Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-92-02

Constraint-basierte Plazierung in multimodalen Dokumenten am Beispiel des Layout-Managers in WIP

Wolfgang Maaß

Januar 1992

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

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

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

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Philips, SEMA Group Systems, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Intelligent Communication Networks
- Intelligent Cooperative Systems.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Prof. Dr. Gerhard Barth
Director

Constraint-basierte Plazierung in multimodalen Dokumenten am Beispiel des Layout-Managers in WIP

Wolfgang Maaß

DFKI-D-92-02

Diese Arbeit wurde durch das Bundesministerium für Forschung und Technologie unterstützt (FKZ ITW-8901 8).

© Deutsches Forschungszentrum für Künstliche Intelligenz 1992

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, Federal Republic of Germany; an acknowledgement of the authors and individual contributors to the work; all applicable portions of this copyright notice. Copying, reproducing, or republishing for any other purpose shall require a licence with payment of fee to Deutsches Forschungszentrum für Künstliche Intelligenz.

Constraint-basierte Plazierung in multimodalen Dokumenten am Beispiel des Layout-Managers in WIP

Wolfgang Maaß
Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI)
Stuhlsatzenhausweg 3
W-6600 Saarbrücken 11
Phone: (+49 681) 302-6261
Fax: (+49 681) 302-5341
E-mail: maass@dfki.uni-sb.de

Zusammenfassung

Bei innovativen intelligenten Benutzerschnittstellen, wie im Beispiel des multimodalen Präsentationssystems WIP, spielt insbesondere die automatische Plazierung von Graphiken und Texten eine wichtige Rolle. Das komplexe Plazierungsproblem läßt sich dabei als Constraint-Satisfaction-Problem auffassen. Zu dessen Lösung haben wir das System CLAY, welches ein integraler Bestandteil des Layout-Managers von WIP ist, entwickelt. Das Constraint-Solver-Modell CLAY, basierend auf der Kopplung zweier dedizierter Constraint-Solver, erlaubt die effiziente Verarbeitung komplexer graphischer Beziehungen, wie sie besonders im funktionalen Layout vorherrschen. CLAY stellt einen effizienten und flexiblen Mechanismus zur deklarativen Spezifikation und Lösung graphischer Gestaltungsprobleme dar. In dieser Arbeit werden dem Constraint-Solver-Modell zugrundeliegende abstrakte Algorithmen sowie den Constraint-Definitionssprachen vorgestellt und an Hand von Beispielen illustriert.

Inhaltsverzeichnis

Wie viel weißt du, o Mensch, der Schöpfung König,
Der du, was sehbar siehst, was meßbar mißt,
Wie viel weißt du! und wieder, ach, wie wenig,
Weil, was scheint, doch nur ein äußres ist.

Und steigst du in die Tiefe der Gedanken,
Wie findest du den Rückweg in die Welt?
Du armer König, dessen Reiche schwanken,
Der eine Krone trägt, allein kein Zepter hält.

Zu dem Gewölb von deinen strengen Schlüssen
Stellt sich der Schlußstein nun und nimmer ein,
Und die Empfindung, Flügel an den Füßen
Entschwebt der Haft und ruft hinfliegend: Nein!

Denn etwas ist, du magst's wie weit entfernen,
Das dich umspinnt mit unsichtbarem Netz,
Das, wenn du liebst, du aufschaust zu den Sternen,
Dich unterwerfend dasteht: das Gesetz.

Franz Grillparzer

Kapitel 1

Einleitung

Mit ein bißchen Manipulation kann man den Leser geschickt durchs Blatt leiten. Auf die Plazierung kommt es an.

Norbert Küpper, Zeitungsdesigner

Zur Präsentation von Informationen wird im allgemeinen eine integrierte Zusammensetzung von Texten und Graphiken verwendet. Das Zusammenspiel beider Ausgabemodi soll es dem Leser erlauben, innerhalb kürzester Zeit die dargebotenen Informationen zu verstehen. Dies ist besonders wichtig, da eine unkoordinierte Flut an Informationen den Leser vor die teilweise sehr schwierige Aufgabe stellt, die für ihn wichtigen Punkte eines solchen multimodalen Dokumentes zu erkennen. Auf der anderen Seite steht der Layouter unter dem ständigen Druck der Erhöhung der Produktivität durch immer innovativere und effizientere Verfahren. Eine Entlastung könnte dabei der Einsatz von neuen wissensbasierten Techniken bieten.

Bei der automatischen Erzeugung von multimodalen Dokumenten fällt dem Layout eine zentrale Rolle zu. Dabei ist unter dem Layout eines Dokumentes die Bestimmung einer adäquaten äußeren Form zu verstehen, die gewünschten ästhetischen Charakteristika genügt und darüberhinaus gewisse funktionale Anforderungen erfüllt. Es ist somitnicht das freie, künstlerische Design von Dokumenten gemeint, in welchem dem Designer erlaubt ist, beliebige Modellierungen an seinen Texten und Graphiken vorzunehmen. Psychologische Untersuchungen haben gezeigt, daß eine klare und logische Gestaltung der Form einer Präsentation ihre *Effizienz* und *Expressivität*, d.h. Lesbarkeit, Verständlichkeit und Glaubwürdigkeit, erheblich fördern. Ein gutes funktionales Layout zeichnet sich dabei vor allem durch seinen transparenten, sachlichen und ästhetischen Charakter aus (vgl. [Braun 87]).

Wir konzentrieren uns in dieser Arbeit ausschließlich auf das beim Entwurf von

Dokumenten relevante funktionale Layout. Im Gegensatz zum künstlerischen Layout steht hierbei mehr die Übermittlung des Inhaltes als die äußere Form der Darstellung im Vordergrund.

1.1 Motivation

Im allgemeinen ist der Begriff des Layouts nur schwer zu fassen, da nur sehr ungenaue und sich teilweise widersprechende Definitionen von Layout existieren. Eine mögliche Definition findet sich im Lexikon für Kunst [Alscher 68].

Definition 1.1 *Layout ist ein genau ausgeführter Entwurf für die wirksame Gestaltung eines Druckwerkes in besonderer Satzordnung oder mit verschiedensten typographischen Elementen wie Bilder, Schmuck, Anzeigen usw. und u.a. ist es auch für die Werbezwecke gebräuchlich.*

Eine zentrale Stellung nimmt demnach die Plazierung der Layoutobjekte ein. Sie entscheidet wesentlich darüber, ob ein Dokument gewissen ästhetisch-künstlerischen Gesichtspunkten genügt. Eine andere, etwas allgemeiner gehaltene Definition, ist wie folgt (vgl. [Tanner 90]).

Definition 1.2 *Unter einem Layout versteht man die Gestaltung (Aufmachung) einer oder mehrer Seiten. Man könnte auch sagen, daß damit das generelle Erscheinungsbild einer Drucksache gemeint ist. Ein Layout zeichnet sich durch ein einheitliches Konzept aus, das sich z.B. in einem einheitlichen Gestaltungsraster, einheitlicher Paginierung, Text- und Bildverwendung, kurz in einer gewissen Durchgängigkeit (Regelmäßigkeit) ausdrückt.*

1.2 Das WIP-Projekt

Der Präsentation von Informationen kommt bei der Gestaltung von intelligenten Benutzerschnittstellen eine herausragende Bedeutung zu, welche die Aufgabe haben dem Benutzer in adäquater Weise Informationen darzubieten. Aus dieser Motivation heraus ist das Projekt WIP (**W**issensbasierte **I**nformations**p**räsentation) entstanden, welches am Deutschen Forschungszentrum für Künstlische Intelligenz (DFKI) in Saarbrücken durchgeführt wird. WIP fungiert als unidirektionale Schnittstelle eines wissenverarbeitenden Hilfesystems. In WIP werden jedoch nicht Standardtechniken, wie z.B. die Ausgabe vorgefertigter Erklärungstexte oder vordefinierter Graphiken eingesetzt, vielmehr muß WIP in Abhängigkeit von Generierungsparametern wie Präsentationsziel, Präsentationssituation, Betriebsmittelbeschränkung und Zielsprache eine für den jeweiligen Benutzer angemessene Präsentationsform intelligent auswählen (s. Abb. 1.1). Zur Bewältigung dieser Aufgabe benötigt das WIP-System zur Wissensdarbietung nicht nur anwendungsspezifisches Fachwissen,

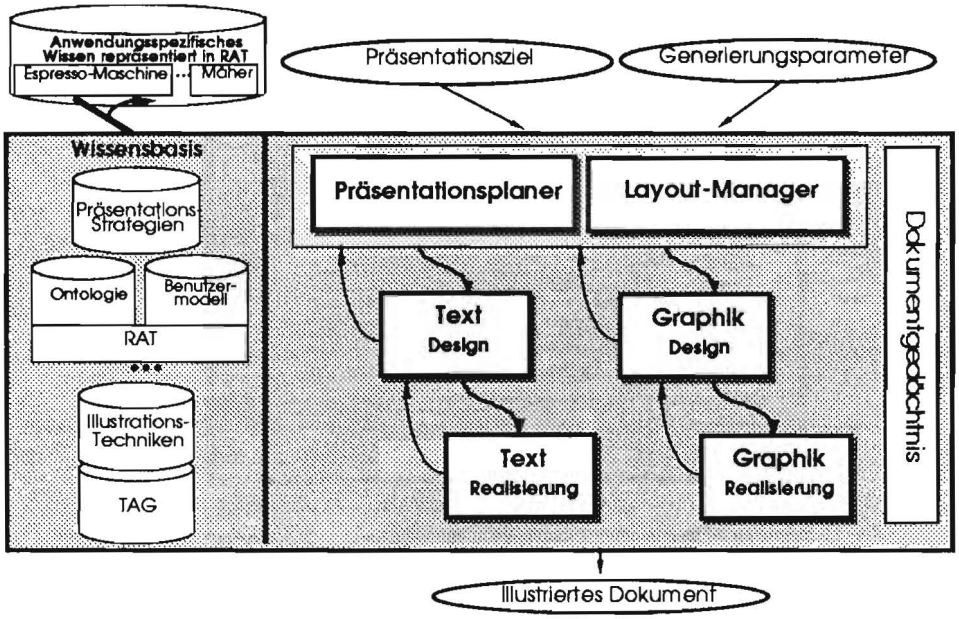

Abbildung 1.1: Die funktionale Sicht von WIP

sondern gerade auch *Alltagsintelligenz*, die auf Heuristiken und Erfahrungswerten aus Alltagstheorien beruhen.

Die WIP-Architektur erlaubt eine große Freiheit in Bezug auf den Dokumenterstellungsprozeß. Während des Prozesses dirigieren der Präsentationsplaner und der Layout-Manager die Generatoren unterschiedlicher Modalität.

Für die inkrementelle Erzeugung von Graphiken und Texten verwendet WIP zwei parallele Verarbeitungskaskaden. Dabei wird das Design eines multimodalen Dokumentes als ein nichtmonotoner Prozeß verstanden. Dieser besteht im allgemeinen aus zahlreichen Rücknahmen von vorläufigen Ergebnissen, massivem Neuplanen oder Planveränderungen, Verhandlungen zwischen den korrespondierenden Design- und Realisierungskomponenten zum Erhalt einer feinkörnigen und optimalen Aufteilung der Arbeit zwischen den selektierten Präsentationsmodi. Für eine ausführliche Darstellung des Projektes WIP sei auf [Wahlster et al. 91a, Wahlster et al. 91b] verwiesen.

Betrachtet man den Layout-Manager (vgl. [Graf 91]), so besteht seine Hauptaufgabe in der Auswertung und Erhaltung einer Reihe von semantisch-pragmatischen Beziehungen, welche durch den Präsentationsplaner vorgegeben werden (s. Abb. 1.2). Diese beziehen sich auf das Arrangement von Graphik- und Textfragmenten, die von den verschiedenen Generatoren erzeugt werden. Mittels dieser Beziehungen bestimmt der Layout - Manager die Größe der Fragmente und die exakten Koordinaten für deren Plazierung auf dem Dokument.

Im folgenden seien zur uniformen Behandlung die Ergebnisstrukturen der verschiedenen Generierungskomponenten[1] mit dem Begriff *Objekt* bezeichnet. Abstraktionen von Objekten, die für das Layout wesentliche Informationen enthalten, seien

[1] Momentan sind die Generierungskomponenten für Graphiken und Texte realisiert und in WIP integriert

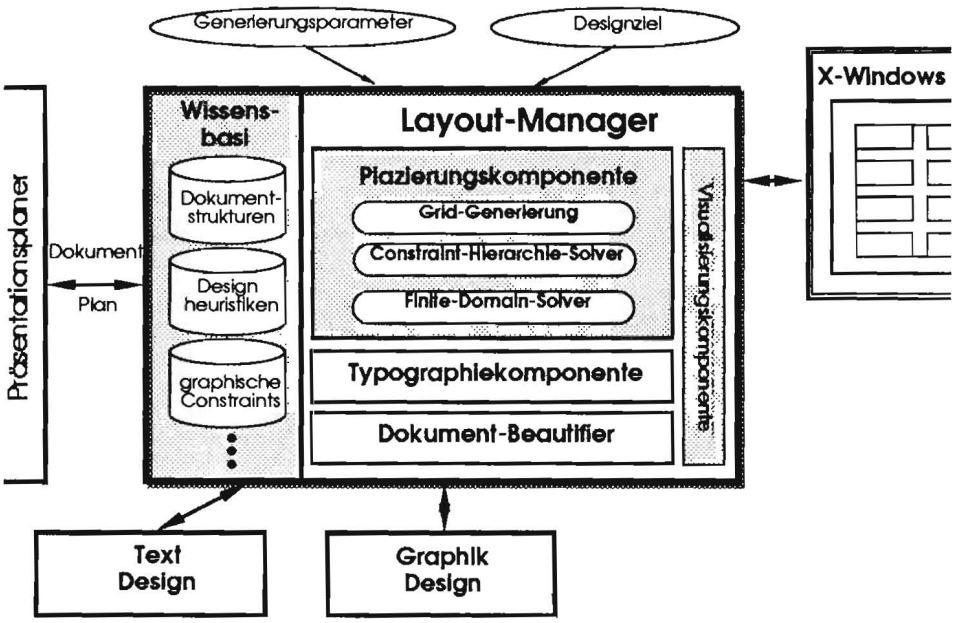

Abbildung 1.2: Der Layout-Manager in WIP

mit dem Begriff *Layoutobjekt* bezeichnet. Der Begriff des Layoutobjekts wird in Abschnitt 3.1 genauer gefaßt.

1.3 Aufbau der Arbeit

Diese Arbeit ist im Rahmen der Mitarbeit im WIP-Projekt entstanden.
Kapitel 2 zeigt bekannte Ansätze zur Generierung eines Layouts sowie deren Verarbeitungsformalismen auf. Weiterhin werden bekannte Systeme vorgestellt, die sich mit ähnlichen Thematiken beschäftigt haben.
Kapitel 3 behandelt die für das Layout wichtigen Beziehungen und Verarbeitungsschritte und die Evaluierung diesbezüglicher Techniken.
Kapitel 4 beschreibt den der Verarbeitung zugrundeliegenden abstrakten Algorithmus sowie das Constraint-Solver-Modell.
Kapitel 5 stellt Aspekte der Implementierung vor.
Kapitel 6 rundet die Arbeit mit einem Beispiel für ein, mit dem beschriebenen Berechnungsmodell erzeugtes, automatisches Layout ab.
Im Anhang befinden sich u.a. die aktuell definierten Constraints und weitere Beschreibungen des Systems.

Kapitel 2

Stand der Forschung

Es wird in diesem Kapitel das Layout aus der Sicht des Graphik-Designs und der mathematisch-wissensbasierten Ansätze betrachtet. Danach werden verschiedene Formalismen vorgestellt, die der Repräsentation und Verarbeitung im Bereich des Layouts dienen. In deren Mittelpunkt steht dabei der constraint-basierte Ansatz.

2.1 Layout

Der Begriff des Layouts soll nun in annähernder Weise formalisiert und der Fokus dieser Arbeit auf einen Teilbereich ausgerichtet werden. Anschließend werden Methodiken des Graphik-Designs und des mathematischen insbesondere des wissensbasierten Bereichs diskutiert. Die Verbindung dieser beiden Ansätze wird in den darauf folgenden Kapiteln die Arbeit bestimmen.

2.1.1 Graphik-Design-Methoden

Diese Arbeit stellt das funktionale Layout eines Dokumentes in den Vordergrund. Der Begriff der Funktionalität bestimmt daher eine Selektion von Wesentlichem und Unwesentlichem, von Notwendigem und Unnötigem. Da die Selektion an sich nicht eindeutig formalisierbar ist, unterliegt man dabei sinn- und zweckmäßigen Optionen, die von Experten aus Erfahrung oder aus dem Gefühl entschieden werden. Unter diesen möglichen Ansätzen gibt es im Graphik-Design allgemein anerkannte Darstellungsformen, die sich formalisieren und operationalisieren lassen. Leithilfen wie der Gestaltungsraster (vgl. [Müller-Brockmann 81]) erlauben es desweiteren Teile der allgemeinen Vorgehensweise bei der Erstellung eines funktionalen Layouts zu systematisieren. Unterschiedliche Ansätze zur rechnergestützen Verarbeitung solcher Systematiken sind bereits entwickelt worden, jedoch werden in diesen das Wissen der Layoutexperten wenig oder gar nicht berücksichtigt.

Das Layout technischer Dokumente ist zwar in seiner Form wesentlich klarer strukturiert als das freie, künstlerische Layout, jedoch trifft man auch hier auf die Verwendung von intuitiven Techniken. Teilweise wird nach rein subjektiven Gesichtpunk-

ten das Layout eines Dokumentes bestimmt, ohne daß der Layouter die vollzogenen Schritte strukturiert darlegen kann.

Unter den möglichen Ansätzen zur Erzeugung eines funktionalen Layouts hat sich im Bereich des Graphik-Desgins besonders die Verwendung eines Gestaltungsrasters als sehr vorteilhaft erwiesen. Der Gestaltungsraster unterstützt den Layouter in der strukturierten Vorgehensweise, damit das Dokument gewissen Bedingungen, wie Konsistenz und Kohärenz in der Darstellung genügt.

Ausgehend vom funktionalen Layout von Dokumenten, erfährt man bei der Verwendung eines Gestaltungsrasters, als Grundlage der Anordnung eine recht starke Einschränkung der Freiheit, in Bezug auf die Positionierbarkeit von Texten und Graphiken. Auf der anderen Seite unterstützt der Raster aber ein Layout, welches wesentliche Informationen klar strukturiert und gut erkennbar hervorheben soll (vgl. [Müller-Brockmann 81]). Die Bestimmung eines Rasters erfordert eine genaue Analyse der darzustellenden Objekte und des Ausgabemediums. Beim Layout einer Zeitung hält man sich dabei meist an ein von Experten ermittelten Grundraster, auf dem nur minimale Änderungen vorgenommen werden dürfen, damit der Charakter des Layouts gewährleistet bleibt.

Erst mit der Systematisierung von Organisationsdrucksachen und der Kombination von Bild und Text bei Büchern, Broschüren und Prospekten hat der Gestaltungsraster seine eigentliche Bedeutung bekommen (vgl. [Tanner 90]). Mit der Erfindung des Buchdrucks mußte das Verhältnis von Text zu freier Fläche auf einer Buchseite fixiert werden. Als Orientierung dienten die bis dahin von Hand geschriebenen und illustrierten Bücher. Ihre Anordnungen wurden zur Grundlage auch für die Festlegung der Textkolumne des gedruckten Buchs. Daraus entwickelte sich dann der Satzspiegel. Allmählich wurden Harmonievorstellungen in der Zeit nach Gutenberg jedoch zunehmend von drucktechnischen Kriterien und darüber hinaus vom *Goldenen Schnitt*[1] bestimmt. Auch dem Druckbereich von Handzetteln, Aushängen und ähnlichen Gelegenheitsdrucksachen lag in früherer Zeit kein klarer Gestaltungsraster zugrunde. Bei Textinformationen stellte man mehrere Satzkolumnen aneinander, orientierte sich aber ansonsten an elementaren Gestaltungsgesetzen wie Symmetrie oder Asymmetrie oder begnügte sich mit der Hervorhebung von Titeln und anderen wichtigen Informationselementen.

Der Satz wurde auf der Mittelachse gestellt oder erfolgte in Block- oder Flattersatz. In der Frühzeit des Buchdrucks wurden Druckstücke mit sparsamen Mitteln, aber solcher Perfektion hergestellt, daß man heute den Begriff einer Druckkultur rechtfertigen kann. Die Harmonie von Schrift und freiem Raum entstand nahezu zwangsläufig. Entscheidungsstellen wie Fuß- und Kopfsteg wurden zum Goldenen Schnitt oder in Annäherung daran festgelegt. Der Drucker war ein Fachmann, der seine Arbeit nicht zu systematisieren brauchte, weil der gesamte Handlungsprozeß in seiner Hand lag. Der Gestaltungsraster unserer Zeit war überflüssig. Mit der Entstehung von periodisch erscheinenden Schriften änderten sich die Satzgewohnheiten, und einfache Raster für die Vereinheitlichung von Druckseiten wurden notwendig.

[1] Der Goldene Schnitt (*stetige Teilung*) ist die Teilung einer Strecke so, daß die ganze Strecke a zum größeren Teil q sich wie der größere Teil q zum kleineren p verhält, also $a{:}q{=}q{:}p$; Nährungswert 13:8. Zieht man p von q ab, so wird auch q stetig geteilt usw. (vgl. [Institut 82]).

Jetzt entstanden klassische typographische Formen wie mehrspaltiger Satz, Haupt- und Zwischenüberschriften sowie Marginalie, also aus dem Haupttext herausgezogene Textinhalte etc. Aber auch typographische Unsitten, Sperren von Minuskeln, Unterstreichen, Versalien im Fließtext etc. haben hier ihren Ursprung. Da ein Gestaltungsraster seine Qualität nur mit einer geordneten Typographie zur Geltung bringt, wird hier von Gestalter und Autor eine konsequente Auffassung verlangt.

Bei der Inflation von Druckstücken, mit der wir heute leben, spielt die Übersichtlichkeit der Information eine wichtige Rolle. Der Gestaltungsraster hilft die Bestandteile auf dem Druckstück in geordnete Informationseinheiten zu gliedern. Bild- und Textteile sollen ihre Schwerpunkte haben und ihr Ablauf muß klar sein. Die Herstellung von Broschüren, Zeitschriften, Katalogen etc. ist heute ohne Organisation und Systematik eines Gestaltungsrasters undenkbar. Der Gestalter sollte bei der Bearbeitung eines umfangreichen Druckstückes nicht ständig veränderte Bildmaße oder verschiedene Satzbreiten entwickeln, er muß den Inhalt der Aufgabe lösen. Die Festlegung eines Baukastens für Bildformate gemäß dem Gestaltungsraster erfolgt einmalig, wie auch die Festlegung der Eigenheiten der Typographie eine nur einmal zu treffende Entscheidung darstellt. Die konsequente Gestaltung mit einem Gestaltungsraster bedeutet auch Zeit- und Kostenersparnis, weil die Festlegung vieler unterschiedlicher Bildformate und Satzanweisungen wie auch die Reproduktion und Satzherstellung bei ständig wechselnden Maßeinheiten aufwendiger wäre. Bei der Verwendung eines Gestaltungsrasters wird eine gegebene Dokumentfläche durch vertikal und horizontal angeordnete, gleichgroße Felder eingeteilt (s. Abb. 2.3). Die Rasterfelder sind dabei an den horizontalen, wie vertikalen Seiten durch jeweils gleichbleibende Abstände voneinander getrennt. Diese Abstände richten sich, wie die Größe der Felder, den sog. *universellen Rasterfeldern*, nach den darzustellenden Objekten. Entscheidend ist dabei u.a. die Schriftart und die verschiedenen Größen der Graphiken. Sie haben die Aufgabe, Objekte voneinander optisch zu trennen und dadurch das gesamte Layout zu strukturieren. Werden die Abstände durch ein Layoutobjekt überdeckt, muß die Größe vorher so gewählt worden sein, daß das Objekt genau mit den Rändern der betreffenden Rasterfelder übereinstimmt. In die Bestimmung des Abstandes müssen somit die Größe der Graphiken und der Schriftarten mit eingehen. Ein universelles Rasterfeld entspricht der kleinsten, auf dem Dokument anordbaren Objektes. Texte werden immer durch Leerzeichen auf die volle Größe eines universellen Rasterfeldes erweitert, wohingegen Graphiken nur in minimaler Weise verändert werden dürfen. In manchen Ansätzen ist eine beliebige Beschneidung bzw. Vergrößerung von Graphiken zwar möglich, jedoch widerspricht dieses Vorgehen der Verwendung des Rasters als Hilfsmittel.

Soll auf dem Gestaltungsraster ein Text angeordnet werden, so muß dieser in seiner Breite mit der einer ganzen Anzahl von universellen Rasterfeldern übereinstimmen. In der Höhe muß der Text nicht unbedingt die Spalte füllen, sondern der verbleibende Rest bleibt für das gesamte Dokument frei. Im Gegensatz dazu müssen Graphiken eine ganze Anzahl von universellen Resterfeldern in der Höhe und der Breite überdecken, um nicht ungewollte Spannungen zu erzeugen (s. Abb. 2.1 und 2.2).

Der Raster darf dadurch nicht zu grob sein, weil dann die Graphiken nur schwerlich wenigstens ein Feld überdecken können. Es soll aber auch nicht zu fein sein, da dann der Sinn nicht mehr ganz erfüllt ist. Letztendlich führt ein beliebig feiner

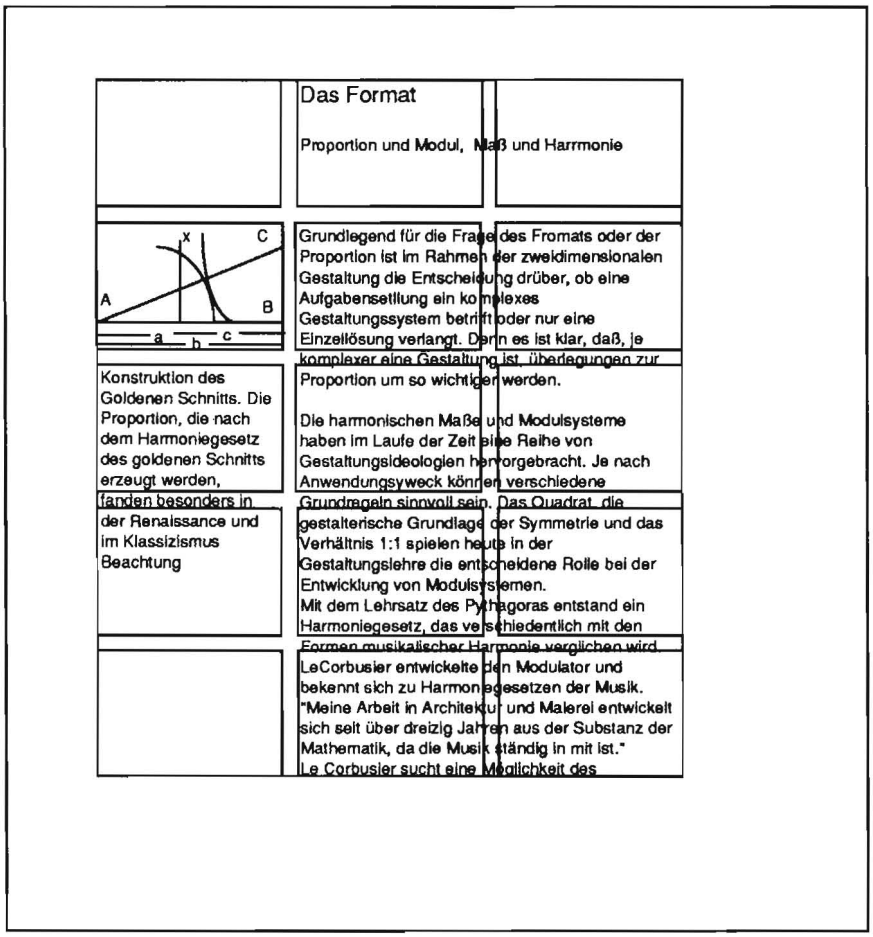

Abbildung 2.1: Layout mit Raster

Raster wieder zur vollkommen freien Plazierung, was der Intention dieses Ansatzes widerspricht. Die Verwendung des Rasters stellt zwar eine Einschränkung der künstlerischen Freiheit dar, kann aber bei richtiger Anwendung auch eine Hilfe für die Entfaltung eines künstlerischen Layouts sein.

2.1.2 Mathematische und wissensbasierte Ansätze

Einige Versuche Teilbereiche des Layouts zu formalisieren haben ihre Grundlagen in mathematischen Methoden. Sie behandeln die Anordnung von Objekten, die meist rechteckiger Form sind, auf einer zweidimensionalen Fläche. Solche Verfahren werden allgemein als *Packing*-Verfahren bezeichnet (vgl. [Beach 85]). Die Schwierigkeit bei der Anwendung dieser Algorithmen besteht darin, daß aufgrund der Komplexität des Problems die allgemeine Lösung zur Klasse der NP-vollständigen Probleme gehört. Verbesserungen, wie das *Random Pack-* Problem, welches die Bestimmung einer Anordnung für eine ungeordnete Menge von beliebigen Rechtecken auf einer Seite mit vorgeschriebener Länge und Breite bzw. der Bestimmung des minimalen Rechteckes, in das die Objekte gepackt werden können, betrifft, gehört ebenfalls zur Klasse der NP-vollständigen Probleme. Auch eine Einschränkung auf *Partition-* bzw. *Bin Packing* ergibt eine NP-Komplexität des Problems. Eine weitere Ein-

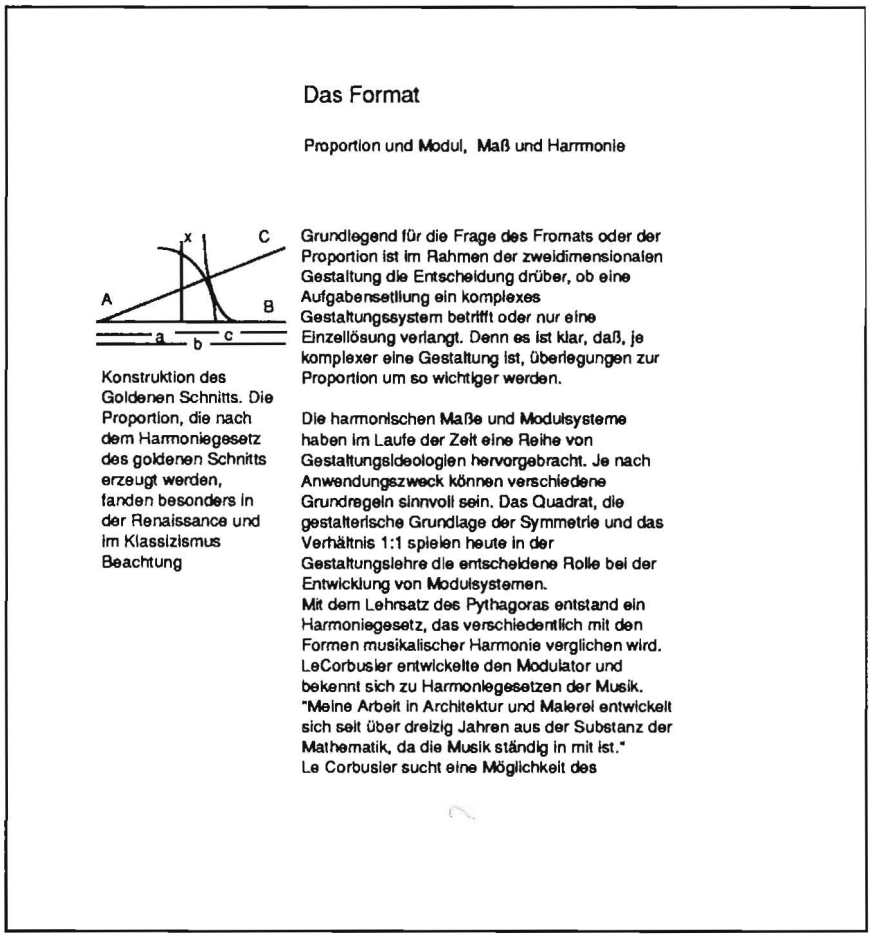

Abbildung 2.2: Layout ohne Raster

schränkung auf *Stub Packing* reduziert das Verfahren auf die Klasse $O(n^2)$ für n Rechtecke.

Erst die Angabe einer Ordnungsstruktur über die zu plazierenden Rechtecke ermöglicht es, die Komplexität des Algorithmus in der Klasse $O(n)$ zu lösen. Der Ausdruck Packing-Verfahren deutet an, daß der Schwerpunkt auf der platzoptimierenden Anordung von Objekten auf einer vorgegebenen Fläche liegt. Untersuchungen, ob diese Verfahren in der automatischen Layoutgenerierung verwendet werden können sind von Beach gemacht worden. I.a. kann diese Frage verneint werden, da sie weder funktionale noch ästhetische Aspekte berücksichtigen. Anwendungen dieser mathematischen Verfahren hat man im Bereich des VLSI-Designs entwickelt. Mit deren Hilfe werden dort verschiedenartige Objekte (meist integrierte Schaltungen) so angeordnet, daß Optima für Leitungswegen und Flächenausnutzungen gefunden werden.

Der Raster ist in verschiedenen DTP-Programmen integriert, was aber einem Laien im Normalfall wenig nutzt, da dieser meist nicht mit den ausgefeilten Techniken der Layouter bekannt ist. In solchen Fällen kommt das Prinzip *mehr Masse als Klasse* in seiner ganzen Tragweite zum Zuge, was zum Großteil vollkommen *überladene* Dokumente zur Folge hat. Ausgehend von dieser Problematik ist es sinnvoll das Expertenwissen eines Layouters zu formalisieren und zwecks Erstellung eines Do-

kumentes zu verarbeiten. Die meisten der bisherigen Systeme zur Automatisierung des Layouts beziehen sich im wesentlichen auf die Unterstützung von Routinedesignaufgaben, für die bereits eine Vielzahl von Designheuristiken bekannt sind und die somit eine effektive Kontrolle des Designraums ermöglichen. In letzter Zeit rücken Systeme zum konzeptionellen Design stärker in den Vordergrund. Die überwiegende Mehrheit dieser Systeme versteht dabei Layout als kombinatorisches Problem über einem diskreten Suchraum (vgl. [Navinchandra 91]).

Zur Verarbeitung und Repräsentation von layoutspezifischem Wissen haben sich solche Formalismen als sehr vorteilhaft herausgestellt, die Darstellungsformen, welche aus Texten und Graphiken bestehen, unterstützen. Verschiedene einfache Versuche zur wissensbasierten Layoutsynthese haben Wege aufgezeigt, an denen man sich bei der Formalisierung und der Verarbeitung halten kann.

