Schemakorrespondenzaxiome unterstützen die paargrammatische Spezifikation inkrementeller Integrationswerkzeuge

Stefan Gruner

Lehrstuhl für Informatik III
der Math.-Naturwissensch. Fakultät
der Rhein.-Westf. Techn. Hochschule,
Ahornstraße 55 / D-52056 Aachen

Contents (Summary)
In this paper, a new method supporting the well-known graph grammar specification approach to developing fine-grained and incrementally operating integration tools is presented.

Section 1, the introductory section, sketches the historical and economical preconditions which induced the need for integration tools as well as the application of integration tools in a working environment. The essential function of any integration tool seems to be some partial reduction of creative to mechanic work. Structural knowledge of the source and target integration domains is most important for the specification of such tools.

Section 2 gives an overview of related integration topics. Much work has already been done in the fields of software engineering, but also in modern data base applications the integration of different data base schemas becomes more and more important. Most of all integration tools in software engineering are focussed on batch transformation or message interchange on a low declarative level. In opposite, the data bench approaches are usually concerned with the very schema integration and fail to develop sophisticated rules for transforming the data themselves.

Section 3 sketches the graph grammar specification of an integration task by example. The documents to be integrated are represented as graphs. A set of graph transformation rules for each graph allows the documents to be manipulated in a working environment. Thereby, integration is specified by coupling corresponding graph productions of the source and target side to each other. The gestalt of syntactically correct document graphs is determined by graph schemas that mainly consist of structure definitions. The example motivates the new method of coupling already the graph schemas in order to enforce a proper choice of integrating graph transformation rules. Thus, the presented approach is to combine the advantages of dynamic graph transformation and static schema integration, which the attention is focussed on here.

Section 4 explaines the semantics of the new correspondence edges between graph schemas in detail.Therefore, it turns out to be important again that graph schemas are required to be —mathematically speaking— lattices, a property which has been demanded for practical reasons already in earlier literature. Correspondence edges, also called integration axioms, offer the possibility to ensure useful transformations as well as to explicitely prohibit undesired results.

Section 5 concludes this paper with some remarks about further work to be done. It would be very useful to construct an editor for defining integration axioms "by mouseclick" and to incorporate this editor into the already existing integration tool development environment. A bigger integration task in the field of chemical process engineering will be used to further examine the applicability of the schema correspondence method.

1 Einführung

Die Herstellung komplexer S y s t e m e, seien es materielle Apparate und Anlagen oder immaterielle Programme für digitale Rechenautomaten, ist ein Charakteristikum der heutigen Produktionsweise, die sich von der Arbeitsorganisation früherer Gesellschaften vor allem in der Art der Arbeitsteilung zwischen Arbeitskräften und Waren unterscheidet. Früher wurde die Arbeit durch die Zünfte hauptsächlich nach Warenart verteilt, z.B. auf Faßbinder, Radmacher, Tischler u.s.w. Spätestens mit Einführung der Fließbandarbeit durch Ford wird der größte Anteil der gesellschaftlichen Arbeit nach Arbeitsschritten geteilt, sodaß —obiges Beispiel abschließend— ein Holzarbeiter sowohl an der Herstellung eines Fasses als auch eines Rades oder Tisches beteiligt sein könnte. Die eben skizzierte ökonomische Entwicklung spiegelt sich in der Wissenschaft der Informatik wider [1] —man vergleiche nur das Konzept des Algorithmus mit Taylors Untersuchungen zur Arbeitseffizienz— und wirkt dadurch beschleunigend auf die Welt der Arbeit zurück. Der Produktionsprozeß einer Ware wird nämlich in vielen Fällen am Rechner schon vorgeplant; die dazu notwendigen Programme aber sind selbst wieder Waren, die in sog. Softwarefabriken in Arbeitsteilung hergestellt werden.

Nicht nur die Arbeitsteilung, sondern auch die Arbeit selbst kann hinsichtlich ihres Ablaufes und ihrer Ergebnisse noch differenzierter beschrieben werden. Mechanische Arbeit ist stark determiniert, d.h.: sowohl das Arbeitsprodukt als auch die zu seiner Realisierung notwendigen Arbeitsschritte sind zu Beginn der Arbeit schon sehr genau bekannt. Kreative Arbeit hingegen ist schwach determiniert, d.h.: sowohl über das Arbeitsprodukt als auch über die Schritte dorthin sind zu Beginn der Arbeit nur ganz vage Vorstellungen vorhanden [2]. Charakteristisch für kreative Tätigkeiten, zu denen Entwurfsarbeiten jeglicher Art gezählt werden müssen, ist daher ihre Vorläufigkeit und Fehleranfälligkeit, wodurch all ihre Ergebnisse als letztlich provisorisch erscheinen. In der Praxis bedeutet dies, daß viele Entwurfsschritte in den einzelnen Arbeitsteilen bis zur allgemeinen Zufriedenheit rückgängig gemacht und von neuem getan werden müssen. Da alle Arbeitsteile voneinander abhängen und Änderungen in einem einzelnen Teil in allen anderen Teilen angemessen beantwortet werden müssen, ist die Wahrscheinlichkeit, daß dabei schließlich heilloses Durcheinander entsteht, nicht gering. Als Beispiel wäre hier wieder die Herstellung eines großen Softwaresystems zu nennen, die mit der Übersetzung der vage formulierten Wünsche des Kunden in eine semiformale Spezifikationssprache beginnt, daraus die modulare Grobarchitektur definiert und deren Funktionalität dann im Detail implementiert. Schließlich wird dazu ein Handbuch, welches alle wesentlichen Teile des Softwaresystems beschreibt und erläutert, verfaßt.

Ein gutes I n t e g r a t i o n s w e r k z e u g ist in einer solchen Situation eine unschätzbare Hilfe. Es kann Änderungen in einem Arbeitsteil an die anderen korrekt weitermelden und die dort in Konsequenz notwendigen Änderungen u.U. sogar automatisch durchsetzen. So hofft man die Konsistenz des Gesamtsystems stets zu wahren. Integrationswerkzeuge dienen also der Organisation verteilter, rechnerunterstützter Entwurfsarbeit und kompensieren einen wesentlichen Teil der Kommunikation, mit der die Reduktion der Komplexität in den einzelnen Arbeitsteilen erkauft werden muß[1]. Her-

[1]Der Begriff 'Werkzeug' hat sich für Hilfsprogramme dieser Art eingebürgert, was die Subsumption sowohl körperlicher als auch geistiger Tätigkeit unter den Begriff 'Arbeit' voraussetzt. Falls man Wert auf Differenzierung legt, könnte man von Rechnerprogrammen auch als 'Denkzeuge' sprechen [3], da sie uns in ihrem Einsatz geistige Mühen ersparen, so wie Werkzeuge uns körperliche. Durch die

stellung und Anwendung eines Intgrationswerkzeuges bedeuten daher nichts anderes als die Verbesserung des Verständnisses eines bestimmten Ausschnitts eines in seiner Gesamtheit noch sehr kreativen Prozesses. Der Pfad eines gesamten Entwicklungsprozesses durch bekanntes und unbekanntes Terrain wird in [4] anschaulich dargestellt. Die wesentliche Funktion eines jeden Integrationswerkzeuges erweist sich hiernach als partielle Reduktion kreativer zu mechanischer Arbeit. Die mit dieser Reduktion verbundene Steigerung der Arbeitseffizienz begründet die wachsende Nachfrage an solchen Werkzeugen. Integrationswerkzeuge für die Softwareentwicklung nach oben erwähnter Methode und für weitere Integrationsprobleme sind bereits implementiert und in [2] ausführlich beschrieben.

Integrationswerkzeuge lassen sich hinsichtlich ihrer inneren Methodik nach ganz unterschiedlichen Kriterien klassifizieren [2]. Hinsichtlich ihrer Stellung im Arbeitsprozeß unterscheidet man die Werkzeuge hauptsächlich in direkt oder indirekt unterstützende. In direkt unterstützenden Werkzeugen ist Handlungswissen gespeichert; sie erleichtern dem Benutzer die Wahl des nächsten Arbeitsschrittes. Indirekt unterstützende Werkzeuge speichern Materialwissen und erleichtern dem Benutzer die korrekte Bearbeitung seiner Materie, ohne ihm die Reihenfolge seiner Handlungen vorzuschreiben. Letzteren gilt die Aufmerksamkeit in diesem Aufsatz.

Voraussetzung für werkzeugunterstützte Integration sind möglichst genaue Kenntnisse der Zusammenhänge zwischen den einzelnen Integrationsbereichen. O.b.d.A. werden hier lediglich zwei Bereiche unterschieden, zwischen denen ein Integrationswerkzeug die Brücke schlagen soll. Für mehrere Arbeitsteile werden dann lediglich mehrere zweiseitige Integrationswerkzeuge benötigt. Die Zwischenergebnisse der Entwurfsarbeit werden auf beiden Seiten in Dokumenten gespeichert. Das Integrationswissen, welches dem Integrationswerkzeug mitgegeben werden muß, stellt eine Relation zwischen den beiden Integrationsdokumenten dar. Integrationswissen ist Strukturwissen. Je feiner die einzelnen Dokumente in strukturelle Einheiten, sog. Inkremente, zerlegbar sind, umso präziser und reichhaltiger wird die Darstellung der Abhängigkeiten zwischen beiden Seiten.

In diesem Aufsatz wird eine neue Methode vorgestellt, solche Zusammenhänge zu explizieren. Die S c h e m a k o r r e s p o n d e n z m e t h o d e erleichtert nicht nur dem versierten Spezifikator den Bau neuer Integrationswerkzeuge —der Entwurf von Integrationswerkzeugen ist ja seinerseits ein nicht triviales Integrationsproblem— sondern gibt auch dem laienhaften Auftraggeber eines Integrationswerkzeuges eine simple Bildersprache, mittels derer die Anforderungen an das gewünschte Werkzeug hinreichend genau beschrieben werden können, zur Hand.

Kapitel 2 nennt zunächst einige verwandte Arbeiten und erläutert deren Unterschiede zum hier vorgestellten Ansatz. Kapitel 3 motiviert anhand eines Beispiels aus dem praxisrelevanten Gebiet der Renovierung alternder Anwendungssoftware (Reengineering, Redesign) die bewährte graphgrammatische Methode der Integratorenspezifikation. Darauf bauend stellt Kapitel 4 die Schemakorrespondenzmethode vor, durch welche graphgrammatische Spezifikationen deutlich erleichtert und verbessert werden sollen. Kapitel 5 beschließt diesen Aufsatz und weist die Richtung der weiteren Forschung.

---

fortschreitende Auflösung des cartesianischen Dualismus in der modernen Philosophie könnte diese Differenzierung allerdings eines Tages gänzlich obsolet erscheinen.

2 Vergleich mit anderen Arbeiten

In den Schriften zur S o f t w a r e t e c h n i k sind bereits zahlreiche Integrationslösungen vorgestellt worden. Da ihre Aufzählung den Rahmen dieses Aufsatzes sprengen würde, werden hier exemplarisch nur die wesentlichen Integrationskonzepte besprochen, die den verschiedenen Werkzeugen unterliegen. Häufig handelt es sich dabei um problemspezifische Ansätze, die sich als nur bedingt verallgemeinerbar erweisen.

2.1 Syntaxunabhängige Integration

Einige arbeitsbereichsübergreifende Integrationsansätze berücksichtigen nur Dokumente als ganze, jedoch nicht deren innere Struktur. Dazu gehören verschiedene CAD- und Datenverwaltungssysteme im Ingenieurwesen [5,6] und Konfigurationsverwalter in der Softwaretechnik [7,8], welche z.B. dafür sorgen, daß die korrekte Reihenfolge verschiedener Programmversionen bezüglich der Kompatibilitätsordnung erhalten bleibt, oder daß neue Module nicht ungeprüft in das Gesamtsystem eingefügt werden.

