Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-92-03

LAYLAB:

Ein System zur automatischen Plazierung von Text-Bild-Kombinationen in multimodalen Dokumenten

**Wolfgang Maaß, Thomas Schiffmann,
Dudung Soetopo, Winfried Graf**

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

LAYLAB: Ein System zur automatischen Plazierung von Text-Bild-Kombinationen in multimodalen Dokumenten

Wolfgang Maaß, Thomas Schiffmann, Dudung Soetopo, Winfried Graf

DFKI-D-92-03

Diese Arbeit ist der Abschlußbericht des Fortgeschrittenenpraktikums *Wissensbasierte Graphikgenerierung* (Wahlster mit Graf), das im SS´90 und WS ´91 vom Fachbereich Informatik der Universität des Saarlandes und dem DFKI veranstaltet wurde.

LAYLAB:
Ein System zur automatischen Plazierung von Text-Bild-Kombinationen in multimodalen Dokumenten

Wolfgang Maaß Thomas Schiffmann Dudung Soetopo
Winfried Graf

Bericht des Fortgeschrittenenpraktikums
"Wissensbasierte Graphikgenerierung"
im SS '90 und WS '91

Universität des Saarlandes
Fachbereich Informatik
und
Deutsches Forschungszentrum für Künstliche Intelligenz

Zusammenfassung

Im Bereich der intelligenten Benutzerschnittstellen besteht derzeit, bedingt durch die wachsende Komplexität der von wissensbasierten Anwendungssystemen zu übermittelden Information, ein zunehmender Bedarf an Werkzeugen zur flexiblen und effizienten Informationspräsentation. Neben Sprache spielt beim Design von (elektronischen) Dokumenten die Verwendung von graphischen Darstellungen sowie eine Kombinationen dieser beiden Medien zur Informationsvermittlung eine entscheidende Rolle. Während heute in der Regel solche Graphiken noch manuell erstellt werden, z.B. mittels interaktiver 3D-Graphikeditoren, bestand die Aufgabe in diesem Fortgeschrittenenpraktikum darin, Graphiken in Abhängigkeit von bestimmten Generierungsparametern wie Präsentationsziel, Präsentationssituation, Zielgruppe, Ausgabemedium, etc. automatisch zu erzeugen. Eine zentrale Rolle spielte dabei die Repräsentation von Wissen über die Verwendung von Graphik. Zum einen sollte Wissen über Objekte und Darstellungstechniken in den Entwurfsprozeß einfließen, zum anderen war der Bildinhalt zu repräsentieren, um beispielsweise darauf natürlichsprachlich Bezug nehmen zu können. In dieser Arbeitsgruppe stand die Entwicklung einer automatischen Plazierungskomponente zur Gestaltung des Layouts von multimodalen Dokumenten im Vordergrund, während sich zwei weitere Gruppen mit der wissensbasierten Anwendung spezieller Graphiktechniken (z.B. Explosion, Aufriß, Annotation) befaßten.

Die vorliegende Arbeit beschreibt die prototypische Implementierung des Systems *LayLab*, einem Experimentiersystem zum automatischen Layout multimodaler Präsentationen, das im Kontext des WIP-Projektes entwickelt wurde. Als Programmierungumgebung standen hierzu Symbolics Lisp-Maschinen und Apple MacIvory Workstations zur Verfügung. Bei der Entwicklung konnte auf einen Lisp-basierten interaktiven 3D-Graphikeditor (S-Geometry) sowie die objektorientierte Symbolics Fensterumgebung zurückgegriffen werden.

Inhaltsverzeichnis

1 Einführung

1.1 Motivation und Aufgabenstellung

Seit Beginn der Entwicklung der Kultur versucht der Mensch, Informationen aufzuzeichnen und zu speichern. Diese Aufgabe wird umso komplexer, je größer die Menge der Informationen ist.

Mindestens ebenso wichtig ist es jedoch, gespeicherte oder auch neu erzeugte Informationen einem Benutzer in verständlicher Form zur Verfügung zu stellen. Eines der Grundprobleme bei der Vermittlung von Informationen ist ihre Aufbereitung in Form und Inhalt.

Es ist intuitiv klar und auch durch psychologische Untersuchen empirisch erwiesen (vgl. z.B. [Miller 87]), daß die klare und logische Gliederung einer Information ihre Lesbarkeit und Verständlichkeit für den Benutzer erheblich fördert. Zudem wird dadurch die Glaubwürdigkeit der Information unterstützt. Anzustreben ist daher, eine Anordnung von Layoutobjekten zu schaffen, die transparent, sachlich, funktional und ästhetisch ist.