Es werden im folgenden Ansätze vorgestellt, die eine strukturierte, und auch wissensbasierte, Vorgehensweise zur Erstellung eines Layouts zum Ziel haben.

Eine Verbindung der mathematischen Verfahren und einem wissensbasierten Ansatz stellt das System zur Erstellung eines Tabellen-Layouts von Beach dar, in dem das Layout und das Design von Tabellen automatisiert wird. Dadurch kann für verschiedene Ausgabemedien ein unterschiedliches Format generiert werden und der Benutzer kann auf bestimmte Teile des Designs Kontrolle ausüben, während andere über einen sog. *default style* bestimmt werden (vgl. [Beach 85]).

Der in Beach vorgestellte Ansatz zur Tabellen-Komposition legt als zentrale Idee eine Trennung von Inhalt und Form von Tabellen zugrunde. Basierend auf einer objekt-orientierten Dokumentstruktur wird das Layout der Tabelle über Grids spezifiziert. Die Tabellen-Anordnung bzw. -Topologie wird durch einen zweidimensionalen Raster bestimmt. Das eigentliche Tabellen-Layout bzw. die Tabellen-Geometrie wird dann aus der Topologie, sowie der Größe der Einträge über einen Constraint-Satisfaction-Algorithmus berechnet.

Ein weiterer wissensbasierter Ansatz ist das System *GRID* (Graphical Interface Design System) von Feiner (vgl. [Feiner 88]). In stark vereinfachter Weise wird in diesem System ein Layout für Text- und Graphikblöcke erstellt. Das Layout in GRIDS wird mit Hilfe eines Rasters erstellt, welches sich nur nach Aspekten der Lesbarkeit bestimmt. Ist der Raster festgelegt, so werden alle Objekte auf ein Vielfaches der Rasterfelder vergrößert oder verkleinert.

Die logische Struktur des Dokumentes wird durch eine sogenannte *prototype display grammar* repräsentiert. Dies ist eine Hierarchie, in der, ausgehend von den Graphik- und Textblöcken, logisch höhere Strukturen aggregiert werden können.

Das Layout wird über eine *Generate-and-Test*-Strategie erstellt, wobei das einzige Weigerungskriterium des Systems das Nichtpassen der darzustellenden Objekte auf einem Raster ist.

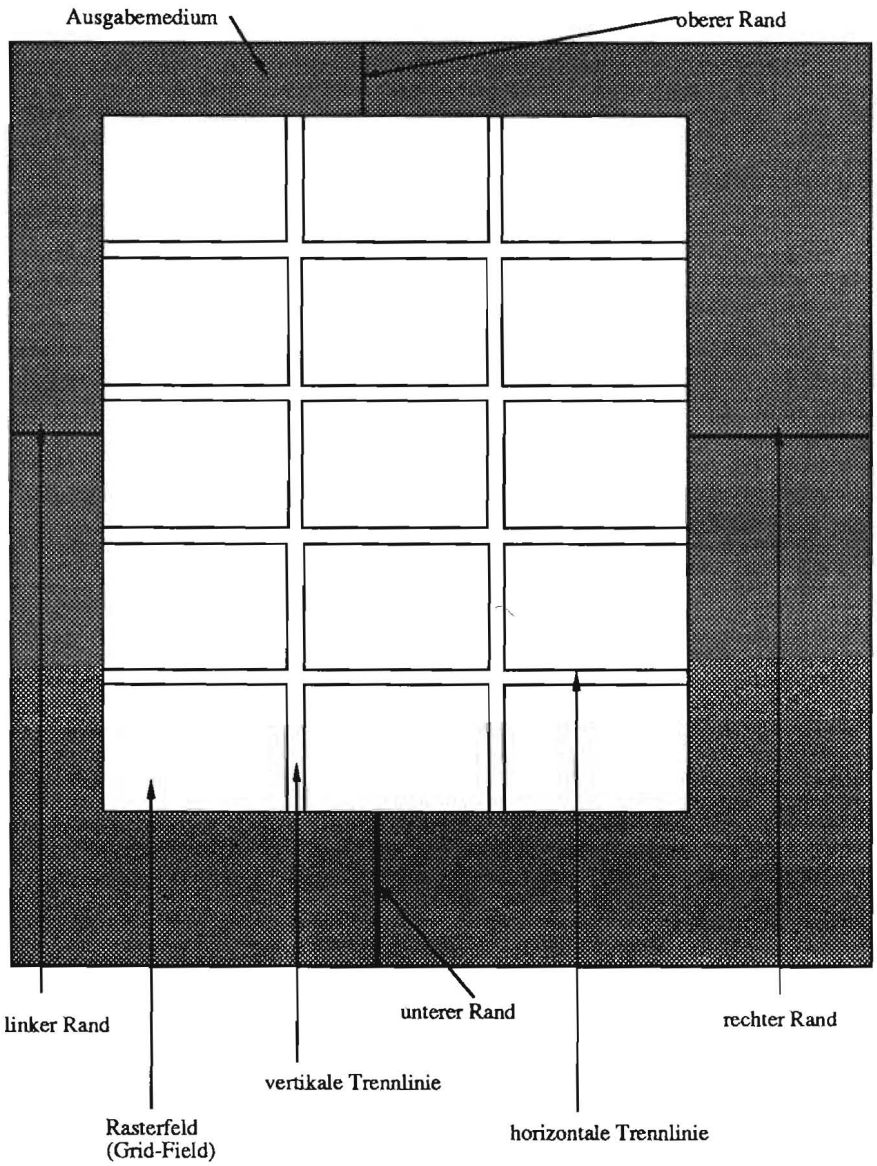

Abbildung 2.3: Beispiel eines dreispaltigen Rasters

2.2 Constraint-basierte Wissensrepräsentation

In allgemeiner Form bilden Constraints eine Möglichkeit Relationen auf einer endlichen Menge von Objekten zu erfüllen und zu erhalten. Constraints lassen sich unter verschiedenen Aspekten betrachten: als ein Berechnungsmodell, als effiziente Kontrollstrategie oder als Wissensrepräsentationsformalismus. Dabei hat sich gezeigt, daß Constraints bei bestimmten Problemen, die bei Verwendung von herkömmlichen Formalismen einer kombinatorischen Explosion unterliegen, ein effizientes Berechnungsmodell liefern. In diesen Fällen schränken Constraints den Suchraum des Problems auf die Bereiche ein, die für eine Lösung in Frage kommen (vgl. n-Damen-Problem in [van Hentenryck 89]). Betrachtet man den Constraint-Formalismus als Berechnungsmodell, so läßt sich dieser als Programmiersprache, ähnlich PROLOG auffassen. Dabei lassen sich Constraints, ähnlich Klauseln, deklarativ definieren und durch, vom Benutzer größtenteils unabhängiger Komponenten, anwenden und erhalten.

Wird ein Problem mittels Constraint-Techniken gelöst, so spricht man allgemein von einem Constraint-Satisfaction-Problem *(CSP)*. Ein solches *CSP* unterteilt sich in zwei Phasen. Die erste Phase beschäftigt sich mit der Erfüllung von Constraints (*constraint satisfaction*), wobei die Menge von Wertetupeln bestimmt wird, die alle Einschränkungen eines Constraint-Modells erfüllen.

Die Propagierung des Constraints (*constraint propagation*) bildet die zweite Phase, in der Änderungen von Variablenbelegungen, die durch das Constraint hervorgerufen worden sind, über das Constraint-Netz weitergegeben werden. Im einfachsten Fall werden alle übrigen Constraints neu berechnet, jedoch erlauben die meisten Constraint-Solver eine inkrementelle Propagierung, so daß nur solche Constraints neu ausgewertet werden, die direkt durch die aktuelle Änderung betroffen sind.

Die Ausweitung der Constraint-Theorie auf ein Programmiersprachenparadigma ist durch die Entwicklung des *Constraint-Logik-Programmier-Schemas (CLP)* erreicht worden. Ähnlich wie logische Programmiersprachen, verwendet *CLP* das Resolutionsprinzip (vgl. [Robinson 65]), jedoch wird das Konzept der syntaktischen Unifikation im Herbrand Universum durch Constraint-Lösen über der Domäne der Anwendung ersetzt. Das *CLP-Schema* ist zwar keine Erweiterung der logischen Programmierung, aber stellt einen allgemeinen Rahmen dar, von dem aus verschiedene PROLOG-Erweiterungen (z.B. CLP($\mathcal{R}$) [Jaffar et al. 87] und PROLOG III [Colmerauer 90]) abgeleitet werden können. Jede Instanz des Schemas ist eine *CLP*-Sprache, die eine eigene Anwendungsdomäne und einen eigenen Constraint-Lösungs-Mechanismus besitzt. Ein dabei wichtiges Ziel ist die Effizienz der Constraint-Sprache durch automatische Mechanismen wie der *Delay*-Technik (CLP($\mathcal{R}$), *CHIP*) und *Check Rules* (*CHIP*) oder von *Guarded Rules* und *Residuierung* (vgl. [Smolka 91]) zu steigern.

In den bekannten *CLP*-Sprachen ist die ganze Funktionalität von herkömmliche logischen Programmiersprachen integriert, so daß die Expressivität und Mächtigkeit des Berechnungsmodells, gepaart mit der Einfachheit und der Sauberkeit der Semantik, wie es beispielsweise bei PROLOG der Fall ist, zur Verfügung steht. Zusätzlich bieten *CLP*-Sprachen Möglichkeiten zur Berechnung von impliziten Ergebnissen, die

z.B. in PROLOG nicht möglich sind. Implizite Ergebnisse werden dann erzeugt, wenn keine weitere Vereinfachung der Constraints möglich sind [2].

Das Layout wird wesentlich durch die Beziehungen zwischen den Layoutobjekten bestimmt. Beziehungen, wie sie im Layout vorherrschen lassen sich durch mathematische Relationen formalisieren. Sie können dabei auf eine beliebige Anzahl von Layoutobjekten gleichzeitig wirken und beschreiben dabei einen Zusammenhang zwischen den Objekten, der sich in einer topologischen Anordnung zueinander ausdrückt. Constraints haben sich dabei zur Verarbeitung und Repräsentation von Relationen auf beliebigen Objekten als nahezu ideal herausgestellt.

2.2.1 Constraint-Formalismen

Die allgemeine Aufgabe eines Constraint-Formalismus besteht in einer Menge von Variablen über einem bestimmten Wertebereich und einer Menge von Prädikaten, die die Werte der Variablen simultan erfüllen müssen (vgl. [Mackworth 77]). Bezieht man sich hierbei auf unäre und binäre Prädikate, so sind Formeln der folgenden Form zu beweisen:

$$(\exists x_1)(\exists x_2 ... (\exists x_n) \quad P_1(x_1) \wedge P_2(x_2) \wedge ... \wedge P_n(x_n) \wedge \\ P_{12}(x_1, x_2) \wedge P_{13}(x_1, x_3) \wedge ... \wedge P_{n-1,n}(x_{n-1}, x_n),$$

wobei P_{ij} nur in der Formel enthalten ist, wenn $i < j$ ist. Die Variablen sind über dem Definitionsbereich $D = D_1 \times ... \times D_n$ definiert.

Diese Prädikate entsprechen Kanten im assoziierten Graphen, die auf Grund der Bidirektionalität von Prädikaten ungerichtet sind. Ein Knoten i in diesem Graphen besteht aus dem Eintrag zu i und der Variablen v_i assoziierten Domäne.

[2] Dies ist z.B. bei nichtlinearen Constraints der Fall, in denen von n Parametern weniger als $n-1$ bekannt sind. In solchen Fällen ist das Constraint nicht weiter reduzierbar und wird in Systemen wie CLP($\mathcal{R}$) und *CHIP* mittels einem *Delay*-Mechanismus solange nicht betrachtet, bis nur noch ein Parameter unbekannt ist. Kann das Constraint während des Programmlaufs nicht gelöst werden, so wird es als Antwort ausgegeben
Beispiel aus [Lassez 87]:

```
mortgage (P, Time, I, B, MP) :-
    Time <= 1,
    B + MP = P * (1 + I).
mortgage (P, Time, i, B, MP) :-
    1 < Time,
    mortgage (P * (1 + I) - MP, Time - 1, I, B, MP).

?-Time=5, I=0.1, B=0, mortgage(P, Time, I, B, MP).

Antwort: MP = 0.263797 * P
```

Ein Knoten i heißt *konsistent*, wenn:

$$(\forall x)[x \in D_i] \subset P_i(x) \text{ gilt.}$$

Entsprechend ist eine *Kante arc(i,j) konsistent*, wenn:

$$(\forall x)[x \in D_i] \supset (\exists y)(Y \in D_j) \wedge P_{ij}(x,y)$$

gilt.

Dies wird erreicht, wenn in D_i alle Elemente entfernt werden, die in D_j mittles P_{ij} keine korrespondierenden Elemente haben.

Die folgende Operation hält die Domäne D_i , relativ zum Prädikat P_{ij} , konsistent:

$$D_i \leftarrow D_i \cap \{x \mid (\exists y)(y \in D_j) \wedge P_{ij}(x,y)\}.$$

Ein Netzwerk heißt *knoten-* und *kantenkonsistent*, wenn alle Knoten und Kanten konsistent sind. Die Knotenkonsistenz ist durch einfache Auswertung der Operation zu erhalten. Die Kantenkonsistenz kann durch einer Prozedur, welche einen Fixpunkt sucht erreicht werden, oder durch eine verfeinerte Propagierungstechnik. Eine solche sollte nur die durch die Änderung betroffenen Kanten neu auswerten. Dies kann in einem Durchlauf erfolgen, solange das Netz keine Zyklen enthält.

Eine Generalisierung des Konsistenz-Konzeptes auf p-näre Relationen stellt die folgende Definition dar ($1 \leq p, k \leq n$):

Definition 2.1 *Gegeben sei eine Belegung von k-1 Variablen, die alle Constraints unter diesen Variablen erfüllt. Dann ist ein Netzwerk k- konsistent, wenn es möglich ist eine Belegung für die k-te Variable zu finden, so daß alle Constraints unter den k Variablen erfüllt sind.*

Unter den möglichen Ansätzen die sich hieraus entwickelt haben, sind im wesentlichen die der *Label Inference* und der *Value Inference* zu nennen (vgl. [van Hentenryck 89]), die sich in ihren Domänen unterscheiden. Im ersten Ansatz werden endliche, diskrete Mengen und im zweiten feste, einzelne Werte aus einer beliebigen, d.h. auch kompakten Menge wie den reellen Zahlen verwendet.

In Abhängigkeit von der Darstellungsform der Wertemengen werden Constraints vielfach als extensional, prädikativ oder konstruktiv bezeichnet. Wobei im extensionalen Fall die Wertemengen in ausgewerteter und in den anderen Fällen in nicht ausgewerteter Form vorliegen.

2.2.2 Constraint-Systeme

Der Begriff des Constraint-Formalismus ist in der Informatik nicht eindeutig festgelegt. Von daher orientiere ich mich im folgenden an den Notationen, wie etwa in *den Projekten von A.Borning (vgl. [Borning 79, Borning 81, Borning & Duisberg* 86, Freeman-Benson et al. 90, Borning et al. 87, Maloney et al. 89, B.N.Freeman-Benson et al. 88, Borning et al. 89, B.N.Freeman-Benson & Wilson 90]) und in CHIP (vgl. [Dincbas et al. 88, van Hentenryck 89]) verwendet werden.

- CONSTRAINTS

  Eine der ersten Anwendungen von Constraints in einem größeren interaktiven System ist CONSTRAINTS von Sussman und Steele (vgl. [Sussman & Steele 80]). Hierbei sind Constraints speziell auf das Problem zur Repräsentation und Verarbeitung von elektrotechnischen Gesetzen in elektrischen Schaltkreisen angewandt worden. CONSTRAINTS arbeitet dabei über der Domäne der festen reelen Zahlen und stellt als Basis-Constraints einfache arithmetische Operationen zur Verfügung. Die Propagierungskomponente, zuzüglich einer Komponente zur symbolischen Manipulation von algebraischen Ausdrücken, leitet die daraus entstehenden Ergebnisse an das Netz weiter und Konsequenzen ab. Sussman und Steele verwenden den Ausdruck von *hierarchischen Constraint-Netzwerken* um Aggregationen beschreiben, die Teilstrukturen zu einem gesamten *Gebilde* verbinden. Der Constraint-Formalismus ist durch das Konzept der Dekomposition stark bestimmt, da besonders in der Anwendungsdomäne *CAD* die schrittweise Verfeinerung eine wesentliche Rolle spielt. Dazu wird ein *Gebilde* solange in Teilstrukturen zerlegt, bis einfache Formen von Constraints angewandt werden können. Entsprechend dem *Superpositionsprinzip* entspricht die Zusammenfassung aller Teil-Constraint-Netze dem Constraint-Netz des ganzen *Gebildes.* In CONSTRAINTS sind noch keine Mechanismen vorgesehen, die den Suchraum einschränken.

- CONSAT

  Einen Versuch das Constraint-Konzept als allgemeines Berechnungsmodell zu verwenden stellt CONSAT dar (vgl. [Güsgen 87] und [Güsgen & Hertzberg 87]). Dieses System baut wesentlich auf den Methoden von CONSTRAINTS auf, die um das Konzept der endlichen Domänen erweitert wurden. In CONSAT sind formal eine Reihe von verschiedenen Constraint-Ansätzen integriert worden. So können extensionale sowie intensionale Relationen und konstruktive oder gefilterte Constraints verarbeitet werden. Es sind u.a. auch unendliche Domänen zugelassen, was eine unendliche Propagierung mit sich bringt.

  Die Allgemeinheit wiederum ist dafür verantwortlich, daß CONSAT in seiner Effizienz nicht das leistet, was für das spezielle Anwendungsgebiet des Layouts gefordert wird. Die Propagierungstechnik, die dabei verwandt wird, ist wie in den meisten Constraint-Systemen durch *lokale Propagierung* gekennzeichnet. Sie gewährleistet lokale Konsistenz, die durch Erweiterungen in einigen Problembereichen, auf Grund der Verwendung einer Rücksetzprozedur, zu globaler Konsistenz ausgeweitet werden kann.

Die Verarbeitung von endlichen Domänen ist eine Stärke, jedoch ist die Verarbeitung relativ einfach konstruiert. Effizienzsteigernde Techniken, wie das Beschneiden des Suchraums sind in CONSAT nicht vorgesehen, so daß hierdurch starke Einbußen in der Laufzeit des Systems entstehen.

CONSAT ist in das hybride Expertensystemtool BABYLON eingebunden, in dem es als Repäsentations- und Verarbeitungsformalismus dient (vgl. [Christaller et al. 89]).

- Techniken des DeltaBlue-Algorithmus

  Der *DeltaBlue*-Algorithmus ist eine Instanz von *DeltaStar* (vgl. [B.N.Freeman-Benson & Wilson 90]) einer Klasse von inkrementellen Constraint-Solvern. *DeltaStar* ist eine Verallgemeinerung einer Reihe von Constraint-Solver, die mit dem Begriff der *Spektrum*-Algorithmen bezeichnet werden. Zu diesen Algorithmen zählen die folgenden Typen von Constraint-Solvern:

Typ	**Eigenschaften**
Red	ein allgemeiner langsamer Solver
Orange	basierend auf dem Simplex-Algorithmus
Yellow	langsamer Relaxation verwendender Solver
Green	Solver für endliche Wertemengen
Blue	schneller Solver ohne Zykelverarbeitung
DeltaBlue	inkrementelle Version von Blue

  Ein wesentlicher Aspekt der Constraint-Solver ist die inkrementelle Arbeitweise. Es ist für eine effiziente Verarbeitung von Constraints notwendig nur dort Berechnungen im Constraint-Netz auszuführen, wo Änderungen auftreten. Desweiteren leistet *DeltaStar* die Verarbeitung verschieden gewichteter Constraints. Constraints werden hierzu in die Klassen der *unbedingten* und *optionalen* Art unterteilt, wobei die erste die bisherigen *flachen* Constraints[3] enthält und die zweite solche, die nach Möglichkeit erfüllt werden sollen. Diese Eigenschaft ist besonders dann wichtig, wenn mittels Constraints allgemeingültige und speziellere Eigenschaften repräsentiert werden sollen. In anderen Constraint-Systemen werden bisher nur bedingte Constraints verwendet, die der klassischen, zweiwertigen Logik entsprechen. Dabei ist ein Constraint entweder erfüllbar oder nicht. Im Fehlerfall bricht der gesamte Prozeß ab. Im DeltaBlue-Algorithmus hingegen wird zwischen solchen unterschieden, die in der bekannten *unbedingten* Form gelten müssen, und solchen, die nur dann gelten sollen, wenn die Constraint-Lösungskomponente dies zuläßt. Kann ein optionales Constraint nicht erfüllt werden, so wird dieses *blockiert*, aber der

[3] Flach bedeutet in diesem Zusammenhang, daß nur Constraints der unbedingten Art verwendet werden. Alle Constraints haben dabei die gleiche Wichtung und der Verarbeitungsprozeß bricht ab, wenn eines dieser Constraints nicht erfüllt werden kann.

Lösungsprozeß wird nicht mit einer Fehlermeldung abgebrochen, sondern arbeitet die verbleibenden Constraints weiter ab. Nur für den Fall, daß ein *unbedingtes* Constraint nicht erfüllt werden kann, verhält sich der DeltaBlue-Algorithmus wie herkömmliche Constraint-Systeme. Somit erhält man eine Hierarchie in einem Constraint-Netz. Die Klassifizierung der Constraints geschieht mittels Wichtungen, wobei das stärkste Gewicht die Klasse der *unbedingten* Constraints bestimmt.

Der *DeltaBlue*-Algorithmus ist ein Vertreter der Wertinferenz-Klasse, der den Variablen feste Werte über einer beliebigen, d.h. auch kompakten Domäne zuordnet. Für Borning et.al. ist ein Constraint eine Relation über einer Domäne D. Die Domäne D bestimmt die Constraint Prädikatensymbole Π_D der Sprache, welche auch die Gleichheit enthalten müssen. Ein Constraint ist somit ein Ausdruck der Form :

$$p(t_1, ..., t_n),$$

wobei p ein n-stelliges Symbol in Π_D und t_i ein Term ist.

Ein *gewichtetes* Constraint ist ein Constraint mit einer Wichtung, geschrieben sc, wobei s eine Wichtung und c ein Constraint ist. Wichtungen werden üblicherweise benannt und haben als Werte ganze Zahlen von 0 bis n, wobei der Wert 0 den Constraints vorbehalten ist, die unbedingt erfüllt werden müssen.

Eine Constraint-Hierarchie ist eine Multimenge von gewichteten Constraints. Sei eine Constraint-Hierarchie H gegeben, so beschreibt H_0 die unbedingten Constraints in H, wobei die Wichtungen entfernt sind. In der gleichen Weise sind $H_1, H_2, ..., H_n$ für die Ebenen $1, 2, \ldots, n$ definiert. Desweiteren gilt $H_k = \emptyset$ *für alle* $k > n$.

Eine Lösung einer Constraint-Hierarchie H besteht aus einer Belegung der freien Variablen in H, d.h., es existiert eine Funktion, die die freien Variablen in H mit Elementen aus dem Wertebereich D unifiziert.

Sei die Menge aller Lösungen wie folgt induktiv definiert :

$$\begin{aligned} S_0 &= \{\theta \mid \forall c \in H_0\ c\theta \textit{ ist erfüllt }\} \\ S &= \{\theta \mid \theta \epsilon S_0 \wedge \forall \sigma \epsilon S_0\ \neg \textit{better}(\sigma, \theta, H)\}, \end{aligned}$$

wobei θ eine Belegung von c ist.

Es gibt mehrere Kandidaten für die Vergleichsfunktion *better*. Es muß gegeben sein, daß die Funktion irreflexiv und transitiv ist. Im allgemeinen stellt die Funktion *better* keine totale Ordnung dar. Zur Definition einer Vergleichungsfunktion wird eine Fehlerfunktion $e(c\theta)$ benötigt, die gleich 0 ist, wenn das Constraint c mit der Belegung θ erfüllt ist, und 1 wenn dies nicht der Fall ist. Somit läßt sich die Vergleichsfunktion *locally-better* wie folgt definieren (vgl. [B.N.Freeman-Benson & Wilson 90]):

Definition 2.2 *Eine Belegung θ ist locally-better als eine andere Belegung σ, wenn für alle Constraints von Ebene 1 bis $k-1$ der Fehler nach Anwendung von θ gleich dem von σ ist und ab Ebene k der Fehler für wenigstens ein Constraint kleiner und für alle anderen kleiner oder gleich dem von σ ist.*

$$
\begin{aligned}
&locally - better(\theta, \sigma, H) \equiv \\
&\quad \exists k > 0 \text{ so daß gilt} \\
&\qquad \forall i \in 1 \ldots k-1 \; \forall p \in H_i \; e(p\theta) = e(p\sigma) \\
&\qquad \wedge \; \exists q \in H_k \; e(q\theta) < e(q\sigma) \\
&\qquad \wedge \; \forall r \in H_k \; e(r\theta) \leq e(r\sigma)
\end{aligned}
$$

Die Semantik der Constraints werden durch die Constraint-*Methoden* bestimmt. Eine Constraint-*Methode* erfüllt ein Constraint, falls sie ausgeführt wird. Jede Methode bestimmt den Wert von einer oder mehreren Variablen, welche in der Outputmenge liegen, in Abhängigkeit von den Variablen, die in der Inputmenge liegen. Für jede mögliche Kombination von Input- und Output-Variablen, die gebraucht werden, muß eine Methode definiert sein. Eine Methode heißt *blockiert*, falls eine der Output-Variablen bereits bestimmt ist. Ein Constraint heißt *blockiert*, falls alle seine Methoden blockiert sind. Constraint-Modelle, die wie im Ansatz von Borning beschrieben werden, gehören zur Klasse, die mit konstruktiven Constraints arbeiten. Der Wert einer jeden Constraint-Variablen ist genau durch einen Wert definiert.

- Techniken in *CHIP*

  CHIP[4] (vgl. [Dincbas et al. 88, van Hentenryck 89]) ist eine neue Logiksprache in der Aspekte der Logik-Programmierung mit effizienten Techniken des Constraint-Lösens verknüpft werden. Es unterscheidet sich von herkömmlichen Logiksprachen durch die Berechnungsbereiche sowie verbesserte Suchprozeduren. Als Berechnungsbereiche sind in *CHIP* folgende integriert:

  - Auf endliche Domänen (finite domains) beschränkte Terme
  - Boolesche Ausdrücke
  - Lineare rationale Terme

  Für alle drei Bereiche werden verschiedene Algorithmen verwendet. Für Terme über endlichen Domänen werden Konsistenz-Test-Verfahren, für Boolesche Terme Gleichungs-Lösungsverfahren und für rationale Terme Simplex-ähnliche Verfahren verwendet. Obwohl CHIP wie CONSAT nicht für eine spezielle Anwendungsdomäne entwickelt wurde, ist großer Wert auf die Effizienz der

[4] Constraint Handling In Prolog

verwendeten Techniken gelegt worden. Die Sprache genügt damit nicht nur akademischen Zwecken, sondern erfüllt bezogen auf das Anwendungsgebiet von Logiksprachen alle Vorausetzungen, um auf eine Reihe von verschiedenartigen Problemen angewandt zu werden.

Eine für die automatische Layout-Erstellung interessante Technik ist die der *Finite Domains.* In diesem Anwendungsbereich treten häufig Beziehungen zwischen Objekten auf, die sich durch Intervalle über natürlichen Zahlen ausdrücken lassen.

Definition 2.3 *Eine Domäne ist eine nichtleere, endliche Menge von Konstanten.*

Eine grundlegende Eigenschaft in *CHIP* bei der Verarbeitung von diskreten kombinatorischen Problemen ist die Möglichkeit auf Domämen-Variablen, d.h. Variablen, die über endlichen Domänen definiert sind, zu arbeiten. *CHIP* unterscheidet zwischen Variablen, die über Konstanten, und über endlichen Mengen natürlicher Zahlen definiert sind. Weiterhin können arithmetische Terme über endlichen Domänen verarbeiten werden.

Eine ganze Reihe von vordefinierten Constraints über Domänen-Variablen werden von *CHIP* angeboten. Dabei handelt es sich nicht nur um arithmetische, sondern auch um symbolische und benutzerdefinierte Constraints. Arithmetische Constraints beispielsweise sind für alle Terme X und Y die folgenden *wohlgeformten* Constraints:

$$X > Y, X >= Y, X < Y, X =< Y, X = Y, X \neq X$$

Symbolische Constraints stellen sonstige, nicht-arithmetische Beziehungen zwischen Domänen-Variablen dar. Beispielsweise ist $element(Nb, List, Var)$ ein Constraint, welches wahr ist, wenn Var das Nb-te Element der Liste $List$ ist. Symbolische Constraints erlauben häufig natürliche Probleme auszudrücken und diese effizient zu lösen. Neben diesen beiden Constraints-Arten kann der Benutzer selbstdefinierte Constraints dem System hinzugeben.

Um diese Constraints über Finite Domains lösen zu können, werden in *CHIP* Konsistenztechniken angewendet. Diese schränken die Belegungen und somit den Such- und Lösungsraum mit jedem Constraint weiter ein. Sie sind dabei meist nicht in der Lage Constraints zu lösen, so daß das Lösen diskreter kombinatorischer Probleme aus der Iteration der folgenden beiden Schritte

besteht:

1. Propagierung der Constraints so weit wie möglich
2. Treffen einer Auswahl

Diese Schleife wird solange durchlaufen, bis eine Lösung gefunden worden ist. Um Konsistenztechniken in Logik-Programmierung zu integrieren, haben sich die folgenden drei Inferenzregeln herausgestellt (vgl. [van Hentenryck 89]):

- Forward Checking Inference Rule (FCIR)
- Looking Ahead Inference Rule (LAIR)
- Partial Looking Ahead Inference Rule (PLAIR)

Jede dieser Inferenzregeln stellt eine Art dar, den Suchraum zu beschneiden. Alle primitiven Constraints über Domänen-Variablen in *CHIP* können als effiziente Spezialisierungen dieser Inferenz-Regeln betrachtet werden.

Eine interessante Technik, auf die in Kapitel 4.2 bezug genommen wird, ist die des *Forward Checkings.*

Definition 2.4 *Sei $p(t_1, ..., t_n)$ ein Atom. $p(t_1, ..., t_n)$ ist forward-checkable, falls*

1. *p ein Constraint ist, und*
2. *genau ein t_i existiert, welches Domänen-Variable ist, und alle anderen t_j sind instanziiert.*

Durch Anwendung von FCIR kann der Suchraum *a priori* eingeschränkt werden, da durch Anwendung dieser Regel ein Teil der möglichen Instanziierungen der Domänen-Variablen durch das Constraint herausgenommen werden.

Ist ein Constraint *forward-checkable*, so wird die zu berechnende Domänen-Variable durch die Menge aller Werte belegt, die unter diesem Constraint gelten. Ist die Belegung einer Domänen-Variablen auf diese Art bis auf genau einen Wert festgelegt, so wird die Variable mit diesem Wert instantiiert. *Forward Checking* erlaubt für den Fall, daß keine Lösung im Suchraum unterhalb der aktuellen Position existiert, dieses frühzeitig zu erkennen. Im Gegensatz zu herkömmlichen Suchtechniken, wie sie z.B. in Prolog verwandt werden, werden alle Teile des Suchbaums nicht mehr in Betracht gezogen, die die als inkonsistent bewiesene Constraint-Menge als Teilmenge enthalten. Diese Technik kann in etwa mit einer automatischen Einfügung eines *Cuts* in PROLOG verglichen werden.

2.2.3 Anwendungen constraint-basierter Ansätze im geometrischen Layout

Anwendungen von Constraint-Techniken finden sich frühzeitig in Systemen wieder, die sich mit geometrischen Layout befassen. Die meisten Systeme betrachten die interaktive Benutzung von graphischen Schnittstellen, mit denen der Benutzer neue geometrische Objekte einfügen und mit alten verbinden kann. Der Benutzer ist somit die Ursache für das Einfügen oder auch Löschen eines Constraints. Constraints werden in den bekannten Systemen als berechnende Verbindungen zwischen verschiedenen Objekten betrachtet und nur ansatzweise als Repräsentationsformalismus von graphischem Wissen.

- **Sketchpad von Sutherland**
  I.Sutherland entwickelte schon 1960 am MIT das System Sketchpad. Es erlaubt dem Benutzer die Erstellung von geometrischen Figuren durch Zusammenfügen von geometrischen Primitiven. Das Zusammenfügen solcher Primitiven erfolgt via Constraints, die die Beziehungen untereinander repräsentieren (vgl. [Sutherland 63]).

- **ThinglabI und II von Borning**
  Die Weiterentwicklung der Ideen von *Sketchpad* sind in *Thinglab I* und *II* integriert worden(vgl. [Borning 79], [Maloney et al. 89] und [B.N.Freeman-Benson et al. 88]). In *Thinglab II* wird die Idee der hierarchisierten Constraints aufgegriffen und angewandt. Die Anwendung von *Thinglab I* und *II* liegen in der Simulation von elektrischen Schaltkreisen. Durch Constraints wird beispielsweise das Ohmsche Gesetz nachgebildet (vgl. [Borning et al. 87] und [Borning et al. 89]).

- **Andere Systeme**
  Andere, zum Teil ältere Systeme als die oben genannten, die in der Erstellung von geometrischen Layout Anwendung finden, sind *Juno* von G.Nelson (vgl. [Nelson 85]), *Magritte* von J.Gosling (vgl. [Gosling 83]) und *IDEAL* von C. Van Dyk (vgl. [van Wyck 82]).

Erfahrungen aus diesen Systemen zeigen, daß nur eine effiziente Gestaltung des Constraint-Solvers eine akzeptable Laufzeit des Systems bewirkt. Von daher sind einige interessante Ansätze entwickelt worden, die nicht nur für interaktive Systeme von Bedeutung sind.

In keinem der angesprochenen Systeme ist der Aspekt der wissensbasierten Layoutgenerierung behandelt worden. Vielmehr stehen dabei einfache Beziehungen zwischen einfachen Objekten im Vordergrund (vgl. [Borning 81]), deren Zusammenwirken in einem Netz von Constraints bestimmt wird. Constraints als eine Repräsentationsform für Wissen aus dem Bereich des Graphik-Desgins zu verwenden ist im Layout-Managers von WIP erstmalig verwendet worden (vgl. [Graf 91, Graf & Maaß 91, Wahlster et al. 91a, Wahlster et al. 91b, Graf 90]).