Hypertextsysteme [9,10] erlauben hingegen zwar die Definition beliebiger, feingranularar Beziehungen zwischen den Inkrementen der zu integrierenden Dokumente, gebrauchen aber die syntaktische Regularität der Dokumentstrukturen ebenfalls nicht. So bleibt jegliche Konsistenzkontrolle auf die Suche nach fehlenden Verbindungen beschränkt und unterstützt die semantische Korrektheit der Gesamtkonfiguration nur spärlich.

Nachrichtenverteiler (broadcast message server) dienen dazu, Änderungsnachrichten an interessierte Werkzeuge in der Entwicklungsumgebung zu vermitteln [19,20,21]. Die Werkzeuge in dieser Umgebung sind autonom und bearbeiten ganz verschiedene Datenstrukturen. Allein die Kommunikation gewährleistet die Konsistenz der gesamten Konfiguration. Partielle (inkrementelle), bidirektionale und benutzergesteuerte Integration ist somit möglich. Integrationswerkzeuge mit diesem Mechanismus bedürfen jedoch aufwendiger Protokollschichten auf semantisch niedrigem Niveau. Leicht können sich dabei unerwünschte Löcher in der Datenverkapselung auftun.

2.2 Syntaxbasierte Integration

Operationen auf Syntaxbäumen eignen sich ebenfalls zur Integration. Mit solchen aus der Compilertechnik stammenden Methoden werden in [11,12,13] die zu integrierenden Dokumente in einem Syntaxbaum dargestellt, dessen Fragmente zu verschiedenen Sprachen gehören können. Die zur Integration erforderliche Information wird durch die Teilbäume propagiert.

Attributgesteuerte Integration ist aber auch zwischen zwei voneinander getrennten Syntaxbäumen möglich. Spezielle Tunnelattribute dienen dabei als Vermittler. Die Systeme Gandalf [14] und Centaur [15] beruhen auf dieser Technik. Sie erfüllen ihren Zweck, wenn nur die P r ü f u n g der Konsistenz der beteiligten Dokumente verlangt wird (passive Integration), sind aber mit invasiv integrierenden Operationen, welche inkonsistente Zustände reparieren sollen, überfordert. Der Ansatz der attributgesteuerten Integration wird daher in [16] weiterentwickelt, wo die berechneten Werte der Integrationsattribute eines Dokumentbaumes selbst wieder Syntaxbäume darstellen,

nämlich die der entsprechenden konsistenten Zustände des Zieldokumentes. So wird das Zieldokument dem veränderten Quelldokument angepaßt.

Auch mit der Baumtransformationssprache TXL [17,18] lassen sich Dokumentübersetzungen spezifizieren, wenn die zugrundeliegenden Strukturen als Bäume oder Listen darstellbar sind. Insbesondere sind mit TXL, analog zu fortschrittlichen funktionalen Programmiersprachen wie Miranda oder Haskell, Muster auf hohem deklarativen Niveau definierbar, die in den Dokumenten gesucht und ersetzt werden. In den attributgesteuerten Ansätzen müßten solche Musterersetzungen auf verschlungenen Wegen "zu Fuß" programmiert werden.

Das Konzept dieser Art von Integrationssystemen ist deterministisch, unidirektional und total. Es daher für viele Entwicklungsaufgaben, in denen der Benutzer zwischen verschiedenen Integrations- oder Transformationsalternativen zu wählen hat, nur mäßig geeignet. Totale Integration ist unakzeptabel ineffizient, wo sehr große Dokumente sehr häufig, aber immer nur geringfügig verändert werden. Unidirektionale Integration zwingt der Dokumententwicklung eine Reihenfolge auf und verbietet sich somit dort, wo die verschiedenen Dokumente in Arbeitsteilung simultan modifiziert werden. Die Darstellung der Integrationsinformation in strukturierten Attributen ist vergleichsweise kompliziert. Partielle (inkrementelle) Nachtransformationen werden nicht unterstützt, und das Datenmodell des attributierten Baumes ist deutlich weniger aussagekräftig als das allgemeinere Modell des attributierten Graphen, welches in den folgenden Ansätzen verwendet wird.

Die Idee, zum Zwecke der Transformation zwei Grammatiken aneinander zu koppeln, indem man Korrespondenzen zwischen deren Produktionen definiert, ist seit mehr als fünfundzwanzig Jahren bekannt [22]. Zwei Dokumente sind nach dieser Definition genau dann zueinander konsistent, wenn sie aus dem leeren Symbol durch "parallele" Folgen von jeweils miteinander korrespondierenden Produktionen entstanden sind.

Spätere Publikationen sprechen von "Tripelgraphgrammatiken", um in diesen Begriff nicht nur die beiden beteiligten Graphgrammatiken, sondern auch die dazugehörigen Kopplungsdefinitionen einzuschließen. Zunächst wurde die Kopplung tatsächlich mit einer dritten Grammatik definiert [23]. Später hingegen wurden dazu nur noch Tabellen verwendet [24]. Um nicht den Eindruck zu erwecken, es würden drei Grammatiken miteinander verbunden, soll hier wieder, wie ursprünglich, von "Paargrammatiken" die Rede sein. Der paargrammatische Ansatz, auf dem auch diese Arbeit beruht, ermöglicht partielle, bidirektionale und indeterministische (benutzergesteuerte) Integration.

2.3 Integration von Datenbankschemata

Auf dem Gebiet der D a t e n b a n k s y s t e m e sind sowohl die Migration als auch die Zusammenfassung von Datenbeständen seit Jahren Gegenstand der Forschung zur Datenintegration. Aufgabe der Schemaintegration ist es, veraltete, zumeist relationale Datenbankschemata durch neue, zumeist objektorientierte zu ersetzen, ohne dabei den wertvollen Inhalt der alten Datenbank zu verlieren. Das Analogon in der Softwaretechnik ist die Programmigration (Renovierung, Reengineering, Redesign), die Algorithmen aus den Lumpen veralteter Sprachen befreit und in zeitgemäße Sprachen kleidet — vgl. Kapitel 3. Daneben stehen verschiedene Ansätze, heterogene Daten, d.h. Daten von ganz unterschiedlicher Struktur, in einer föderierten Datenbank zu verwah-

ren oder mehrere datentypspezifische Banken virtuell wie eine einzige zu benutzen. Aus der reichhaltigen Literatur werden im folgenden nur Arbeiten zur Schemaintegration erwähnt, da dieses Thema —jedoch außerhalb des Datenbankkontextes— dasjenige ist, mit welchem dieser Aufsatz sich befaßt.

Schemaintegration mit Hilfe eines Metamodells wird in [25] beschrieben. Zunächst wird aus den beiden Datenmodellen zugrundeliegenden Konzepten (oder Begriffen) ein Metamodell formuliert. Dann wird das eine Schema in das Metamodell projiziert und so von allen irrelevanten Spezifika befreit. Ein letzer Schritt füllt dieses Gerippe mit dem Fleisch des anderen Datenmodells und erzeugt somit das gewünschte neue Schema. Grundlage der Implementation ist ein auf der Prädikatenlogik basierender Formalismus.

Die Idee des Metamodells in [25] stammt aus der Softwaretechnik [26] und ist auch in [27] wiederzuerkennen, allerdings nur implizit. Diese Arbeit beschreibt die Migration eines relationalen Datenbankschemas in ein objektorientiertes Schema mit Hilfe der graphgrammatischen Regelmethode, welche in Kapitel 3 dieses Aufsatzes anschaulich dargestellt wird.

Die Autoren von [27] generieren die Transformationsregeln aus den Paarungsdefinitionen der Grammatiken bereits automatisch. Sie vermeiden damit zwar Mühen und Fehler bei der Regelspezifikation, werden dadurch jedoch die Schwierigkeit der detaillierten Paarungsdefinitionen als solche nicht los. Auch ihr Ansatz könnte von der unten entwickelten Schemakorrespondenzmethode noch profitieren. Zwei Schemata durch Korrespondenzkanten direkt miteinander zu integrieren ist bedeutend weniger kompliziert als auf ihnen, wie in [26], den Überbau eines im schlimmsten Falle nicht einmal wiederverwendbaren Metamodells zu errichten.

2.4 Zusammenfassung

Die Beiträge zur Schemaintegration bei Datenbanken sind dem in diesem Aufsatz entwickelten Thema am nächsten verwandt. Sie beschreiben Integrationsbeziehungen statisch und lassen die dynamischen Aspekte der Transformation außer acht. Komplementär dazu ist der Ansatz von TXL, welcher sich ganz auf die Transformationsregeln konzentriert. Die Kopplung von Erzeugendensystemen zu Paaren liegt konzeptuell dazwischen. Weil Integrationsregeln den aktuellen Kontext berücksichtigen, ist die Kopplung von Produktionen ausdrucksmächtiger als rein statische Kopplungen, aber auch auf höherem deklarativen Niveau —und somit besser verständlich— als die reinen Transformationssysteme. Daher sind Paargrammatiken für die Spezifikation von Integrationswerkzeugen sehr gut geeignet. Schwierig bleibt dabei dennoch die Definition der u.U. großen Menge von Produktionenpaaren. Diese Aufgabe wird jedoch einfacher, wenn die dynamischen Paarregeln mit statischen Korrespondenzdefinitionen auf sehr hohem deklarativen Niveau verglichen werden können. Diesen Mechanismus, der ungefähr die gleiche Funktion erfüllt wie Typdefinitionen in typsicheren Programmiersprachen, macht sich bisher kein paargrammatischer Spezifikationsansatz zunutze. Diese Lücke in der bisherigen Forschung soll mit dem vorliegenden Aufsatz geschlossen werden.

3 Integration mit Paargraphgrammatiken

Die Grammatiken, die in dem hier verfolgten Ansatz zur Spezifikation von Integrationswerkzeugen aneinander gekoppelt werden, sind Graphgrammatiken. Ihr Vorteil gegenüber den üblichen Chomskygrammatiken besteht darin, daß ihre Satzformen nicht eindimensionale Zeichenreihen, sondern mehrdimensionale Graphen darstellen, welche die zu integrierenden Dokumente mit ihren vielen internen Abhängigkeiten auf intuitive Weise modellieren. Hierfür hat sich die Graphtransformationssprache PROGRES[2] bereits vielfach bewährt [2,7,24,26,27]. Das Prinzip der Sprache ist leicht verständlich, sodaß auch der Leser ohne Vorkenntnisse den weiteren Ausführungen folgen kann. Eine informelle, didaktische Einführung in PROGRES würde den Rahmen dieses Aufsatzes sprengen; in [2] findet sich das nötige. Syntax und Semantik von PROGRES werden (abgesehen von einigen mittlerweiligen Erweiterungen) in [28] formal definiert. Das folgende Beispiel erläutert in Grundzügen eine paargraphgrammatische Integrationsspezifikation und motiviert deren Erweiterung durch statische Korrespondenzen in Kapitel 4.

3.1 Beispiel

Ein sehr großes, nicht dokumentiertes und somit in seinem Gesamtverhalten nur unvollständig verstandenes Programm in einer veralteten Programmiersprache, z.B. Cobol, soll durch ein Programm von möglichst der gleichen Funktionalität in einer modernen Sprache ersetzt werden. Solche Probleme werden sich der "Datenarchäologie" in Zukunft immer häufiger und dringender stellen, da viele Anwendungsprogramme in kommerziellen Betrieben oder wissenschaftlichen Instituten noch aus der Gründerzeit der elektronischen Datenverarbeitung stammen und nach dem Gesetz der wachsenden Entropie allmählich verfallen.

3.1.1 Methode

Ähnlich wie in Kapitel 2 angedeutet, ist auch hier eine schrittweise Problemlösung vorteilhaft. Zunächst versucht man den alten Quellkode in die abstrakte Architekturbeschreibungssprache MIL [29] (Module Interconnection Language) zu transformieren. Das Ergebnis ist dann relativ leicht in den neuen Zielkode transformierbar. Mit der ersten Transformation befaßt sich dieses Beispiel. Voraussetzung ist, daß sowohl Coboldokumente als auch Dokumente in MIL als Graphen dargestellt sind; (dies ist wiederum ein Integrationsproblem, welches an dieser Stelle aber als bereits gelöst angenommen sei). Dann wird die Integrationsaufgabe durch geeignete Kopplung der zugehörigen Graphgrammatiken erledigt.