Dies zu erreichen ist normalerweise die Aufgabe von Graphik-Designern. Deren Vorgehen ist ein iterativer Annäherungsprozeß an eine endgültige Form des Layouts, das sehr stark von Heuristiken, Intuition und Erfahrung geprägt ist.

Ebenso spielen das ästhetische Empfinden sowie persönliche Vorlieben eine ausschlaggebende Rolle für das Ergebnis der Arbeit. Aus diesem Grund ist es schwierig, eine einheitliche und funktionale Form zu finden, die einerseits eine uniforme, kohärente und konsistente Gestaltung von Dokumenten erlaubt, die auf der anderen Seite aber auch flexibel ist für unterschiedliche Inhalte.

Da die manuelle Gestaltung eines Dokuments eine hochkomplexe Arbeit darstellt und anderseits die üblichen Produktionstechniken stark auf Massenproduktion ausgelegt sind, ist es nicht üblich, daß ein Layout flexibel an den jeweiligen Leser angepaßt wird. Statt dessen wird von einem 'Durchschnittsbenutzer' ausgegangen und die Präsentation der Informationen auf diesen abgestimmt.

Mit dem System *LayLab* soll ein Beitrag geleistet werden zur automatischen und flexiblen Gestaltung von Dokumenten. Dabei soll durch einen wissensbasierten Ansatz sowohl eine beträchtliche Reduzierung des Zeit- und Arbeitsaufwandes bei der Erstellung erzielt werden, als auch die Möglichkeit geschaffen werden, durch Einbeziehung verschiedener Generierungsparameter auf die jeweilige Situation und auf individuelle Bedürfnisse eines Benutzers (z.B. schlechte Sehfähigkeit) einzugehen (vgl. auch [Graf 92]).

Ausgangspunkt des Systems ist eine Menge von Layoutobjekten, d.h. von Graphiken und Texten, die auf einem Dokument anzuordnen sind. Von diesen Objekten ist zunächst nur die Größe von Interesse und die Tatsache, ob es sich um einen Text oder um eine Graphik

handelt.

Daneben können die Objekte in inhaltlichen Beziehungen zueinander stehen, das bedeutet, daß mehrere Objekte z.B. einen Kontrast oder eine Sequenz bilden. Diese *semantischen Relationen*, die vorgegeben sind, sollen in der Anordnung der Objekte zum Ausdruck kommen. Außerdem soll diese Anordnung den Regeln für gutes Layout genügen.

1.2 Definition von Layout

Definition (Der große Brockhaus, 17. Auflage, 1970): "*Layout* (engl. Plan, Skizze) Entwurf der Druckvorlage für die Gestaltung von Buch- und Presseerzeugnissen mit einem Lageplan der Text- und Bildelemente. Bei Werbemitteln ist das Layout ein Schemaaufriß zur Aufgliederung von Illustration, Schlagzeile, Zwischentitel und fortlaufenden Text, so daß aus den einzelnen Elementen ein geschlossenes, auf psychologische Wirkung abzielendes Gesamtbild entsteht."

Speziell umfaßt dieser Entwurf

- die Anordnung von Texten und Graphiken auf einem Dokument,
- die Konstruktion des Satzspiegels (Aufteilung eines Dokuments in ein oder mehrere Spalten),
- die Bestimmung der Randproportionen und der Abstände zwischen den einzelnen Objekten.

Es wird hier unterschieden zwischen *funktionalem* und *künstlerischem* Layout (vgl.: [Stankowski & Duschek 89]). Beim funktionalen Layout steht die Übermittlung der inhaltlichen Informationen im Mittelpunkt, während beim künstlerischen Layout (vgl. z.B. [Arnheim 69]) auch durch die äußere Form der Darstellung Informationen an den Benutzer übermittelt werden sollen (vgl. Abb. 1).

Da sich das WIP-System, in das diese Arbeit integriert ist, mit funktionaler und sachlicher Präsentation beschäftigt und die Behandlung von künstlerichem Layout durch die fehlende Möglichkeit der Formalisierung den Rahmen dieser Arbeit sprengen würde, werden wir uns im folgenden ausschließlich mit funktionalem Layout beschäftigen.