Kapitel 3

Allgemeine Vorgehensweise beim automatischen Layout multimodaler Dokumente

Wie sehr der Mensch nach Wissenschaft verborgener Dinge ringt,
So bleibt ihm doch unzählig viel, davon er sagt: Mich düngt.
Friedrich von Logau

Many of the intractable problems have the feature
that a series of correct guesses would suffice to solve them
E. Charniak, D. McDermott, Artificial Intelligence

In diesem Kapitel wird das zu behandelnde Problem der automatischen Layoutgenerierung genauer analysiert und entsprechende Verarbeitungsformalismen diesbezüglich evaluiert. Anschließend werden Repräsentationsformen der am Prozeß beteiligten Objekte und den zugeordneten Beziehungen sowie deren Zusammenspiel vorgestellt. Abschließend wird auf die Möglichkeit der Rücknahme getroffener Entscheidungen eingegangen.

3.1 Layoutobjekte

Da die Objekte, die von den Generatoren in WIP erzeugt werden, unterschiedlicher Modalität sind, ist es notwendig diese auf eine für das Layout spezifische Repräsentationsform zu bringen, welche im weiteren mit Layoutobjekten gleichgesetzt wird. Die Vorraussetzungen, die diese Abbildung zu erfüllen hat sind:

- Abbilden der unterschiedlich-modalen Layoutobjekte auf inhaltsabstrahierende Strukturen
- Erhaltung der geometrischen Ausmaße der Layoutobjekte

Die geometrische Form der Layoutobjekte wird durch Approximierung der Ausgabeobjekte der Generatoren mittels dem minimalen, das Objekt umschreibende Rechteck erhalten (s. Abb. 3.1). Das Rechteck wird über die linke obere Ecke, die Höhe und die Breite in seiner Größe festgelegt. Diese Daten werden zusammen mit den inhaltsabhängigen Angaben über Beziehungen der Objekte zueinander in die Struktur *LOBJEKTE* abgelegt.

$$lobjekt : OBJEKTE \rightarrow LOBJEKTE,$$

mit OBJEKTE, der Menge der generierten Objekte, LOBJEKTE $\subseteq N \times N \times N \times N \times R$, mit N die Menge der natürlichen Zahlen und R die Menge aller möglichen Relationen, wobei für $o \epsilon OBJEKTE$ gilt:

$$\begin{aligned} pr_1(lobjekt(o)) &= x - koordinate(o) \\ pr_2(lobjekt(o)) &= y - koordinate(o) \\ pr_3(lobjekt(o)) &= höhe(o) \\ pr_4(lobjekt(o)) &= breite(o) \\ pr_5(lobjekt(o)) &= lok - rel(o) \\ pr_6(lobjekt(o)) &= glob - rel(o), \end{aligned}$$

mit der naheliegenden Definition der Funktionen *x-koordinate, y-koordinate, höhe* und *breite* und der Projektionsfunktion pr_i $(i \epsilon \{1,2,3,4,5,6\})$. Die Zugriffsfunktionen *lok-rel* und *glob-rel* liefern die jeweilig für das Layoutobjekt relevanten lokalen und globalen Relationen.

3.2 Beziehungen zwischen Layoutobjekten

Das für das Layout relevante Wissen läßt sich durch die Art der Beziehungen zwischen den Layoutobjekten klassifizieren. Die eine Klasse enthält lokale Beziehungen, welche Layoutobjekte zu Einheiten zusammenfassen, die sehr stark voneinander abhängen. Die andere Klasse enthält globale Beziehungen, durch die Wissen über geometrische Beziehungen zwischen Einheiten von Layoutobjekten dargestellt werden. Globale Beziehungen bestimmen dabei, im Gegensatz zu lokalen Beziehungen, nur relative Lagen der Einheiten zueinander. Sie sind im allgemeinen Strategien

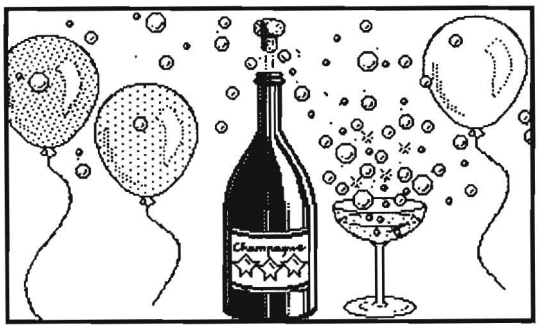

Abbildung 3.1: Minimales Rechteck um ein Graphikobjekt

zur Anordnung von Einheiten, die durch lokale Beziehungen miteinander verknüpft sind.

Beide Arten, lokale und globale Beziehungen, lassen sich durch Constraint-Formalismen repräsentieren und in effizienter Weise verarbeiten, da Constraints dem allgemeinen Begriff der Relation und somit den Beziehungen sehr stark nachempfunden sind. Neben dieser Eigenschaft, erlauben Constraints mittels Propagierungstechniken Änderungen an andere Layoutobjekte weiterzuleiten.

3.2.1 Lokale Beziehungen

Lokale Beziehungen beschreiben semantisch-pragmatische Relationen. Sie stützen sich zum Teil auf die Relationen, die durch Mann/Thompson in ihrer RST-Theorie (vgl. [Mann & Thompson 88]) vorgestellt worden sind. Sie beschreiben enge Kopplungen zwischen Objekten, die für die Aussage der Relation die zwischen den Objekten besteht, entscheidend sind. Bespielsweise können Objekte, die durch die Relation *sequence* verbunden sind, nicht in beliebiger Weise angordnet oder geteilt werden, ohne den Verlust der Aussage zu vermeiden (s. Abb. 3.2 und 3.3). Neben den Relationen der RST-Theorie werden eine Reihe weiterer, teils elementarer Relationen, wie die Verknüpfung zwischen Graphik- und Textobjekten, betrachtet. Lokale Beziehungen verknüpfen Layoutobjekte durch starke Bindungen miteinander zu *Einheiten*, die für den weiteren Verarbeitungsprozeß als atomar zu betrachten sind.

Im Beispiel (s. Abb. 3.2) wird der Arbeitsvorgang durch eine Sequenz beschrieben. Die Sequenz ist dabei aus drei Bildern aufgebaut, die durch zugehörige Texte unterstützt werden. Eine übliche Darstellungsweise einer Sequenz ist dabei verwendet worden, in der die Graphiken horizontal nebeneinander und auf gleicher Höhe angeordnet werden. Die Texte kommen direkt unterhalb der Graphiken zu stehen. Eine weitere alternative Darstellungsweise ist es Graphiken vertikal übereinander anzuordnen, wobei die Texte rechts von den Graphiken stehen. Je nach Platz oder Dokumenttyp werden diese beiden Formen gewählt. Alle anderen Präsentationsformen einer Sequenz sind zum schnellen Erfassen einer Information i.a. ungeeignet (s. Abb. 3.3).

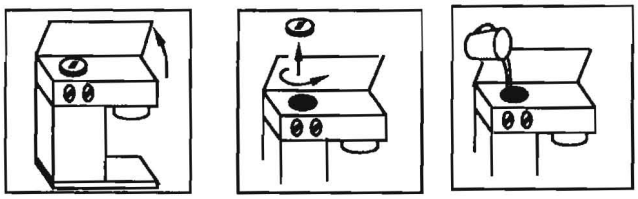

Abbildung 3.2: Illustration einer Sequenz am Beispiel des Wassereinfüllens einer Espressomaschine

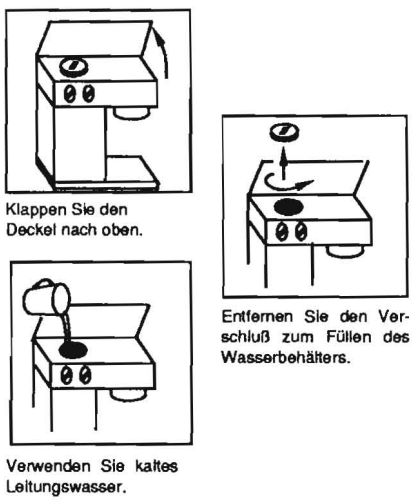

Abbildung 3.3: Schlechte Anordnung einer Sequenz

Die durch lokale Beziehungen verbunden Layoutobjekte werden zu *Einheiten* zusammengefaßt. Dazu wird, ähnlich der Vorgehensweise bei der Bestimmung der geometrischen Ausmaße von Layoutobjekten, das minimal umschreibende Rechteck berechnet, welches alle Layoutobjekte umfaßt (s. Abb. 3.4). Das Minimum der Belegung aller X- und Y-Variablen bestimmen die Belegung der linken oberen Ecke der Einheit sowie das Maximum der Höhen und die Addition der Breiten die Höhe und die Breite der Einheit.

Einheiten bilden, entsprechend den Layoutobjekten für lokale Beziehungen, die atomaren Objekte für globale Beziehungen.

3.2.2 Reformatierung

In manchen Fällen kommt man mit einigen Entscheidungen nicht zu einer Lösung des Problems. Diese muüssen zurückgenommen und durch Alternativen ersetzt werden. Dieser Vorgang wird im folgenden mit *Reformatierung* bezeichnet. Dabei werden zwei Schritte unterschieden:

1. Reformatierung einer globalen Beziehung

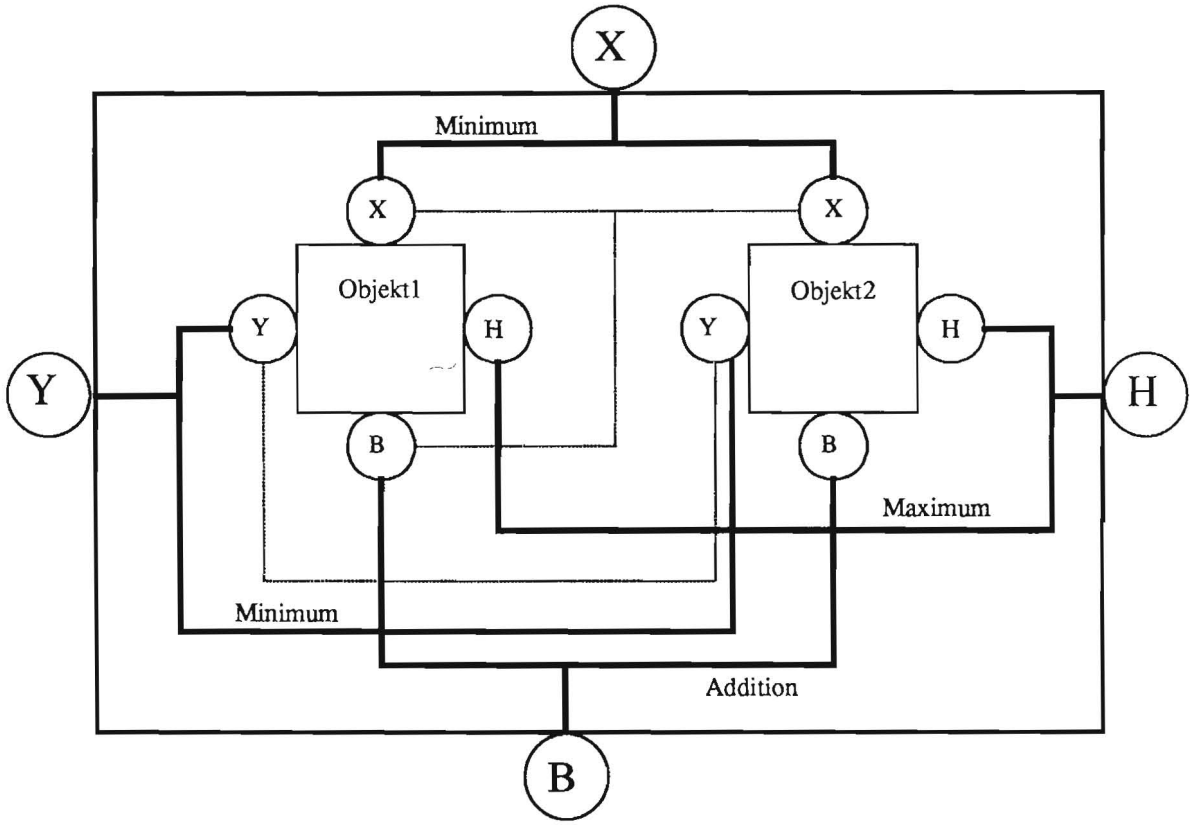

Abbildung 3.4: Aggregierung einer Einheit aus Layoutobjekten

2. Reformatierung einer lokalen Beziehung

Die Vorgehensweise, die dabei modelliert werden soll, wird von Experten durch Umordnen des gegebenen Layouts ausgeführt. Dabei versucht dieser zuerst die Struktur der lokalen Beziehungen zu erhalten und erst, wenn dies zu keinem Ergebnis führt, werden auch diese *aufgebrochen* und durch alternative Anordnungen ersetzt.

Für die Verarbeitung bedeutet dies, daß wenn ein globales Constraint nicht erfüllbar ist, so wird eine Reformatierung einer lokalen Beziehung vorgenommen. Ist es nicht möglich alle Einheiten in der gegebenen Form auf dem Dokument zu plazieren, so wird der zweite Schritt der Reformatierung ausgeführt. Hierbei wird die betroffene Einheit an die Verarbeitungskomponente der lokalen Beziehungen zur Auswahl einer anderen geometrischen Anordnung der entsprechenden Relation zurückgegeben. Durch Auswahl einer anderen, in der Preferenz niedrigeren Methode, wird das Constraint auf die Layoutobjekte der Einheit angewandt. Diese Methode bewirkt im Normalfall eine andere geometrische Gestalt der Einheit. Diese Einheit wird an die Verarbeitungskomponente globaler Beziehungen zurückgegeben, woraufhin ein neuer Versuch gestartet wird diese Einheit zu plazieren.

Das Verfahren bricht ab, wenn keine alternativen Darstellungsformen der globalen und lokalen Beziehungen zur Auswahl stehen. In diesem Fall wird eine Fehlermeldung an den Layout-Manager ausgegeben, der eine Konfliktlösungsstrategie

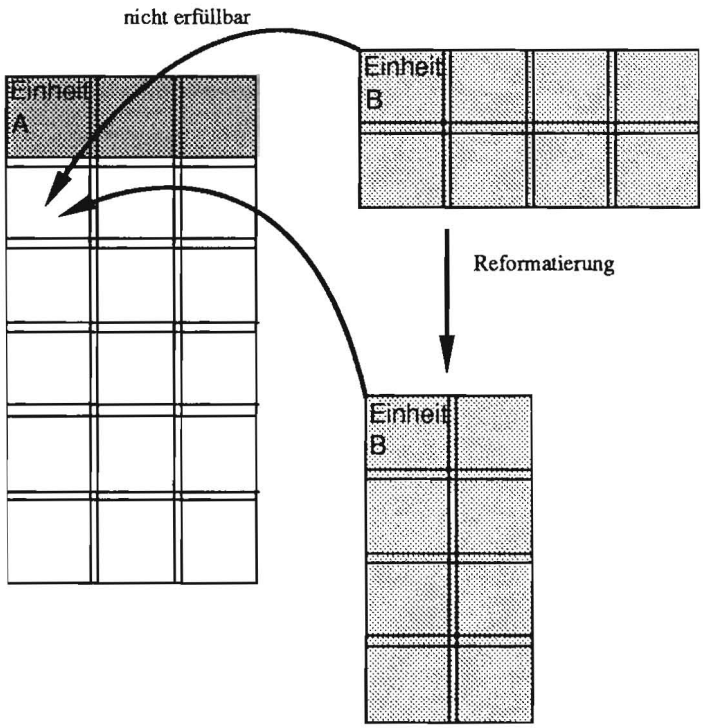

Abbildung 3.5: Reformatierung bei der Plazierung einer *Sequenz*

ausführt.

Im Beispiel (s. Abb. 3.5) hat eine *Sequenz* die alternativen horizontalen und vertikalen Darstellungsformen. Die horizontale Version ist dabei der vertikalen vorzuziehen. Vorgegeben sei eine globale Anordnungsheuristik, die die Objekte zuerst *rechtsvon* und dann *unter* der assoziierten Einheit zu verbinden sucht. Wie in der Abbildung gezeigt, kann die Einheit B weder *rechtsvon* noch *unter* Einheit A plaziert werden. Somit wird eine alternative Darstellungsweise einer *Sequenz* ausgewählt, die *unter* die Einheit A positioniert werden kann.

3.2.3 Globale Beziehungen

Zwischen den Einheiten, gibt es unterschiedliche globale Beziehungen. Teilweise sind solche globalen Beziehungen von Anfang an bekannt, wie es beispielsweise im Zeitungslayout bei Überschriften und dem *Aufmacher* der Fall ist, wobei der Aufmacher immer direkt unterhalb der Überschrift plaziert wird (s. Abb. 3.6). Zu beachten ist hierbei, daß jede dieser Einheiten aus verschiedenen Layoutobjekten bestehen kann. Z.B. kann der *Aufmacher* aus einer Reihe von Text- und Graphikblöcken bestehen.

Für manche Einheiten sind von Anfang an bestimmte Positionen auf dem Dokument vorgesehen, wie es beim *Logo* der Fall ist, welches meist rechtsoben steht. Solche Relationen sind unär in bezug auf die Einheiten und treten somit nicht in Beziehung zu anderen Einheiten. Im allgemeinen Fall sind Einheiten a priori nicht mit anderen oder einer festen Position auf dem Dokument assoziiert. Einheiten solcher

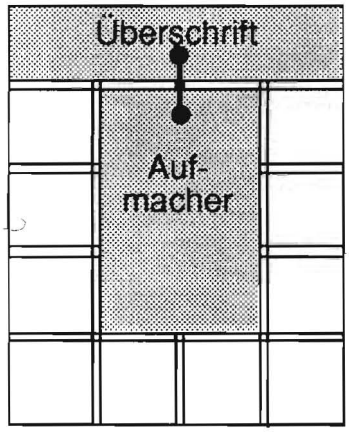

Abbildung 3.6: Vorgegebene Beziehung *unter* der Einheit des *Aufmachers* unter der *Überschrift*

Art werden entsprechend Heuristiken an bereits plazierte Einheiten durch globale Beziehungen angebunden. Die für den westeuropäischen Kulturraum natürlichste Art der Plazierungsstrategie besteht darin, die Einheiten von oben links nach unten rechts anzuordnen, da sie der normalen Leserichtung des Lesers entspricht. Globale Beziehungen verknüpfen dadurch Einheiten, die teilweise vorher nicht miteinander assoziiert sind.

Nachdem Objekte durch lokale Beziehungen zu Einheiten verbunden sind, werden diese mit Hilfe von globalen Beziehungen auf der Dokumentfläche angeordnet. Zuerst werden alle Einheiten plaziert, die fest vorgegebene Positionen einnehmen sollen. Danach werden die restlichen entsprechend ihrer Reihenfolge durch Anwendung des Plazierungsprozesses positioniert.

Globale Beziehungen verbinden Einheiten miteinander, die in keiner oder nur schwacher Beziehung zueinander stehen. Sie sind heuristischer Natur, die in Abhängigkeit vom Dokumenttyp und den Objekten variieren können. Im einfachsten Fall verwendet man eine Heuristik, die Einheiten nach der Form zuerst *nebeneinander*, sonst *untereinander* anordnet. In anderen Fällen kann eine Anordnung nach der Form zuerst *untereinander*, dann *übereinander* sinnvoll sein.

Die sequentielle Anordnung der Einheiten bedingt, daß für jede Einheit nur dann ein fester Platz auf dem Raster bestimmt werden kann, wenn zum einen eine feste Ortsangabe für eine Einheit vorgegeben wird, wie das bei Logos der Fall ist, oder zum anderen die Dokumentfläche nur noch einen bestimmten Platz zur Anordnung zur Verfügung stellt. Ansonsten muß der Einheit der gesamte Freiraum auf der Dokumentfläche offengehalten werden.

Die erste auf der Dokumentfläche zu plazierende Einheit kann an keine Einheit durch eine globale Beziehung angebunden werden. Dazu sind um die Dokumentfläche *virtuelle* Einheiten angeordnet, an die die erste Einheit mittels dem entsprechenden Constraint der Heuristik verbunden werden kann (s. Abb. 3.7). Nachdem die erste Einheit, für die die gesamte Dokumentfläche zur Verfügung steht in das Constraint-Netz eingebunden ist, werden die virtuellen Einheiten blockiert, so daß sie nicht mehr mit neuen Einheiten verbunden werden können.

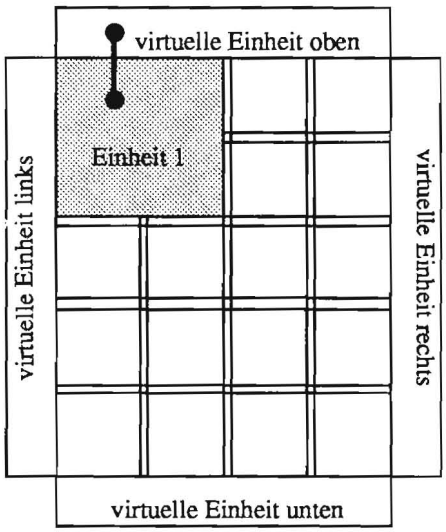

Abbildung 3.7: Beziehung zwischen der ersten Einheit einer Dokumentseite und der oberen virtuellen Einheit

Eine Einheit ist auf der Dokumentfläche, entsprechend den für sie geltenden Constraints, frei plazierbar, bis die Belegung der X- und Y-Domänenvariabale auf einen Wert eingeschränkt sind. In diesem Fall beansprucht die Einheit einen festen Bereich der Dokumentfläche, der daraufhin gesperrt wird.

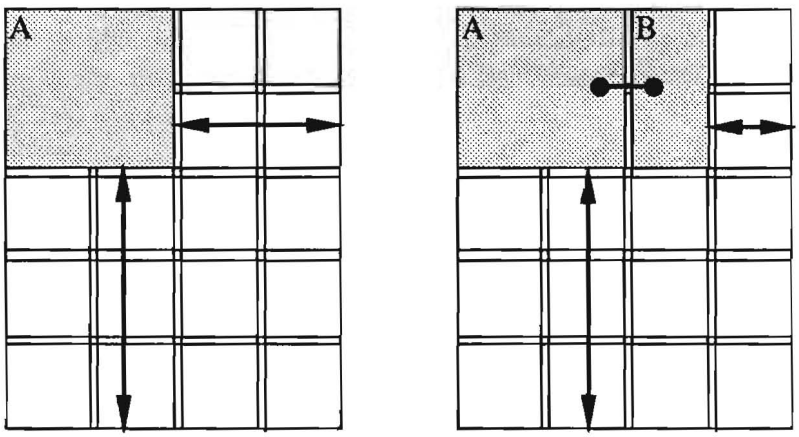

Abbildung 3.8: Einschränkung des Plazierungsbereichs einer Einheit A durch eine andere Einheit B

Um eine Verbindung zwischen primär nichtzusammenhängenden Einheiten herstellen zu können, werden die angesprochenen Heuristiken verwandt. Über diese wird eine zu plazierende Einheit mit einer bereits plazierten Einheit verbunden. Die Auswahl einer Einheit als Konnektor für eine neue Einheit wird durch Kontrollstrukturen gesteuert, die für alle vier Seiten jeweils eine Liste unterhält, an deren entsprechende Seite eine neue Einheit mittels einem globalen Constraint angebunden werden kann. Eine Seite einer Einheit gilt dabei als belegt, wenn schon eine andere Einheit an dieser Seite angebunden ist. Ist eine Einheit als potentieller Konnektor ausgewählt worden, so wird das entsprechende globale Constraint über diesen Einheiten ausgewertet, welches, ausgehend von den Werten des Konnektors, die Werte

für die zu plazierende Einheit berechnet. Durch das Einfügen der neuen Einheit in das Constraint-Netz werden Rückwirkungen an die anderen Einheiten berechnet und durch das Netz propagiert.

3.3 Konzeptuelle Sicht

Durch die Verwendung eines rasterbasierten Ansatzes wird zum einen das funktionelle und ästethische Layout unterstützt und zum anderen der Suchraum für die Position der Elemente stark verkleinert.

In den Layoutprozeß gehen semantisch-pragmatische Beziehungen, Generierungsparameter, wie z.B. der Dokumenttyp, und die zu plazierenden Layoutobjekte ein. Zur Verarbeitung dieser Eingabedaten sind Vorgehensweisen der Illustrationtechnik in Form von sog. *graphischen Wissen* (vgl. [Graf & Maaß 91]) repräsentiert. Dem graphischen Wissen liegen dabei exakte, aber auch heuristische Techniken zugrunde. Die für den Layoutprozeß relevanten Beziehungen lassen sich in zwei Arten klassifizieren, die sich zum einen lokal auf Layoutobjekte und zum anderen global auf die Vorgehensweise des Prozesses beziehen. Lokale Beziehungen fassen ein oder mehrere Layoutobjekte zu einer Einheit zusammen, welche für globale Beziehungen die kleinsten Einheiten bilden. Beide Arten der Beziehungen beschreiben topologische Anordnungen von Layoutobjekten, die dadurch eine bestimmte Wirkung beim Leser erzeugen sollen. Die Unterteilung in zwei Klassen entspricht der Vorgehensweise eines Layouters, der zusammengehörige Objekte möglichst nach von ihm bevorzugten Strukturen anordnet und diese auf der Dokumentfläche mit anderen zu verbinden sucht. Dieser Prozeß läßt sich in ähnlicher Weise durch iteratives Anwenden von lokalen und globalen Beziehungen modellieren. Entsprechend einem Experten, werden erst Einheiten durch lokale Beziehungen hergestellt, die anschließend durch globale Beziehungen miteinander verknüpft werden. Kommt es dabei zu Konflikten, so werden entweder globale oder lokale Beziehungen zurückgenommen, um sie in einer alternativen Weise anzuwenden.

.

Im folgenden werden die in Kapitel 2 beschriebenen Constraint-Solver auf ihre Anwendung für die Repräsentation und Verarbeitung im automatischen Layout untersucht.

3.4 Evaluierung von Constraint-Systemen bezüglich Layout

Folgende Merkmale muß der Constraint-Formalismus leisten, um die gesteckten Ziele der constraint-basierten Plazierung, wie sie in WIP gefordert sind (vgl. [Graf & Maaß 91]), zu erreichen:

- Effizienz
- Deklarativität
- Leichte Modifizierbarkeit
- Hierarchisierbarkeit von Constraints

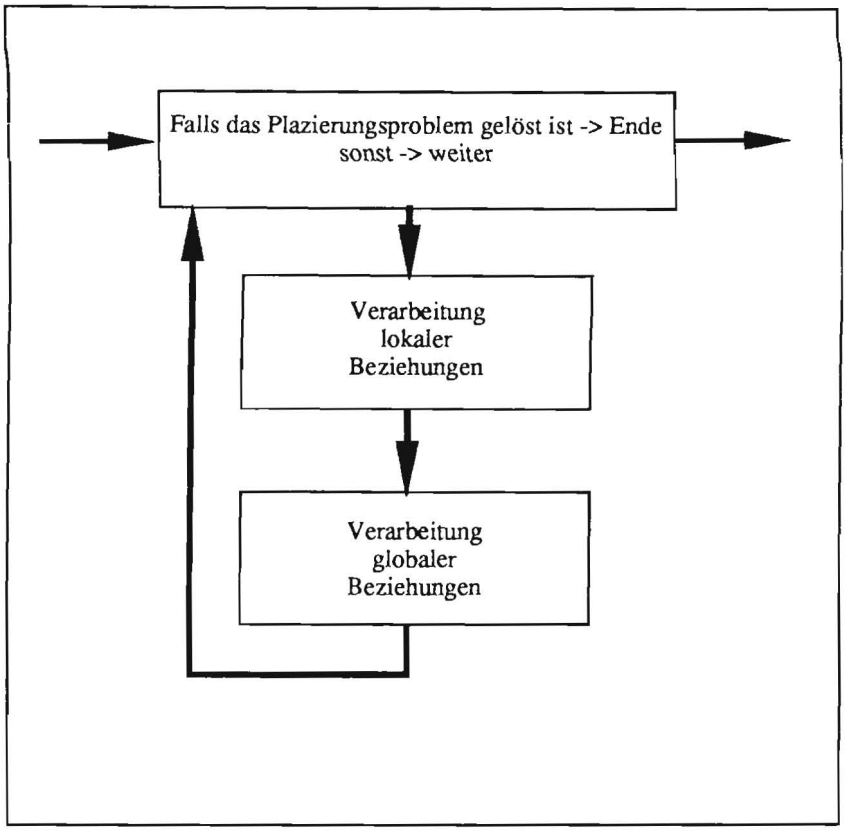

Abbildung 3.9: Allgemeine Vorgehensweise des Plazierungsproblems

- Verarbeitung von Gleichungen und Ungleichungen
- Verarbeitung von Intervallen
- Repräsentation und Verarbeitung von dynamischen Beziehungen

Im folgenden werden die einzelnen Systeme, die für den Gebrauch in WIP relevant sind, untersucht. Die Auswahl der Constraint Solver beschränkt sich auf die, deren Arbeitsweise bekannt ist oder die als Implementierung vorliegen.

3.4.1 CONSAT

Dieser Constraint-Solver liegt in einer implementierten Form vor und konnte getestet werden. Ein wesentliches Problem ist durch die Konzeption von CONSAT als allgemeiner Constraint-Solver bedingt. Die Evaluierung von CONSAT bezüglich einer Verwendung in der Domäne der automatischen Layoutgenerierung zeigte, daß CONSAT nicht die dort geforderten speziellen Ansprüchen erfüllen kann .

Wichtungen, d.h. Constraints unterschiedlicher Stärke, sind in CONSAT nicht vorgesehen. Die Verarbeitung von diskreten Intervallen ist eine Stärke, jedoch ist die Verarbeitung relativ einfach konstruiert und dadurch recht ineffizient. Dadurch ist CONSAT für die speziellen Bedürfnisse der automatischen Layout-Generierung zu umständlich und somit zu langsam.

3.4.2 DeltaBlue

Eine hervorragende Erweiterung bietet der Constraint-Formalismus von Borning et. al. (vgl. [Borning et al. 87, B.N.Freeman-Benson et al. 88, Borning et al. 89]), der zum Aufbau von herkömmlichen flachen Constraint-Netzen (vgl. [Sussman & Steele 80] und [Güsgen 87]) Hierarchiesierungsmöglichkeiten zwischen Constraints einführte (vgl. Kapitel 2).

Optionale Constraints, die gültig sind, solange nichts dagegenspricht finden z.B. Verwendung in der Festlegung des Ursprungs einer Einheit von Layoutobjekten. A priori wird dieser auf die Koordinaten (0,0) gesetzt, bis der Verarbeitungsprozeß andere bestimmt hat.

Dieses Beispiel zeigt die Notwendigkeit von Wichtungen von Constraints, weil hierbei die Constraints, welche die Koordinaten auf (0,0) festlegen, durch neue überschieben werden müssen.

Ein weiterer sehr wichtiger Punkt ist die Effizienz des Constraint-Solvers. Frühe Systeme zeichnen sich durch hohe Laufzeiten aus, die eine Verwendung in interaktiven Systemen wie ThingLab I und II ausschlossen (vgl. [Borning 79] und [Maloney et al. 89]). Stattdessen sind effiziente Solver entwickelt worden, wie sie dann letztendlich in Thinglab I und II verwendet wurden.

Die Wichtungen der Constraints erlauben den Aufbau von Hierachien unter den Constraints und desweiteren die Propagierung effizient zu gestalten. In *ThinglabII* wird der DeltaBlue-Algorithmus verwandt, um geometrische Objekte in Beziehung zu setzen und zu verwalten. Eine Stärke von *DeltaBlue* ist die rücksetzungsfreie Verarbeitung von Constraints. Dies ist dadurch möglich, daß alle Informationen fürdas Einfügen bzw. Löschen eines Constraints bei den Variablen gespeichert werden. Daneben ist es möglich die aufgebaute Constraint-Hierarchie aus dem Constraint-Solver zu entnehmen und für erneute Aufrufe zu verwenden. In *ThinglabII* wird der DeltaBlue-Algorithmus verwandt, um geometrische Objekte in Beziehung zu setzen und zu verwalten.

3.4.3 CHIP

Die *CSP*-Sprache *CHIP* (vgl. [Dincbas et al. 88, van Hentenryck 89]) ist mittlerweile, wie PROLOG III (vgl. [Colmerauer 90]), ein kommerzielles Produkt geworden, welches zu Beginn dieser Arbeit nicht zur Verfügung stand. Von den in *CHIP* verwendeten Berechnungsbereichen *boolesche Ausdrücke*, *lineare rationale Ausdrücke* und *endliche Domänen* ist besonders das letzte Konzept für die automatische Layoutgenerierung interessant. Betrachet man die Inferenzregeln, die in *CHIP* verwendet werden können, so stellt sich heraus, daß die für das Layout die *Forward-Checking-Inferenz-Regel (FCIR)* die beste ist. Die *Look-Ahead-Inferenz-Regel (LAIR)* scheidet für eine inkrementelle Verwendung aus, da sie voraussetzt, daß alle Constraints bekannt sind. Im Gegensatz dazu betrachtet *FCIR* immer nur die Constraints, die sich aktuell im Constraint-Netz befinden. Kommt es bei diesen zu einem Konflikt,

der eine Lösung dieses Constraint-Netzes ausschließt, kann die Suche in diese Richtung abgebrochen werden. Dadurch ermöglicht diese Konsistenztechnik eine starke Einschränkung des Suchraums.

Kapitel 4

Constraint-Solver Modell CLAY

Das Universum besteht aus Materie in Bewegung
Descartes
Die Natur wird von universellen Gesetzen regiert
Newton
Wissen ist Macht
Bacon

Es wird im folgenden das System CLAY (*c*onstraintbasiertes *Lay*out) beschrieben, in dem das Layoutproblem als Constraint-Satisfaction-Problem *(CSP)* formalisiert und mittels Konsistenztechniken gelöst wird (vgl. [Graf & Maaß 91]). Dabei wird gezeigt, daß sich Constraint-Formalismen sehr gut sowohl zur deklarativen Wissensrepäsentation, als auch als Berechnungsmodell und effiziente Kontrollstrategie eignen. Wir verwenden Techniken des DeltaBlue-Algorithmus (vgl. [Freeman-Benson et al. 90]) und aus CHIP (vgl. [Dincbas et al. 88]). Dieser modellbasierte Ansatz ermöglicht es, sich im Gegensatz zu bisherigen regelbasierten Ansätzen, durch die Einbeziehung von strukturellem Wissen über den Layoutprozeß, auf einige Heuristiken in Form von kompiliertem Wissen zu beschränken. Dadurch kann graphisches Wissen unabhängig von domänenspezifischen Designstrategien verarbeitet werden.