3.1.2 Ein Graphenschema für Cobol

So wie alle syntaktisch korrekten Cobolprogramme eines gemeinsam haben, nämlich Sätze der Cobolgrammatik zu sein, so haben auch deren verschiedene Graphrepräsentationen ein gemeinsames Schema, welches ihre Gestalt konditioniert.

[2]vormals PROGRESS

"Alle Cobolprogramme bestehen im Wesentlichen aus Berechnungen und Daten. Sowohl Berechnungen als auch Daten sind Programmfragmente, welche mit einer bestimmten Zeilennummer beginnen und mit einer bestimmten Zeilennummer enden. Berechnungen konsumieren Daten und können andere Berechnungen aufrufen." Dieser Beschreibung entspricht ein simples Graphenschema für Cobolgraphen, welches in Abbildung 3a) zu sehen ist:

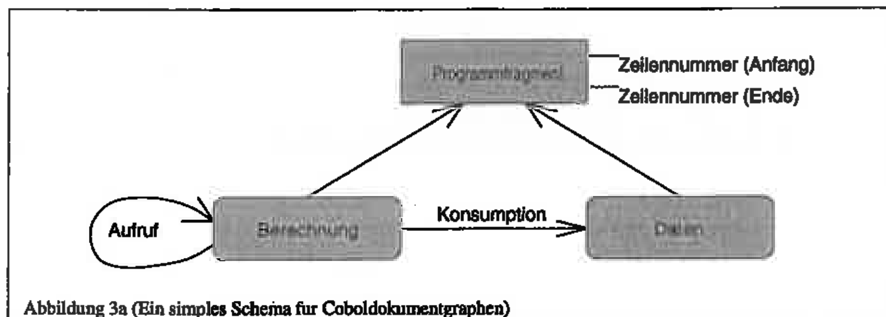


Abbildung 3a (Ein simples Schema für Coboldokumentgraphen)

Ein Rechteck mit abgerundeten Ecken ist ein T y p. Der Typ mit der Aufschrift "Daten" repräsentiert also alle Graphknoten, welche Daten, z.B. die natürlichen Zahlen, beinhalten. Der Typ "Berechnung" repräsentiert analog alle Graphknoten, die eine Berechnung auf Daten beinhalten. Diese zweidimensionale Beziehung zwischen Berechnungsknoten und Datenknoten der Dokumentgraphen wird durch den Pfeil mit der Aufschrift "Konsumption" zwischen den Typen "Berechnung" und "Daten" im Graphenschema dargestellt. Analog stellt der Pfeil mit der Aufschrift "Aufruf" vom und zum Typen "Berechnung" dar, daß eine Berechnung wiederum eine Berechnung aufrufen kann.

Ein "echtes" Rechteck ist eine abstrakte K l a s s e. Sie repräsentiert die Gemeinsamkeiten der ihr zugeordneten Unterklassen oder Typen. Die Klasse "Programmfragment" repräsentiert alle Graphknoten der Dokumentgraphen, welche als Attribute zwei Zeilennummern beinhalten. Dies ist in dem Beispiel sowohl für Knoten vom Typ "Berechnung" als auch für Knoten vom Typ "Daten" der Fall. Sie e r b e n also diese gemeinsame Eigenschaft von ihrer Klasse "Programmfragment". Die unbeschrifteten Pfeile repräsentieren immer diese Erbrelation. Typen repräsentieren Gegenstände, besitzen also Instanzen, und bilden immer die Blätter der Vererbungshierarchie. Die abstrakten Klassen besitzen hingegen keine eigenen Instanzen, sondern versammeln nur die Instanzen der zu den jeweiligen Klassen gehörigen Typen.

Das Schema 3a) ist freilich viel zu grob, um als Grundlage eines brauchbaren Integrators für MIL- und Coboldokumente dienen zu können. Zur Veranschaulichung des Prinzips ist es aber wohl geeignet, da zu viele realistische Details vom hier Wesentlichen nur ablenken würden.

3.1.3 Vervollständigung zur Graphgrammatik

Ist das Schema definiert, wird es durch Produktionen zur Graphgrammatik vervollständigt. Abbildung 3b) zeigt eine Produktion, welche zu einem schon vorhandenen Programmfragment vom Typ "Daten" ein neues Programmfragment vom Typ "Berechnung" definiert und die vom Graphenschema vorgeschriebene Konsumptionsrelation zwischen beiden einrichtet.

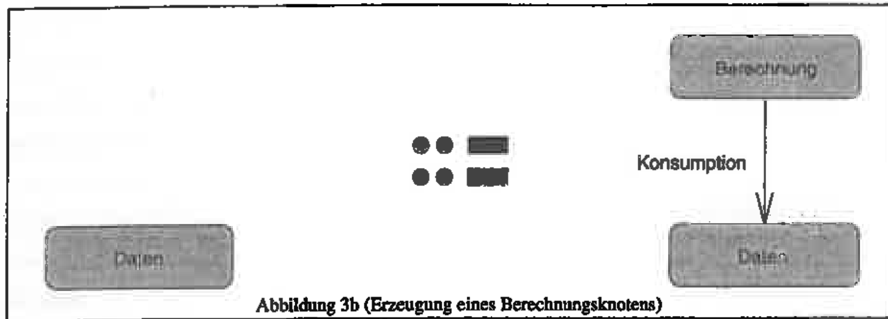


Abbildung 3b (Erzeugung eines Berechnungsknotens)

Viele und in Wirklichkeit weitaus kompliziertere Produktionen sind zur Spezifikation einer brauchbaren Graphgrammatik, welche alle dem Graphenschema entsprechenden Graphen erzeugen kann, erforderlich. Da in diesem Kapitel aber lediglich das Entwurfsprinzip vorgeführt wird, soll zur Illustration dieses eine Exemplar einer Graphproduktion genügen.

3.1.4 Graphenschema und Produktionen für MIL

Ganz analog müssen nun die Charakteristika der Sprache MIL als Graphenschema dargestellt werden. Sie sind folgendermaßen zu beschreiben: "Dokumente der Sprache MIL bestehen aus Einheiten, die einen eindeutigen Namen tragen. Einheiten sind sowohl Spezifikationen, als auch deren Implementationen. Module sind besondere Spezifikationen, welche einander benutzen. Sie werden in Datenmodule und Funktionsmodule unterschieden." Nota bene: Die Spezifikation definiert die Schnittstelle zum Inhalt oder das Verhalten einer Einheit unabhängig von der Implementation des Inhaltes oder des Verhaltens. So können mehrere Implementationsvarianten vorkommen oder fehlerhafte Implementationen leicht entfernt und ersetzt werden, ohne daß das ganze Dokument in Mitleidenschaft gerät [29]. Der obigen Beschreibung entspricht das Graphenschema in Abbildung 3c):

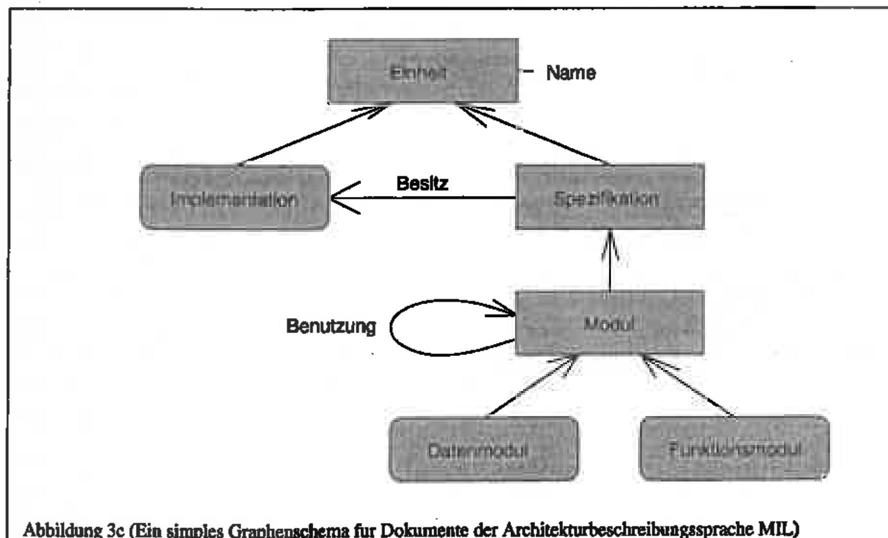


Abbildung 3c (Ein simples Graphenschema fur Dokumente der Architekturbeschreibungssprache MIL)

Wiederum stellen die Rechtecke mit den runden Ecken im Graphenschema die Typen dar, welchen die Knoten der Dokumentgraphen zugeordnet werden. Wiederum werden

die gemeinsamen Eigenschaften der Typen durch die Klassen zusammenfassend repräsentiert. Wiederum stellen beschriftete Pfeile im Graphenschema zweidimensionale Relationen, welche niemals verletzt werden dürfen, zwischen den Dokumentgraphknoten in MIL dar. Wiederum bezeichnen unbeschriftete Pfeile die besondere Erbrelation der Klassenhierarchie.

Nachdem das Graphenschema definiert ist, wird es durch Graphproduktionen, welche alle erlaubten (schematreuen) Dokumentgraphen erzeugen können, zur Graphgrammatik erweitert. In den folgenden Abbildungen 3d) und 3e) wird zum Beispiel gezeigt, wie der Spezifikation eines Datenmoduls eine Implementation hinzugefügt, bzw. zu einem Datenmodul ein benutzendes Funktionsmodul erzeugt wird.

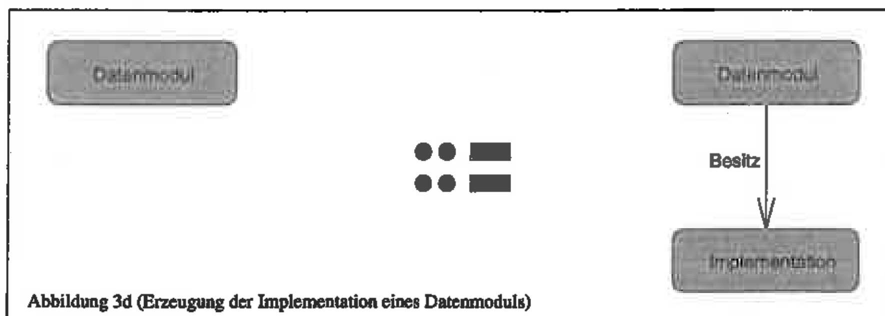


Abbildung 3d (Erzeugung der Implementation eines Datenmoduls)

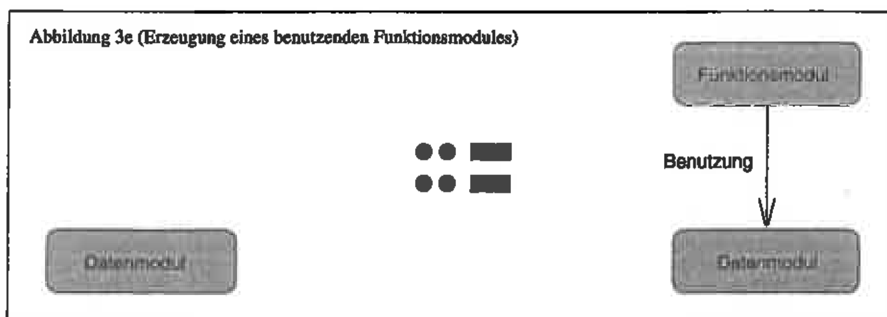


Abbildung 3e (Erzeugung eines benutzenden Funktionsmodules)

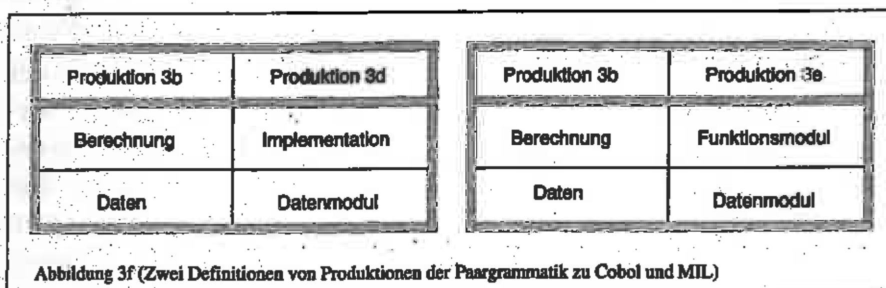
Nota bene: Bei Anwendung der letzteren Produktion auf einen vorhandenen Dokumentgraphen spielt es keine Rolle, ob auf den vorgefundenen Graphknoten des Typs "Datenmodul" auf der linken Regelseite bereits andere Benutzungspfeile zeigen. Ein Datenmodul kann ja von beliebig vielen Funktionsmodulen benutzt werden.

