rubin, A. Goldberg: *Object Behavior Analysis*, Communications of the ACM, vol. 35, no. 9, pp. 48-62, September 1992

[Ris 70] W. Risse: *Logik der Neuzeit*, 2. Band 1640 - 1780, Friedrich Fromann Verlag (Günther Holzboog), Stuttgart - Bad Cannstadt, 1970

[RMD 98] V.E. van Reijswoud, J.B.F. Mulder, J.L.G. Dietz, *Communicative action-based business process and information systems modelling with DEMO*, Information Systems Journal, vol. 9, iss. 2, pp. 117-139, April 1998

[Ros 77] T.D. Ross: *Structured analysis (SA): A language for communcating ideas*, IEEE Transactions on Software Engineering, vol. SE-3, no. 1, 1977

[Rum 95] J. Rumbaugh: *OMT: The functional model*, Journal of Object-Oriented Programming, vol. 8, no. 1, pp. 10-14, March/April 1995

[Rum 96] J. Rumbaugh: *OMT Insights, Perspectives on Modeling from the Journal of Object-Oriented Programming*, SIGS Books, New York, 1996

[R-W 00] T. Rock, R. Wille: *Ein TOSCANA-Erkundungssystem zur Literatursuche*, in: [S-W 00], 2000

[Schi 97] B. Schienmann: *Objektorientierter Fachentwurf*, Texte zur Informatik, Band 20, Teubner, Leipzig, 1997

[Sche 99] P. Schefe: *Softwaretechnik und Erkenntnistheorie*, Informatik-Spektrum, Band 22, Heft 2, S. 122-135, April 1999

[SIS 87] Swedish Institute for Standards: *SIMULA Standard, DATA processing - Programming languages - SIMULA*, Swedish Standard SS 63 61 14, Stockholm 1987

[S-M 88] S. Shlaer, S.J. Mellor: *Object-Oriented Systems Analysis, Modeling the World in Data*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, New Jersey, 1988

[S-M 92] S. Shlaer, S.J. Mellor: *Object Lifecycles, Modeling the World in States*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, New Jersey, 1992

[S-M 93] S. Shlaer, S.J. Mellor: *The Shlaer-Mellor Method*, Project Technology, Inc., "*http://www.projtech.com/info/smmethod.html*", 1993

[Sne 96] G. Snelting: *Reengineering of configurations based on mathematical concept analysis*, ACM Transactions on Software Engineering and Methodology, 5(2), pp.146--189, April 1996

[Som 92] I. Sommerville: *Software Engineering*, 4th edition, Addison-Wesley Publishing Company, 1992

[Sow 92] J.F. Sowa: *Conceptual Graphs Summary*, in: T.E. Nagle, J.A. Nagl, L.L. Gerholz, P.W. Eklund (Eds.): Conceptual Structures - Current Research and Practice, Ellis Horwood Workshops, pp. 3-51, 1992

[SSB 91] H. Schaschinger, H. Sikora, I. Bäuchler: *Objektorientierte Analyse- und Designmethoden - Überblick und kritische Betrachtung*, Softwaretechnik-Trends 11.4, S. 32-43, November 1991

[S-T 99] G. Snelting, F. Tip: *Reengineering Class Hierarchies Using Concept Analysis*, Forschungsbericht MIP-9910, Universität Passau, Fakultät für Mathematik & Informatik, 1998

[Stan 95] Standish Group International, Inc.: *CHAOS*, Sample Research Paper, "*http://standishgroup.com/visitor/chaos.htm*", 1995

[Stan 96] Standish Group International, Inc.: *Unfinished Voyages*, Sample Research Paper, "*http://standishgroup.com/visitor/voyages.htm*", 1996

[Ste 93] W. Stein: *Objektorientierte Analysemethoden - ein Vergleich*, Informatik-Spektrum 16, S. 317-332, 1993

[Str 92] B. Stroustrup, *Die C++ - Programmiersprache*, 2. überarbeitete Auflage, Addison-Wesley, 1992

[S-W 00] G.Stumme, R.Wille (Hrsg.): *Begriffliche Wissensverarbeitung. Methoden und Anwendungen*, Springer-Verlag, Berlin-Heidelberg, 2000

[Szy 98] C. Szyperski: *Component Software, Beyond Object-Oriented Programming*, ACM Press, Addison-Wesley, 1998