Die Strukturierung des Layout-Wissens durch lokale und globale Beziehungen erfordert aus Effizienzgründen zwei verschiedene Ansätze zur Verarbeitung. Lokale und globale Beziehungen lassen sich in deklarativer Weise durch konstruktive Constraints über dem Bereich der natürlichen Zahlen bestimmen.

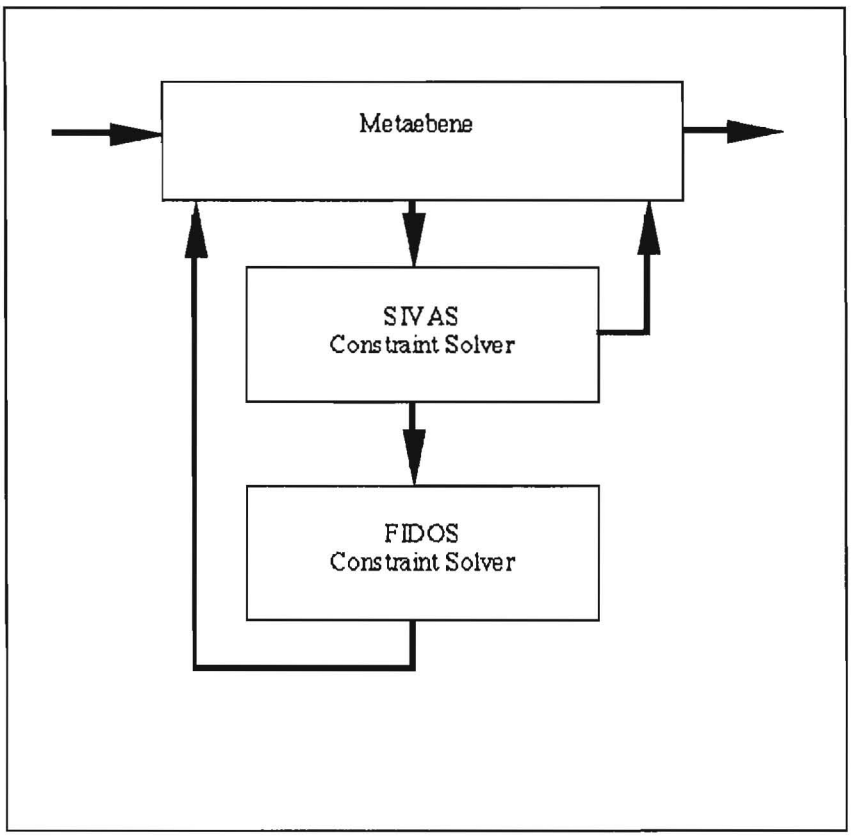

Abbildung 4.1: Kontrollfluß zwischen den Systemkomponenten von CLAY

Das der Verarbeitung zugrundeliegende Constraint-Solver-Modell CLAY ist durch die Kopplung zweier dedizierter Constraint-Solver charakterisiert (s. Abb. 4.1), deren zugrundeliegende Formalismen im folgenden genauer vorgestellt werden. Durch das Zusammenspiel beider Solver wird die Auswertung und Erhaltung aller lokalen und globalen Beziehungen erreicht und eine konsistente Lösung bestimmt.

4.1 Formale Arbeitsweise von SIVAS

SIVAS (**Si**ngle-**Va**lue **S**olver) ist ein inkrementeller Constraint-Solver, der Hierarchien verarbeiten kann, wozu er den in Kapitel 2 vorgestellten DeltaBlue-Algorithmus verwendet (vgl. [B.N.Freeman-Benson et al. 88, Freeman-Benson et al. 90]).

Von SIVAS werden lokale Beziehungen verarbeitet, die topologische Anordnungen von Layoutobjekten zueinander bestimmen. Über eine Eingabeliste erhält SIVAS alle diese Beziehungen mit den dazugehörigen Layoutobjekten. Die Namen dieser Beziehungen bestimmen eindeutig ein Constraint, welches mit den Werten der Layoutobjekte instanziiert wird. Die X- und Y-Koordinate, Höhe und Breite werden entsprechend dem Constraint an die übrigen Layoutobjekte weiterpropagiert.

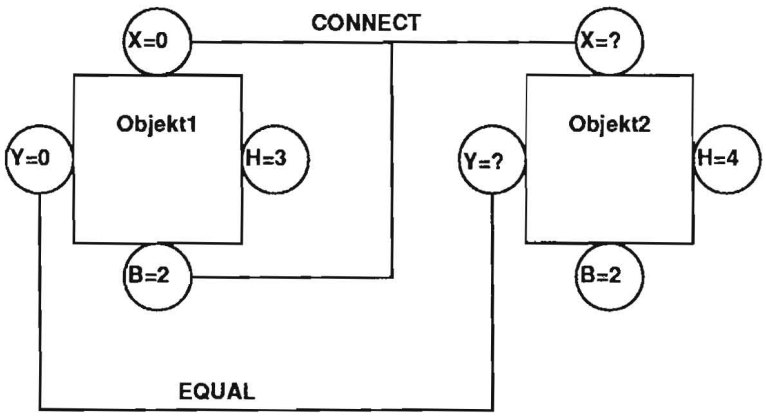

Abbildung 4.2: Constraint-Netz der lokalen Beziehung Kontrast vor der Propagierung der Werte

Der Constraint-Solver arbeitet mit gewichteten Constraints, de die Propagierung steuern. Wirken zwei verschiedene Constraints auf eine Variable eines Layoutobjektes, so bindet das stärkere. Sind beide Constraints gleich stark sind, bleibt die Belegung der Variablen unverändert. In der ersten Phase werden die Variablen der Layoutobjekte durch mittlere Constraints gebunden, so daß in einer späteren Phase neue Werte in dieses Netz eingetragen werden können. Durch diese Hierarchiesierung werden in einer späteren Phase die relativen Koordinaten in absolute Koordinaten umgewandelt werden.

Der inkrementelle Solver selektiert, entsprechend den Wichtungen mit denen die Variablen gebunden sind, Methoden[1] zur Lösung eines Constraints. Einem jeden Constraint wird eine Wichtung mitgegeben, wodurch die Wichtigkeit festgelegt wird, mit der dieses Constraint das bestehende Netz beeinflussen kann. Variablen erhalten jeweils die Wichtung des schwächsten Constraints, welches die Variable bindet. Um eine Variable durch ein Constraint verändern zu können, muß die Wichtung des Constraints größer als die der Variablen sein. Wichtet man alle Constraints mit der

[1] Analog zu Bornings objektorientierter Sicht werden die rechten Seiten eines Constraints mit dem Begriff *Methode* bezeichnet (vgl. [Borning et al. 87])

gleichen Stärke, so erhält man ein Constraint-Netz, in dem nur ungebundene Variablen gebunden werden können. Dies entspricht der herkömmlichen Verarbeitung flacher Constraints.

Es werden im folgenden die Komponenten vorgestellt, die eine Verarbeitung von einer Vielzahl von verschiedenen semantisch-pragmatischen Beziehungen erlauben. Dazu zählen zum einen die Basis-Constraints und darauf aufbauend die Grund-Constraints als Grundlage der Modellierung lokaler Beziehungen und zum anderen die aggregierten Constraints, in statischer wie dynamischer Form, welche eine direkte Umsetzung lokaler Beziehungen in den Constraint-Formalismus erlauben. Eine Propagierungskomponente leitet Veränderungen im Constraint-Netz an Variablen über bestehende Constraints weiter, die durch diese Veränderung betroffen sind.

4.1.1 Basis-Constraints

Die Constraints von SIVAS können auf einem beliebigen skalaren Wertebereich definiert werden. Der rasterbasierte Ansatz, der bei der automatischen Layoutgenerierung verfolgt wird, bedingt eine Diskretisierung des Suchraums. Layoutobjekte können nur auf endlich vielen, fest definierten Positionen auf dem Raster positioniert werden. Somit ist eine Abbildung der Dokumentfläche auf ein zweidimensionales Feld über dem Bereich der positiven natürlichen Zahlen als Repräsentation der Dokumentfläche ausreichend. Dies hat zu Folge, daß Basis-Constraints ebenfalls über diesem Bereich der positiven, natürlichen definiert sind.

Als Basis-Constraints werden die arithmetischen Funktionen des zugrundeliegenden LISP-Systems verwendet, die entsprechend von SIVAS ausgewertet werden.

4.1.2 Grund-Constraints

Aufbauend auf diesen Basis-Constraints, repräsentieren Grund- Constraints die einfachsten, geometrischen Beziehungen zwischen zu plazierenden Objekten. Entsprechend der Multidirektionalität von Constraints bestimmen Grund-Constraints die Werte der einzelnen Parameter durch verschiedene Methoden. Je nach Belegung der Eingabe-Variablen wird eine von diesen Methoden ausgewählt, um die noch freie Variable zu binden.

Eine n-stellige Relation R kann durch ein Grund-Constraint C in folgender Form repräsentiert werden[2]:

$$\text{(DefSIConstraint Constraint-Name } (Method_1 \vee \ldots \vee Method_n))$$

Dadurch, daß Basis-Constraints funktionalen Charachter haben, werden die Variablen, die durch Grund-Constraints belegt werden, auch durch einen festen, diskreten Wert belegt. Die Belegung dieser Variablen wird durch die Propagierungs-Komponente an das Netz weitergeleitet, so daß lokale Änderungen globale Auswirkungen haben können.

[2]Dabei bestimmt die m-te Methode den Wert des m-ten Parameters.

Ein Beispiel für ein Grund-Constraint ist *CONNECT*, welches wie folgt definiert ist[3]:

$$\text{CONNECT}\ (\ X\ Y\ Z\ ) \leftrightarrow (\ -\ Y\ Z\ ) \vee (\ +\ X\ Z\ ) \vee (\ -\ Y\ X\ ),$$

wobei das erste Element der Methodenliste X, das zweite Y und das dritte Z berechnet.

Die Bedeutung des Constraints *CONNECT* für die Anwendung im Layout (s. Abb. 4.3) ist, daß die Variablen X und *Breite* von Layoutobjekt *LO1* und X von Layoutobjekt *LO2* so in Beziehung stehen, daß von den Gleichungen[4]

$$\begin{aligned} &x\ (\ LO1\ ) = x\ (\ LO2\ ) - breite\ (\ LO1\ ) \\ &x\ (\ LO2\ ) = x\ (\ LO1\ ) + breite\ (\ LO1\ ) \\ &breite\ (\ LO1\ ) = x\ (\ LO2\ ) - x\ (\ LO1\ ) \end{aligned}$$

diejenigen ausgewertet wird, die die noch unbestimmte Variable bestimmt.

Mit solchen Grund-Constraints werden Objekte untereinander in Beziehung gesetzt und Berechnungen über ihren Platz relativ zueinander ausgewertet.

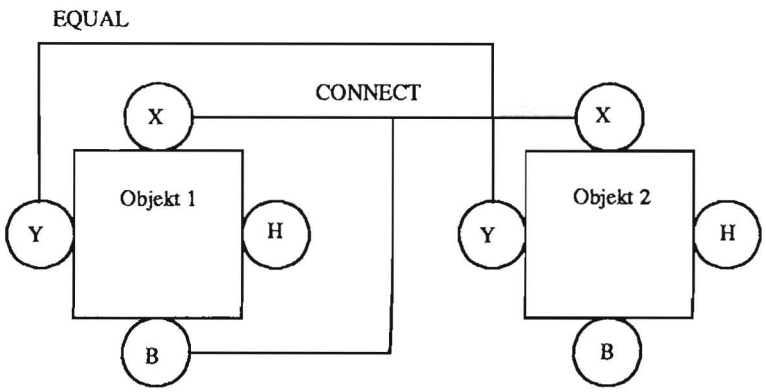

Abbildung 4.3: Grund-Constraint *CONNECT* zur Verbindung zweier Layoutobjekte

4.1.3 Aggregierte Constraints

Das Ziel der aggregierten Constraints ist die Repräsentation von semantisch-pragmatischen Beziehungen. Aggregierte Constraints snd induktiv über dem Bereich der Grund-Constraints definiert. Die deklarative Definition von aggregierten Constraints erlaubt somit die Repräsentation von komplexen geometrischen Zusammenhängen. Die Methoden der aggregierten Constraints unterscheiden sich in der Bedeutung von denen der Grund-Constraints. Aggregierte Constraints bestimmen Zusammenhänge zwischen verschiedenen Parametern durch Anwendung einer Reihe

[3]In diesem Beispiel sind X, Y, und Z Parameter, die untereinander in Beziehung gesetzt werden.
[4]Die Funktionen x und *breite* liefern den Wert der entsprechenden Variable des Layoutobjektes

von Grund- oder auch aggregierten Constraints. Eine Methode gilt dann als erfüllt, wenn alle Constraints dieser Methode erfüllt sind. Die Reihenfolge der Methoden bestimmt dabei keine Zuordnung zu den Paramtern, wie es bei Grund-Constraints der Fall ist, sondern es wird durch die Reihenfolge der Methoden eine Präferenz festgelegt, da der Solving-Prozeß die erste Methode auswählt, die erfolgreich ausgewertet werden kann. Jede Methode eines aggregierten Constraints ist für den Layout-Prozeß eine alternative Repräsentation einer semantisch-pragmatischen Beziehung.

Diese Technik entspricht der Auswahlstrategie von *PROLOG*. Durch Umsortieren kann dadurch auf den Constraintlösungsprozeß eingewirkt werden. Im Gegensatz zu *PROLOG* können bei SIVAS keine Endlosschleifen entstehen, da der Solver zykelfrei arbeitet und nur eine endliche Anzahl von Eingabevariablen verwendet. Somit ist die Terminierung des Constraint-Lösungsprozesses gewährleistet.

Statische aggregierte Constraints

Ein statisches aggregiertes Constraints dient der Repräsentation von statischen geometrischen Beziehungen, die sich zwischen beliebigen Variablen ergeben. Hierbei sind die Constraints in der Anzahl ihrer Parameter fest bestimmt. Diese Constraints geben feste Beziehungen wieder, wie sie beispielsweise von Sussmann/Steele (vgl. [Sussman & Steele 80]) zwischen Objekten einer Temperatur-Anzeigenumwandlung beschrieben werden.

Ein Beispiel für ein statisches aggregiertes Constraint ist *CONTRAST*. Ausgegangen wird davon, daß ein Kontrast nur zwischen zwei Bilder bestehen kann.

Die prädikatenlogische Repräsentation des Constraints *CONTRAST* ist wie folgt[5] :

$$\begin{aligned}&\text{CONTRAST } (\ B_1 B_2\ ) \leftrightarrow \\ &\qquad ((\text{CONNECT X}(B_1)\ \text{B}(B_1)\ \text{X}(B_2)) \wedge (\text{EQUAL Y}(B_1)\ \text{Y}(B_2))) \vee \\ &\qquad ((\text{CONNECT Y}(B_1)\ \text{H}(B_1)\ \text{Y}(B_2)) \wedge (\text{EQUAL X}(B_1)\ \text{X}(B_2)))\end{aligned}$$

[5]In den Beispielen werden getypte Objekte als Eingabe der Constraints verwendet, um den Bezug zum Layout direkt zu zeigen. Ein Objekt besteht hierbei aus einer X- und Y-Komponente, sowie einer Breite und einer Höhe.

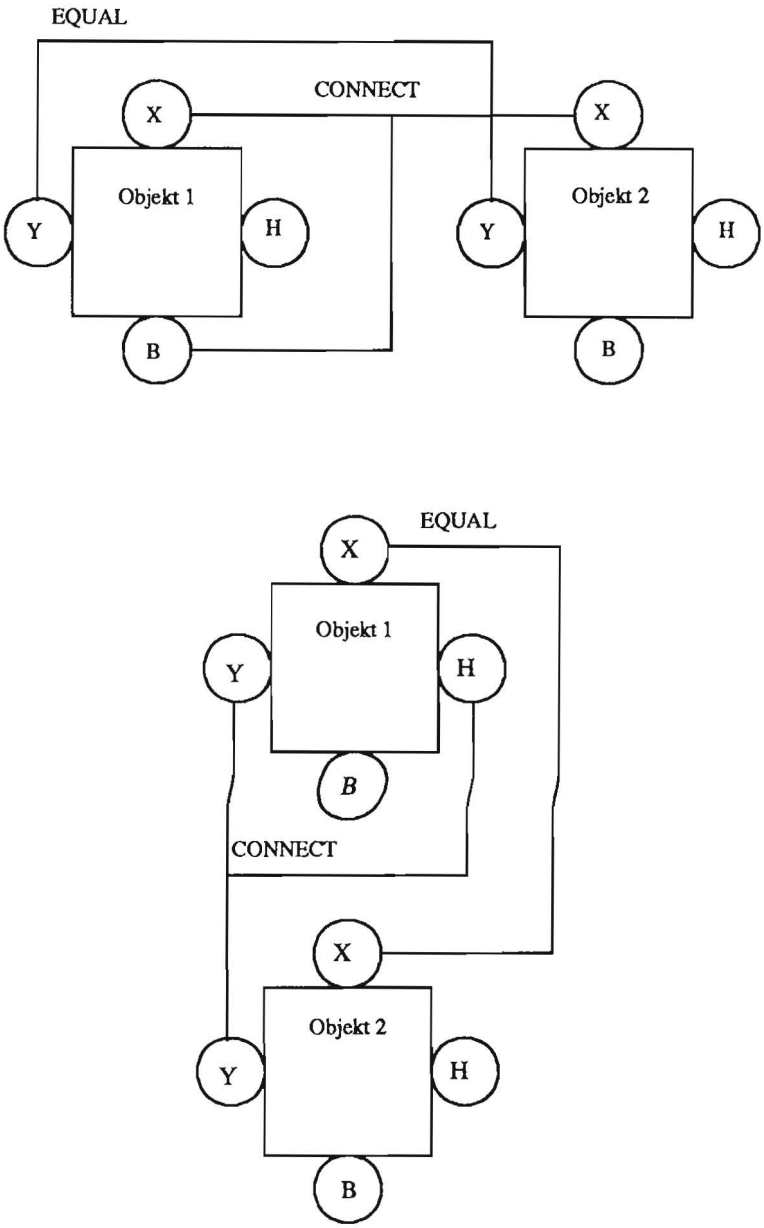

Abbildung 4.4: Statisches aggregiertes Constraint *CONTRAST*

Die Erfüllbarkeit des Constraints *CONTRAST* ist dann gegeben, wenn eine der Methoden gültig ist. Das Grund-Constraint *EQUAL* repräsentiert die Gleichheit in der Belegung zweier Variablen. Die Semantik des Constraints bewirkt eine horizontale bzw. im zweiten Fall eine vertikale Alignierung der Objekte. Die Alternative wird dann angewandt, wenn die erste Methode nicht erfüllt werden kann (s. Abb 4.4).

Dynamische aggregierte Constraints

Die Verwendung von statischen aggregierten Constraints reicht aus, wenn das Constraint-System nur fest definierte Beziehungen erhalten muß. Sollen jedoch auch Beziehungen verarbeitet werden, die in der Anzahl der zu betrachtenden Parameter nicht beschränkt sind, reichen sie nicht mehr aus. Dazu ist der Constraint-Formalismus von SIVAS um dynamische aggregierte Constraints erweitert worden, die zur Laufzeit entsprechend den Anforderungen generiert werden.

Sie repräsentieren Beziehungen, die eine Reihe von gleichartigen Gruppierungen von Layoutobjekten zu einer Einheit zusammenfassen. Beispielsweise besteht eine Sequenz aus einer beliebigen Anzahl von Graphik-Text-Blöcken. Diese Blöcke haben eine feste Struktur und repräsentieren in der Kombination die gesamte Relation.

Beispiel eines dynamischen aggregierten Constraints ist *SEQUENZ*[6]:

SEQUENZ ($B_1 B_2$) $\leftrightarrow$
((EQUAL X(B_1) X(B_2)) $\wedge$
(CONNECT Y(B_1) H(B_1) Y(B_2)) $\wedge$
(EQUAL Y(B_1) SUCC-Y) $\wedge$
(CONNECT SUCC-X X(B_1) B(1)))
$\vee$
((EQUAL Y(B_1) Y(B_2)) $\wedge$
(CONNECT X(B_1) B(B_1) X(B_2))) $\wedge$
(EQUAL X(B_1) SUCC-X)$\wedge$
(CONNECT SUCC-Y Y(B_1) H(B_1)))

Es sind hierbei zwei alternative Methoden für die Plazierung, wie in Bild beschrieben, angegeben. Die erste Methode des dynamischen aggregierten Constraints *sequence* repräsentiert eine horizontale (s. Abb. 4.5), die zweite eine vertikale Anordnung (s. Abb.4.6) der einzelnen Graphik-Text-Blöcke zueinander. In der ersten Methode werden durch die beiden ersten Constraints *EQUAL* und *CONNECT* jeweils die Beziehungen zwischen den X- und Y-Koordinaten des Layoutobjektes *G1* und des Layoutobjektes *T1* hergestellt. Die nächsten beiden Constraints *CONNECT* und *EQUAL* verknüpfen das Layoutobjekt *G1* mit dem nachfolgenden Layoutobjekt *G2*, falls dieses existiert. In entsprechender Weise bilden die ersten beiden Constraints der zweiten Methode eine Verknüpfung des Graphikblocks *G1* und des Layoutobjektes *T1* in horizontler Ebene, wenn die gesamte Relation vertikal ausgerichtet werden soll. Die durch die jeweils ersten beiden Constraints verbundenen Layoutobjekte sind die Grundeinheiten, aus denen sich die gesamte Relation aufbaut.

[6] *succ-x* und *succ-y* bestimmen die X- und Y-Variable des jeweiligen Nachfolgeobjektes

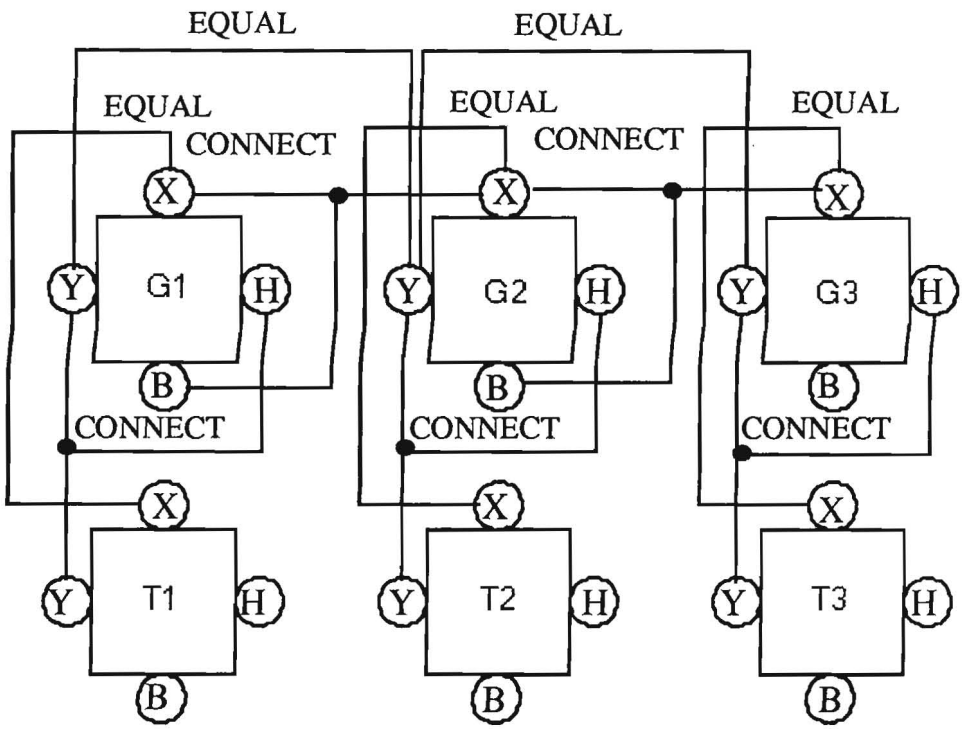

Abbildung 4.5: 1. Alternative einer Instanz des dynamischen aggregierte Constraints *SEQUENCE* mit drei Grundeinheiten

4.1.4 Constraint-Lösung und Propagierung

Die durch den Solver hervorgerufenen Veränderungen der Variablenbindungen werden durch die Propagierungskomponente an das Gesamtnetz weitergeleitet. Durch die inkrememtelle Arbeitsweise dieser Komponente, erreicht man eine effiziente Verarbeitung, die vor allen Dingen bei stark variablen Constraint-Netzen, wie im Falle des Layouts, notwendig ist. In Beispiel (s. Abb. 4.10) wird gezeigt, wie die Propagierungskomponente von den Wichtungen der Constraints und den Variablen beeinflußt wird. Eine Variablenbelegung kann nur dann verändert werden, wenn:

1. die zu belegende Variable noch nicht gebunden ist oder
2. die Wichtung des Constraints stärker ist als das, welches aktuell die Variable bindet oder
3. die Belegungen der anderen Variablen des Constraints sich geändert haben.

Der erste Punkt kommt bei der Auswahl der Methode zum tragen, wenn der Solver bestimmt, welche Variable durch das Constraint gebunden werden soll. Die beiden nächsten Punkte sind Teil der Propagierungsphase, in der bestimmt wird, welche Constraints erneut auszuwerten sind.

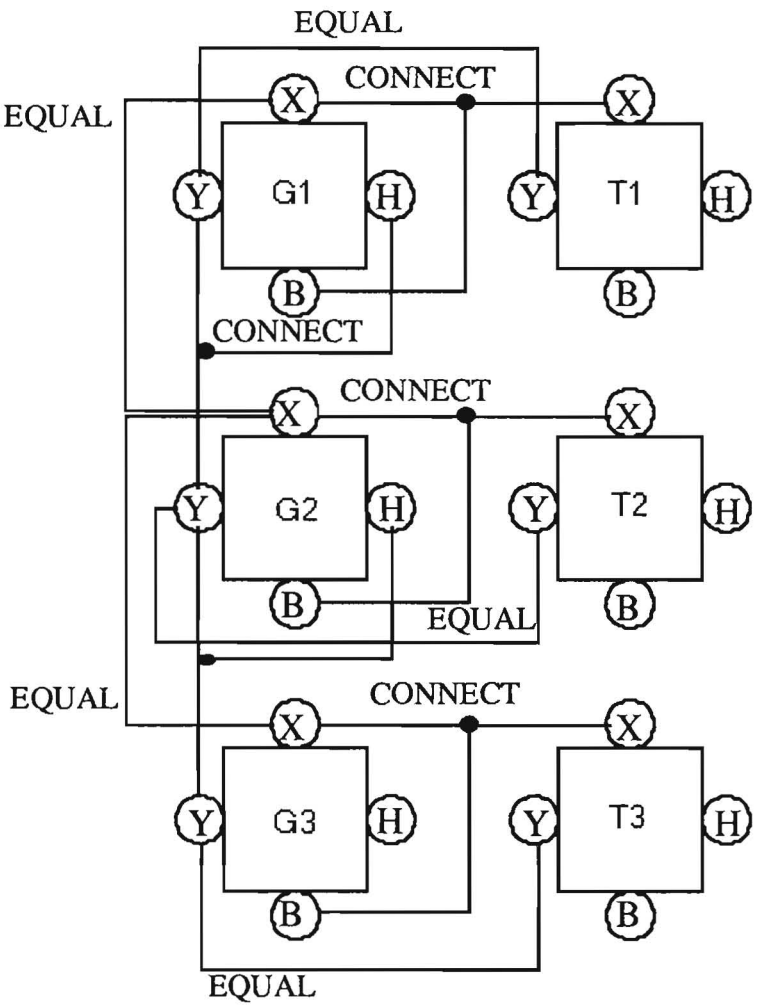

Abbildung 4.6: 2. Alternative einer Instanz des dynamische aggregierte Constraints *SEQUENCE* mit drei Grundeinheiten

Der Algorithmus traversiert die Hierarchie von den stärksten abwärts zu den schwächsten Constraints, bis alle Constraints entweder erfüllt oder blockiert sind.Für jedes Constraint gilt folgendes:

- Hat das Constraint eine Methode, deren Input-Variablen belegt und deren Output-Variablen frei sind, so benutze diese Methode um das Constraint zu erfüllen und werte das stärkste verbleibende Constraint der Hierarchie aus.
- Sind alle Methoden des Constraints blockiert, so markiere das Constraint als geblockt und betrachte es nicht weiter. Fahre in der Hierarchie weiter fort.
- Sind die beiden ersten Punkte nicht erfüllt, so mache mit dem Constraint nichts und fahre ebenfalls in der Hierarchie fort.

constraint	level	satisfied?	method
A + B = C	required	yes	$C \leftarrow A + B$
C + D = E	required	yes	$E \leftarrow C + D$
A stay	strong	yes	A stay
D stay	medium	yes	D stay
B stay	weak	yes	B stay
E stay	very_weak	no	—

Abbildung 4.7: Variablenbelegung des Constraint-Netzes

Weiterentwicklungen dieser einfachen Constraint-Solving-Strategie sind inkrementelle Solver, wie sie von Borning (vgl. [Freeman-Benson et al. 90, Borning et al. 87]) vorgestellt wurden. Zugrunde liegt ein inkrementeller Constraint-Solver, der durch die Verwendung von Wichtungen der Constraints die Möglichkeit der Hierarchisierung von Constraints beinhält.

Die Wichtung einer Variablen stellt die Wichtung des schwächsten Constraint dar, welches überschrieben werden muß, um den Wert der Variablen zu verändern. Wird ein neues Constraint in die Hierarchie eingefügt, so werden folgende Schritte ausgeführt:

1. Prüfe, ob das neue Constraint durch Anwendung einer Methode erfüllt werden kann
2. Suche das Constraint, welches überschrieben werden kann
3. Bestimme die Teile der Belegungen, die verändert werden müssen, um das neue Constraint zu erfüllen.

Im folgenden Beispiel wird die Veränderung einer Hierarchie durch Einfügen eines neuen stärker gewichteten Constraints für die Variable E gezeigt.

Die graphische Darstellung dieser Hierarchie ist in Bild (s. Abb. 4.8) dargestellt. Die Pfeile repräsentieren Constraints, wobei an den Kanten die Wichtungen eingetragen sind. Gestrichelt dargestellte Constraints sind nicht erfüllte, sog. *blockierte* Constraints.

Wird im vorliegenden Beispiel (s. Abb. 4.10) ein Constraint der Wichtung *required* eingefügt, welches die Belegung von E verändert, so hat das Auswirkungen auf den Rest des Netzes. Die Wichtung einer Variablen ist wie folgt bestimmt:

Definition 4.1 *Sei Variable v durch die Methode m des Constraint c bestimmt. Die Wichtung von v ist das Minimum der Wichtungen von c und den Wichtungen der Inputvariablen von m.*

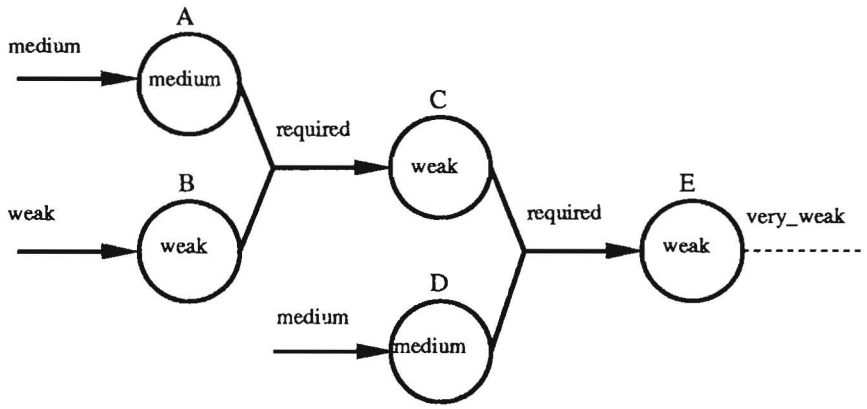

Abbildung 4.8: Beispiel eines Constraint-Netzes

constraint	level	satisfied?	method
A + B = C	required	yes	$B \leftarrow C - A$
C + D = E	required	yes	$C \leftarrow E - D$
E edit	required	yes	E $\leftarrow$ input value
A stay	strong	yes	A stay
D stay	medium	yes	D stay
B stay	weak	yes	B stay

Abbildung 4.9: Variablenbelegung nach Einfügung des unbedingten Constraints

Wird im obigen Beispiel das Constraint mit der Wichtung *required* in das Netz an die Variable E angelegt, so prüft der Algorithmus, ob das Constraint anwendbar ist. Da E eine Wichtung *weak* hat, kann das Constraint diese Variable neu belegen. Die Variable E erhält die Wichtung *required*, da keine Inputvariablen vorliegen. Daraufhin beginnt die Propagierung, welche eine Neuberechnung des Constraint-Netzes solange weiterführt, bis entweder keine noch nicht ausgeführte Constraints mehr vorhanden sind oder diese Constraints zu schwach sind um neue Belegungen zu berechnen.

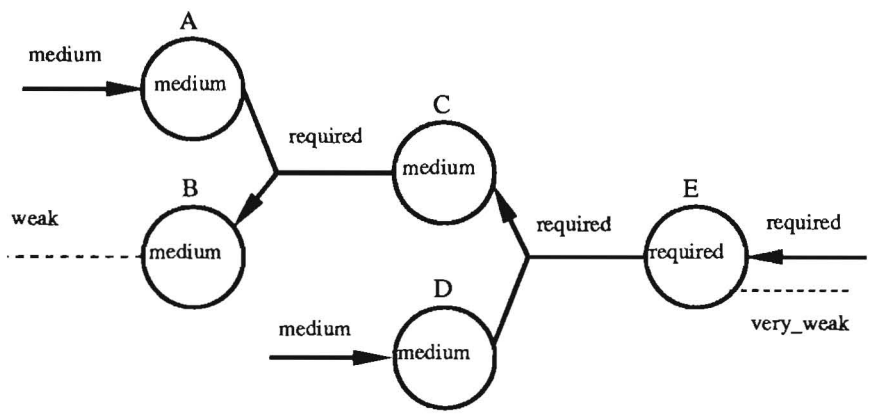

Abbildung 4.10: Constraint-Netz nach Einfügen eines neuen Constraints

Im Beispiel heißt dies, daß das Constraint, welches C, D und E miteinander in Verbindung setzt, zurückgenommen werden muß, da die Inputvariable C nur mit der Wichtung *weak* versehen ist und das neue Constraint stärker ist. Das alte Constraint zwischen C, D und E wird umgekehrt, woran man die Multidirekionalität der Constraints erkennen kann, und wird für die Belegung der Variable C verwendet. C wird unter den Variablen des Constraints asgewählt, da sie die schwächste ist. Die Berechnung der Wichtung der Variablen C ergibt die Wichtung *medium*. Da *medium* stärker als die Wichtung von B ist, wird auch das Constraint zwischen A, B und C in entsprechender Weise umgekehrt und die Wichtungen neu berechnet. Daraufhin bricht die Propagierung ab und hat, entsprechend der Definition, den Zustand lokaler Konsistenz erreicht. Bedingt durch diese Art der Propagierung muß die Zyklenfreiheit gewährleistet sein. Ist ein Zyklus vorhanden, so bricht das System mit einer Fehlermeldung ab.