3.1.5 Paargrammatische Integration

Im nächsten Entwurfsschritt werden die Graphgrammatiken der MIL- und Coboldokumentgraphen zu einer Paargrammatik verschmolzen, indem man einzelne Produktionen jeweils der Quell- und Zieldokumentgrammatik als zusammengehörig definiert. Das heißt: wird das Quelldokument mittels einer Quellproduktion p verändert, so muß auch das Zieldokument, um die Konsistenz der Gesamtkonfiguration zu wahren, einer Zielproduktion p' unterworfen werden, welche per Korrespondenzdefinitionem mit p verbunden ist. Paargrammatiken sind prinzipiell symmetrisch, legen also nicht fest, welche ihrer beiden Seiten Quell- und welche Zielseite sein soll. Diese werden erst später bei der Bestimmung der Werkzeugfunktionalität unterschieden.

Man beachte, daß es sich bei der Kenntnis, welche Produktionen miteinander verbunden werden müssen, damit ein brauchbares Integrationswerkzeug entsteht, um Expertenwissen (Erfahrungswissen) handelt, welches aus den beiden nur syntaktischen Graphgrammatiken nicht abgeleitet werden kann. Offensichtliche Ähnlichkeiten zweier Grammatiken, welche gewisse unmittelbar einsichtige Korrespondenzdefinitionen geradezu verlangen, widersprechen dem nicht, sondern deuten nur auf "fortgeschrittenes Wissen" hin, welches sich nicht erst in den Korrespondenzdefinitionen, sondern bereits in der Dokumentmodellierung niederschlägt. Dennoch ist ein auf diesem Fundament gebauter Integrator kein Expertensystem im Sinne von [30], da er u.a. weder Inferenz- oder Deduktionsmechanismen bereitstellt, noch über Lernfunktionen zur Veränderung seiner Wissensbasis verfügt.

In unserem Beispiel weiß man aus Erfahrung, daß Berechnungen in Cobol sowohl einem expliziten Funktionsmodul in MIL entsprechen, als auch in der Implementation eines Datenmoduls verborgen werden können. Daher muß die Produktion 3c) sowohl mit Produktion 3d) als auch —alternativ— mit Produktion 3e) verknüpft werden, wodurch zwei Produktionen der entsprechenden Paargrammatik entstehen. Abbildung 3f) zeigt tabellarisch beide Produktionenverknüpfungen:



Produktion 3b	Produktion 3d
Berechnung	Implementation
Daten	Datenmodul

Produktion 3b	Produktion 3e
Berechnung	Funktionsmodul
Daten	Datenmodul

Abbildung 3f (Zwei Definitionen von Produktionen der Paargrammatik zu Cobol und MIL)

Aus diesen Paarproduktionen werden im letzten Schritt der Spezifikation die Transformationsregeln des entstehenden Integrationswerkzeuges abgeleitet. Dazu gibt es im allgemeinen verschiedene Möglichkeiten. Sinnvoll sind Transformationsregeln, welche zwischen schon vorhandenen Quell- und Zieldokumenten ein Integrationsdokument einrichten, desweiteren identische Transformationen, welche lediglich das Vorkommen gewisser erwünschter oder unerwünschter Situationen in der Konstellation der Dokumente prüfen und schließlich Produktionen, die aus dem Quelldokument das Zieldokument samt Integrationsdokument erzeugen. Um letztere handelt es sich in diesem Beispiel. Die Abbildungen 3g) und 3h) zeigen alternative konsistente Erzeugungen von Knoten im Zieldokument für die Erzeugung einer Berechnung im Quelldokument, einschließlich der Korrespondenzknoten des Integrationsdokumentes.

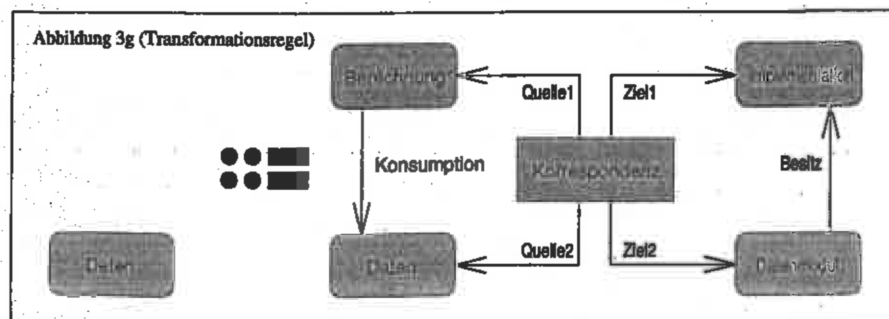
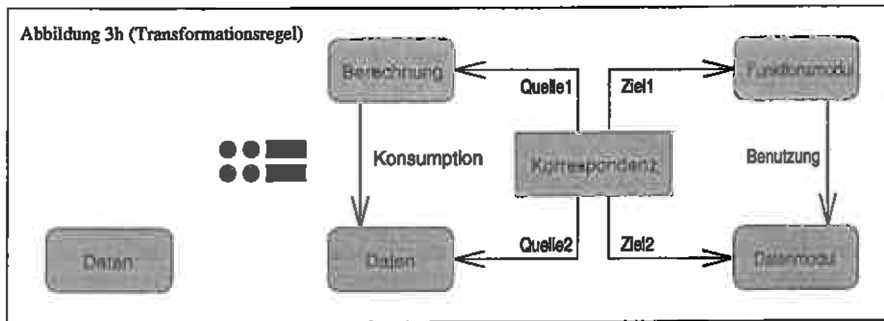


Abbildung 3g (Transformationsregel)

Abbildung 3h (Transformationsregel)



Es fällt auf, daß beide Produktionen die gleiche linke Seite besitzen. Sie wären in der entsprechenden Situation also beide anwendbar. Generell obliegt dem Benutzer des Integrationswerkzeuges die Lösung solcher indeterministischen Situationen, falls das Werkzeug die Auswahl einer der passenden Regeln nicht selbst kontextabhängig korrekt zu treffen imstande ist. Viele und weitaus kompliziertere solcher Transformationsregeln sind zur Spezifikation eines brauchbaren Integrationswerkzeuges nötig.

3.2 Zusammenfassung

Am Beispiel eines Integrationsproblems aus der Softwaretechnik wurde die von der Sprache PROGRES unterstützte Graphenspezifikationsmethode andeutungsweise vorgestellt. Die Graphenschemata konditionieren nur die Gestalt der zugehörigen Dokumentgraphen, enthalten aber selbst noch keine Integrationsinformation. Diese ist erst in den Definitionen der Paargrammatik enthalten. Die Korrespondenzdefinitionen zwischen zusammengehörigen Produktionen, welche der Bildung der Transformationsregeln eines Integrationswerkzeuges vorausgehen, sind völlig beliebig. Hier setzt die im Einführungskapitel angekündigte neue Spezifikationsmethode an, welche in Kapitel 4 ausführlich behandelt wird. Es ist nämlich durchaus möglich, die in Abb. 3g) und 3h) durch die Korrespondenzknoten dargestellten Integrationsbeziehungen bereits in den Graphenschemata zu konditionieren, wie die folgende Abbildung 3j) andeutet:

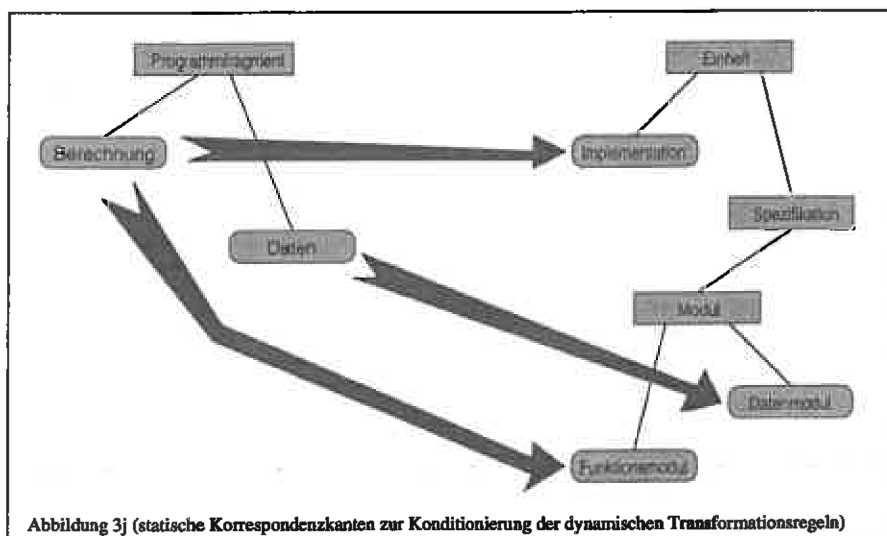


Abbildung 3j (statische Korrespondenzkanten zur Konditionierung der dynamischen Transformationsregeln)

Die Pfeile stellen statische Integrationsbedingungen zwischen Klassen oder Typen der beiden Schemata dar, im Gegensatz zu den dynamischen Integrationsregeln, die für die Dokumentgraphen gelten. Die intraschematischen Relationen (bis auf die Erbrelation) fehlen in der Abbildung, da sie für die neue Methode bis jetzt noch irrelevant sind; mögliche Erweiterungen beschreibt Kapitel 5.

Das folgende Kapitel handelt von der genauen Bedeutung der Korrespondenzkanten und ihrem Nutzen für die Spezifikation der Transformationsregeln eines Integrationswerkzeuges.

4 Korrespondenzen auf Graphenschemata

Das vorige Kapitel deutete mit einem längeren Beispiel an, wie Graphenschemata die Gestalt der ihnen zugehörigen Graphen konditionieren und wie Integrationsbeziehungen zwischen den Dokumentgraphen nicht allein durch Paarung der dynamischen Graphproduktionen, sondern bereits durch entsprechende statische Zuordnungen zwischen den Graphenschemata formuliert werden können. Diese Vorstellungen werden im folgenden verallgemeinert und präzisiert. Dazu muß nach einigen vorbereitenden Definitionen zunächst der Begriff "Schema" definiert werden. Dann erst ist es möglich, eine Semantik der Schemaknoten und der zwischen ihnen definierten Korrespondenzkanten zu formulieren. Hierbei erweist sich die aus praktischen Gründen in PROGRES implementierte Verbandseigenschaft der Schemata, nämlich daß zu je zwei Klassen genau eine größte gemeinsame Unterklasse und genau eine kleinste gemeinsame Oberklasse existiert [31,32], erneut von großem Nutzen. Sie ermöglicht nämlich Metaregeln auf Korrespondenzkanten, deren Logik in der Definition der Klassensemantik selbst ruht und von den Interpretationen der Klassen und Typen eines Schemas durch verschiedene Integrationswerkzeuge unabhängig ist.