Ein gutes Layout ist transparent, sachlich, funktional und ästhetisch (vgl. [Braun 87]). Konkret bedeutet das eine klare Aufteilung von Bild- und Textelementen durch das Layout. Es verdeutlicht die inhaltlichen Zusammenhänge bzw. Gegensätze des dargestellten Materials und macht eine eindeutige Zuordnung der Objekte zueinander möglich. Befinden sich auf einer Dokumentseite mehrere zusammengehörige Einheiten (z.B. mehrere

Zeitungsartikel), so ist es Aufgabe des Layouts, dem Leser die Zusammengehörigkeiten

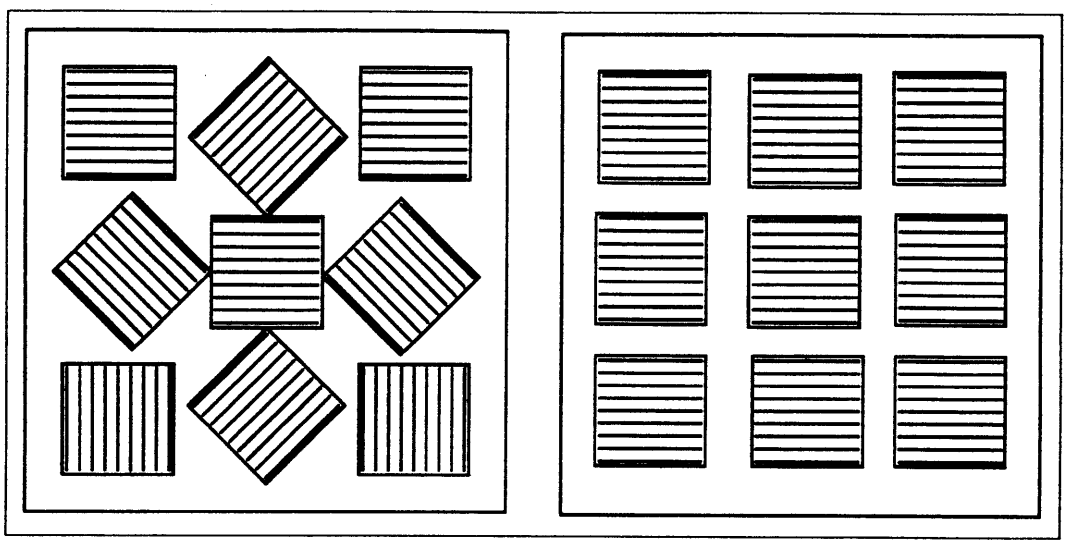


Abbildung 1: Künstlerisches versus funktionales Layout

Das Layout legt ebenfalls die Anzahl der Spalten und die Spaltenbreite fest. Diese muß so gewählt sein, daß beim üblichen Betrachtungsabstand eine gute Lesbarkeit des Textes gewährleistet ist. Dabei ist neben dem Zeilendurchschuß (Abstand zweier aufeinanderfolgender Zeilen), der gewählten Schriftart (Font) und der Schriftgröße die Anzahl der

Als eine Möglichkeit dazu haben sich intelligente, multimodale Präsentationssysteme herausgestellt, die einen wichtigen Baustein für Benutzerschnittstellen der nächsten Generation darstellen.

Das Ziel des Projektes WIP (Wissenbasierte Informationspräsentation) besteht in der automatischen Generierung von multimodalen Dokumenten (Kombination natürlicher Sprache, Graphik, Animation und Gestik) in Abhängigkeit von spezifischen Generierungsparametern. Dabei soll die Präsentation durch Variationen im Layout und der Modi- /Medienkombination auf den individuellen Benutzer und die jeweilige Präsentationssituation zugeschnitten sein (vgl. [Wahlster et al. 89, Wahlster et al. 91, Wahlster et al. 92]).

Ein wesentlicher Unterschied zu bisherigen Systemen besteht hier in der koordinierten Generierung von Texten und Graphiken auf der Basis einer gemeinsamen Repräsentation.