4.2 Formale Arbeitsweise von *FIDOS*

FIDOS (**Fi**nite-**Do**main-**S**olver) ist ein inkrementeller Constraint-Solver, basierend auf der *Label-Inference*-Technik (vgl. [van Hentenryck 89]). Ähnlich wie bei SIVAS, werden durch FIDOS Constraints über dem Bereich der natürlichen Zahlen verarbeitet, jedoch werden den Variablen nicht einzelne skalare Werte sondern Wertemengen zugeordnet. Dies erlaubt die Bestimmung einer Lösung entsprechend der in *CHIP* verwendeten Inferenztechnik des *Forward-Checkings*. Die Gesamtheit der Belegungen aller Variablen in einem gegebenen Constraint-Netz bestimmt den maximalen Lösungsraum. Wenn also eine Variable durch ein Constraint mit der leeren Belegung gebunden wird, besitzt daß gegebene Constraint-Netz, bezogen auf die aktuelle Belegung, keine konsistente Lösung. Die Begründung liegt dabei in der Monotonie der Constraints in FIDOS. Wenn ein Constraint-Netz A, bestehend aus n Constraints, für eine Variable X eine leere Belegung erzeugt, so hat ein Constraint-Netz B, welches das Netz A als Teilmenge enthält, ebenfalls keine Belegung für die Variable X. Dies wiederum bedeutet, daß alle Netze, die A als Teilnetz beinhalten, inkonsistent sind. Ist ein Netz als inkonsistent erkannt worden, so kann der Constraint-Lösungs-Prozeß abgebrochen werden und eine Rücksetz-Prozedur einen neuen Ansatz suchen. Mittels dieser Technik kann die Suche nach einer konsistenten Lösung stark eingeschränkt werden.

Constraints werden entsprechend SIVAS über dem Bereich der Basis-Constraints definiert, die sich in beliebiger Weise aggregieren lassen. Basis-Constraints sind arithmetische oder mengenwertige Funktionen über dem Bereich der natürlichen Zahlen.

Definition 4.2 *Ein Constraint heißt erfüllbar, wenn die Belegung aller Input- und Output-Variablen des Constraints ungleich der leeren Menge sind.*

Hierauf aufbauend ist die Erfüllbarkeit eines Constraint-Netzes definiert:

Definition 4.3 *Ein Constraint-Netz heißt erfüllbar, wenn alle Constraints des Netzes erfüllbar sind.*

4.2.1 Basis-Constraints

Basis-Constraints von FIDOS haben wie bei SIVAS den Charachter von Funktionen. Sie bestimmen aus der Belegung gegebener Domänen-Variablen den Wertebereich, der neu zu belegenden Ausgabevariablen. Der gesamte Berechnungsumfang der zugrundeliegenden LISP-Funktionen stehe dazu zur Verfügung.

4.2.2 Grund-Constraints

In Bezug auf die Constraints über Finite Domains von *CHIP*, umfassen Grund-Constraints von FIDOS sowohl die arithmetischen, als auch symbolische Constraints. Für die Verarbeitung sind vor allen Dingen die Mengenoperationen auf den Domänen der Variablen von besonderer Wichtigkeit. Sie bestimmen die Einschränkungen der Domänen, die durch zusammengesetze Constraints den Domänen-Variablen zugewiesen werden.

Basis-Constraints sind in Form von LISP-Funktionen in FIDOS integriert und können in einfacher Weise modifiziert werden. Ein Beispiel hierfür ist das Grund-Constraint *FDSUB*, welches den Durchschnitt der Belegungen zweier Domänen-Variablen bestimmt:

(*DefGroundFDConstraint* FDSUB (domain1 domain2)
(set-difference domain1 domain2))

4.2.3 Aggregierte Constraints

Über dem Bereich der Grund-Constraints lassen sich aggregierte Constraints definieren. Es wird zwischen unidirektionalen und bidirektionalen aggregierten Constraints unterschieden, die entweder nur in Richtung der neu zu belegenden Variablen oder auch in Richtung der Eingabevariablen wirken. Im Normalfall wirkt die Auswertung eines Constraints in beide Richtungen, da die Belegung einer Variablen Rückwirkungen auf die Variablen hat, die in diesem Constraint als Eingabe dienen. Die Belegung einer Variablen entspricht der maximalen Lösungsmenge einer Variablen in einem gegebenen Constraint-Netz. Wird eine Variable zu einem Zeitpunkt durch einen Wert dieser aktuellen Lösungsmenge fest gebunden, so gibt es im Constraint-Netz konsistente Belegungen aller Variablen, die das Netz erfüllen.

Unidirektionale Constraints bestimmen die Belegung der Domänen-Variablen. Dies ist z.B. der Fall, wenn eine Einheit auf einem bestimmten Bereich der Layout-Fläche fixiert werden soll. Dadurch haben unidirektionale Constraints die Semantik einer Zuweisung.

Sollen Domänen-Variablen einer Layouteinheit durch eine andere Einheit über ein Constraint belegt werden, so werden bidirektionale Constraints verwendet. Sie belegen zuerst die Domänen-Variablen der zu belegenden Einheit und im Gegenzug die der Einheit, die als Eingabe dieses Constraints gilt. Dadurch muß bei der Definition eines zweidirektionalen Constraints bestimmt werden, wie die Domänen-Variablen der beiden Layout-Einheiten sich gegenseitig beeinflussen.

Der allgemeine Aufruf zur Definition eines bidirektionalen Constraints wird durch die Funktion *DefFDConstraint* bestimmt:

(*DefFDConstraint* Constraint-Name
(<Liste der Einschränkungen für Einheit2>
<Liste der Einschränkungen für Einheit1>)

Am Beispiel des Constraints *RechtsVon* soll die Definition eines aggregierten Constraints illustriert werden[7].

(*DefFDConstraint* Rechtsvon $(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8) \leftrightarrow$
(FDADD (FDMIN X_1) X_4) $\wedge$;;; x2-min
(FDMAXRIGHT MIN X_8 X_2))$\wedge$;;; x2-max
(FDMIN X_2)$\wedge$;;; y2-min
(FDDIFF (FDMAX X_2)(FDDIFF X_7 X_3)) ;;; y2-max
$\wedge$
(FDMIN X_1)$\wedge$;;; x1-min
(FDDIFF (FDMAX X_5) X_4)$\wedge$;;; x1-max
(FDMIN X_2)$\wedge$;;; y1-min
(FDDIFF (FDMAX X_6)(FDDIFF X_3 X_7)) ;;; y1-max

Die Definition unterteilt die Wirkungsweise des Constraints in einen Teil, der auf die Domänen-Variablen der zu belegenden Einheit und in einen, der auf die Variablen der belegenden Einheit wirkt. Die Konjunkte bestimmen jeweils die untere und obere Grenze des Wertebereichs auf dem die Variable unter dem aktuellen Constraint-Netz gültig ist. Beispielsweise bestimmen im Beispiel *Rechtvon* die ersten beiden Konjunkte die unter und obere Grenze der Belegung der Variablen X_1. Entsprechendes gilt für die übrigen Konjunkte. Die Definition ist dabei durch die Verwendung von jeweils acht Variablen pro Constraint, stark auf die Gegebenheiten in der automatischen Layoutgenerierung ausgerichtet.

4.2.4 Constraint-Lösung und Propagierung

Der Constraint-Solving-Prozeß wird durch die Instantiierung der Parameter eines aggregierten Constraints mit den Domänen-Variablen der Einheiten gestartet. Dadurch, daß das Constraint immer auf die Domänen-Variablen aller Einheiten wirkt, muß es zuerst auf Variablen der zu bindenden Einheit angewendet werden. Die Einschränkungen, die sich dadurch ergeben, werden über das Constraint-Netz an alle Domänen-Variablen propagiert, die durch diese Änderungen betroffen sind. Danach wird das Constraint in Richtung der Domänen-Variablen der anderen Einheit ausgewertet, und deren Änderungen über das Netz propagiert.

Die Propagierung wird durch Informationen gesteuert, die jedes Constraint und jede Domänen-Variable enthält. Unidirektionale Constraints bilden im Verarbeitungsprozeß eine Untermenge von bidirektionalen Constraints. Die Verarbeitung eines bidirektionalen Constraints unterteilt sich in folgende vier Schritte:

[7]Um den Bezug zur automatischen Layoutgenerierung herzustellen, werden die ersten vier Parameter mit der X-, Y-, Breite und Höhe-Variablen der zweiten und die zweiten vier mit denen der ersten Einheit gematcht. Allgemein kann FIDOS aber Constraints zwischen beliebigen Variablen, d.h unabhängig vom Bezug zum Layout auswerten und erhalten.

1. Auswertung des Constraints in Richtung der Ausgabevariablen
2. Propagierung der Änderungen durch das aktuelle Constraint-Netz
3. Auswertung des Constraints in Richtung der Eingabevariablen
4. Propagierung der Änderungen durch das aktuelle Constraint-Netz

Ist ein Constraint gelöst und sind die Änderungen propagiert worden, so befindet sich das Netz entweder in einem konsistenten Zustand oder zumindest eine Domänen-Variable ist durch die leere Menge belegt. Da die Belegung einer Variablen monoton fallend ist, d.h. eine Belegung kann nur verkleinert werden, kann es keine Lösung dieses Constraint-Netzes geben.

Definition 4.4 *Eine Menge $\mathcal{R}$ von aggregierten Constraints heißt konsistent, wenn es keine Variable $x \in \mathcal{V}$ gibt, wobei $\mathcal{V}$ die Menge der zu $\mathcal{R}$ assoziierten Domänen-Variablen ist, deren Belegung die leere Menge ist.*

Definition 4.5 *Eine Menge $\mathcal{R}$ von aggregierten Constraints heißt inkonsistent, wenn sie nicht konsistent ist.*

Definition 4.6 *Eine Menge $\mathcal{R}$ von aggregierten Constraints heißt minimal-inkonsistent, wenn:*

$$\exists r \in \mathcal{R} : \mathcal{R} \setminus r \text{ ist konsistent.}$$

Lemma 4.1 *Sei $\mathcal{R}$ die Menge aller aggregierten Constraints und $\mathcal{P} \subset \mathcal{R}$ minimal-inkonsistent.*

So gilt:

$$\forall r \in \mathcal{R} : \mathcal{P} \cup r \text{ ist inkonsistent.}$$

Beweis:

Vor.: Für alle $r \in \mathcal{R}$ gilt[8] *:*

$$x_i \supseteq pr_i(r(x_1, \ldots, x_n)), \text{ für } 0 \leq i \leq n,$$

d.h., daß nach Anwendung eines Constraints r die Belegung einer Variablen in der Anzahl höchstens kleiner geworden ist.

Da $\mathcal{P}$ minimal-inkonsistent ist, gibt es eine Variable x_i für die $x_i = \emptyset$ gilt.

[8] pr_i ist die Projektion auf die i-te Komponente der Relation. Sie liefert alle gültigen Belegungen der Variabeln x_i

Falls r die Variable x_i nicht betrifft, so folgt, daß wenn x_i vor Anwendung von r mit der leeren Menge belegt war, so auch danach.

Betrifft r die Variable x_i, so folgt auf Grund der Monotonieeigenschaft der Constraints in FIDOS:

$$pr_i(c(x_1, \ldots, x_i, \ldots, x_n)) \subseteq x_i$$

Daraus folgt mit $x_i = \emptyset$:

$$pr_i(c(x_1, \ldots, x_i, \ldots, x_n)) = \emptyset$$

□

4.2.5 Globale Konsistenz

Die Erfüllbarkeit von Constraint-Netzen unter FIDOS definiert eine lokale Konsistenz der Constraints untereinander. Die lokale Konsistenz ist aber nicht ausreichend um eine global-konsistente Lösung des Constraint-Netzes zu bestimmen (s. Abb. 4.11).

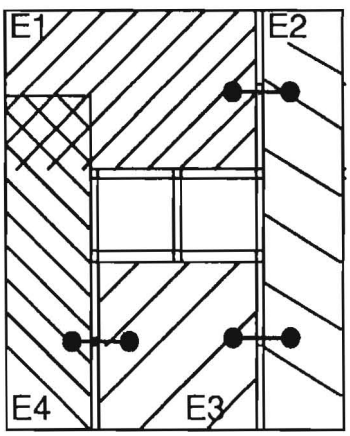

Abbildung 4.11: Globale Konsistenz

In Abbildung 4.11 ist ein Szenario dargestellt, in dem vier Einheiten auf einem Raster plaziert werden sollen. Dies ist aber unter Anwendung der eingezeichneten globalen Constrains nicht möglich, da die vierte Einheit mit der ersten in Konflikt steht. Es besteht keine direkte Verbindung zwischen diesen beiden Einheiten, so daß dies mittels lokaler Konsistenzüberprüfung nicht erkannt wird.

Um nicht durch Erweiterungen des Constraint-Solvers eine Effizienzeinbußen bei der Bestimmung eines Layouts einzugehen, wird dem Solver eine Kontrollstruktur zugeordnet, die während der Auswertung der Constraints protokolliert, ob die berechnete Belegung der Variablen global-konsistent ist. Die Kontrollstruktur verwaltet hierbei das vorgegebene Raster, auf dem die Objekte angeordnet werden sollen.

Ein Ziel der Vorgehensweise von FIDOS ist der Erhalt der maximalen Lösungsmenge. Dazu werden jeder Einheit alle Koordinaten des Rasters als potentielle

Plazierungspunkte der linken, oberen Ecke zugeordnet, die durch aktuell evaluierte Constraints möglich sind. Die bedeutet, daß eine Einheit solange nicht fest an eine Position gebunden ist, wie sie entweder nicht eine a priori fixierte Einheit ist, oder die Constraints noch andere Position zulassen. Wird eine Einheit aber auf eine Position festgelegt, so kann die gesamte Fläche die es damit auf dem Raster belegt nicht mehr durch andere Einheiten verwendet werden. Somit hat eine Einheit Auswirkungen auf andere Einheiten, die möglicherweise nicht mit dieser durch ein globales Constraint verbunden ist. Eine Verwaltungsstruktur blockiert somit die belegte Fläche auf dem Raster und streicht somit diese für alle anderen Einheiten.

4.2.6 Bestimmung der absoluten Koordinaten

Nach Auswertung aller lokalen und globalen Beziehungen ist die maximale Lösungsmenge bestimmt. Aus dieser Menge wird entsprechend dem Dokumenttyp eine Möglichkeit aus einer Menge von globalen Beziehungen ausgewählt, indem jede Variable in FIDOS genau ein Wert ihrer Lösungsmenge zugeordnet wird. Dabei können beliebige Reihenfolgen in der Verwendung globaler Beziehungen verwendet werden. Im Normalfall wird der erste Wert einer jeden Menge ausgewählt, so daß die Einheit in ihrer Plazierung nach oben links tendieren. Alternative Auswahlmöglichkeiten, wie die Wahl des mittleren Wertes plazieren die Objekte tendenziell in die Mitte der Dokumentfläche.

Ist die Auswahl getroffen, werden alle diese Werte, die die Koordinaten der linken, oberen Ecke einer Einheit bestimmen an den Solver SIVAS weitergegeben. Da bis zu diesem Zeitpunkt alle Layoutobjekte, die zu einer Einheit zusammengefaßt worden sind, relativ zum Nullpunkt der Einheit positioniert worden sind, werden sie nun mittels der Koordinaten aus FIDOS absolut zum Ursprung des Rasters ausgerichtet. Dieser Vorgang wird durch *SET*-Constraints[9] von SIVAS ausgeführt, die die größtmögliche Wichtung besitzen und somit alle Constraints überschreiben, die bisher die X- und Y-Koordinaten der linken oberen Ecke einer Einheit bestimmt haben. Durch Festsetzen der linken oberen Ecke einer Einheit wird durch die Propagierung erreicht, daß alle Layoutobjekte ihre absolute Position zum Ursprung des Rasters erhalten.

[9] Das *SET*-Constraint legt die Belegung einer Variablen aus SIVAS auf einen bestimmten Wert fest.

4.3 Definition der Constraint-Sprachen in CLAY

4.3.1 Die Constraint-Sprache von SIVAS

Für die Definition von Constraints für den Constraint-Solver SIVAS stehen zwei verschiedene Funktionen zur Verfügung. Mittels der Funktion *DefSIConstraint* lassen sich einfache aggregierte Constraints und durch die Funktion *DefSIDynConstraint* dynamische aggregierte Constraints definieren.

Die Syntax des Aufrufs wird durch die folgenden Definitionen festgelegt:

Aufruf:

DefSIConstraint	Constraint-Name ((*S-Method*) ... (*S-Method*))
DefDynConstraint	Constraint-Name ((*D-Method*) ... (*D-Method*))

S-Method	::	((*S-Constraint*) ... (*S-Constraint*))
S-Constraint	::	Constraint-Name <*S-Param-List*>
S-Param-List	::	*StaticVariable*...*StaticVariable*
StaticVariable	::	?*Number* | !*Number*
D-Method	::	((*D-Constraint*) ... (*D-Constraint*))
D-Constraint	::	*Constraint-Name* <*D-Param- List*>
D-Param-List	::	*DynamicVariable*...*DynamicVariable*
DynamicVariable	::	^*Number* | ^^*Number* | !*Number*
Number	::	$\mathcal{N}$

4.3.2 Die Constraint-Sprache von FIDOS

Für die Definition von Constraints für den Constraint-Solver FIDOS steht die Funktion *DefFDConstraint* zur Verfügung, womit sich aggregierte Constraints lassen.

Die Syntax des Aufrufs wird durch die folgenden Definitionen festgelegt:

Aufruf:

DefFDConstraint	Constraint-Name ((*B-Method*) (*B-Method*))

B-Method	::	((*F-Constraint*) ... (*F-Constraint*))
F-Constraint	::	Constraint-Name <*F-Param-List*>
F-Param-List	::	*F-Param*...*F-Param*
F-Param	::	F-Constraint | F-Variable
F-Variable	::	?*Number*

Kapitel 5

Implementierung

Das in Kapitel 4 formal beschriebene System CLAY ist in Common-Lisp auf einem MicroExplorer entwickelt und von dort auf MacIvorys und Symbolics-Lisp-Maschinen unter Genera 8.01 portiert worden. Zusätzlich zur Implementation der abstrakten Algorithmen besteht eine Testumgebung, die die Eingabe von Layoutobjekten mit lokalen und globalen sowie der Auswahl von verschiedenen Dokumenttypen und Heuristiken unterstützt (vgl. auch [Maaß et al. 91]). Auf den Einsatz von PROLOG ist auf Grund der Einbindung in das Projekt WIP verzichtet worden, da in diesem die Sprache LISP verwendet wird. Desweiteren war es das Hauptziel eine effiziente Verarbeitung von layoutspezifischem Wissen zu gewährleisten, so daß LISP, ähnlich wie im Projekt CHIP [1], als nahezu ideal anzusehen ist. Besonders ist dabei hervorzuheben, daß LISP eine symbolverarbeitende und funktionaleSprache ist, wodurch die Implementation von Basisfunktionen der Constraint-Solver entfallen konnte, was bei Verwendung einer imperativen Sprache wie *C* nicht der Fall gewesen wäre. Die Portabilität war ausschlaggebend dafür Common-LISP zu verwenden.

In diesem Kapitel werden die abstrakten Algorithmen von SIVAS, FIDOS und CLAY sowie die Strukturen der jeweiligen Constraints und die Schnittstellenfunktionen mit den entsprechenden Übergabestrukturen behandelt. Anschließend wird die Testumgebung beschrieben.

5.1 Abstrakte Algorithmen

Die Verarbeitungsprozedur, die dem Constraint-Solver-Modelle zugrunde liegt, welcher lokale und globale Beziehungen sowie den dazugehörigen Layoutobjekten verarbeitet, wird durch den abstrakten Algorithmus *LAYOUT-SOLVE* beschrieben[2]. Die Eingabeparameter bestehen dabei aus dem zu belegenden Gestaltungsraster *grid*, einer Anzahl von lokalen und globalen Beziehungen *lokRel* und *globRel* sowie dem Parameter *heuristic*, der die Steuerung der globalen Beziehungen bestimmt.

[1] In CHIP wird PROLOG und in einigen Fällen aus Effizienzgründen C verwendet.

[2] Die Beschreibung des Algorithmus lehnt sich an eine PASCAL-ähnliche Notation an, wobei die Kontrollstrukturen in entsprechender Weise zu verstehen sind.

```
LAYOUT-SOLVE ( grid, lokRel, globRel, heuristic )
lrel = NEXT ( lokRel );
while lobjects ≠ ∅ do
    lrel = NEXT ( lokRel );
    grel = NEXT ( globRel, lrel );
    objs = GET-OBJ ( lrel );
    types = GET-TYPES ( grel );
    unit = SIVAS ( objs, types, lrel, grid );
    if unit ≠ ∅
        res = FIDOS ( unit, grel, grid );
    else
        exit ( error );
    while res == error
        unit = REFORMAT-LREL ( objs, types, lrel, grid );
        if unit = ∅
            exit ( error );
        else
            res = FIDOS ( unit, grel, grid );
        endif
    endwhile
lrel = NEXT ( lokRel );
endwhile
```

Abbildung 5.1: Abstrakter Algorithmus LAYOUT-SOLVE

Der Algorithmus läuft über die Liste der lokalen Beziehungen *lokRel* und wendet ensprechende lokale Constraints vermittels dem Constraint-Solver SIVAS auf die assoziierten Layoutobjekte *objs* an[3]. Diese bilden eine Einheit *unit*, der a priori eine globale Beziehung *grel* zgeordnet sein kann. Ist dies der Fall, so wird diese, ansonsten eine globale Beziehung entsprechend der Heuristik *heuristic*, durch den Constraint-Solver FIDOS auf die Einheit angewandt. Kommt es dabei zu einer Inkonsistenz, die von FIDOS durch alternative Constraints nicht aufgefangen werden kann, so wird die Funktion *REFORM-LREL* aufgerufen, die die lokale Beziehung *lrel* durch eine alternative Methode auswertet, und die so entstandene Einheit *unit* wieder an den Solver FIDOS zurückgibt. Diese Schleife wird solange durchlaufen, bis entweder alle lokalen und globalen Beziehungen erfüllt werden können, oder eine Inkonsistenz nicht behoben werden kann. Eine Inkonsistenz entsteht, wenn entweder eine lokale Beziehung durch SIVAS, oder eine globale Beziehung durch FIDOS nicht gelöst werden kann[4].

5.1.1 SIVAS

Der abstrakte Algorithmus basiert auf Arbeiten von Freeman-Benson et. al. (vgl. [Freeman-Benson et al. 90]). Als wesentliche Erweiterungen sind die Verarbeitung von aggregierten statischen und dynamischen Constraints hinzugekommen. Entsprechend der gegebenen lokalen Beziehung *lokrel*, wird in der Funktion *BUILDUP-CONSTRAINT* ein passendes lokales Constraint ausgewählt, oder im Falle eines dynamischen Constraints zur Laufzeit generiert. Dieses wird anschließend in der Funktion *INCREMENTALLYSATISFY* ausgewertet. Dazu wird die erste Methode des Constraints ausgewählt, deren Ausgabevariable eine schwächere Wichtung als das Constraint besitzt. Kann dadurch keine Methode bestimmt werden, so bricht der Algorithmus ab und liefert im Fall, daß es sich um ein unbedingtes (d.h. von der Wichtung *REQUIRED*) Constraint handelt einen Fehler. Im anderen Fall werden alle bisherigen Constraints zurückgenommen, die die Ausgabevariable *outvariable* bisher bestimmt haben. Anschließend wird die Methode ausgewertet. Alle zurückgenommenen Constraints *oldconstraints* werden dann, falls dies möglich ist, in anderer Richtung ausgewertet, wodurch die aktuelle Änderung durch das Netz propagiert wird (s. Abb 5.2).

5.1.2 FIDOS

Mittels dem abstrakten Algorithmus FIDOS (s. Abb. 5.3) werden, speziell für den Layoutprozeß, Einheiten durch globale Beziehungen an andere Einheiten gebunden.

[3]Bei der Verarbeitung dieser Constraints kann, im Gegensatz zu herkömmlichen logischen Programmiersprachen, auf ein Unifikation verzichtet werden, da die Instantiierung der Parameter mit den entsprechenden Variablen der Layoutobjekte genau bestimmt ist. Jeder Variablen ist genau ein Parameter eines Constraints zugeordnet, was bedeutet, daß die Reihenfolge, mit der die Constraints instantiiert werden, von größter Wichtigkeit ist.

[4]Eine Beziehung heißt dabei *gelöst*, wenn das assoziierte Constraint im Kontext des aktuellen Constraint-Netzes in konsistenter Weise durch den Constraint-Formalismus ausgewertet werden kann.

```
SIVAS ( lokrel )
    constraint = BUILDUPCONSTRAINT ( lokrel );
    INCREMENTALLYSATISFY ( constraint );


INCREMENTALLYSATISFY ( constraint )
    method = SELECTMETHOD ( constraint );
    if method == error
        exit ( error )
    endif
    if constraint-type ( constraint ) ≠ compound
        outvariable = OUT-VARIABLE ( method );
        if Strength ( outvariable ) ≥ Strength ( constraint )
            if Strength ( constraint ) ≠ *REQUIRED*
                exit ( error );
            else
                exit ( t );
            endif
        else
            oldconstraints = Ancestors ( outvariable );
            RETRACT ( overriddenconstraints );
            SATISFY ( method );
            loop oc oldconstraints
                INCREMENTALLYSATISFY ( oc );
            endloop
        endif
    endif
endif
```

Abbildung 5.2: Abstrakter Algorithmus SIVAS

Der Algorithmus hat als Grundlage Aspekte des *Forward-Checkings* aus *CHIP* integriert, die den Suchraum a priori einschränken können. Als Eingabe dient die Liste der zu plazierenden Einheiten *unitlist* sowie dem Dokument entsprechende Dokumenttyp *docutype*. Die Hauptschleife läuft dabei solange über die Liste der Einheiten, bis entweder alle Einheiten mittels globalen Constraints in das aktuelle Constraint-Netz eingebunden sind, oder eine Inkonsistenz zu einer Ausnahmebehandlung führt. Ist eine Einheit noch nicht eingebunden, wird, entsprechend dem Dokumenttyp, ein globales Constraint der Heuristik *heuristic* entnommen.

Für den Fall, daß der Einheit auf dem Raster keine feste Position zugeordnet werden soll, d.h. daß die Einheit nicht vom Typ *FIXED* ist, wird diese an eine andere Einheit *connector* mittels dem globalen Constraint angebunden. Dazu wird die normiert Form des Constraints durch die Funktion *BUILDUPFDCONSTRAINT* erzeugt und durch die Funktion *EXECCONSTRAINT* ausgewertet. In dieser Funktion (s. Abb. 5.4) werden alle Beziehungen zu anderen Einheiten von *unit1* entfernt, die bereits durch *unit2* bestehen. Danach wird das Constraint in Richtung *unit1* ausgewertet und anschließend die Konnektoren angepaßt und die Änderungen durch die Propagierungskomponente an das Netz weitergeleitet. Nachdem das Constraint in Richtung der Einheit *unit2* ausgewertet worden ist, werden auch diese Änderungen propagiert. Zum Abschluß werden alle Einheiten an den Seiten blockiert, an die eine der beiden Einheiten direkt angrenzt und wodurch keine weitere Einheit dazwischen plaziert werden kann.

In der Propagierungskomponente werden alle Einheiten neu berechnet, die direkt mit einer der beiden Einheiten durch ein globales Constraint in Beziehung stehen. Dazu werden alle die Constraints, die zu diesen Einheiten gehören und seit der letzten Anwendung eines Constraints zwischen *cu* und *unit* hinzugekommen sind, neu ausgewertet. Dabei ist auf die Richtung zu achten, in die das Constraint ausgewertet worden ist, was am Modus des Constraint zu erkennen ist.

```
FIDOS ( unitlist, docutype )
    heuristic = SELECTHEURISTIC ( docutype );
    loop until unitlist
        constname = NEXTCONSTRAINT ( heuristic );
        while ( not solved ) and ( constname )
            connector = NEXTCONNECTOR ();
            if Type ( unit ) == FIXED
                FIXUNIT ( unit );
                exit ( t );
            else
                while connector and ( not solved )
                    constraint = BUILDUPFDCONSTRAINT
                                 ( constname );
                    solved = EXECCONSTRAINT
                             ( constraint connector unit );
                    connector = NEXTCONNECTOR();
                endwhile
            endif
        endwhile
    endloop
```

Abbildung 5.3: Abstrakter Algorithmus FIDOS

```
EXECCONSTRAINT ( constraint unit1 unit2 )
    REMOVEUNITDEPENDENCIES ( unit1 unit2 );
    if ( not EXECONMETHOD ( unit1 unit2 ))
        exit ( error );
    endif
    UPDATECONNECTORS( constraint );
    PROPAGATION ( unit2 );
    if ( not EXECBACKMETHOD ( constraint ))
        exit ( error );
    endif
    PROPAGATION ( unit1 );
    LOCKASSOCUNITS ( unit1 unit2 );


PROPAGATION ( unit )
    cunits = CONNECTEDUNITS ( unit );
    loop cu cunits
        constraintlist = SELECTLASTOCCUR ( cu unit );
        loop constraint constraintlist
            mode = SELECTMODE ( constraint );
            if mode == ON
                EXECONMETHOD ( constraint );
            else
                EXECBACKMETHOD ( constraint );
            endif
        endloop
        PROPAGATION ( cu );
    endloop
```

Abbildung 5.4: Abstrakte Algorithmen zur Lösung und Propagierung eines Constraints in FIDOS

(defstruct constraint	
(name nil)	Constraint-Name
(type nil)	*Compound* | *Primitive*
(strength *AbsoluteWeakest*)	Wichtung des Constraints
(methods *none*)	Liste der CMethoden
(variables)	Benutzte Ein- und Ausgabevariablen
(whichMethod))	Aktuell verwendete CMethode
(defstruct cmethod	
(code)	Berechnungsteil
(inputs)	Eingabevariablen
(outputs))	Ausgabevariablen
(defstruct variable	
(name nil)	Name der Variablen
(value nil)	aktuelle Belegung
(constraints nil)	Liste der Constraints in denen die Variable verwendet wird
(walkStrength *AbsoluteWeakest*)	Wichtung der Variablen
(usesMe *LeereMenge*)	Liste der Constraints die Variable als Eingabevariable verwenden
(isDeterminedBy *LeereMenge*)	Name des Constraints, welches die aktuelle Belegung bestimmt
(ancestors nil)	Liste der Variablen, die im Constraint *isDeterminedBy* als Eingabe dienen
(isfixedto nil))	Fixierung zu bestimmter abs. Position

Abbildung 5.5: Strukturen *Constraints*, *CMethod* und *Variable* in SIVAS

5.2 Strukturen und Schnittstellenfunktionen

5.2.1 Constraints und Strukturen in SIVAS

Die wichtigsten Strukturen in SIVAS sind die für Constraints, die entsprechenden Methoden und die Variablen (s. Tabelle5.5),wobei diese Strukturen intern in SIVAS verwendet werden.

```
(defstruct FDConstraint
( Name nil )                    Name des Constraints
( Type nil )                    Typ
( XOnMethod nil )               Methode in X-Hin-Richtung
( YOnMethod nil )               Methode in Y-Hin-Richtung
( XBackMethod nil )             Methode in X-Rückrichtung
( YBackMethod nil )             Methode in Y-Rückrichtung
( Variables nil ))              Liste der verwandten Variablen

(defstruct FDVariable
( Name nil )                    Variablenname
( BlockName nil )               Name der assoziierten Einheits
( Domain nil )                  Belegung
( UsedVars nil )                Eingabevariablen zur Belegung
( UsedMethods nil ))            Zur Belegung verwandte Methoden

(defstruct FDBlock
( Name nil )                    Name der Einheit
( XName nil )                   Name der X-Variablen
( YName nil )                   Name der Y-Variablen
( HName nil )                   Name der H-Variablen
( BName nil )                   Name der B-Variablen
( Type 'COMMON )                Typ
( IsConstrainedWith nil ))      Liste der angewandten Constraints
```

Abbildung 5.6: Strukturen *FDConstraint*, *FDVariable* und *FDBlock* in FIDOS

5.2.2 Constraints und Strukturen in FIDOS

Es sollen jetzt die wichtigsten internen Strukturen, die in *FIDOS* in der Verarbeitung von globalen Constraints Verwendung finden, vorgestellt werden (s. Abb. 5.6) :

Wichtig ist dabei, daß die Constraint-Struktur stark auf die Domäne der automatischen Layoutgenerierung abgestimmt ist. Soll der Constraint-Solver FIDOS auf andere Problembereiche angewandt werden, so brauchen nur die Strukturmerkmale XOnMethod, YOnMethod, XBackMethod und YBackMethod in nur eine Hin- und eine Rückmethode umgewandelt werden, so erhält man einen Solver für normale binäre Constraints im Sinne von A. Mackworth (vgl. [Mackworth 77]).

5.2.3 Schnittstellenstrukturen

Die Schnittstellenfunktion *LayoutControl* zum System CLAY wird wesentlich durch den Übergabeparameter *Relationenliste* bestimmt. In diesem sind die lokalen Beziehungen (s. Struktur *Relation*) mit den entsprechenden Layoutobjekten (s. Struktur

(defstruct Relation	
(Name nil)	Name der lokalen Beziehung
(Objekte nil)	Liste der assoziierten Layoutobjekte
(IsFixedTo nil)	Fixierung, falls definiert
(ConnectedBy nil)	Name der verbundenen Einheit
(ConnectedWith nil))	Name der a priori verbundenen Einheit
(defstruct Objekt	
(Name nil)	Layoutobjektname
(Typ nil)	*Graphik* oder *Text*
(XKoordinate 0)	X-Koordinate der linken, oberen Ecke
(XWalkStrength *VERYWEAK*)	Wichtung der Bindung
(YKoordinate 0)	Y-Koordinate der linken, oberen Ecke
(YWalkStrength *VERYWEAK*)	Wichtung der Bindung
(Hoehe nil)	Höhe
(Breite nil))	Breite
(defstruct gframe	
(Name nil)	Name des das Layoutobjekt umschreibende Rechtecks
(x-point nil)	X-Koordinate der linken, oberen Ecke
(y-point nil)	Y-Koordinate der linken, oberen Ecke
(hoehe nil)	Höhe
(breite nil))	Breite

Abbildung 5.7: Definition der Übergabestrukturen

Objekt) aufgelistet. Von den Objekten sind a priori die Höhe und die Breite bekannt (s. Abb. 5.7).