4.1 Die Syntax der Schemakorrespondenzsprache

Definition 4.1 (Symbolvorrat)
Zur Korrespondenzsprache L_S zählen folgende Symbole: der dünne Pfeil, der dicke Pfeil, das Rechteck und das Rundeck wie in Abbildung 4a) zu sehen. *CLASS* sei die Menge aller Rechtecke, *TYPE* die Menge aller Rundecke. Zur Unterscheidung jeweils verschiedener Rechtecke bzw. Rundecke dürfen diese mit zusätzlichen Aufschriften versehen werden.

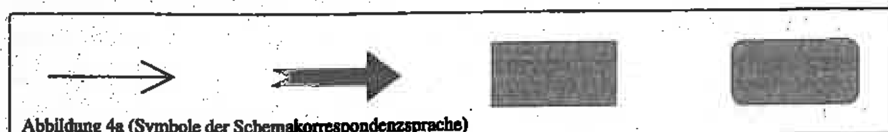


Abbildung 4a (Symbole der Schemakorrespondenzsprache)

Diese Zeichen sind bereits alle aus Kapitel 3 bekannt. Der dünne Pfeil stellt die Erbrelation zwischen Klassen dar, der dicke Pfeil wird im folgenden als Schemakorrespondenzsymbol definiert. Das Symbol mit den abgerundeten Ecken ist ein Typ. Typen sind besondere Klassen, welche Gegenstände, nämlich Knoten der Dokumentgraphen repräsentieren. Die Gegenstände eines Typs werden in der Literatur auch seine Instanzen genannt. Die Menge aller Instanzen eines Typs ist seine Extension. Alle anderen Klassen werden durch die Rechtecke dargestellt. Sie ordnen Typen und andere Klassen

nach Eigenschaften, repräsentieren aber selbst keine Gegenstände. Im folgenden gelte übrigens immer die Konvention, daß kleine Variable k Elemente aus *CLASS*, kleine Variable t Elemente aus *TYPE* und kleine Variable a, b, e Elemente aus $CLASS \uplus TYPE$ bezeichnen.

Definition 4.2 (intraschematische Verbindungen)
Für die Verknüpfung der Symbole $t, t_i \in TYPE$, $k, k_j \in CLASS$,
$e, e_i \in CLASS \uplus TYPE$ und $\rightarrow$ aus L_S gelten folgende Bedingungen:
a) $\forall t \exists! k$: $t \rightarrow k$
b) $\nexists e$: $e \rightarrow e$
c) $\nexists n \in \mathbb{N}$: $e_1 \rightarrow \cdots \rightarrow e_n \rightarrow e_1$
d) $\forall e \rightarrow k_i, e \rightarrow k_j$: $k_i \neq k_j$
e) $\nexists n \in \mathbb{N}$: $e_1 \rightarrow e_2 \rightarrow \cdots \rightarrow e_n$, $e_1 \rightarrow e_n$
f) $\forall \rightarrow \exists e_1, e_2$: $e_1 \rightarrow e_2$.

Hiermit ist definiert, wie dünne Pfeile, Rechtecke und Rundecke überhaupt miteinander verbunden werden dürfen und wie nicht. Bedingung a) weist jedem Typen genau eine Oberklasse zu und reduziert damit die Möglichkeit der Mehrfachvererbung auf die abstrakten Klassen. Dies fördert, wie später zu sehen, die Anschaulichkeit der Schemata. Bedingungen b) und c) verbieten Vererbungszyklen, Bedingung d), e) und f) verbieten redundante bzw. unvollständige Pfeile.

Die Schemadefinition gebraucht ferner noch folgende Begriffe:

Definition 4.3 (Oberklasse, Unterklasse, Größenrelation)
Eine Klasse k_o heißt Oberklasse gdw. gilt: $\exists k$: $k \rightarrow k_o$.
Eine Klasse k_u heißt Unterklasse gdw. gilt: $\exists k$: $k_u \rightarrow k$.
Das Symbol $\rightarrow$ induziert eine reflexive und transitive
Größenrelation $\leq$ auf Klassen wie folgt:
a) $\forall k_u \rightarrow k_o$: $k_u \leq k_o$
b) $\forall k$: $k \leq k$
c) $\forall k_u \leq k, k \leq k_o$: $k_u \leq k_o$.

Dies ist allgemein bekannt und bedarf keiner ausführlichen Erläuterung. Damit wird nun der Schemabegriff folgendermaßen bestimmt:

Definition 4.4 (Schema)
Ein nach den Regeln von Definition 4.2 gebildetes Geflecht S
von Symbolen e aus L_S heißt Schema gdw. gilt:
a) $\forall e \in S \exists! \hat{k}$: $e \leq \hat{k}$
b) $\forall e \in S \exists! \check{k}$: $\check{k} \leq e$
c) $\forall e_1, e_2 \exists! kgo$: $e_1 \leq kgo \wedge e_2 \leq kgo \wedge \forall e$: $e_1 \leq e, e_2 \leq e \Rightarrow kgo \leq e$
d) $\forall e_1, e_2 \exists! ggu$: $ggu \leq e_1 \wedge ggu \leq e_2 \wedge \forall e$: $e \leq e_1, e \leq e_2 \Rightarrow e \leq ggu$
e) $\exists t \in S$
f) $\forall e \rightarrow t$: $e = \check{k}$.

Aus den Forderung a) bis d) folgt die Verbandseigenschaft der Schemata. Forderung e) verbietet gegenstandslose Schemata. Der Bezeichner *ggu* steht für "größte gemeinsame Unterklasse", *kgo* entsprechend für "kleinste gemeinsame Oberklasse". Die Forderung f) erzwingt o.B.d.A., daß die Typen —abgesehen von der untersten, in dem Beispiel in Kapitel 3 unsichtbaren "leeren" Klasse $\check{k}$— die bezüglich $\leq$ kleinsten Ele-

mente in S bilden. So wie z.B. alle Cobolgraphen ihrem Schema entsprechen müssen, um korrekte Cobolgraphen darzustellen, so müssen auch alle Graphenschemata den in Definition 4.4 aufgestellten Bedingungen genügen, um überhaupt als Schema zu gelten.

Abbildung 4b) zeigt ein Beispiel eines Schemas mit mehreren Klassen und Typen sowie das einfachstmögliche, nur aus einem einzigen Typen und seinen per Definitionem notwendigen Klassen gebildete Schema.

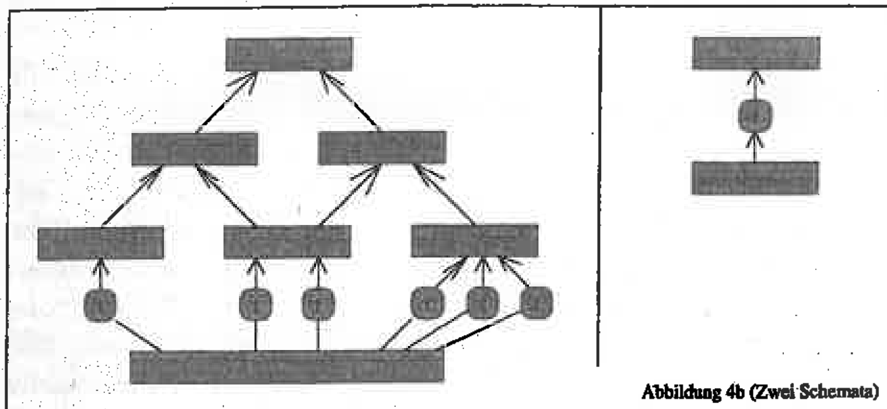


Abbildung 4b (Zwei Schemata)

Nun endlich sind alle Voraussetzungen gegeben, um statische Korrespondenzen, welche die Spezifikation von Integrationsregeln erleichtern und die Prüfung der Konsistenz dieser Regeln ermöglichen, auf Graphenschemata zu definieren.

Hauptdefinition 4.5 (Schemakorrespondenzaxiom)
Seien S_1, S_2 Schemata wie oben definiert. Dann heißt ein dicker Pfeil $e_1 \Longrightarrow e_2$ Schemakorrespondenzaxiom (kurz: Korrespondenzaxiom, Axiom), wenn gilt: e_1 ist ein Typ oder eine Klasse aus S_1, e_2 ist ein Typ oder eine Klasse aus S_2.

"Axiom" wird ein solcher Pfeil deshalb genannt, weil er vom Spezifikator eines Integrationswerkzeuges gesetzt wird. Von den gegebenen Schemakorrespondenzen hängen nun alle weiteren Schritte in der Spezifikation eines Integrationswerkzeuges ab. Nota bene: Sind S_1 und S_2 identisch, wird durch Schemakorrespondenzaxiome eine "klassische" Programmtransformation definiert.

So wie es in den oberen Abschnitten nicht möglich war, beliebige Geflechte mit dünnen Pfeilen zu bilden, so müssen nun auch alle Konstellationen von dicken Pfeilen zwischen zwei Schemata bestimmte Bedingungen befolgen, um nicht in sich widersprüchlich zu sein. So entstehen M e t a r e g e l n für Schemakorrespondenzaxiome, die im folgenden eingeführt werden. Dazu muß aber erst die Semantik der oben eingeführten syntaktischen Begriffe erläutert sein.

4.2 Die Semantik der Schemakorrespondenzsprache

Definition 4.6 (Semantik von Typen und Klassen)
Sei t ein Typ. Seine Semantik lautet $[[t]] := \{t\}$
Sei $\check{k}$ die "leere" Klasse. Ihre Semantik lautet $[[\check{k}]] := \{\}$
Sei k_n eine Oberklasse mit Unterklassen $u_1, \ldots, u_n (n > 0)$.
Dann gelte $[[k_n]] := [[u_1]] \cup \cdots \cup [[u_n]]$.

Diese Mengensemantik entspricht der Alltagsvorstellung von Begriffen und Oberbegriffen. Ungewöhnlich scheint allenfalls, daß das Typsymbol t zur Definition seiner eigenen Semantik verwendet wird. Dieser dem Logiker Herbrand zu verdankende Kniff der Selbstinterpretation [33] befreit die Überlegungen in diesem Kapitel vom "Datenbankproblem", nämlich: nicht zu wissen, ob Typen im Graphenschema tatsächlich Extensionen in Gestalt der den Typen zuzuordnenden Knoten in den Dokumentgraphen besitzen. Die Semantik des Typen ist daher ein Symbol für alle seine möglichen Extensionen.

Aus der Definition der Semantik von Typen und Klassen folgt trivialerweise die Semantik der dünnen Pfeile als die Erbrelation: eine Klasse, die Quelle eines dünnen Pfeiles ist, besitzt mindestens alle Eigenschaften derjenigen Klasse, auf welche dieser dünne Pfeil weist. Die Erbrelation ist selbstverständlich transitiv, auch wenn keine entsprechenden "Abkürzungspfeile" (siehe Definition 4.2e) im Schema vorkommen. Man beachte, daß die Semantik nur für die einzelnen Symbole eines Schemas definiert wurde, nicht aber für das Schema als solches, weil dieses ja die in den Semantikdefinitionen verwendeten Klassenbeziehungen zugrundelegt. Die semantische Definition eines Korrespondenzaxioms ist weniger anschaulich und bedarf einiger vorbereitender Bemerkungen.

Zunächst muß eine b r a u c h b a r e Bedeutung für das einzelne Schemakorrespondenzaxiom erwogen werden. Soll es eine bestimmte Transformation "gestatten"? Wäre dann jegliche Transformation für solche Dokumentgraphknoten, für deren Typen und Oberklassen kein Korrespondenzaxiom festgesetzt ist, "verboten"? Oder soll das Korrespondenzaxiom eine bestimmte Transformation "erzwingen"? Wären dann aber beliebige Transformationen für Graphknoten mit Schemaknoten ohne Axiom "gestattet"? Sind diese Fragen beantwortet, muß die Bedeutung mehrerer Korrespondenzaxiome zueinander bestimmt werden. Sollen sie alle zugleich gelten? Sollen sie Alternativen darstellen, zwischen denen gewählt werden darf? Eine sinnvolle Wahl wird im folgenden getroffen und begründet.