[Tic 85] W. F. Tichy: *RCS - A System for Version Control*, Software Practices & Experience, July 1985

[Tog 00] TogetherSoft LLC: *Together/J*, "*http://www.togethersoft.com/together*", 2000

[TYF 86] T.J. Teorey, D. Yang, J.P. Fry: *A logical design methodology for relational databases using the extended entity-relationship model*, ACM Computing Surveys, vol 18, no. 2, June 1986

[Vog 96] F. Vogt: *Formale Begriffsanalyse mit C++, Datenstrukturen und Algorithmen*, Springer-Verlag, Berlin Heidelberg, 1996

[Vog 97] F. Vogt: *Supporting Communication in Software Engineering: An Approach Based on Formal Concept Analysis*, Preprint Nr. 1926, Technische Universität Darmstadt, Fachbereich Mathematik, 1997

[Vog 99] F. Vogt: *Anaconda - Interaktiver Editor für ConScript-Dateien*, "*http://www.mathematik.tu-darmstadt.de/ags/ag1/Software/Anaconda/Welcome_de.html*", 1999

[Vos 99] G. Vossen: *Datenbankmodelle, Datenbanksprachen und Datenbankmanagement-Systeme*, 3. Auflage, Oldenbourg Verlag, München, 1999

[Wei 88] M. Weitz: *Theories of Concepts, A History of the Mayor Philosophical Tradition*, Routledge, London, 1988

[W-F 86] T. Winograd, F. Flores: *Understanding Computers and Cognition: A New Foundation for Design*, Norwood, NJ: Ablex, 1986

[Wil 95] R. Wille: *Begriffsdenken: Von der griechischen Philosophie bis zur Künstlichen Intelligenz heute*, Preprint Nr. 1724, Technische Universität Darmstadt, Fachbereich Mathematik, 1995

[Wir 75] N. Wirth: *Algorithmen und Datenstrukturen*, Teubner Studienbücher Informatik, Stuttgart, 1975

[Wir 83] N. Wirth: *Systematisches Programmieren*, 4. Auflage, Teubner Studienbücher Informatik, Stuttgart, 1983

[W-N 95] K. Waldén, J.-M. Nerson: *Seamless Object-Oriented Software Architecture: Analysis and Design of Reliable Systems*, Prentice Hall, Object-Oriented Series, 1995

[W-O 92] F. Weltz, R. G. Ortmann: *Das Software-Projekt, Projektmanagement in der Praxis*, Campus-Verlag, Frankfurt am Main 1992

[WWW 90] R. Wirfs-Brock, B. Wilkerson, L. Wiener: *Designing Object-Oriented Software*, Prentice Hall Englewood Cliffs, New Jersey, 1990

[Y-C 79] E. Yourdon, L.L. Constantine: *Structured Design, Fundamentals of a Discipline of Computer Program and Systems Design*, Yourdon Press, Computing Series, Prentice Hall Englewood Cliffs, New Jersey, 1979

[ZGHK 97] A. Zendler, S. Gastinger, W. Hesse, P. Kosiuczenko: *Advanced Concepts, Life Cycle Models and Tools for Object-Oriented Software Development*, FAST - Forschungsinstitut für Angewandte Software-Technologie, Reihe Softwaretechnik 7, Tectum-Verlag, Marburg, 1997

Lebenslauf

Name		Stephan Düwel
Geboren am	30.08.1967	in Paderborn
Staatsangehörigkeit		Deutsch
Familienstand		verheiratet, ein Kind
Schulausbildung	08/78 - 05/87	Gymnasium Schloß Neuhaus
	05/87	Abschluß: Allgemeine Hochschulreife
Wehrdienst	07/87 - 09/88	Wuppertal und Augustdorf
Studium	10/88 - 09/94	Diplom-Mathematik mit Nebenfach Informatik an der Universität/GH Paderborn
	09/92 - 04/93	Auslandsstudium an der Universität Lüttich (Mathematik)
	09/94	Abschluß als Diplom-Mathematiker mit einer Arbeit im Gebiet der topologischen Vektorräume
	10/94 - 04/95	Ergänzungsstudium Mathematik und Informatik Lehramt
Berufliche Tätigkeit	04/95 - 04/00	Wissenschaftlicher Mitarbeiter am Fachbereich Mathematik und Informatik der Philipps-Universität Marburg, Fachgebiet: Software Engineering