5.2.4 Schnittstellenfunktion

Als Schnittstellenfunktion des Systems CLAY dient die bereits erwähnte Funktion *LayoutControl*, deren Aufruf sich wie folgt gestaltet:

(LayoutControl GridX GridY RelationenListe DocuType)

Die Parameter *GridX* und *GridY* bestimmen die Größe des Rasters in Form von positiven natürlichen Zahlen. Der dritte Parameter entspricht der in Abschnitt (s. Abb. 5.2.3) beschrieben Liste. Abschließend enthält *DocuType* Informationen über den zugrundeliegenden Dokumenttyp.

Die Ausgabe des Systems besteht ebenfalls aus einer einfachen linearen Liste von Strukturen der Art *gframe*, die die topologischen Angaben für die einzelnen Objekte enthält, welche mit den Layoutobjekten assoziiert sind. Diese Liste wird verwendet, um mittels dem Typ des Layoutobjektes entsprechenden Prozeduren die Informationen modusspezifisch auf der Dokumentfläche auszugeben.

5.3 Layouttestumgebung CLAY

Die CLAY -Testumgebung bietet Möglichkeiten die Umsetzung semantisch-pragmatischer sowie globaler Beziehungen in entsprechende Constraints und deren Zusammenspiel zu testen. CLAY besitzt Eingabemöglichkeiten für Layoutobjekte, Auswahlmöglichkeiten für verschiedenen Dokumenttypen und unterschiedliche Heuristiken für die Auswertung globaler Beziehungen.

Die in diesem Abschnitt beschriebenen Eigenschaften des Systems werden durch Bildschirmabzüge im Anhang Systembeschreibung ergänzt.

5.3.1 Initialzustand

Das System kann auf den Symbolics-Lisp-Maschinen in einfacher Weise durch den folgenden Aufruf gestartet werden:

load system clay

Danach ist es möglich durch die Tastenkombination $< SELECT >< Y >$ jederzeit zur Testumgebung zu wechseln. Danach bietet sich einem die Arbeitsumgebung wie sie in Abbildung 6.16 gezeigt wird.

5.3.2 Lokale Constraints

Hat der Benutzer über den Menüpunkt *Documenttype* einen passenden Dokumenttyp ausgewählt, so kann man über den Menüpunkt *Local Relations* für lokale Beziehungen passende Layoutobjekte definieren.

Abbildung 6.17 zeigt die Auswertung der lokalen Beziehung *Kontrast* und Abbildung 6.18 die der lokalen Beziehung *Sequenz.*

5.3.3 Globale Constraints

Die Steuerung der globalen Constraints kann über einen entsprechenden Eintrag während der Zusammenstellung einer Einheit festgelegt werden oder aber über den Menüpunkt *Global Relations* bestimmt werden. Dazu kann der Benutzer verschiedene vordefinierte Heuristiken auswählen.

5.3.4 Graphische und textuelle Darstellung der Constraints

Zur Beschreibung der im System verwandten Constraints ist der Menüpunkt *Description* geeignet. Damit können Informationen über die Definition der einzelnen lokalen wie globalen Constraints in textueller wie graphischer Art (in Form eines Graphen) ausgegeben werden. Hierzu sei auf die Abbildungen 6.21, ?? und 6.23 verwiesen.

Zusätzlich können die Belegungen der Variablen von SIVAS und FIDOS sowie die Ausführungsliste (s. Abb. 6.32) angezeigt werden. Die Ausführungsliste ist ein Art von Erklärungsausgabe, die es erlaubt die ausgeführten Schritte zur Erstellung des Layouts nachzuvollziehen.

5.3.5 Dokumenttypen und Layout mehrerer Seiten

Hat man die Layoutobjekte mit den entsprechenden Beziehungen zusammengestellt, so kann man sich das passende Layout auf verschiedenen Dokumenttypen anzeigen lassen (s. Abb. 6.24,6.26,6.25 und 6.27).

Neben dem Layout von nur einer Dokumentseite ist das System auch in der Lage das Layout mehrerer Seiten zu generieren. In diesem Fall kann über den Menüpunkt *Select Page* eine gewünschte Dokumentseite ausgewählt werden. Siehe hierzu die Abbildungen 6.28, 6.30, 6.29 und 6.31, die das Layout mit und ohne Darstellung des Rasters zeigen.

Kapitel 6

Beispiel einer wissensbasierten Layout-Generierung

Es soll nun an Hand eines Beispiels die Arbeitsweise des Systems CLAY erklärt werden. Für die verschiedenen Module des Systems seien die in Tabelle 6.1 beschriebenen Layoutobjekte gegebenen, welche durch die jeweiligen Generatoren erzeugt worden sind. Die im folgenden verwendeten Koordinaten, Höhen und Breiten der Layoutobjekte sind in Rasterkoordinaten angegeben, d.h. daß eine Einheit die Höhe bzw. Breite eines universellen Rasterfeldes beträgt. Dabei sind der Übersichtlichkeit wegen, hauptsächlich Layoutobjekte der Höhe und Breite 1 verwendet worden. Die vorgegebenen semantisch-pragmatischer Beziehungen sind unter der Rubrik *Lokale Beziehung* vermerkt. Die Layoutobjekte, die dadurch verbunden werden sollen, bilden eine gemeinsame Zeile. Das Ziel des Beispiels ist es die vorgegebenen Layoutobjekte, unter Berücksichtigung der lokalen Beziehungen, auf dem Raster eines technischen Dokumentes (Bedienungsanleitung) im Hochformat anzuordnen.

6.1 Lokale Beziehungen

Wie in Kapitel 4.1 beschrieben, werden durch lokale Beziehungen Layoutobjekte zusammengefaßt, die einen engen inhaltlichen Zusammenhang aufweisen. Der Plazierungsprozeß muß diese, durch lokale Beziehungen zu Einheiten zusammengebundenen Objekte, auf dem Dokument der Relation entsprechend anordnen. Es stehen hierzu i.A. eine Reihe von alternativen Darstellungsformen der lokalen Beziehungen zur Verfügung. Für einen Graphik-Text-Block gilt dabei, daß die übliche Verwendung die Graphik links von oder alternativ über den zugeordneten Text zu plazieren, gegeben ist. Text hingegen ist eine variable Beziehung, die sich den topologischen Gegebenheiten anpaßt und horizontal oder vertikal ausgerichtet werden kann[1]. Um in dieser initialen Phase des Prozesses Werte in die jeweiligen Constraint-Netze der

[1] Die Formatierung eines Textblocks wird von einer noch in der Entwicklung befindlichen Komponente für Typographie übernommen

Lokale Bez.	Layoutobjekt	Höhe	Breite
G-T-Block	Graphik	1	1
	Text	1	1
G-T-Block	Graphik	1	1
	Text	1	1
SEQUENZ	Graphik	1	1
	Text	1	1
	Graphik	1	1
	Text	1	1
	Graphik	1	1
	Text	1	1
TEXT	Text	2	1

Abbildung 6.1: Tabelle der lokalen Beziehungen mit assoziierten Layoutobjekten

$$\text{G-T-Block}\ (\ B_1B_2\ ) \leftrightarrow$$
$$((\text{CONNECT}\ \text{X}(B_1)\ \text{B}(B_1)\ \text{X}(B_2)) \wedge (\text{EQUAL}\ \text{Y}(B_1)\ \text{Y}(B_2))) \vee$$
$$((\text{CONNECT}\ \text{Y}(B_1)\ \text{H}(B_1)\ \text{Y}(B_2)) \wedge (\text{EQUAL}\ \text{X}(B_1)\ \text{X}(B_2)))$$

Abbildung 6.2: Prädikatenlogische Definition des statischen aggregierten Constraints *G-T-Block*

lokalen Beziehungen propagieren zu können, werden die X- und Y-Koordinaten auf die Werte 0 gesetzt. Dadurch wird eine topologische Anordnung der Layoutobjekte relativ zur linken, oberen Ecke der Einheit erreicht.

Es soll am Beispiel der ersten lokalen Beziehung die Verwendung von statischen, aggregierten Constraints und an der dritten die von dynamischen, aggregierten Constraints demonstriert werden.

6.1.1 Statische lokale Beziehungen

Die erste lokale Beziehung *G-T-Block* verbindet eine Graphik mit einem zugeordneten Text. Die Verwendung der ersten Methode zur Bestimmung einer topologischen Anordnung setzt die beiden Layoutobjekte in horizontaler Ebene nebeneinander, wobei die Graphik links positioniert wird (s. Abb. 6.3). Das dabei verwendete statische aggregierte Constraint *G-T-Block* verknüpft dabei die Variablen der Layoutobjekte, wie es von der prädikatenlogischen Form (s. Abb. 6.2) vorgegeben wird[2].

In gleicher Wiese werden die Layoutobjekte der zweiten lokalen Beziehung miteinander verknüpft.

[2]Die Parameter B_1B_2 und die Zugriffsfunktionen X(B), Y(B), H(B) und B(B) sind dabei in der von Kapitel 4.1 gewohnten Weise zu verstehen

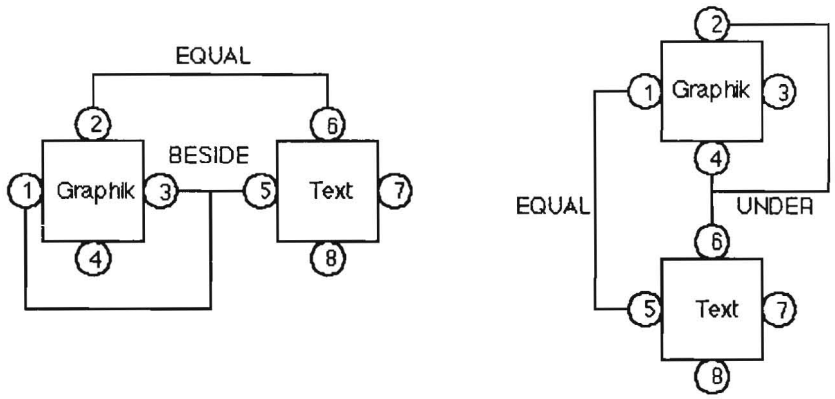

Abbildung 6.3: Alternative Darstellungen eines Graphik-Text-Blockes

6.1.2 Dynamische lokale Beziehungen

Die Anwendung der dritten lokalen Beziehung *Sequenz* erfordert die Verwendung eines dynamischen aggregierten Constraints. Dazu wird, entsprechend der Grundform und der Anzahl dieser in der betrachteten lokalen Beziehung, ein Constraint generiert. Betrachtet man eine Sequenz, so besteht die Grundform aus einer Graphik mit einem zugeordneten Text. Aus dieser Grundform läßt sich die gesamte Beziehung aufbauen, wobei die Anzahl dieser Grundformen nicht festgelegt ist (deshalb dynamisch). Das Constraint besteht zum einen aus Constraints, die Layoutobjekte innerhalb einer Grundform verbinden und zum anderen aus solchen, die Layoutobjekte zwischen Grundformen verbinden. In Abbildung 6.4 ist illustriert, wie dies im Fall der *Sequenz* zu verstehen ist, wobei zwei Grundformen miteinander in Verbindung gesetzt worden sind[3].

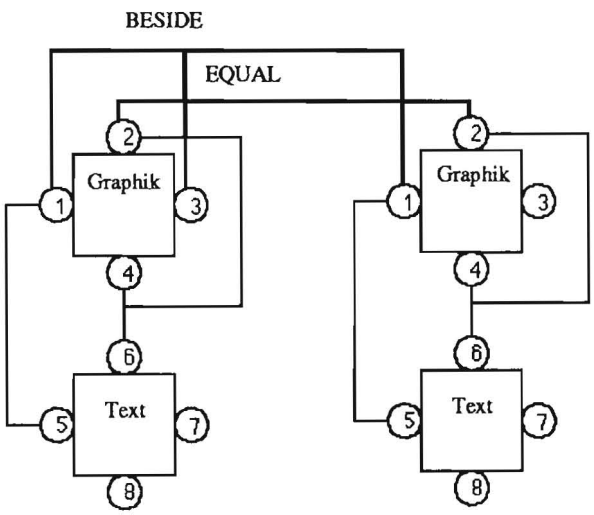

Abbildung 6.4: Beispiel einer dynamischen lokalen Beziehung: Eine 2-er Sequenz

Die prädikatenlogische Form des dynamischen aggregierten Constraints *Sequenz* entspricht der Definition von Abbildung 4.1.3.

[3]Die schmalen Kantenentsprechen dabei Constraints innerhalb einer Grundform und stärkere Kanten solchen zwischen den Grundformen

Einheit	lokale Bez.	X-Wert	Y-Wert	Höhe	Breite
E1	G-T-Block	0	0	1	2
E2	G-T-Block	0	0	1	2
E3	SEQUENZ	0	0	3	2
E4	TEXT	0	0	2	1

Abbildung 6.5: Tabelle der Einheiten

6.2 Globale Beziehungen

Nachdem die lokalen Beziehungen zwischen den Objekten hergestellt worden sind, müssen die so entstandenen Einheiten (s. Abb. 6.5) auf dem Raster plaziert werden. Die einzelnen Einheiten werden im folgenden nacheinander mittels globaler Beziehungen mit anderen Einheiten verbunden, so daß sich nach und nach durch gegenseitige Einengung ein Layout herauskristallisiert.

Die Vorgehensweise bei der Anwendung von globalen Beziehungen, die in der Regel nicht vorgegeben sind, ist durch die Art des Dokumentes bestimmt. Für technische Dokumente im Hochformat ist es üblich von oben links nach unten rechts vorzugehen. Daraus folgt eine Heuristik, die anzuordnende Einheiten zuerst *untereinander*, falls dies nicht möglich ist, *rechtvon* einer anderen Einheit anzubinden. Die Einheit wird mit einer anderen Einheit verbunden, die an der unteren Seite einen freien Konnektor besitzt. Da im Fall der ersten Einheit, die auf einer Dokumentseite zu plazieren ist noch keine plazierte Einheit vorhanden ist, ist das Konzept der *virtuellen Einheiten* eingeführt worden, welche sich rund um das Raster befinden. Da die Einheit nicht über ein explizit vorgegebenes globales Constraint gebunden werden muß, wird eins aus der Heuristik entnommen, was ebenfalls reihenfolgenabhängig ist. In diesem Fall ist dies das globale Constraint *unter*. Die zum globalen Constraint *unter* passende virtuelle Einheit, ist die, welche über dem Raster positioniert ist. An diese wird die erste Einheit *E1* angebunden (s. erste Graphik in Abb. 6.6).

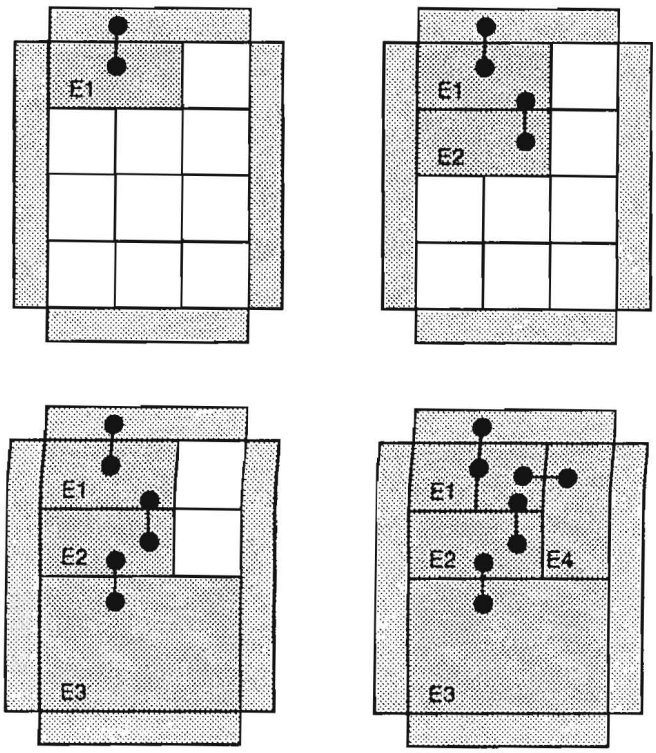

Abbildung 6.6: Plazierung von Einheiten auf einem Raster mit eingezeichneten globalen Constraints

Für die Einheit *E1* steht nach wie vor die gesamte Dokumentfläche als potentielle Position zur Verfügung (s. Tabelle 6.7).

Die Einfügung der zweiten Einheit unterliegt den gleichen Gesichtspunkten wie Einheit *E1*, nur ist bereits eine andere Einheit, nämlich *E1*, auf dem Raster plaziert. Da die Einheit *E1* bis jetzt noch frei auf dem Raster plazierbar ist, mit der einzigen Restriktion, daß *E1* sich unter der oberen virtuellen Einheit befinden soll, was auf einem Rechteck immer gegeben ist, muß der für *E1* freie Platz so eingeschränkt werden, um der Einheit *E2* Platz auf dem Raster zuzuordnen (s. 2. Graphik Abb. 6.6 und Tabelle 6.8).

Nach dem gleichen Schema wird die dritte Einheit *E3* plaziert, wobei eine Propagierungskomponente gewährleistet, daß Einschränkungen auf den Domänen der Einheit *E2* an die Einheit *E1* weitergeleitet werden (s. 3. Graphik in Abb. 6.6 und Tabelle 6.9). Nachdem die dritte Einheit eingefügt worden ist, liegt diese in ihren Koordinaten fest, wodurch dieser Bereich des Rasters als gesperrt markiert wird.

Für die vierte Einheit *E4* besteht keine Möglichkeit unterhalb eines anderen Blocks

Einheit	Variable	Belegung
E1	X	$\{0,1\}$
	Y	$\{0,1,2,3\}$

Abbildung 6.7: Belegung der Variablen von Einheit E1

Einheit	Variable	Belegung
E1	X	$\{0,1\}$
	Y	$\{0,1,2\}$
E2	X	$\{0,1\}$
	Y	$\{1,2,3\}$

Abbildung 6.8: Belegung der Variablen der Einheiten E1 und E2

Einheit	Variable	Belegung
E1	X	$\{0,1\}$
	Y	$\{0\}$
E2	X	$\{0,1\}$
	Y	$\{1\}$
E3	X	$\{0\}$
	Y	$\{2\}$

Abbildung 6.9: Belegung der Variablen der Einheiten E1, E2 und E3

plaziert zu werden, somit wird die nächst Alternative, was im Beispiel das globale Constraint *rechtsvon* ist, ausgewählt. Mittels diesem Constraint wird ein freier Platz an der rechten Seite einer anderen Einheit gesucht, was mit der Einheit *E1* Erfolg hat (s. 4. Graphik in Abb. 6.6 und Tabelle 6.10).

Somit ist die Plazierung für dieses Problem gelöst, was in Tabelle 6.11 dokumentiert ist. Die Koordinaten der Einheiten werden hierzu an die Layoutobjekte weiterpropagiert so, daß jedes Layoutobjekt seinen absoluten Platz findet. Dazu werden auf die Constraint-Netze, die in Abschnitt 6.1 konstruiert worden sind, *SET-* Constraint mit den Werten der Variabeln, die durch die Prozedur der globalen Beziehungen bestimmt worden sind, angewandt.

Einheit	Variable	Belegung
E1	X	$\{0\}$
	Y	$\{0\}$
E2	X	$\{0\}$
	Y	$\{1\}$
E3	X	$\{0\}$
	Y	$\{2\}$
E4	X	$\{2\}$
	Y	$\{0\}$

Abbildung 6.10: Belegung der Variablen der Einheiten E1, E2, E3 und E4

Lokale Bez.	Layoutobjekt	X-Koord.	Y-Koord.	Höhe	Breite
E1	Graphik	0	0	1	1
	Text	1	0	1	1
E2	Graphik	0	1	1	1
	Text	1	1	1	1
E3	Graphik	0	2	1	1
	Text	0	3	1	1
	Graphik	1	2	1	1
	Text	1	3	1	1
	Graphik	2	2	1	1
	Text	2	3	1	1
E4	Text	2	0	2	1

Abbildung 6.11: Absolute Koordinaten der Layoutobjekten

6.3 Reformatierung

Bei dem vorgestellten Problemfall konnten die Layoutobjekte immer ohne Änderungen plaziert werden. Dies ist aber nicht immer der Fall. Beispielsweise kann man eine Dreier-Sequenz nicht horizontal auf einem Raster palzieren, welches nur zwei Rasterfelder breit und drei hoch ist. Unter diesem Umständen ist man gezwungen Reformatierungen an seinen Objekten vorzunehmen. Im Fall der Sequenz wird diese nicht mehr horizontal, sondern vertikal ausgelegt, wodurch sie im Beispiel genau auf das Raster passen würde.

Betrachtet werde dazu das oben eingeführte Beispiel (s. Tabelle 6.1). Das vorgegebene Raster sei zwei Rasterfelder breit und sechs Rasterfelder hoch. Die beiden Graphik-Text-Einheiten können hierbei ohne Probleme mittels der Heuristik plaziert werden. Im dritten Schritt wird die Sequenz verarbeitet, die in ihrer horizontalen Version nicht auf das Raster paßt. In diesem Fall wird die Einheit von der Prozedur *FIDOS* zurückgewiesen und in *SIVAS* reformatiert, wodurch man im Beispiel der Sequenz eine vertikalorientierte Topologie der Einheit erhält. In ähnlicher Weise wird im vierten Schritt Einheit E4 reformatiert (s. Abb. 6.12). Durch diese Reformatierungskomponente ist es möglich gleiche Objekte an die geometrischen Ausmaße des gegebenen Ausgabemediums anzupassen.

Die einzelnen Schritte, die bei der Plazierung im Beispiel gemacht werden, sind in der folgenden Tabelle zusammengefaßt[4].

[4]BU steht für die obere, virtuelle Einheit

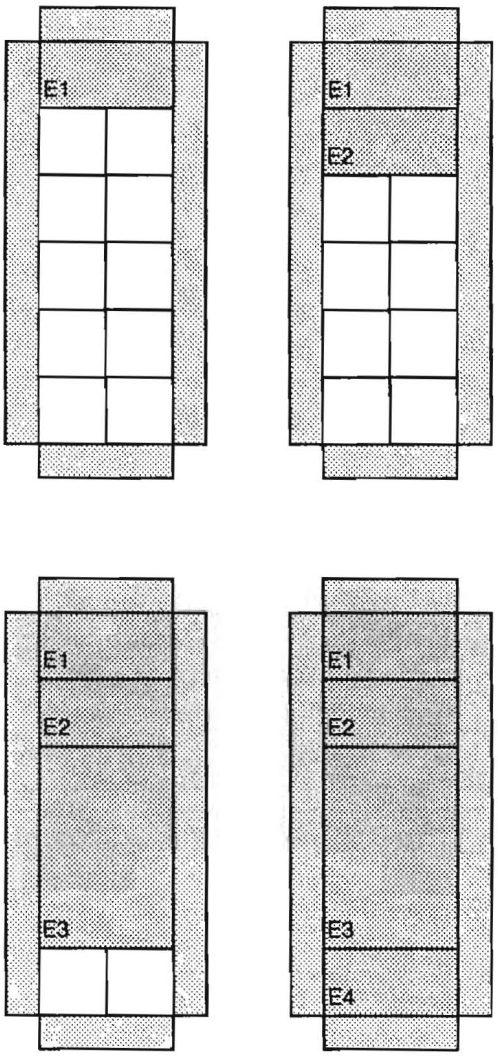

Abbildung 6.12: Plazierung mit Reformatierung der Einheiten E3 und E4

Schritt	Aktion	Resultat
1	UNTER BU E1	Erfolg
2	UNTER E1 E2	Erfolg
3	UNTER E2 E3	*Fehler*
4	REFORMAT E3	Erfolg
5	UNTER E2 E3	Erfolg
6	UNTER E3 E4	*Fehler*
7	REFORMAT E4	Erfolg
8	UNTER E3 E4	Erfolg

6.4 Reformatierung mit Heuristik

Im obigen Beispiel war es bisher nicht möglich Einheiten durch verschiedene globale Beziehungen in das Constraint-Netz zu integrieren. In diesem Beispiel, welches wiederum die Layoutobjekte aus Tabelle 6.1 verwendet, werden alle globalen Con-

straints auf Erfolg getestet. Dieser Versuch wird immer gemacht, jedoch hatte dieser bei den obigen Beispielen keinen Zweck, weswegen auf eine Demonstration bisher verzichtet wurde. Die alternative globalen Constraints werden nun solange getestet, *bis entweder eines eine konsistente Lösung ergibt, oder keine Alternative* mehr zur Verfügung steht. Im letzteren Fall wird die Einheit wie bisher an die Prozedur *SIVAS* zurückgegeben und in bekannter Weise reformatiert.

Als Fläche liegt ein vier-mal-drei-großes Rasterfeld vor. Die Heuristik schreibt eine Untereinander-Anordnung, mit alternativer Rechtsvon-Anordnung vor.

Die Plazierung der ersten Einheit E1 unter der oberen, virtuellen Einheit ist nach wie vor erfüllbar sowie die Anbindung der Einheit E2 unter E1. Im nächsten Schritt wird versucht die dritte Einheit E3 unter die Einheit E2 anzubinden, was jedoch scheitert. Das Gleiche passiert ebenfalls bei allen Versuchen, die Einheit E3 rechts an eine andere Einheit anzubinden, da E3 durch die standardmäßige, horizontale Ausrichtung zu breit ist. Somit wird E3 in eine vertikale Topologie reformatiert (s. Tabelle 6.14 Schritt 4). Die Anwendung des globalen Constraints *unter* führt auch diesmal zu einem Fehler (Schritt 5).Danach ist kein Konnektor mehr frei, der ein Anbinden auf der rechten Seite einer Einheit gestatten würde, so daß das nächste globale Constraint, in diesem Fall *rechtsvon* verwendet wird. Der Versuch Einheit E3 rechts an E1 anzubinden ist erfolgreich (Schritt 6).

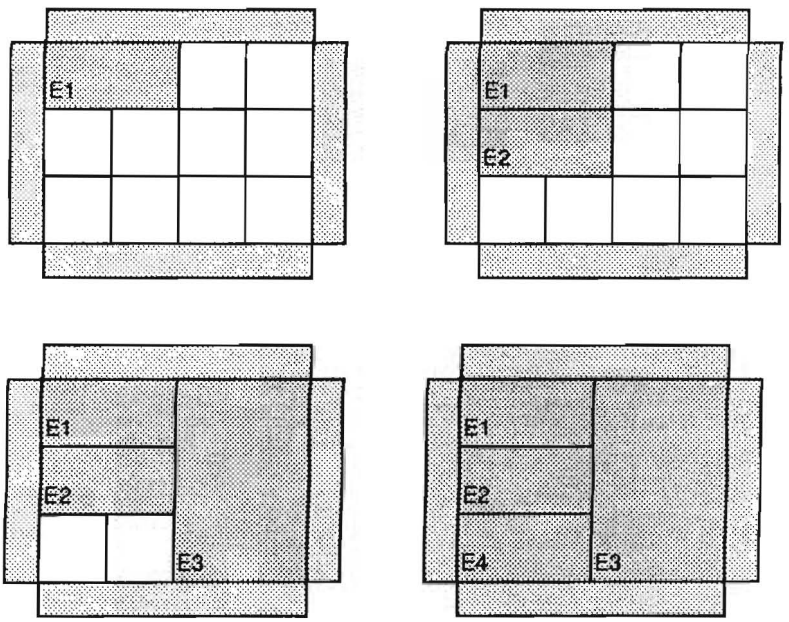

Abbildung 6.13: Plazierung mit Reformatierung und Heuristik-Verwendung

In gleicher Weise ist die Verwendung des globalen Constraints *unter* für Einheit E4 in der vertikalen Form nicht möglich, so daß auch diese, jedoch in eine horizontale Form, reformatiert werden muß (Schritt 7, 8). Anschließend kann sie durch das globale Constraint *unter* mit Einheit E2 verbunden werden.

Schritt	Aktion	Resultat
1	UNTER BU E1	Erfolg
2	UNTER E1 E2	Erfolg
3	UNTER E2 E3	*Fehler*
4	REFORMAT E3	Erfolg
5	UNTER E2 E3	*Fehler*
6	RECHTSVON E1 E3	Erfolg
7	UNTER E2 E4	*Fehler*
8	REFORMAT E4	Erfolg
9	UNTER E2 E4	Erfolg

Abbildung 6.14: Aktionsfolge des Beispiels

Abschließend zeigt Abbildung 6.15 das Layout der drei vorgestellten Beispiele ohne den Raster und die virtuellen Einheiten. Zusätzlich ist die gesamte Dokumentseite, einschließlich den Rändern, dargestellt.

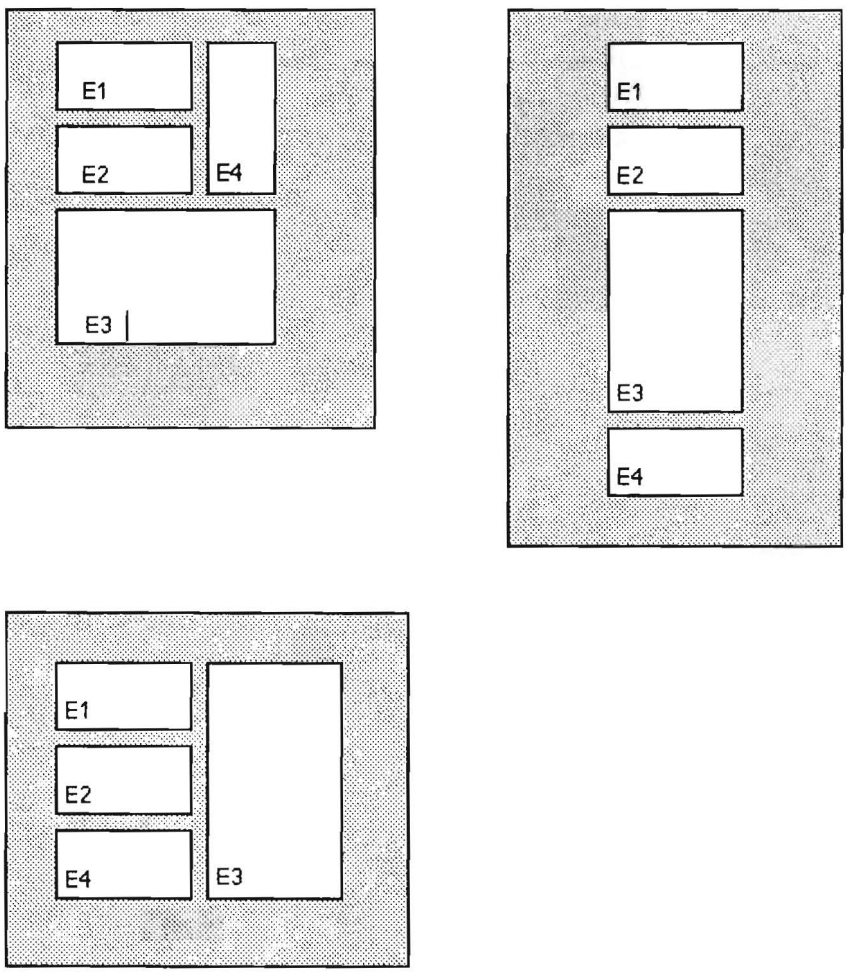

Abbildung 6.15: Layout der Beispiele

Literaturverzeichnis

[Alscher 68] L. **Alscher** (ed.). *Lexikon der Kunst.* Leipzig, 1968.

[Beach 85] R. **Beach**. *Setting Tables and Illustrations with Style.* PhD thesis, Dept. of Computer Science, University of Waterloo, Ontario, 1985.

[B.N.Freeman-Benson & Wilson 90] **B.N.Freeman-Benson** and M. **Wilson**. *DeltaStar, How I Wonder What You Are: A General Algorithm for Incremental Satisfaction of Constraint Hierarchies.* Technical Report 90-05-02, Departement of Computer Science and Engineering, University of Washington, Seattle, April 1990.

[B.N.Freeman-Benson et al. 88] **B.N.Freeman-Benson**, **J.Maloney**, and **A.Borning**. *The DeltaBlue Algorithm: An Incremental Constraint Hierarchy Solver.* Technical report, University of Washington, 1988.

[Borning & Duisberg 86] A. **Borning** and R. **Duisberg**. *Constraint-based tools for building user interfaces.* ACM Transactions on Graphics, 5(4):345–374, October 1986.

[Borning et al. 87] A. **Borning**, R. **Duisberg**, B. **Freeman-Benson**, A. **Kramer**, and M. **Woolf**. *Constraint Hierarchies.* In: Proceedings of OOPSLA '87, pp. 48–60, October 1987.

[Borning et al. 89] A. **Borning**, B. **Freeman-Benson**, and M. **Wilson**. *Constraint Hierarchies.* Department of Computer Science and Engineering, FR-35, University of Washington, 1989.

[Borning 79] A. **Borning**. *ThingLab - A Constraint-Oriented Simulation Laboratory.* PhD thesis, Dept. of Computer Science, Stanford University, Stanford, CA, 1979.

[Borning 81] A. **Borning**. *The Programming language aspects of ThingLab, a constraint-oriented simulation laboratory.* ACM Transactions on Programming Languages and Systems, 3(4):353–387, October 1981.

[Braun 87] G. **Braun** (ed.). *Grundlagen der visuellen Kommunikation.* München: Bruckmann, 1987.

[Christaller et al. 89] T. **Christaller**, F. **di Primio**, and A. **Voß** (eds.). *Die KI-Werkbank Babylon.* Bonn: Addison Wesley, 1989.

[Colmerauer 90] A. **Colmerauer**. *An introduction to PROLOG III.* Communications of the ACM, pp. 70–90, Juli 1990.

[Dincbas et al. 88] M. **Dincbas**, P. **van Hentenryck**, H. **Simonis**, A. **Aggoun**, T. **Graf**, and F. **Berthier**. *The constraint logic programming language CHIP.* In: Proceedings of the International Conferenceon Fifth Generation Computer Systems FGCS-88, pp. 693–702, Tokyo, Japan, Dezember 1988.

[Feiner 88] S. **Feiner**. *A Grid-Based Approach to Automating Display Layout.* In: Proceedings of the Graphics Interface, pp. 192–197, 1988.

[Freeman-Benson et al. 90] B. **Freeman-Benson**, J. **Maloney**, and A. **Borning**. *An Incremental Constraint Solver.* Communications of the ACM, 33(1):54–63, 1990.

[Gosling 83] J. **Gosling**. *Algebraic Constraints.* PhD thesis, Dept. of Computer Science, Carnegie Mellon University, 1983.

[Graf & Maaß 91] W. **Graf** and W. **Maaß**. *Constraint-basierte Verarbeitung graphischen Wissens.* In: Proceedings 4. Internationaler GI- Kongreß Wissensbasierte Systeme - Verteilte KI. Berlin, Germany: Springer-Verlag, October 1991.