Definition 4.7 (Semantik eines Korrespondenzaxioms)
Sei $e_q \Longrightarrow e_z$ ein Schemakorrespondenzaxiom. Dann lautet seine Bedeutung:
$[[e_q \Longrightarrow e_z]] := \{(t_q, t_z) | t_q \in [[e_q]], t_z \in [[e_z]]\}$, bzw.
$[[e_q \Longrightarrow e_z]] := \{(t_q, \perp) | t_q \in [[e_q]], e_z = \check{k}\}$.

Ein Knoten des Typs t_q, welcher aus dem Quelldokument transformiert werden soll, m u ß in einen Knoten vom Typ t_z im Zieldokument transformiert werden, wenn ein Typenpaar (t_q, t_z) in der semantischen Menge eines Schemakorrespondenzaxioms vorkommt (und sofern dem kein anderes Axiom widerspricht). Mit $e \Longrightarrow \check{k}$, wo $\check{k}$ die unterste, leere Klasse des Verbandes, sind Transformationsverbote formulierbar. (Weniger sinnvoll sind vermutlich Erzeugungen aus dem Nichts mit $\check{k} \Longrightarrow e$.)

Die Definition der Semantik mehrerer Korrespondenzaxiome zueinander ist, wie im folgenden gezeigt, durch Erfahrungen aus der Praxis der Integratorenspezifikation begründet. Es genügt übrigens zunächst, die gemeinsame Semantik für jeweils zwei Axiome zu definieren. Die Semantik aller Axiome einer Spezifikation zusammen wird später dargestellt. Zu diesem Zweck werden im folgenden drei Regeln vorgestellt, welche eine anfängliche Korrespondenzspezifikation in eine widerspruchsfreie Abschlußspezifikation überführen, mit welcher dann die gemeinsame Semantik aller Axiome definiert wird.

4.3 Metaregeln für Korrespondenzaxiome

Die Metaregeln auf Schemakorrespondenzaxiomen erfüllen zugleich zwei Aufgaben. Erstens definieren sie den Begriff der gemeinsamen Semantik zweier Axiome in einer semantisch ambivalenten Situation. Zweitens tragen sie zur Auflösung von ambivalenten oder widersprüchlichen Situationen bei. Der Name Metaregel drückt aus, daß es sich um Regeln handelt, die für Korrespondenzaxiome gelten. Keinesfalls dürfen die Metaregeln mit den Graphtransformationsregeln der tiefer liegenden Schicht verwechselt werden.

Definition 4.8 (Zielkollaps)
Seien S_1, S_2 Schemata, e eine Klasse aus S_1, k und u mit $k \neq u$, $u \leq k$ Klassen aus S_2. Dann gelte die Axiomersetzung $(e \Longrightarrow k, e \Longrightarrow u) \succ (e \Longrightarrow u)$.

Definition 4.9 (Quellkollaps)
Seien S_1, S_2 Schemata, k und u mit $k \neq u$, $u \leq k$ Klassen aus S_1, e eine Klasse aus S_2. Dann gelte die Axiomersetzung $(k \Longrightarrow e, u \Longrightarrow e) \succ (k \Longrightarrow e)$.

Jeweils zwei Axiome haben zunächst also genau dann eine gemeinsame Semantik, wenn sie erstens derselben Quelle entspringen oder auf ein gemeinsames Ziel weisen, und zweitens die entsprechenden Ziele (bzw. Quellen) auf der jeweils anderen Seite in Erbrelation zueinander stehen. Dies sind semantisch a m b i v a l e n t e Situationen, und die Metaregeln 4.8 und 4.9 definieren zwei Lösungen dieser Ambivalenzen. Im einen Fall absorbiert ein spezielleres Axiom das allgemeinere, im anderen Fall absorbiert umgekehrt ein allgemeineres Axiom ein spezielleres. Transformationsverbote sind dadurch dominant, d.h.: sie können durch andere Axiome nicht unterlaufen werden.

Diese Definitionen berücksichtigen die übliche Methode in der Spezifikation von Integratoren. Ausgehend von einigen Speziallfällen auf der Quellseite wird zunächst ganz grob das Ziel definiert. Nach und nach werden die Ziele verfeinert, während man zugleich mehr und mehr in der Lage ist, aus den anfänglichen Spezialfällen allgemeinere Gesetzlichkeiten zu abstrahieren.

Die Definition der gemeinsamen Semantik zweier Schemakorrespondenzaxiome induziert einen kleinen Schemakorrespondenzkalkül, welcher eine zwischen zwei Schemata beliebig gesetzte Menge von Axiomen konsequent abschließt und dabei eventuelle Widersprüche aus der ursprünglichen Axiomenmenge entfernt. Zu diesem Zweck wird im folgenden zunächst die von der Ambivalenz verschiedene Widersprüchlichkeit definiert. Anschließend wird eine dritte Metaregel vorgestellt, mit deren Hilfe die obigen Kollapsregeln widersprüchliche Situationen auflösen können.

Definition 4.10 (Widersprüchlichkeit)
Zwei Korrespondenzaxiome $a \Longrightarrow a'$, $b \Longrightarrow b'$ mit $a \neq b$, $a' \neq b'$ sind zueinander widersprüchlich, wenn ihre Quellklassen in anderer Ordnung zueinander stehen als ihre Zielklassen. Folgende widersprüchliche Fälle sind bis auf Variablenumbenennung möglich:
a) $a \leq b$ und $b' \leq a'$ (Überschneidung),
b) $a \leq b$ und $a' \not\leq b'$, $b' \not\leq a'$ (Verzerrung),
c) $a \not\leq b$, $b \not\leq a$ und $a' \leq b'$ (Verklumpung).

Definition 4.11 (Verbandregel)
Seien S_1, S_2 Schemata, $a \neq b$ Klassen aus S_1,
$a' \neq b'$ Klassen aus S_2. Dann gelte die Axiomersetzung
$(a \Longrightarrow a', b \Longrightarrow b') \succ (a \Longrightarrow a', b \Longrightarrow b', kgo \Longrightarrow kgo', ggu \Longrightarrow ggu')$,
wobei kgo die kleinste gemeinsame Oberklasse von a und b, kgo' die kleinste gemeinsame Oberklasse von a' und b', ggu die größte gemeinsame Unterklasse von a und b und ggu' die größte gemeinsame Unterklasse von a' und b' bezüglich $\leq$.

Diese Erweiterung führt sicher nicht zum Widerspruch mit den ursprüglichen Axiomen $a \Longrightarrow a', b \Longrightarrow b'$, denn die neu eingefügten Axiome stehen in der Ordnung ja oberhalb und unterhalb der Axiome aus der Anwendungsvoraussetzung dieser Regel. Man beachte jedoch einen wesentlichen Unterschied der Verbandregel zu den beiden Kollapsregeln! Die Voraussetzung $a \leq b$, welche die Anwendung der Kollapsregel notwendig macht, um eine semantisch ambivalente Situation in eine eindeutige Situation zu verwandeln, ist hier nicht gegeben. Die Vorbedingung der Verbandregel ist in sich widerspruchsfrei. Eigentlich besteht gar kein Grund, eine widerspruchsfreie Situation zu verändern. Das heißt, daß die Verbandregel eine s c h w a c h e Regel ist, die nicht notwendigerweise angewendet werden muß. Sie wird, wie im folgenden dargestellt, nur als in sich selbst widerspruchsfreies Hilfsmittel zur Suche und zur Auflösung widersprüchlicher Situationen verwendet.

4.4 Unterstützung der Spezifikation

Nachdem in den vorherigen Abschnitten gezeigt wurde, daß die Schemakorrespondenzsprache ein in sich schlüssiges Mittel zur statischen Darstellung von Integrationsbeziehungen ist, bleibt zum Schluß dieses Aufsatzes nur noch vorzuführen, wie mit den neuen Schemakorrespondenzaxiomen die bewährte Methode der Regelspezifikation unterfüttert und verbessert werden kann. Hiervon handelt der folgende Abschnitt.

Beginnend mit den Graphenschemata der zu integrierenden Dokumente werden zunächst die gewünschten Axiome frei gesetzt. Dann treten automatisch die Metaregeln hervor, um diese initiale Spezifikation semantisch abzuschließen und von eventuellen Widersprüchen zu befreien. Die abgeschlossene statische Korrespondenzspezifikation dient dann als zuverlässiger Prüfstein, an dem die relative Korrektheit der nun zu formulierenden dynamischen Graphtransformationsregeln gemessen wird. Abbildung 4c) zeigt die Auflösung einer Überschneidung durch kombinierte Applikationen der Metaregeln:

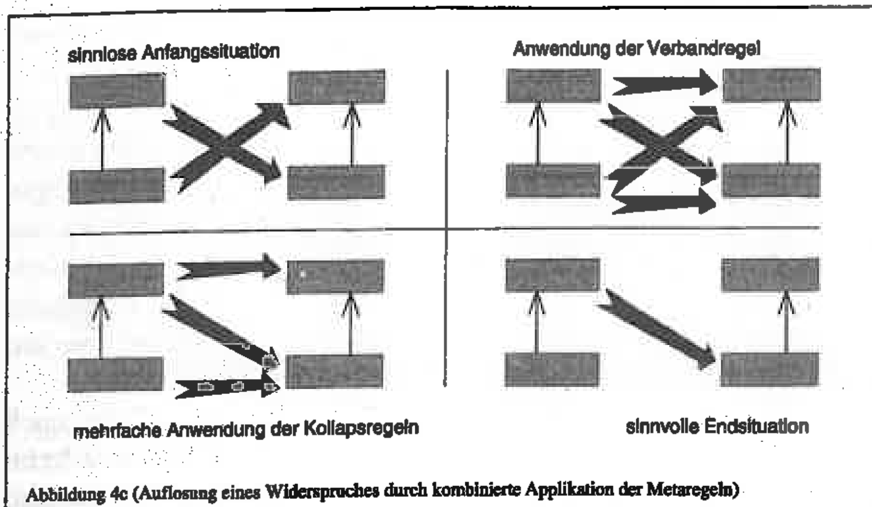


Abbildung 4c (Auflösung eines Widerspruches durch kombinierte Applikation der Metaregeln)

Definition 4.12 (Integrationsspezifikation)
Ein Tripel (S_1, S_2, K_i), wobei S_1, S_2 Schemata und K_i eine Menge der Mächtigkeit $i < \infty$ von Schemakorrespondenzaxiomen heißt initiale Integrationsspezifikation.
Ein Tripel (S_1, S_2, K^*), wobei S_1, S_2 Schemata und K^* diejenige Menge, welche durch Anwendung des unten definierten Algorithmus auf die initiale Axiomenmenge K_i entsteht, heißt abgeschlossene Spezifikation.

Definition 4.13 (Abschlußalgorithmus)
EINGABE: $x = K_i$.
SCHRITT: $x = x \cup \{a_{kgo}, a_{ggu}\}$, wo a_{kgo}, a_{ggu} Axiome als Resultate der Verbandregel.
SCHRITT: $x = x - \{a_{kgo}\}$, falls durch a_{kgo} keine ambivalente Situation entstand.
SCHRITT: $x = x - \{a_{ggu}\}$, falls durch a_{ggu} keine ambivalente Situation entstand.
WIEDERHOLE, bis die Verbandregel nichts neues mehr erzeugt.
SCHRITT: $x = x - \{a_q\}$, wo a_q durch einen Quellkollaps löschbar.
WIEDERHOLE, bis kein Quellkollaps mehr anwendbar.
SCHRITT: $x = x - \{a_z\}$, wo a_k durch einen Zielkollaps löschbar.
WIEDERHOLE, bis kein Zielkollaps mehr anwendbar.

Von der Verbandregel eingefügte Axiome sind also nur dann permanent, wenn sie die Anwendung einer Kollapsregel zur Konfliktbeseitigung vorbereiten. Andernfalls sind die Erzeugnisse der Verbandregel überflüssig und werden sofort wieder gelöscht.

Satz 4.14 (Termination und Widerspruchsfreiheit)
Der Abschlußalgorithmus terminiert und liefert
eindeutig eine widerspruchsfreie Axiomenmenge K^*.

Beweis: Zwischen zwei endlichen Schemata kann die Anwendung der Verbandregel höchstens endlich viele Axiome neu erzeugen, (falls überhaupt). Danach kann die wiederholte Anwendung der Kollapsregeln höchstens endlich viele Axiome löschen. Dies sind aber genau die widersprüchlichen Axiome, denn:

Die Kollapsregeln erzeugen keine Widersprüche, da sie Axiome nur löschen. Die Verbandregel kann zwar neue Axiome erzeugen, jedoch definitionsgemäß widerspruchsfrei. In jeder nach Definition 4.10 widersprüchlichen Situation tritt ein Fall $a \leq b$ auf. Dann ist aber a die *ggu* von beiden und b die *kgo* von beiden. Darauf erzeugt die

Verbandregel sowohl ein allgemeineres Axiom, nämlich an *kgo*, als auch ein spezielleres Axiom, nämlich an *ggu*. Nach Anwendung der Verbandregel sind also sowohl an a, als auch an b jeweils zwei Axiome beteiligt. Jeweils eines davon wird durch eine Kollapsregel gelöscht. Da eine Kollapsregel aber entweder n u r speziellere Axiome löscht (Quellkollaps) oder n u r allgemeinere (Zielkollaps), können nicht zugleich beide von der Verbandregel erzeugten Axiome wieder gelöscht werden, falls die Löschungen auf Quell- und Zielseite einander nicht abwechseln. Der Algorithmus ist aber so definiert, daß er Quell- und Zielkollapse separiert. Da aber sowohl an a als auch an b gelöscht wird, muß folglich entweder an a oder an b ein ursprüngliches Axiom gelöscht werden, welches einen Konflikt verursacht hatte.*qed*

Nach Anwendung des Abschlußalgorithmus kommen also weder ambivalente noch widersprüchliche Situationen mehr vor. Nicht ausgeschlossen sind dadurch jedoch Axiomenpaare, welche zwar eine gemeinsame Quelle (oder ein gemeinsames Ziel) besitzen, deren beiden Ziele (respektive Quellen) aber mit der Größenrelation $\leq$ nicht vergleichbar sind (vgl. Definitionen 4.8 und 4.9). Für diese Fälle muß zur Definition der Semantik der gesamten Schemakorrespondenzspezifikation erst noch eine weitere Axiomenpaarsemantik definiert werden.

Definition 4.15 (Pseudoaxiom)

a) Zwei Axiome $a_1 \Longrightarrow x$ und $a_2 \Longrightarrow y$ mit $a_1 = a_2$ und $x \not\leq y \wedge y \not\leq x$ bilden ein Pseudoaxiom $a_1 \Longrightarrow \{x, y\}$ mit der Semantik $[[a_1 \Longrightarrow x]] \cup [[a_2 \Longrightarrow y]]$.
b) Zwei Axiome $x \Longrightarrow b_1$ und $y \Longrightarrow b_2$ mit $b_1 = b_2$ und $x \not\leq y \wedge y \not\leq x$ bilden ein Pseudoaxiom $\{x, y\} \Longrightarrow b_1$ mit der Semantik $[[x \Longrightarrow b_1]] \cup [[y \Longrightarrow b_2]]$.
c) Zwei Axiome oder Pseudoaxiome $A_1 \Longrightarrow M$ und $A_2 \Longrightarrow N$ mit $M = \{x_1, \ldots, x_m\}, N = \{y_1, \ldots, y_n\}$ und $A_1 = A_2$ und $x_i \not\leq y_j \wedge y_j \not\leq x_i \forall x_i \in M, y_j \in N$ bilden ein Pseudoaxiom $A_1 \Longrightarrow M \cup N$ mit der Semantik $[[A_1 \Longrightarrow M]] \cup [[A_2 \Longrightarrow N]]$.
d) Zwei Axiome oder Pseudoaxiome $M \Longrightarrow B_1$ und $N \Longrightarrow B_2$ mit $M = \{x_1, \ldots, x_m\}, N = \{y_1, \ldots, y_n\}$ und $B_1 = B_2$ und $x_i \not\leq y_j \wedge y_j \not\leq x_i \forall x_i \in M, y_j \in N$ bilden ein Pseudoaxiom $M \cup N \Longrightarrow B_1$ mit der Semantik $[[M \Longrightarrow B_1]] \cup [[N \Longrightarrow B_2]]$.
e) Alle anderen Axiome, welche an keinem Pseudoaxiom gemäß a), b), c) und d) teilnehmen, bilden ihr eigenes, singuläres Pseudoaxiom.

Zwei Axiome, welche die oben definierten Voraussetzungen erfüllen, werden also disjunktiv verallgemeinert. Sie stellen somit die indeterministischen Transformationsalternativen dar, zwischen denen der Benutzer eines Integrationswerkzeuges mitunter zu wählen hat, wie in Kapitel 3 besprochen. Analog zur Vereinigung von Axiomen werden Pseudoaxiome induktiv zu noch größeren Pseudoaxiomen geballt. Man beachte, daß disjunkte Axiome hingegen nicht zu einem Pseudoaxiom vereinigt werden dürfen, weil ein "oberstes", triviales Axiom $\hat{k} \Longrightarrow \hat{k}'$ sonst alle anderen, spezielleren Axiome ad absurdum führen würde. Abbildung 4d) zeigt zum Beispiel eine ambivalenzfreie, widerspruchsfreie Korrespondenzspezifikation, die aus vier Axiomen drei Pseudoaxiome bildet, welche nicht mehr weiter zusammengeballt werden können, da sich ihre rechten und linken Seiten jeweils paarweise voneinander unterscheiden. Die Pseudoaxiome heißen $b \Longrightarrow \{d, e\}$, $c \Longrightarrow \{d, f\}$ und $\{b, c\} \Longrightarrow d$. Wollte man fälschlicherweise die beiden letzteren miteinander vereinen, so wäre plötzlich b nach f transformierbar, was jedoch offensichtlich nicht der Fall ist. Würde hingegen das Axiom $c \Longrightarrow f$ durch ein Axiom $c \Longrightarrow e$ ausgetauscht, so entstünde aus den Pseudoaxiomen $b \Longrightarrow \{d, e\}$ und $c \Longrightarrow \{d, e\}$ ein einziges Pseudoaxiom $\{b, c\} \Longrightarrow \{d, e\}$.

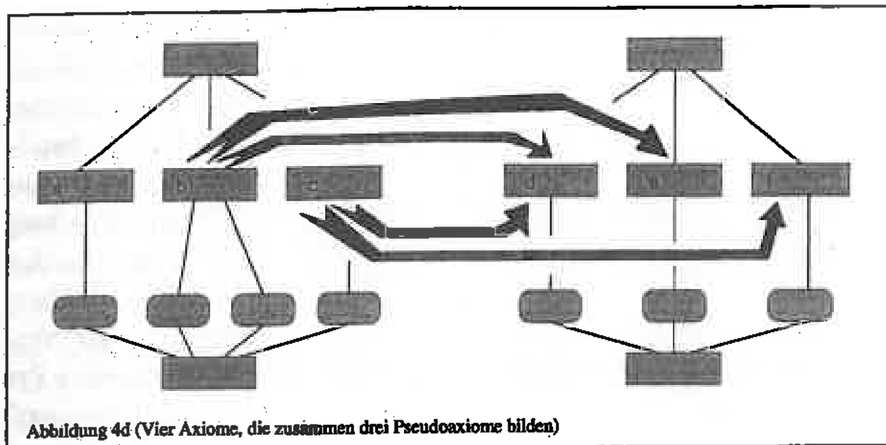


Abbildung 4d (Vier Axiome, die zusammen drei Pseudoaxiome bilden)

Damit ist nun endlich die gemeinsame Semantik aller Axiome einer Korrespondenzspezifikation definierbar.

Definition 4.16 (Semantik der abgeschlossenen Spezifikation)
Sei $I = (S_1, S_2, K^*)$ eine abgeschlossene Integrationsspezifikation.
Dann lautet ihre Semantik $[[(s_1, s_2, K^*)]] :=$
$\{[[A \Longrightarrow B]] | A \Longrightarrow B$ ist Pseudoaxiom $\in K^*\}$.
Dies ist eine Menge von Mengen von Transformationstupeln (t, t').

Definition 4.17 (Statische Semantik einer Paarproduktion)
Sei PP eine Paarproduktion bestehend aus einer tabellarischen Paarung (vgl. z.B. Abbildung 3f) von Quell- und Zielknoten der Typen q_i und z_i.
Dann gelte $[[PP]] := \{(q_i, z_i) | q_i, z_i \in PP\}$.
Es ist kein Zufall, daß die Form von $[[PP]]$ der Form von $[[A \Longrightarrow B]]$ entspricht, da zur Konsistenzprüfung einer Regel alle Pseudoaxiome der abgeschlossenen Spezifikation mit dieser Regel verglichen werden müssen. Dies ermöglichen nun primitive Mengenoperationen, wie folgendermaßen definiert.

Definition 4.18 (Statische Korrektheit einer Transformationsregel)
Eine paargrammatische Produktion PP heißt statisch korrekt bezüglich einer abgeschlossenen Integrationsspezifikation I genau dann wenn gilt:
$(t_q, t_z) \in [[PP]] \wedge (t_q, t_z) \in [[A \Longrightarrow B]]$ für alle Paare (q_i, z_i) in PP
und für alle diejenigen $[[A \Longrightarrow B]]$ in I mit der Eigenschaft $\exists t_x$: $(t_q, t_x) \in [[A \Longrightarrow B]]$.
Das heißt, daß zum Vergleich mit einer Paarproduktion PP all diejenigen Pseudoaxiome heranzuziehen sind, welche über mindestens einen von PP transformierten Quelltypen t_q etwas aussagen. Eine Transformationsregel wird dann verworfen, wenn sie den Zielbereich eines relevanten Pseudoaxioms $A \Longrightarrow B$ verläßt, d.h. wenn sie ein Paar (q_i, z_i) enthält, welches nicht in $[[A \Longrightarrow B]]$ enthalten ist.

4.5 Beispiel

Zum Abschluß dieser formalen Exkursion sei nochmals auf das einführende Beispiel in Kapitel 3 verwiesen. Abbildung 3j) wurde —die Reihenfolge der Ausführungen in Kapitel 4 umkehrend— aus den Produktionen 3g) und 3h) abstrahiert, um das Bedürfnis

nach Schemakorrespondenzen zu motivieren. Deshalb ist es an dieser Stelle trivialerweise unnötig, die Konsistenz von 3g) und 3h) mit den Axiomen aus 3j) verifizieren. Stattdessen werden kurz die Wirkungen des Abschlußalgorithmus auf 3j) besprochen, (wobei die unsichtbaren jeweiligen untersten Klassen $\check{k}$ hier keine Rolle spielen). Die Verbandregel setzt über Berechnung $\Longrightarrow$ Funktionsmodul und Daten $\Longrightarrow$ Datenmodul das transiente Axiom Programmfragment $\Longrightarrow$ Modul, welches sofort wieder gelöscht wird, da kein zu lösender Konflikt an "Modul" oder an "Programmfragment" entstanden. Analog entsteht über Berechnung $\Longrightarrow$ Implementation und Daten $\Longrightarrow$ Datenmodul nur vorübergehend das Axiom Programmfragment $\Longrightarrow$ Einheit. Damit ist das Potential der Verbandregel erschöpft und Kollapsregeln versuchen ihr Werk. Die beiden Axiome am Typen "Berechnung" kollabieren jedoch nicht, da "Funktionsmodul" $\not\leq$ "Implemenation" und "Implementation" $\not\leq$ "Funktionsmodul". Der Abschlußalgorithmus terminiert, und die widerspruchsfreie Spezifikation hat ihre Prüfung tadellos bestanden. Ihre Semantik ist{{(Daten, Datenmodul)}, {(Berechnung, Implementation), (Berechnung, Funktionsmodul)}}, da die Axiome Berechnung $\Longrightarrow$ Implementation und Berechnung $\Longrightarrow$ Funktionsmodul zusammen ein Pseudoaxiom bilden, das indeterministische Auswahl verlangt.