[Graf 90] W. **Graf**. *Spezielle Aspekte des automatischen Layout- Designs bei der koordinierten Generierung von multimodalen Dokumenten.* GI-Workshop 'Multimediale elektronische Dokumente', Heidelberg, November 1990.

[Graf 91] W. **Graf**. *Constraint-Based Processing of Design Knowledge.* In: Proceedings of the AAAI-91 Workshop on Intelligent Multimedia Interfaces, Anaheim, CA, July 1991.

[Güsgen & Hertzberg 87] H.W. **Güsgen** and J. **Hertzberg**. *A Functional View on Constraints.* Arbeitspapier der GMD 252, Ges. für Mathematik und Datenverarbeitung, GMD, St. Augustin, Juni 1987.

[Güsgen 87] H.W. **Güsgen**. *CONSAT - A System for Constraint Satisfaction.* PhD thesis, Gesellschaft für Mathematik und Datenverarbeitung mbH, St. Augustin, 1987.

[Institut 82] Lexikographisches **Institut** (ed.). *Der Große Knaur.* Kirchheim bei München: R. Oldenbourg Graphische Betriebe GmbH, 1982.

[Jaffar et al. 87] J. **Jaffar**, S. **Michaylov**, P.J. **Stuckey**, and R.H.C. **Yap**. *The CLP(R) Language and System.* In: Proceedings of the 4th. International Conference on Logic Programming, Melbourne, Mai 1987.

[Lassez 87] C. **Lassez**. *Constraint Logic Programming.* BYTE, pp. 171–176, August 1987.

[Maaß et al. 91] W. **Maaß**, T. **Schiffmann**, and D. **Soetopo**. *LAYLAB: Ein System zur automatischen Plazierung in multimodalen Dokumenten.* Fortgeschrittenenpraktikum Wissensbasierte Graphikgenerierung, Dept. of Computer Science, University of Saarbrücken, 1991.

[Mackworth 77] A. **Mackworth**. *Consistency in Networks of Relations.* Artificial Intelligence, 8(1):99–118, 1977.

[Maloney et al. 89] J. **Maloney**, A. **Borning**, and B. **Freeman-Benson**. *Constraint Technology for User-Interface Construction in ThingLabII.* In: Proceedings of OOPSLA '89, pp. 381–388, October 1989.

[Mann & Thompson 88] W. **Mann** and S. **Thompson**. *Rhetorical Structure Theory: Towards a Functional Theory of Text Organization.* TEXT, 8(3), 1988.

[Müller-Brockmann 81] J. **Müller-Brockmann** (ed.). *Grid Systems in Graphic Design.* Niederteufen, Switzerland: Verlag Arthur Niggli, 1981.

[Navinchandra 91] D. **Navinchandra** (ed.). *Exploration and Innovation in Design.* Berlin, Germany: Springer-Verlag, 1991.

[Nelson 85] G. **Nelson**. *Juno, a constraint-based graphics system.* Proceedings of the SIGGRAPH '85, 19(3):235–243, 1985.

[Robinson 65] J.A. **Robinson**. *A machine-oriented logic based on the resolution principle.* Journal ACM, 12:23–41, 1965.

[Smolka 91] G. **Smolka**. *Residuation and Guarded Rules for Constraint Logic Programming.* Technical Report 12, Digital Paris Research Laboratory, Juni 1991.

[Sussman & Steele 80] G.J. **Sussman** and G.L. **Steele**. *CONSTRAINTS – A language for expressing almost-hierachical descriptions.* Artificial Intelligence, 14(1):1–39, 1980.

[Sutherland 63] I. **Sutherland**. *Sktechpad: A Man-Machine Graphical Communication System.* In: IFIPS Proceedings of the Spring Joint Computer Conference, pp. 329–345, 1963.

[Tanner 90] A. **Tanner**. *Layout 1.* Technical report, Tanner Dokuments KG, Juni 1990.

[van Hentenryck 89] P. **van Hentenryck** (ed.). *Constraint Satisfaction in Logic Programming.* Cambridge, MA: MIT Press, 1989. Revision of Ph.D. thesis, University of Namur, 1987.

[van Wyck 82] C.J. **van Wyck**. *IDEAL User's Manual.* Technical Report 103, Bell Laboratories, 1982.

[Wahlster et al. 91a] W. **Wahlster**, E. **André**, S. **Bandyopadhyay**, W. **Graf**, and T. **Rist**. *WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation.* In: Computational Theories of Communication and their Applications. Berlin, Germany: Springer-Verlag, 1991. Also DFKI Research Report RR-91-08.

[Wahlster et al. 91b] W. **Wahlster**, E. **André**, W. **Graf**, and T. **Rist**. *Designing Illustrated Texts: How Language Production Is Influenced by Graphics Generation.* In: Proceedings of the 5th Conference of the European Chapter of the Association for Computational Linguistics, pp. 8–14. Berlin, Germany: Springer-Verlag, April 1991. Also DFKI Research Report RR-91-05.

Constraints in SIVAS

Definition des *SIVAS*-Constraints **CONNECT**

Semantik:

CONNECT verbindet 3 Parameter in der Form miteinander, daß die Beziehung ?1 =?2+?3 gilt.

Definition:

```
(DefSIConstraint CONNECT
    (((+ ?2 ?3))
    ((- ?1 ?3))
    ((- ?1 ?2))))
```

Definition des *SIVAS*-Constraints **EQUAL**

Semantik:

EQUAL belegt die jeweils freie Variable mit dem Wert der anderen.

Definition:

```
(DefSIConstraint EQUAL
    (((?2))
    ((?1))))
```

Definition des *SIVAS*-Constraints **IF-LESS-EQUAL**

Semantik:

Dieses Constraint kann zur Kontrolle verwendet werden. Es testet ob zwei Belegungen in einer Kleinergleich-Beziehung zueinander stehen.

Definition:

```
(DefSIConstraint IF-LESS-EQUAL
    (((cond ((≤ ?2 ?3)
    ?2)
    (t error)))))
```

Definition des *SIVAS*-Constraints **TEXT-GRAPHIK**

Semantik:

TEXT-GRAPHIK verbindet 8 Variablen miteinander durch zwei alternative Weisen. Für das Layout bedeutet dies, daß ein Graphik-Layoutobjekt und ein Text-Layoutobjekt entweder nebeneinandeer oder übereinander stehen.

Definition:

```
(DefSIConstraint TEXT-GRAFIK
    (((CONNECT ?5 ?4 ?1)
    (EQUAL ?6 ?2))

    ((CONNECT ?6 ?2 ?3)
    (EQUAL ?5 ?1))))
```

Definition des *SIVAS*-Constraints **KONTRAST**

Semantik:

KONTRAST verbindet 8 Variablen miteinander durch zwei alternative Weisen. Für das Layout gedeutet dies, daß zwei beliebige Layoutobjekte entweder horizontal oder vertikal miteinander verknüpft werden.

Definition:

```
(DefSIConstraint KONTRAST
    (((CONNECT ?5 ?4 ?1)
    (EQUAL ?6 ?2))

    ((CONNECT ?6 ?2 ?3)
    (EQUAL ?5 ?1))))
```

Definition des *SIVAS*-Constraints **TEXT**

Semantik:

TEXT ist eine leeres lokales Constraint, da im Layout ein freistehender Text lokal ungebunden ist.

Definition:

```
(DefSIConstraint TEXT
    ((())))
```

Definition des *SIVAS*-Constraints **GRAPHIK**

Semantik:

Für *GRAPHIK* gilt das Gleiche wie für *TEXT*.

Definition:

```
(DefSIConstraint GRAPHIK
    (((())))
```

Definition des *SIVAS*-Constraints **SET**

Semantik:

Dieses Constraint setzt ein Variable auf einen festen Wert.

Definition:

```
(DefSIConstraint SET
    (((!1))))
```

Definition des *SIVAS*-Constraints **SEQUENZ**

Semantik:

Das dynamische Constraint *SEQUENZ* verbindet 8 Variablen miteinander und baut, falls vorhanden, Beziehungen zu den Nachbarvariablen auf.

Definition:

```
(DefSIDynConstraint SEQUENZ
    (((CONNECT ^6 ^2 ^3)
    (EQUAL ^5 ^1)
    (EQUAL ^2 ^^2)
    (CONNECT ^^1 ^1 ^4))

    ((CONNECT ^5 ^1 ^4)
    (EQUAL ^6 ^2)
    (EQUAL ^1 ^^1)
    (CONNECT ^^2 ^2 ^3)))
    2 2)
```

Constraints in FIDOS

Definition des *FIDOS*-Constraints **RechtsVon**

Semantik:

RechtsVon bindet eine Einheit B rechts an eine Einheit A an.

Definition:

```
(DefFDConstraint RechtsVon ((
    ((FDADD (FDMIN ?1) ?4)                          ;;; dx2-min
    (FDMAXRIGHT MIN ?8 ?2))                         ;;; dx2-max
    ((FDMIN ?2)                                     ;;; dy2-min
    (FDDIFF (FDMAX ?2)(FDDIFF ?7 ?3))))             ;;; dy2-max

    (((FDMIN ?1)                                    ;;; dx1-min
    (FDDIFF (FDMAX ?5) ?4))                         ;;; dx1-max
    ((FDMIN ?2)                                     ;;; dy1-min
    (FDDIFF (FDMAX ?6)(FDDIFF ?3 ?7))))))           ;;; dy1-max
```

Definition des *FIDOS*-Constraints **LinksVon**

Semantik:

LinksVon bindet eine Einheit B links an eine Einheit A an.

Definition:

```
(DefFDConstraint LinksVon ((
    ((FDMAXLEFT MAX ?8 ?2)                          ;;; dx2-min
    (FDDIFF (FDMIN (FDUNION ?1 ?5)) 1))             ;;; dx2-max
    ((FDMIN ?2)                                     ;;; dy2-min
    (FDDIFF (FDMAX ?2)(FDDIFF ?7 ?3))))             ;;; dy2-max

    (((FDADD (FDMIN ?5) ?8)                         ;;; dx1-min
    (FDMAX (FDUNION ?1 ?5)))                        ;;; dx1-max
    ((FDMIN ?6)                                     ;;; dy1-min
    (FDMAX ?6)))))                                  ;;; dy1-max
```

Definition des *FIDOS*-Constraints **Unter**

Semantik:

Unter bindet eine Einheit B unter an eine Einheit A an.

Definition:

```
(DefFDConstraint Unter ((
    ((FDMIN ?1)                                ;;; dx2-min
    (FDDIFF (FDMAX ?1) (FDDIFF ?8 ?4)))        ;;; dx2-max
    ((FDADD (FDMIN ?2) ?3)                     ;;; dy2-min
    (FDMAXDOWN ?1 ?7 MIN)))                    ;;; dy2-max

    (((FDMIN ?1)                               ;;; dx1-min
    (FDDIFF (FDMAX ?1) (FDDIFF ?8 ?4)))        ;;; dx1-max
    ((FDMIN ?2)                                ;;; dy1-min
    (FDDIFF (FDMAX ?6) ?3)))))                 ;;; dy1-max
```

Definition des *FIDOS*-Constraints **Ueber**

Semantik:

Ueber bindet eine Einheit B ueber an eine Einheit A an.

Definition:

```
(DefFDConstraint Ueber ((
    ((FDMIN ?1)                                          ;;; dx2-min
    (FDDIFF (FDMAX ?1)(FDDIFF ?8 ?4)))                   ;;; dx2-max
    ((FDMIN (FDUNION ?2 ?6))                             ;;; dy2-min
    (FDDIFF (FDDIFF (FDMAX ?2) 1)(FDDIFF ?7 ?3))))       ;;; dy2-max

    (((FDMIN ?1)                                         ;;; dx1-min
    (FDDIFF (FDMAX ?5)(FDDIFF ?8 ?4)))                   ;;; dx1-max
    ((FDADD (FDMIN ?6) ?7)                               ;;; dy1-min
    (FDMAX (FDUNION ?6 ?2))))))                          ;;; dy1-max
```

Definition des *FIDOS*-Constraints **Zentriert-Unter**

Semantik:

Zentriert-Unter bindet eine Einheit B zentriert unter an eine Einheit A an.

Definition:

```
(DefFDConstraint Zentriert-Unter ((
    ((FDDIFF (FDADD (FDMIDDLE ?4) (FDMIN ?1)) (FDMIDDLE ?8))   ;;; dx2-min
    (FDADD (FDDIFF (FDMAX ?1) (FDDIFF ?8 ?4))
        (FDDIFF (FDMIDDLE ?4) (FDMIDDLE ?8))))                 ;;; dx2-max
    ((FDADD (FDMIN ?2) ?3)                                     ;;; dy2-min
    (FDMAXDOWN ?1 ?7 MIN)))                                    ;;; dy2-max

    (((FDMIN ?1)                                               ;;; dx1-min
    (FDDIFF (FDMAX ?1) (FDDIFF ?8 ?4)))                        ;;; dx1-max
    ((FDMIN ?2)                                                ;;; dy1-min
    (FDDIFF (FDMAX ?6) ?3)))))                                 ;;; dy1-max
```

Systembeschreibung

Auf den folgenden Seiten befinden sich die Bildschirmabzüge der Layouttestumgebung CLAY . Sie sind in Kapitel 5 genauer beschrieben.

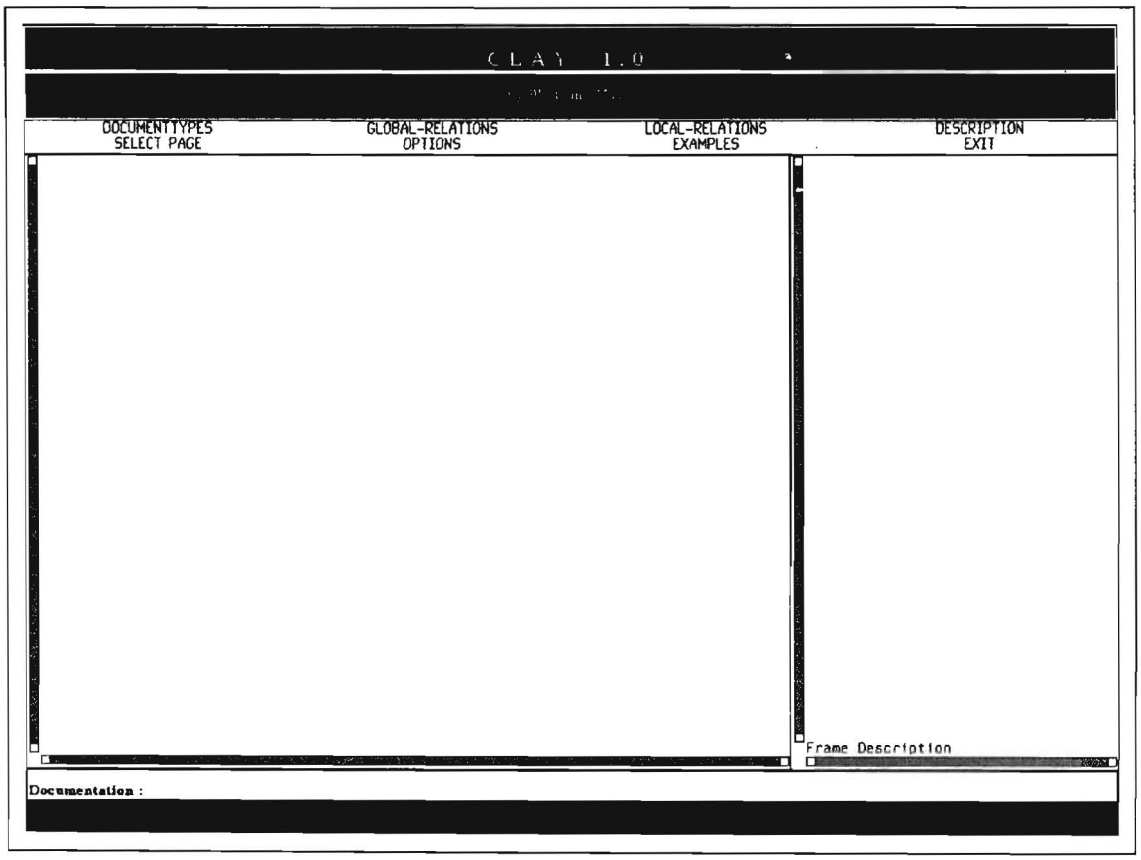

Abbildung 6.16: Der Initialzustand

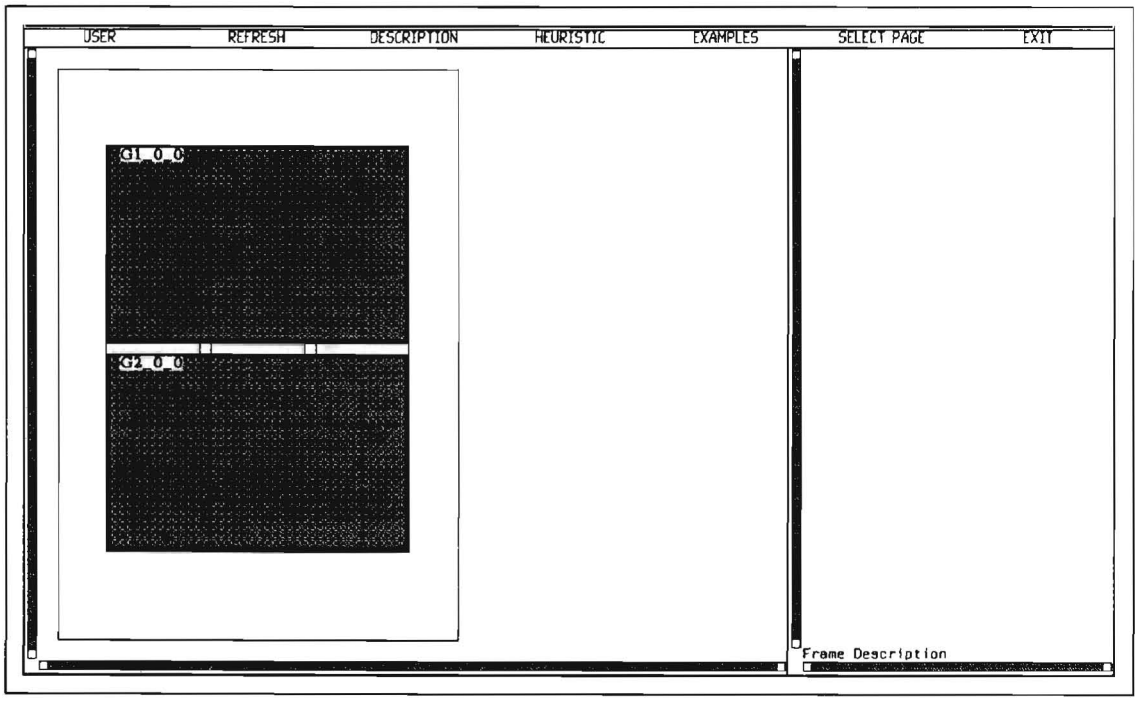

Abbildung 6.17: Beziehung *Kontrast* zwischen zwei Layoutobjekten

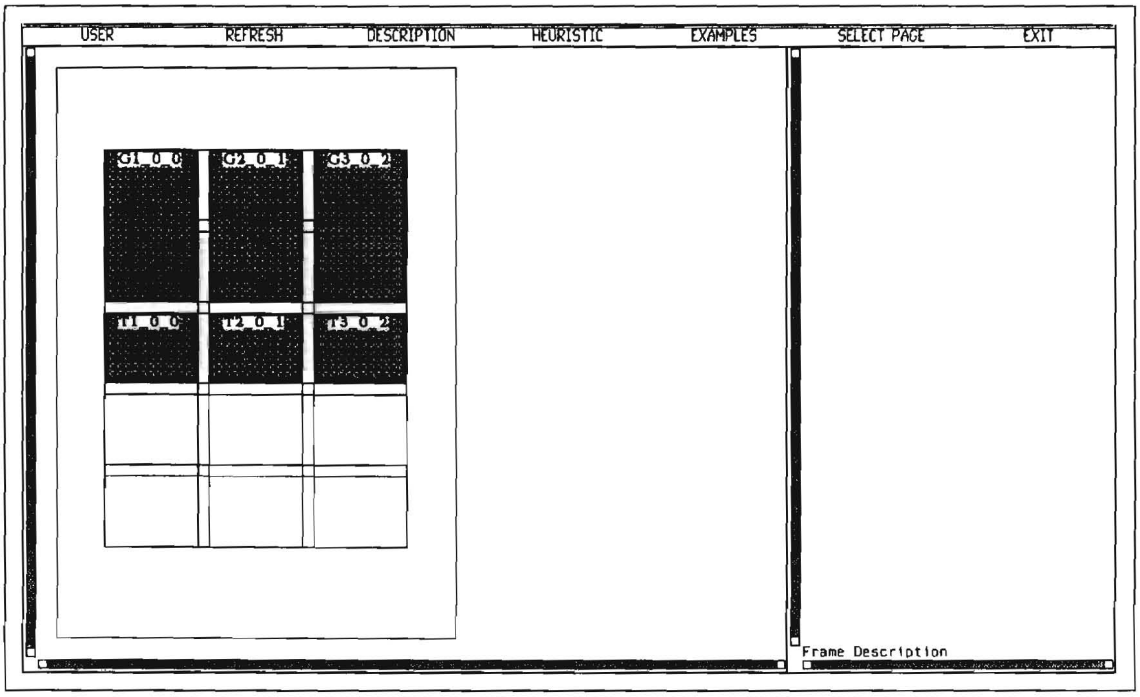

Abbildung 6.18: Beziehung *Sequenz* zwischen drei Texten und drei graphischen Objekten

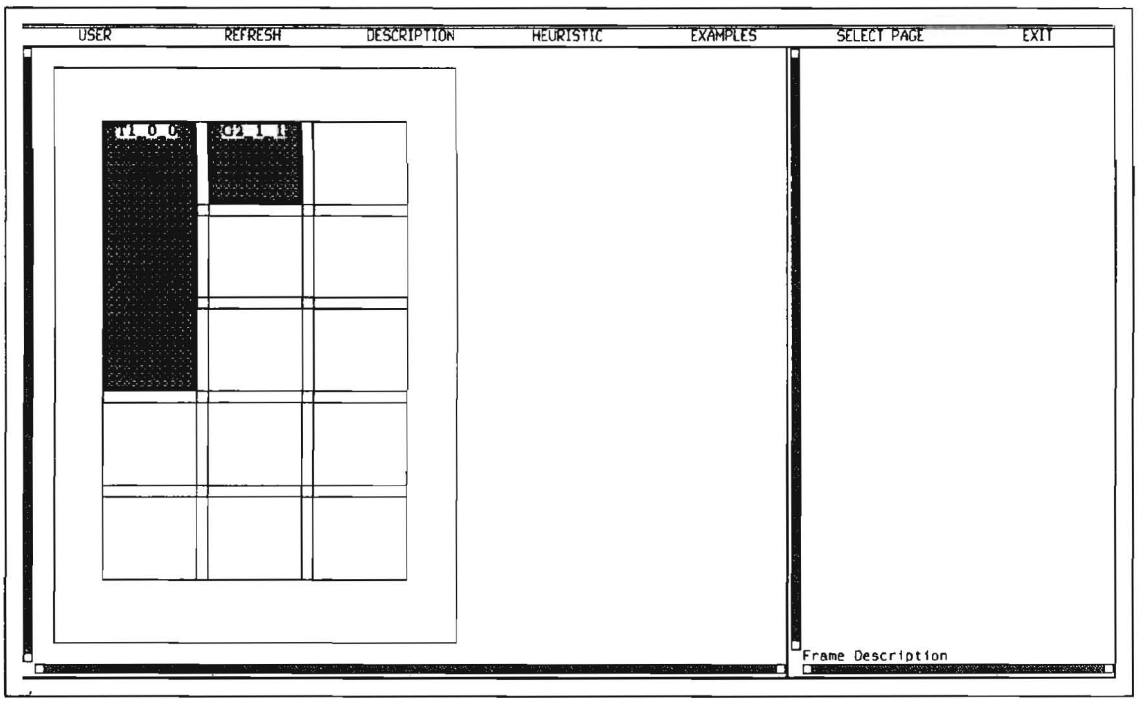

Abbildung 6.19: Beispiellayout unter Verwendung des globalen Constraints *rechts-von*

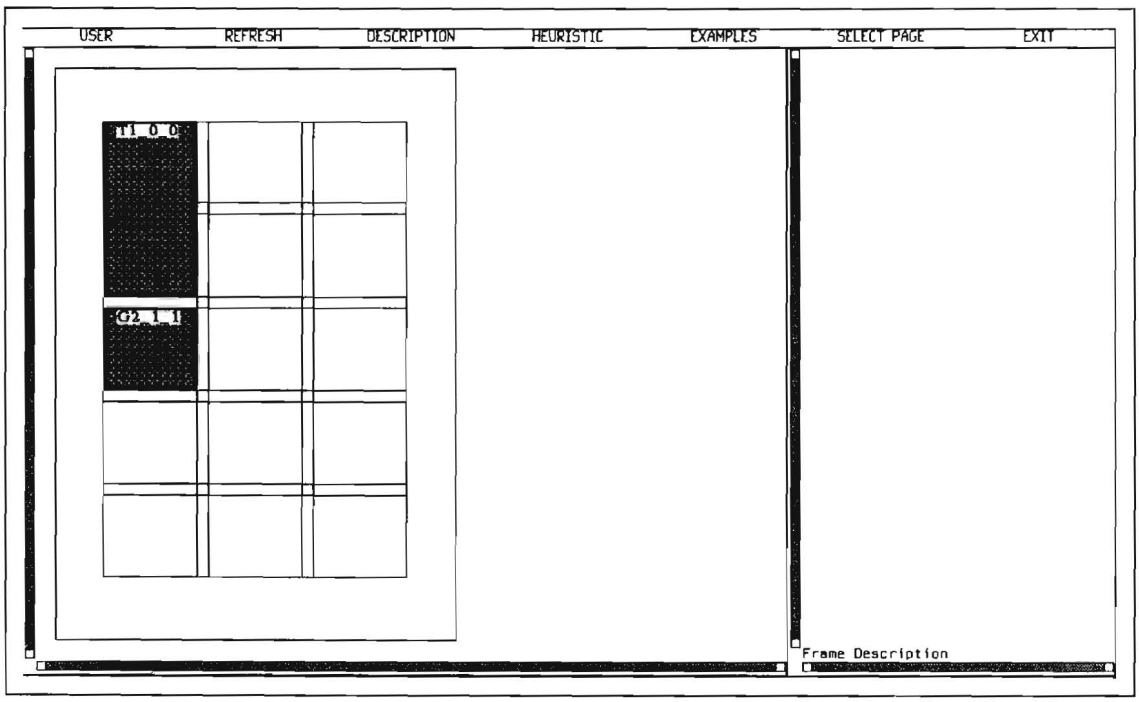

Abbildung 6.20: Beispiellayout unter Verwendung des globalen Constraints *unter*

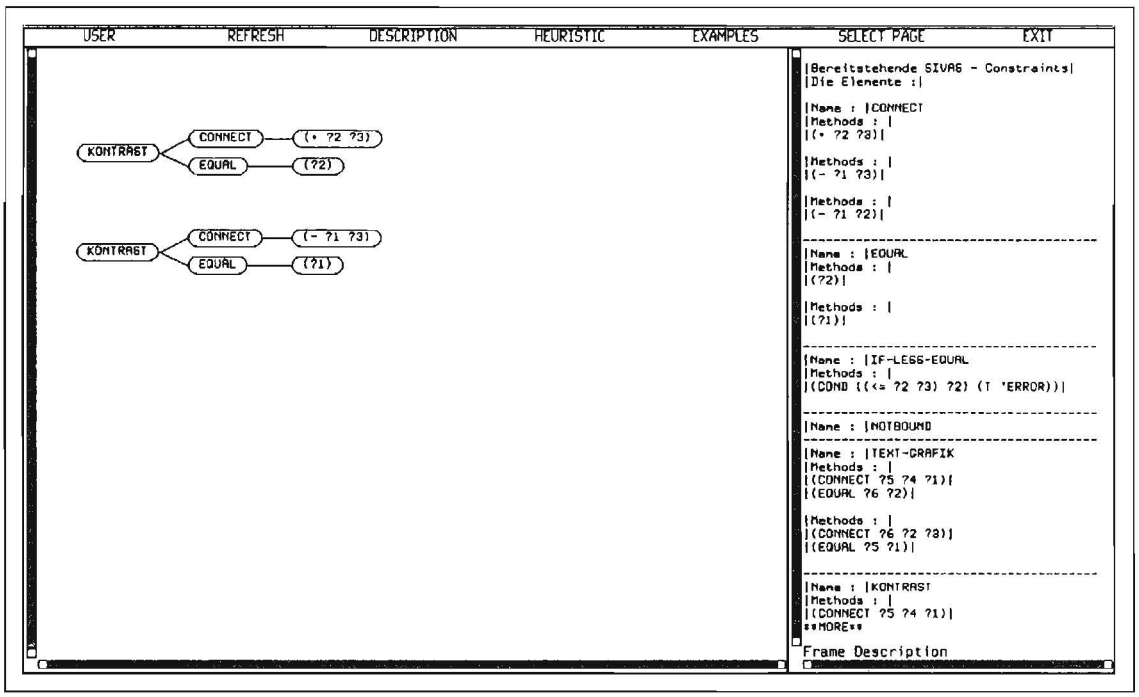

Abbildung 6.21: Graphische Darstellung des Constraints *Kontrast*

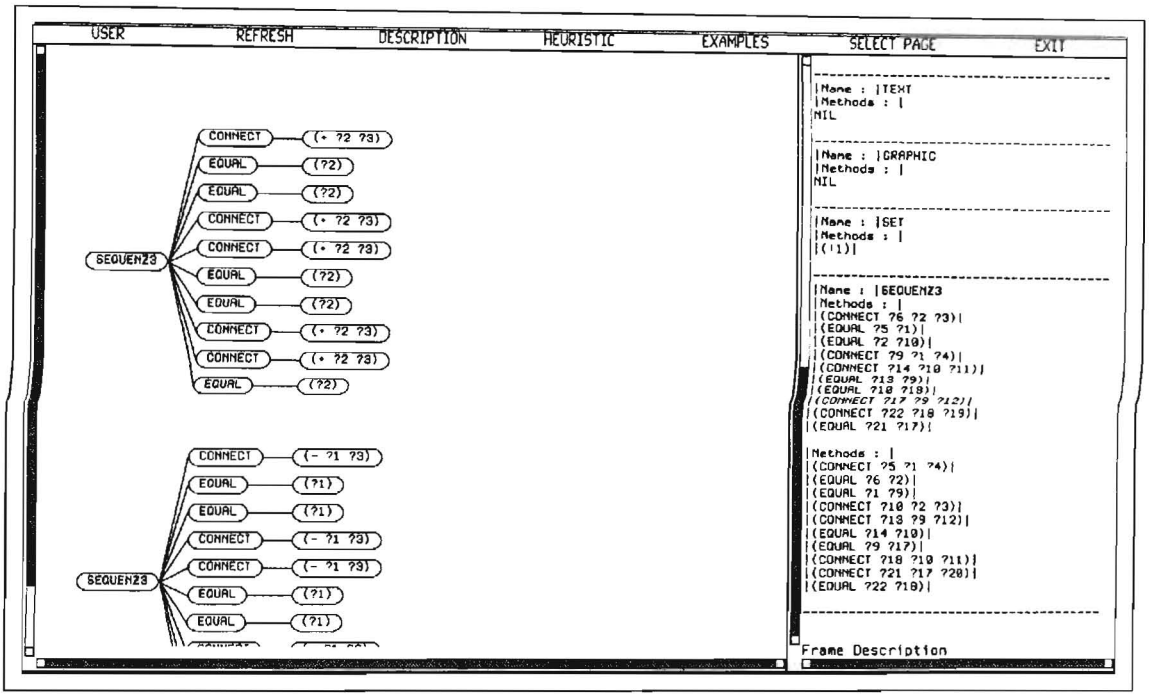

Abbildung 6.22: Graphische Darstellung des Constraints *Sequenz*

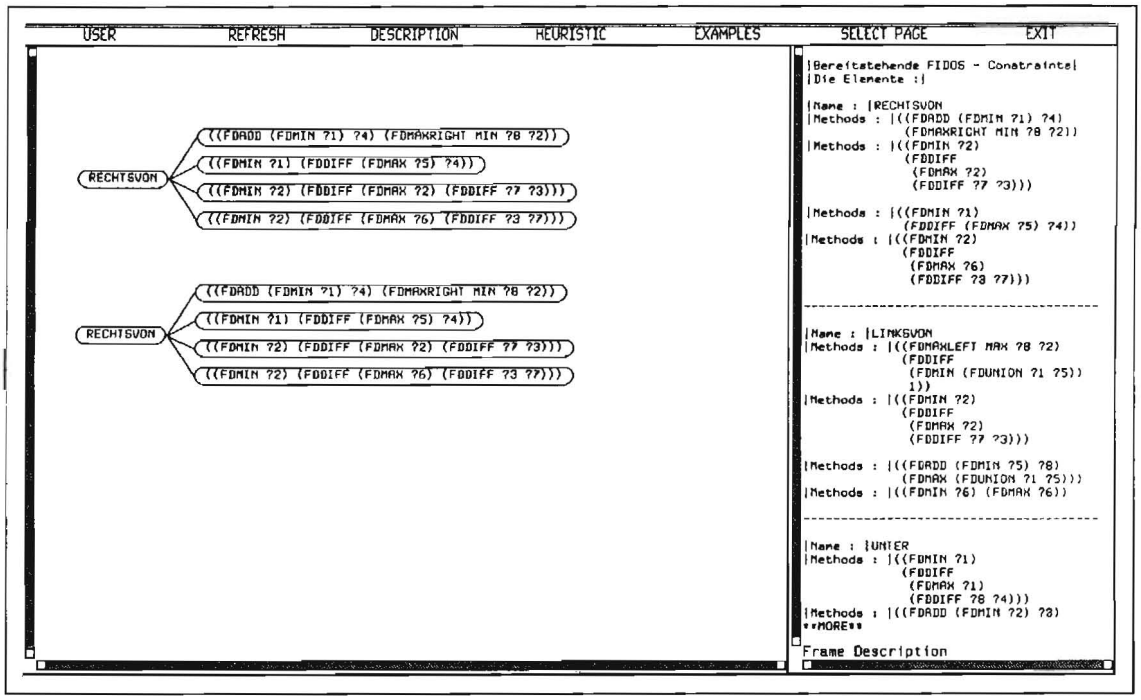

Abbildung 6.23: Graphische Darstellung des Constraints *Rechtsvon*

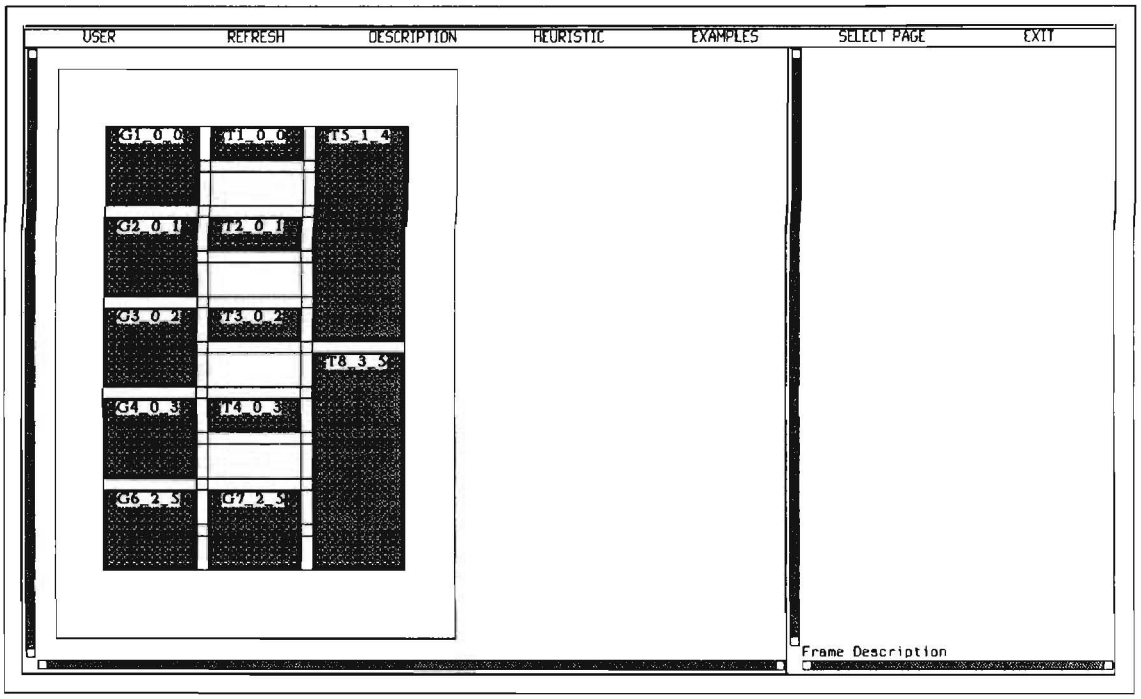

Abbildung 6.24:
Layout von vier Einheiten auf dem Dokumenttyp *Bedienungsanleitung hoch* mit Raster