5 Zusammenfassung und Ausblick

In diesem Aufsatz wurde eine n e u e M e t h o d e zur Spezifikation von Integrationswerkzeugen entwickelt. Sie beruht auf Korrespondenzaxiomen, deren Konsistenz zueinander durch einen Abschlußalgorithmus sichergestellt wird, welcher auf semantisch begründeten Metaregeln für Korrespondenzaxiome fußt. Diese Methode gilt für alle Schemata unabhängig von ihrer anwendungsbedingten Interpretation. Es wurde dargelegt, wie die Schemakorrespondenzmethode die Spezifikation der integrierenden paargrammatischen Graphtransformationen statisch unterstützt, indem sie Korrespondenzaxiome und paargrammatische Produktionen durch Abbildung auf eine gemeinsame Mengenstruktur einander vergleichbar macht. Die Intention dieser Arbeit ist durch die langjährige praktische Bewährung der paargrammatischen Integratorenspezifikation gerechtfertigt, deren Methode in Kapitel 3 anhand eines Beispiels aus der Softwaretechnik erläutert wurde.

Um die neue Methode auch praktikabel zu machen, soll in Zukunft ein Editor entwickelt werden, mit dem der Spezifikator eines Integrationswerkzeuges die Schemakorrespondenzaxiome mit "Maus und Bildschirm" statt mit Bleistift und Papier leicht setzen und ihren automatischen Abschluß durch die Metaregeln beobachten kann. Der Editor muß in die bisherige Umgebung der Graphtransformationssprache PROGRES integriert werden, damit die statische Korrektheit der Graphproduktionen jederzeit überprüfbar ist. Der neue Editor würde somit insgesamt zum nützlichen Teil einer großen Umgebung von Werkzeugen und wiederverwendbaren Programmodulen zum Integratorbau, die in [2] ausführlich beschrieben ist. Denkbar wäre auch eine Vereinigung des Ansatzes mit [27]. Dort werden versuchsweise schon ganze Integrationswerkzeuge aus der Spezifikation der Korrespondenzen der graphgrammatischen Produktionen automatisch generiert. Für die dortige Generierung der Transformationsregeln aus der Spezifikation der Paargrammatik dürfte die hier vorgestellten Formalisierung der statischen Korrespondenzbedingungen recht nützlich sein. Insgesamt ergäbe auch dies ein neues Bauteil im Lager der zukünftigen Integratorenfabrik.

Nicht nur die praktische Anwendung, sondern auch die theoretische Untermauerung der neuen Methode steht noch in den Anfängen. Der vorgestellte Korrespondenzkalkül sagt bisher nur aus, w a s transformiert werden soll, aber nicht, was n i c h t transformiert werden soll. Aussagen wie: "Jedes Element aus dem Quellschema darf in jedes Element des Zielschemas transformiert werden, nur darf a nicht in b transformiert werden", sind in dem oben entwickelten Kalkül noch nicht darstellbar. Eine Erweiterung um negative Axiome ist jedoch denkbar, damit obige Aussage dann durch $\hat{k} \Longrightarrow \hat{k}'$, $a \not\Longrightarrow b$ leicht formalisiert werden kann. Bisher wurden Korrespondenzaxiome auch nur auf Typen und Klassen, also Knoten der Graphenschemata definiert. Intraschematische Relationskanten und Knotenattribute blieben unberücksichtigt. Möglicherweise wäre es hilfreich, auch solche Relationen und Attribute statisch korrespondieren zu lassen.

Komplizierte Integrationsprobleme stellen sich auch in der chemischen Verfahrenstechnik [34]. Im Rahmen einer Dissertation soll die Brauchbarkeit der Schemakorrespondenzmethode anhand eines hinreichend komplexen Integrationsproblems aus der chemischen Verfahrenstechnik exemplarisch nachgewiesen werden.

Insgesamt dürfte man sich davon einen kleinen Fortschritt im Integratorenbau mit allen Konsequenzen in den davon abhängigen informatischen und ingenieurtechnischen Disziplinen erhoffen.

Dank

Aus einer bis dato unveröffentlichten Schrift "Werkzeugunterstützung zum Reengineering betriebswirtschaftlicher Informationssysteme" meiner Kollegin *K.Cremer* stammt das Beispiel aus dem Bereich der Softwaremigration, anhand dessen in Kapitel 3 das Prinzip der paargrammatischen Integratorspezifikation erläutert wurde.

Mein hochgeschätzter Lehrer *A.Schürr* war so bescheiden, sich nicht offiziell Mitautor dieser Arbeit zu nennen. Ihm sind sowohl die wichtigsten Ideen, als auch viele hilfreiche Anmerkungen und die mühselige Korrektur der früheren Entwürfe dieser Arbeit zu verdanken. Seine präzisen Fragen führten manche voreilige Vermutung ad absurdum und verhalfen zu besserer Erkenntnis.

Auch die Diskussion mit den Doktoranden und Gelehrten im Graduiertenkolleg "Informatik und Technik" war meiner Sache förderlich. Die grundsätzliche Skepsis der Professoren *K.Indermark* und *W.Oberschelp* sollte mit diesem Aufsatz nun hoffentlich zerstreut worden sein. Dennoch sei an dieser Stelle nochmals betont, daß die Unterstützung komplizierter Arbeitsprozesse mit strukturellen Zuordnungen weder durch Makebefehle in UNIX ersetzt werden kann, noch in die unentscheidbare Formelwüste der theoretischen Chemie führen wird.

Während der Entstehung dieses Aufsatzes war meine beste Freundin *Irmgard* besonderns geduldig und verständnisvoll zu mir. Ihr widme ich diesen Beitrag.

Schrifttum

[1] P.Schefe, *Informatik – Eine konstruktive Einführung*, 2.Aufl., B.I. Wissenschaftsverlag Mannheim/Wien/Zürich, 1987

[2] M.Nagl, *Building tightly integrated software development environments: the IPSEN approach*, LNCS 1170, Springer-Verlag Berlin, 1996

[3] G.Vollmer, *Auf der Suche nach der Ordnung: Beiträge zu einem naturalistischen Welt- und Menschenbild*, S.Hirzel Wissenschaftliche Verlagsgesellschaft Stuttgart, 1995

[4] K.Pohl, *Three dimensions of requirements engineering: a framework and its applications*, Information Systems 19(3), pp 243-258, 1994

[5] D.Harrison/A.Newton/R.Spickelmeir/T.Barnes, *Electronic CAD frameworks*, Proc. of the IEEE 78(2), pp 393-419, 1990

[6] G.Spur, *Datenbanken für CIM*, Springer-Verlag/TÜV Rheinland G.m.b.H., Köln 1992

[7] R.Conradi/B.Westfechtel, *Version models for software configuration management*, Technischer Bericht (Aachener Informatikberichte 96-10), Rheinisch-Westfälische Technische Hochschule, Aachen 1996

[8] W.Tichy, *Configuration management*, Verlag John Wiley& Sons New York, 1994

[9] J.Conklin, *Hypertext: an introduction and survey*, IEEE Computers (September 1987), pp 17-41

[10] J.Ferrans/D.Hurst/M.Sennett et.al., *Hyperweb: a framework for hypermedia-based environments*, Proc. 5th Symposium on practical software development environments, ACM SIGSOFT software engineering notes 17(5), pp 1-10, 1992

[11] P.Borras et.al., *Centaur: the system*, in P.Henderson, Proc 3rd ACM SIGSOFT/SIGPLAN software engineering symposium on practical software development environments, ACM software engineering notes 13(5), pp 14-24, 1988

[12] A.Habermann/D.Notkin, *Gandalf: software development environments*, Transactions on software engineering 12(12), pp 1117-1127, 1986

[13] G.Kaiser/S.Kaplan, *Parallel and distributed incremental attribute evaluation algorithms for multiuser software development environments*, ACM Transactions on software eng. and methodology 2(1), pp 47-92, 1993

[14] A.Habermann/D.Notkin, *Gandalf: Software Development Environments*, Special Issue on Programming Environments, TOSE 12(12), pp1117-1127, 1986

[15] P.Borras et.al., *Centaur: the system*, P.Henderson, Proc. 3rd Symp. ACM Software Engineering Notes 13(5), pp14-24, 1988

[16] H.Ganzinger/R.Giegerich, *Attribute coupled Grammars*, in Proc. ACM SIGPLAN '84 Symp. on Compiler Construction, ACM SIGPLAN Notices 17(6), pp172-184, 1984

[17] J.Cordy/C.Halpern/E.Promislow, *TXL: a rapid prototyping system for programming language dialects*, Proc. IEEE International conference on computer languages, Miami 1988

[18] J.Cordy/I.Carmichael, *The TXL programming language: syntax and informal semantics, version 7*, External technical report 93-355, Department of Computing and Information Science, Queen's University, Kingston(CDN) 1993

[19] M.Cagan, *The HP softbench environment: an arichtecture for a new generation of software tools*, Hewlett-Packard Journal, pp 36-47, 6/1990

[20] D.Fromme, *HP encapsulator: bridging the generation gap*, Hewlett-Packard Journal, pp 59-68, 6/1990

[21] S.Reiss, *Interacting with the FIELD environment*,
Software practice and experiance vol.20, pp 89-115, 1990
[22] T.Pratt, *Pair Grammars, graph languages and string-to-graph translations*,
Journal of computer and system sciences 5, pp 560-595, Academic Press, 1971
[23] A.Schürr, *Specification of graph translators with triple graph grammars*, in: G.Tinhofer, Proc. WG'94
Int. workshop on graph theoretic concepts in computer science,
LNCS 903, pp.151-163, Springer-Verlag, Berlin 1994
[24] M.Lefering, *Integrationswerkzeuge in einer Softwareentwicklungsumgebung*, Verlag Shaker, Aachen 1995
[25] M.Jeusfeld/U.Johnen, *An executable metamodel for re-engineering of database schemas*, International Journal of cooperative information systems 4(2/3), pp 237-258, 1995
[26] T.Janning, *Requirements engineering und Programmieren im Großen: Integration von Sprachen und Werkzeugen*,
Deutscher Universitätsverlag Wiesbaden, 1992
[27] J.Jahnke/W.Schäfer/A.Zündorf, *A design environment for migrating relational to objectoriented database systems*, ICSM96 und
Technischer Bericht der Universität Paderborn, 1996
[28] A.Schürr, *Operationales Spezifizieren mit programmierten Graphersetzungssystemen*,
Deutscher Universitätsverlag Wiesbaden, 1991
[29] M.Nagl, *Softwaretechnik: Methodisches Programmieren im Großen*,
Springer-Verlag Berlin, 1990
[30] G.Gottlob/T.Frühwirth/W.Horn, *Expertensysteme*,
Reihe "Angewandte Informatik", Springer-Verlag Wien, 1990
[31] G.Birkhoff, *Lattice Theory*, American Math. Soc. 1948
[32] H.Hermes, *Einführung in die Verbandstheorie*,
Springer-Verlag Berlin, 1967
[33] U.Schöning, *Logik für Informatiker*, 3.Aufl.,
B.I. Wissenschaftsverlag Mannheim/Wien/Zürich, 1992
[34] R.Bogusch/B.Lohmann/W.Marquardt, *Computer aided process modeling with Modkit*, Proc. Chemputers Europe III, Frankfurt 1996

SUBMITTED FOR PUBLICATION
ZUR VERÖFFENTLICHUNG EINGEREICHT