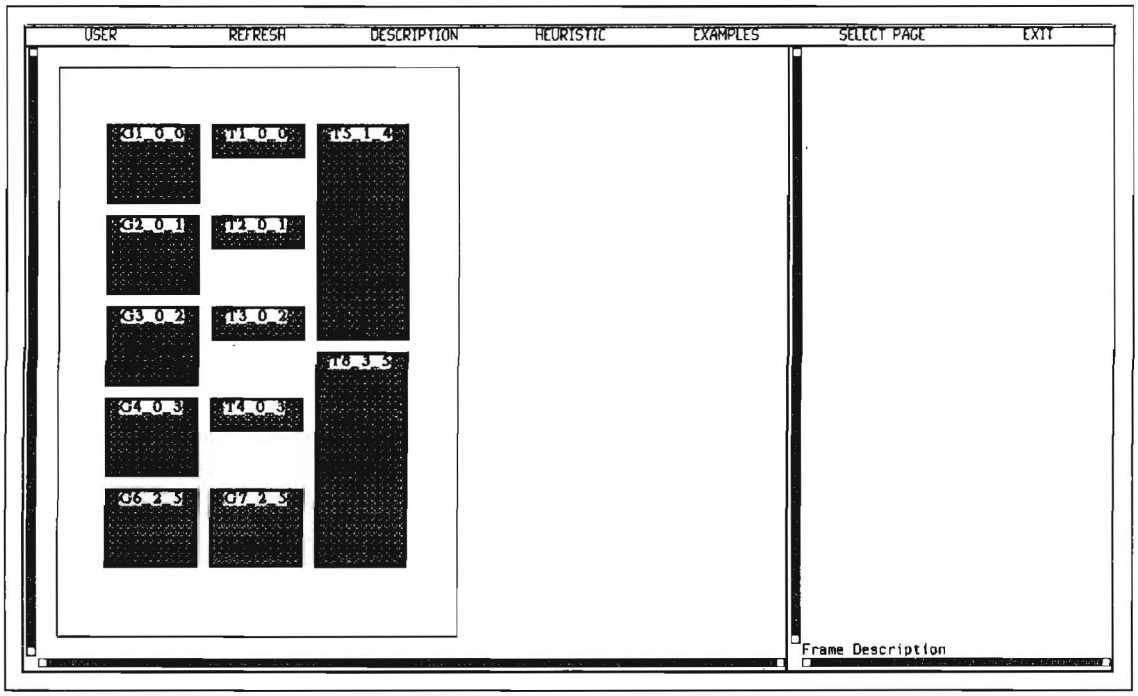

Abbildung 6.25: Layout von vier Einheiten auf dem Dokumenttyp *Bedienungsanleitung hoch*

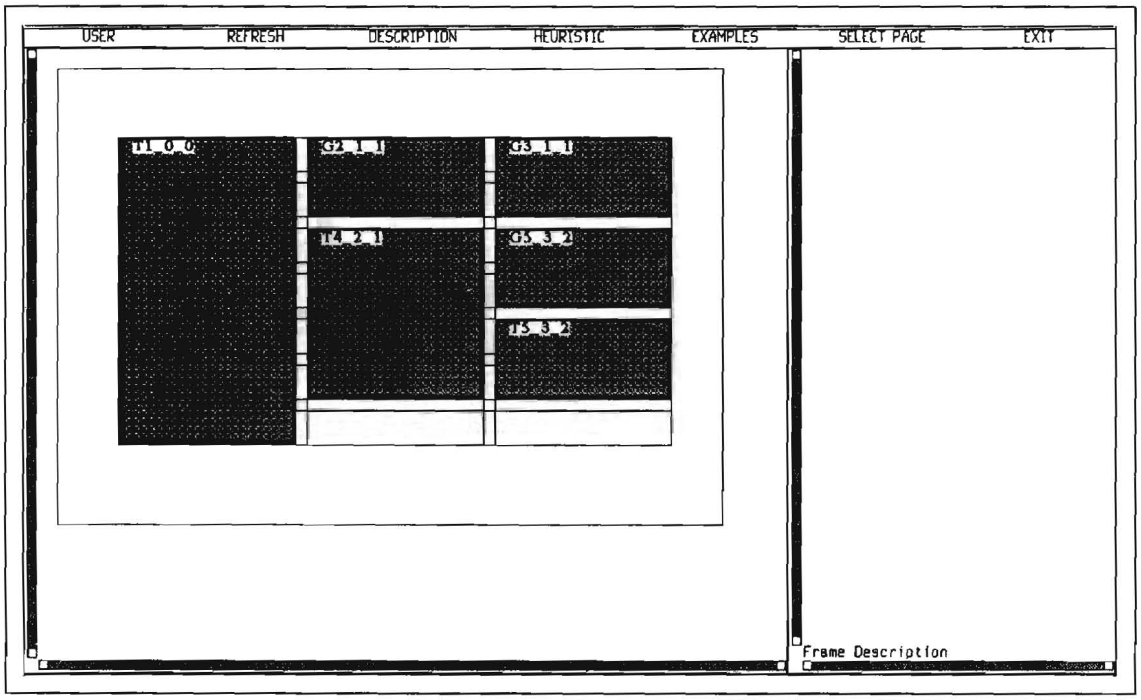

Abbildung 6.26: Layout von vier Einheiten auf dem Dokumenttyp *Bedienungsanleitung breit* mit Raster

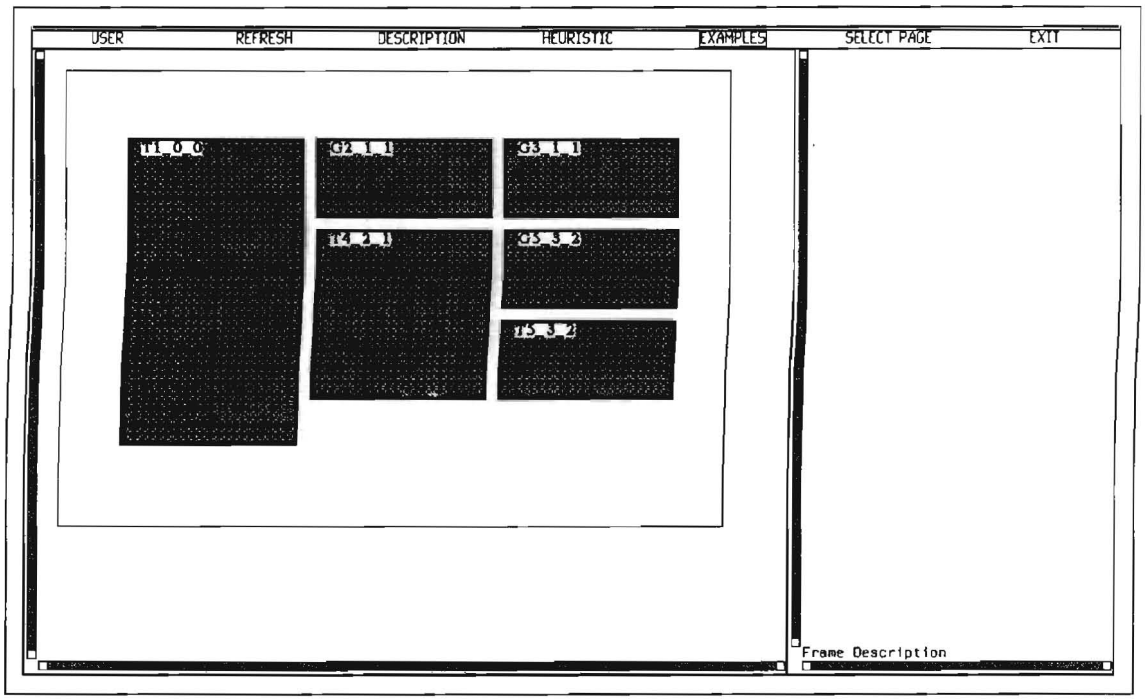

Abbildung 6.27: Layout von vier Einheiten auf dem Dokumenttyp *Bedienungsanleitung breit*

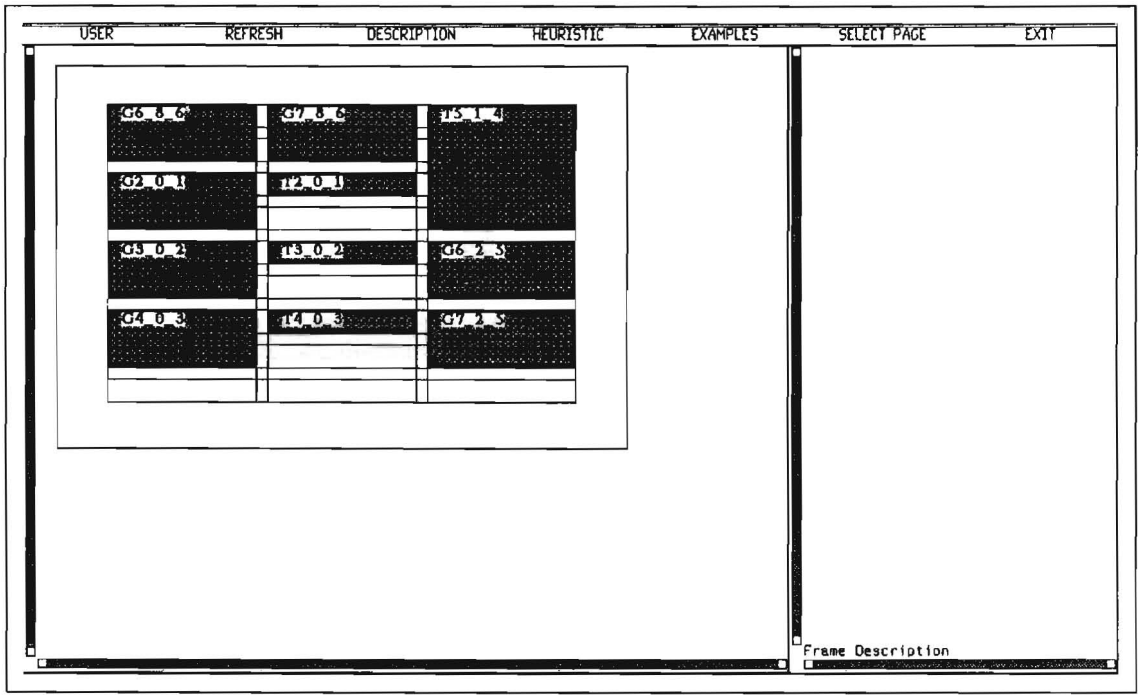

Abbildung 6.28: Layout über mehrere Seiten (Seite 1) mit Raster

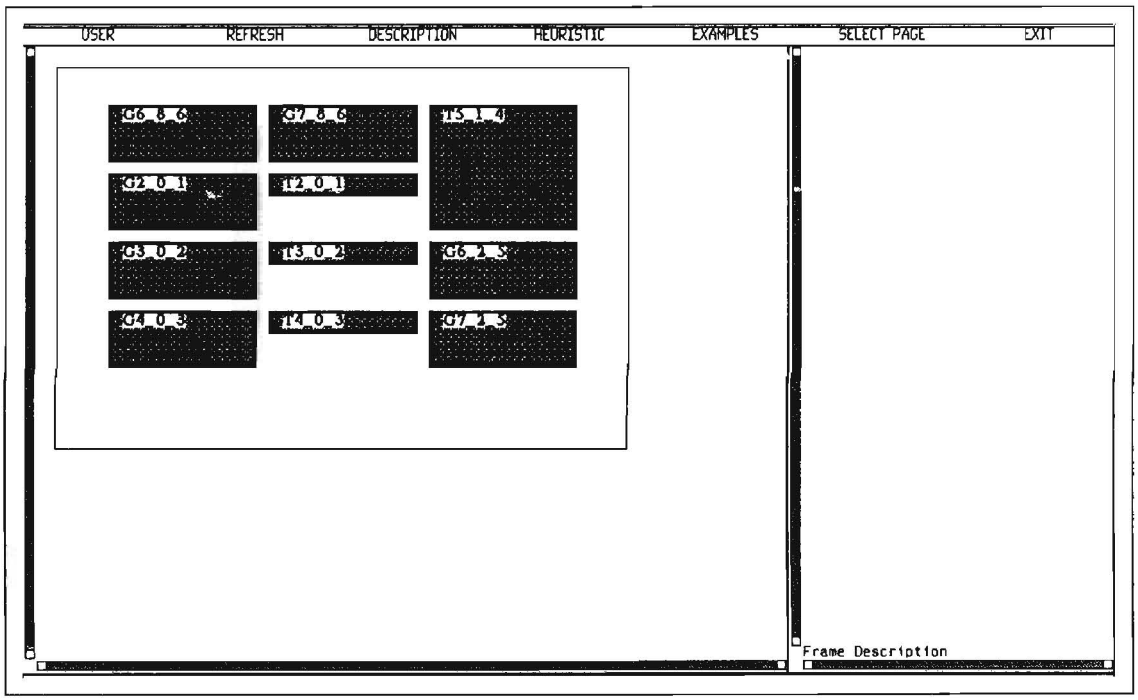

Abbildung 6.29: Layout über mehrere Seiten (Seite 1)

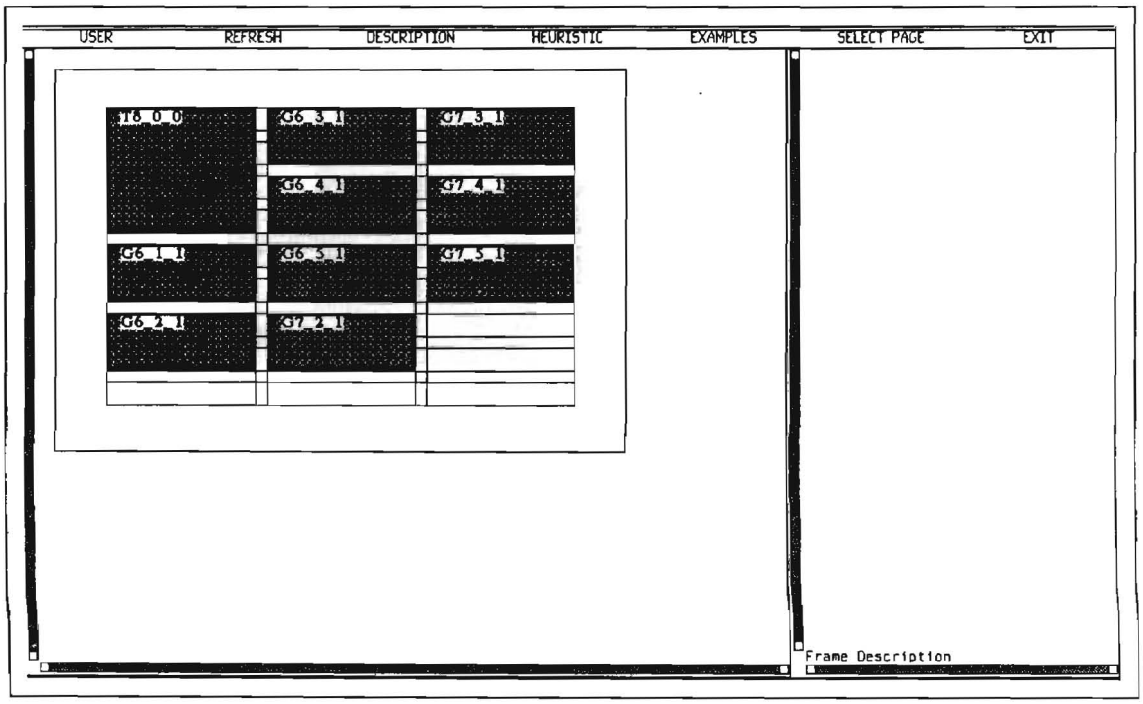

Abbildung 6.30: Layout über mehrere Seiten (Seite 2) mit Raster

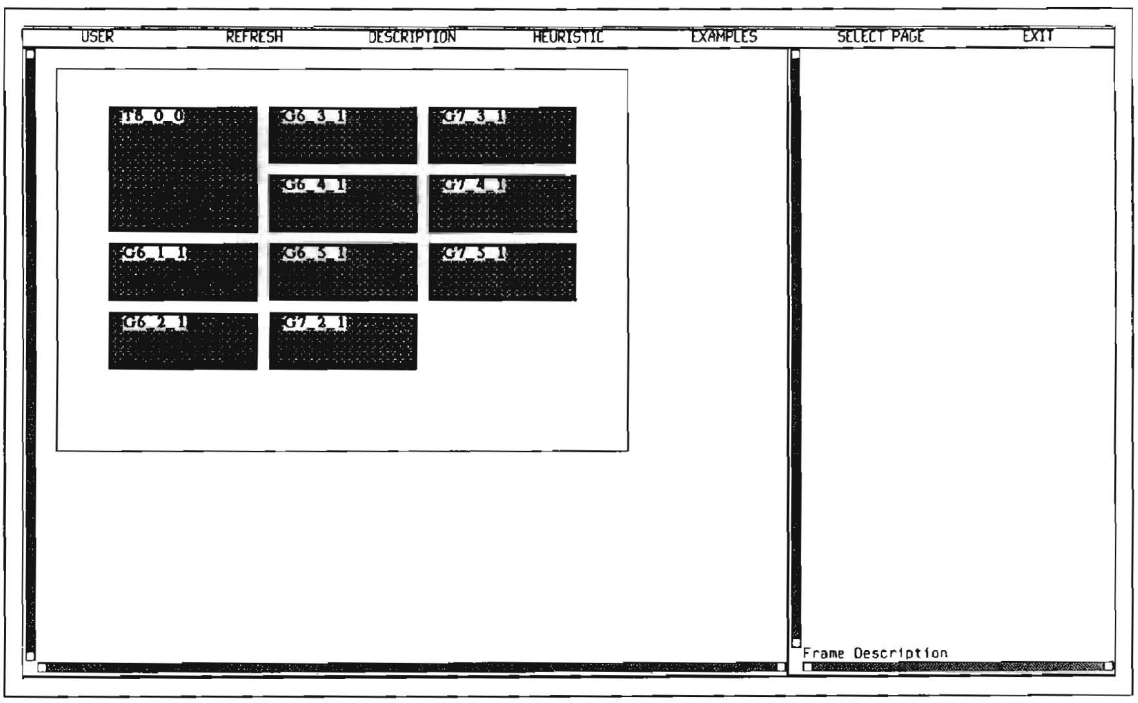

Abbildung 6.31: Layout über mehrere Seiten (Seite 2)

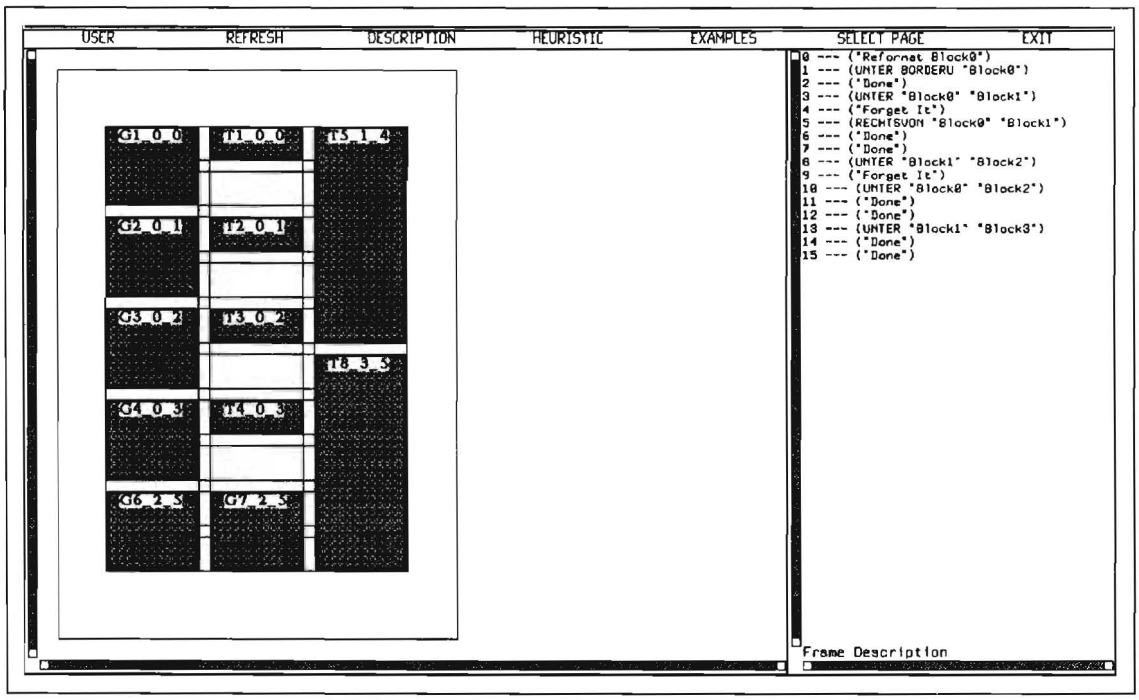

Abbildung 6.32: Aktionsausführungsliste zum dargestellten Layout

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
D-6750 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all publisched papers so far can be ordered from the above address.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-91-08
Wolfgang Wahlster, Elisabeth André, Som Bandyopadhyay, Winfried Graf, Thomas Rist: WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation
23 pages

RR-91-09
Hans-Jürgen Bürckert, Jürgen Müller, Achim Schupeta: RATMAN and its Relation to Other Multi-Agent Testbeds
31 pages

RR-91-10
Franz Baader, Philipp Hanschke: A Scheme for Integrating Concrete Domains into Concept Languages
31 pages

RR-91-11
Bernhard Nebel: Belief Revision and Default Reasoning: Syntax-Based Approaches
37 pages

RR-91-12
J.Mark Gawron, John Nerbonne, Stanley Peters: The Absorption Principle and E-Type Anaphora
33 pages

RR-91-13
Gert Smolka: Residuation and Guarded Rules for Constraint Logic Programming
17 pages

RR-91-14
Peter Breuer, Jürgen Müller: A Two Level Representation for Spatial Relations, Part I
27 pages

RR-91-15
Bernhard Nebel, Gert Smolka: Attributive Description Formalisms ... and the Rest of the World
20 pages

RR-91-16
Stephan Busemann: Using Pattern-Action Rules for the Generation of GPSG Structures from Separate Semantic Representations
18 pages

RR-91-17
Andreas Dengel, Nelson M. Mattos: The Use of Abstraction Concepts for Representing and Structuring Documents
17 pages

RR-91-18
John Nerbonne, Klaus Netter, Abdel Kader Diagne, Ludwig Dickmann, Judith Klein: A Diagnostic Tool for German Syntax
20 pages

RR-91-19
Munindar P. Singh: On the Commitments and Precommitments of Limited Agents
15 pages

RR-91-20
Christoph Klauck, Ansgar Bernardi, Ralf Legleitner FEAT-Rep: Representing Features in CAD/CAM
48 pages

RR-91-21
Klaus Netter: Clause Union and Verb Raising Phenomena in German
38 pages

RR-91-22
Andreas Dengel: Self-Adapting Structuring and Representation of Space
27 pages

RR-91-23
Michael Richter, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner: Akquisition und Repräsentation von technischem Wissen für Planungsaufgaben im Bereich der Fertigungstechnik
24 Seiten

RR-91-24
Jochen Heinsohn: A Hybrid Approach for Modeling Uncertainty in Terminological Logics
22 pages

RR-91-25
Karin Harbusch, Wolfgang Finkler, Anne Schauder: Incremental Syntax Generation with Tree Adjoining Grammars
16 pages

RR-91-26
M. Bauer, S. Biundo, D. Dengler, M. Hecking, J. Koehler, G. Merziger: Integrated Plan Generation and Recognition
- A Logic-Based Approach -
17 pages

RR-91-27
A. Bernardi, H. Boley, Ph. Hanschke, K. Hinkelmann, Ch. Klauck, O. Kühn, R. Legleitner, M. Meyer, M. M. Richter, F. Schmalhofer, G. Schmidt, W. Sommer: ARC-TEC: Acquisition, Representation and Compilation of Technical Knowledge
18 pages

RR-91-28
Rolf Backofen, Harald Trost, Hans Uszkoreit: Linking Typed Feature Formalisms and Terminological Knowledge Representation Languages in Natural Language Front-Ends
11 pages

RR-91-29
Hans Uszkoreit: Strategies for Adding Control Information to Declarative Grammars
17 pages

RR-91-30
Dan Flickinger, John Nerbonne: Inheritance and Complementation: A Case Study of *Easy* Adjectives and Related Nouns
39 pages

RR-91-31
H.-U. Krieger, J. Nerbonne: Feature-Based Inheritance Networks for Computational Lexicons
11 pages

RR-91-32
Rolf Backofen, Lutz Euler, Günther Görz: Towards the Integration of Functions, Relations and Types in an AI Programming Language
14 pages

RR-91-33
Franz Baader, Klaus Schulz: Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures
33 pages

RR-91-34
Bernhard Nebel, Christer Bäckström: On the Computational Complexity of Temporal Projection and some related Problems
35 pages

RR-91-35
Winfried Graf, Wolfgang Maaß: Constraint-basierte Verarbeitung graphischen Wissens
14 Seiten

RR-92-01
Werner Nutt: Unification in Monoidal Theories is Solving Linear Equations over Semirings
57 pages

RR-92-02
Andreas Dengel, Rainer Bleisinger, Rainer Hoch, Frank Hönes, Frank Fein, Michael Malburg: Π_{ODA}: The Paper Interface to ODA
53 pages

RR-92-03
Harold Boley: Extended Logic-plus-Functional Programming
28 pages

RR-92-04
John Nerbonne: Feature-Based Lexicons: An Example and a Comparison to DATR
15 pages

RR-92-05
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Michael Schulte, Rainer Stark: Feature based Integration of CAD and CAPP
19 pages

RR-92-07
Michael Beetz: Decision-theoretic Transformational Planning
22 pages

RR-92-08
Gabriele Merziger: Approaches to Abductive Reasoning - An Overview -
46 pages

RR-92-09
Winfried Graf, Markus A. Thies: Perspektiven zur Kombination von automatischem Animationsdesign und planbasierter Hilfe
15 Seiten

RR-92-11
Susane Biundo, Dietmar Dengler, Jana Koehler:
Deductive Planning and Plan Reuse in a Command Language Environment
13 pages

RR-92-13
Markus A. Thies, Frank Berger:
Planbasierte graphische Hilfe in objektorientierten Benutzungsoberflächen
13 Seiten

RR-92-14
Intelligent User Support in Graphical User Interfaces:
1. InCome: A System to Navigate through Interactions and Plans
   Thomas Fehrle, Markus A. Thies
2. Plan-Based Graphical Help in Object-Oriented User Interfaces
   Markus A. Thies, Frank Berger

22 pages

RR-92-15
Winfried Graf: Constraint-Based Graphical Layout of Multimodal Presentations
23 pages

RR-92-17
Hassan Aït-Kaci, Andreas Podelski, Gert Smolka:
A Feature-based Constraint System for Logic Programming with Entailment
23 pages

RR-92-18
John Nerbonne: Constraint-Based Semantics
21 pages

RR-92-19
Ralf Legleitner, Ansgar Bernardi, Christoph Klauck
PIM: Planning In Manufacturing using Skeletal Plans and Features
17 pages

RR-92-20
John Nerbonne: Representing Grammar, Meaning and Knowledge
18 pages

DFKI Technical Memos

TM-91-05
Jay C. Weber, Andreas Dengel , Rainer Bleisinger:
Theoretical Consideration of Goal Recognition Aspects for Understanding Information in Business Letters
10 pages

TM-91-06
Johannes Stein: Aspects of Cooperating Agents
22 pages

TM-91-08
Munindar P. Singh: Social and Psychological Commitments in Multiagent Systems
11 pages

TM-91-09
Munindar P. Singh: On the Semantics of Protocols Among Distributed Intelligent Agents
18 pages

TM-91-10
Béla Buschauer, Peter Poller, Anne Schauder, Karin Harbusch: Tree Adjoining Grammars mit Unifikation
149 pages

TM-91-11
Peter Wazinski: Generating Spatial Descriptions for Cross-modal References
21 pages

TM-91-12
Klaus Becker, Christoph Klauck, Johannes Schwagereit: FEAT-PATR: Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM
33 Seiten

TM-91-13
Knut Hinkelmann:
Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter
16 pages

TM-91-14
Rainer Bleisinger, Rainer Hoch, Andreas Dengel:
ODA-based modeling for document analysis
14 pages

TM-91-15
Stefan Bussmann: Prototypical Concept Formation
An Alternative Approach to Knowledge Representation
28 pages

TM-92-01
Lijuan Zhang:
Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

DFKI Documents

D-91-06
Gerd Kamp: Entwurf, vergleichende Beschreibung und Integration eines Arbeitsplanerstellungssystems für Drehteile
130 Seiten

D-91-07
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner
TEC REP: Repräsentation von Geometrie- und Technologieinformationen
70 Seiten

D-91-08
Thomas Krause: Globale Datenflußanalyse und horizontale Compilation der relational-funktionalen Sprache RELFUN
137 Seiten

D-91-09
David Powers, Lary Reeker (Eds.):
Proceedings MLNLO'91 - Machine Learning of Natural Language and Ontology
211 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-91-10
Donald R. Steiner, Jürgen Müller (Eds.):
MAAMAW'91: Pre-Proceedings of the 3rd European Workshop on „Modeling Autonomous Agents and Multi-Agent Worlds"
246 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-91-11
*Thilo C. Horstmann:*Distributed Truth Maintenance
61 pages

D-91-12
Bernd Bachmann:
$H^{hiera}C_{on}$ - a Knowledge Representation System with Typed Hierarchies and Constraints
75 pages

D-91-13
International Workshop on Terminological Logics
Organizers: Bernhard Nebel, Christof Peltason, Kai von Luck
131 pages

D-91-14
Erich Achilles, Bernhard Hollunder, Armin Laux, Jörg-Peter Mohren: KRIS : Knowledge Representation and Inference System
- Benutzerhandbuch -
28 Seiten

D-91-15
Harold Boley, Philipp Hanschke, Martin Harm, Knut Hinkelmann, Thomas Labisch, Manfred Meyer, Jörg Müller, Thomas Oltzen, Michael Sintek, Werner Stein, Frank Steinle:
µCAD2NC: A Declarative Lathe-Worplanning Model Transforming CAD-like Geometries into Abstract NC Programs
100 pages

D-91-16
Jörg Thoben, Franz Schmalhofer, Thomas Reinartz:
Wiederholungs-, Varianten- und Neuplanung bei der Fertigung rotationssymmetrischer Drehteile
134 Seiten

D-91-17
Andreas Becker:
Analyse der Planungsverfahren der KI im Hinblick auf ihre Eignung für die Abeitsplanung
86 Seiten

D-91-18
Thomas Reinartz: Definition von Problemklassen im Maschinenbau als eine Begriffsbildungsaufgabe
107 Seiten

D-91-19
Peter Wazinski: Objektlokalisation in graphischen Darstellungen
110 Seiten

D-92-01
Stefan Bussmann: Simulation Environment for Multi-Agent Worlds - Benutzeranleitung
50 Seiten

D-92-02
Wolfgang Maaß: Constraint-basierte Plazierung in multimodalen Dokumenten am Beispiel des Layout-Managers in WIP
111 Seiten

D-92-03
Wolfgan Maaß, Thomas Schiffmann, Dudung Soetopo, Winfried Graf: LAYLAB: Ein System zur automatischen Plazierung von Text-Bild-Kombinationen in multimodalen Dokumenten
41 Seiten

D-92-06
Hans Werner Höper: Systematik zur Beschreibung von Werkstücken in der Terminologie der Featuresprache
392 Seiten

D-92-08
Jochen Heinsohn, Bernhard Hollunder (Eds.):
DFKI Workshop on Taxonomic Reasoning
Proceedings
56 pages

D-92-09
Gernod P. Laufkötter: Implementierungsmöglichkeiten der integrativen Wissensakquisitionsmethode des ARC-TEC-Projektes
86 Seiten

D-92-21
Anne Schauder: Incremental Syntactic Generation of Natural Language with Tree Adjoining Grammars
57 pages

Constraint-basierte Plazierung in multimodalen Dokumenten am Beispiel des Layout-Managers in WIP

Wolfgang Maaß

D-92-02
Document