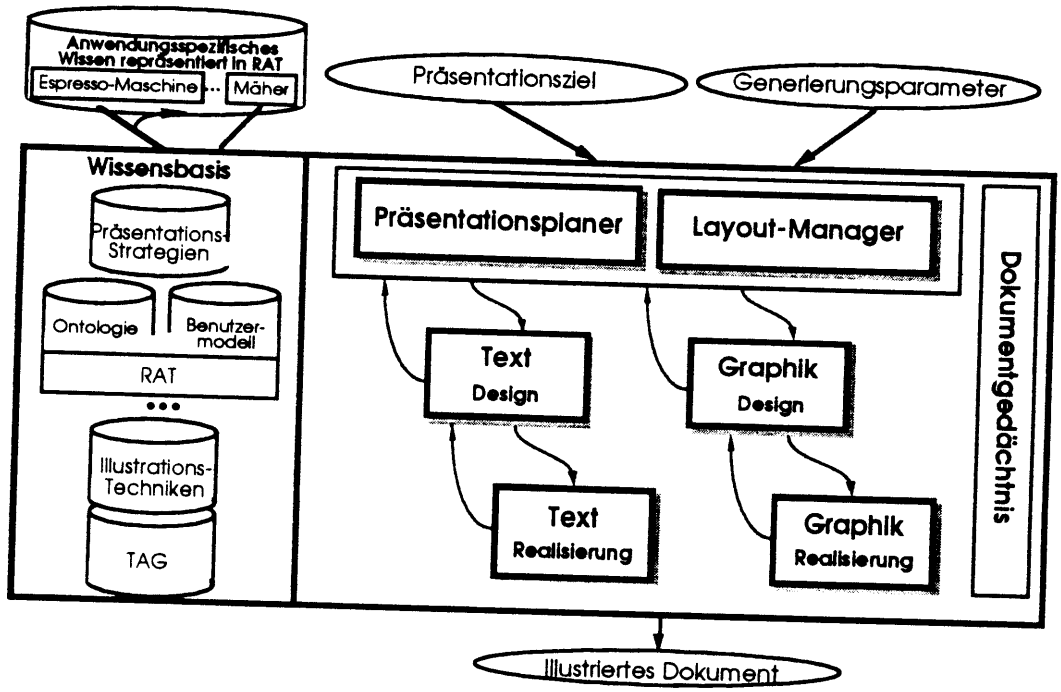


Abbildung 2: Die Architektur des Systems WIP

In unterschiedlichen Präsentationssituationen soll WIP das in einer formalen Wissensrepräsentationssprache vorliegende Wissen auf flexible Weise jeweils angemessen darbieten.

Dabei zeichnet sich das WIP-System durch eine kontextgesteuerte Auswahl der darzubietenden Information aus, sowie deren multimodale und multilinguale Präsentation. Die Architektur von WIP sieht deshalb neben den mediaspezifischen Generatoren einen Präsentationsplaner und einen Layout-Manager vor (vgl. Abb. 2).

Die Architektur des Systems WIP ist mittels zweier paralleler Kaskaden von Prozessoren organisiert, die von einem Präsentationsplaner und einem Layout-Manager moderiert

werden. Der Präsentationsplaner ist dabei verantwortlich für die Bestimmung des Inhalts der Darstellung, sowie für die Wahl und Kombination der verschiedenen Modi (s. [André & Rist 90a, André & Rist 90b]).

1.4 Die Architektur des Systems LayLab

Die Aufgabe des Layout-Managers, der in Teilen durch das hier vorgestellte System *LayLab* realisiert ist, besteht in der Bestimmung eines ästhetisch optimalen Layouts für ein multimodales Dokument unter Berücksichtigung verschiedener Randbedinungen, wie etwa Platzrestriktionen, Ausgabemedium und Benutzer. Für weitere Informationen sei der interessierte Leser an dieser Stelle auf die Arbeiten von [Graf 90, Graf 91, Graf & Maaß 91, Maaß 92] verwiesen.

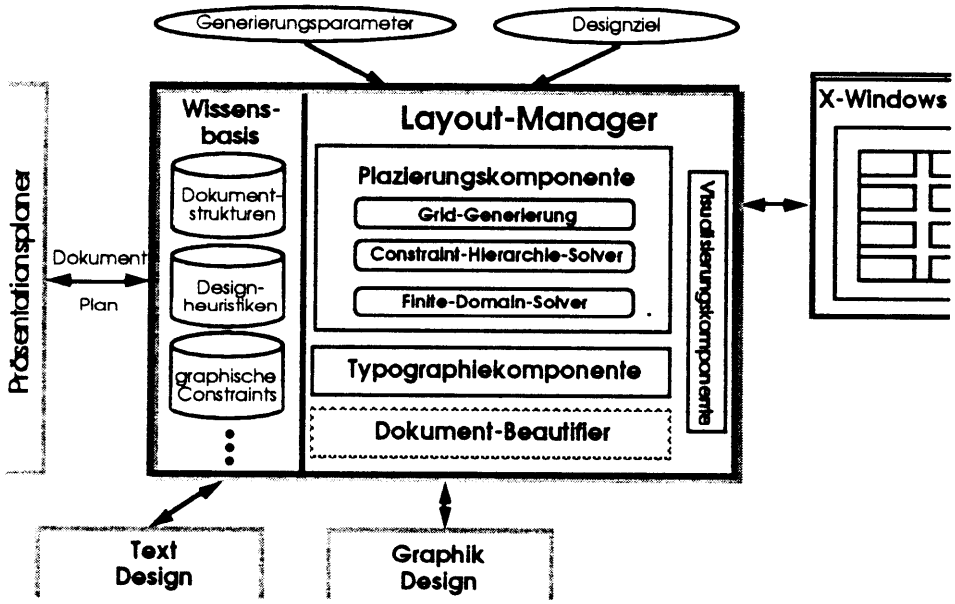
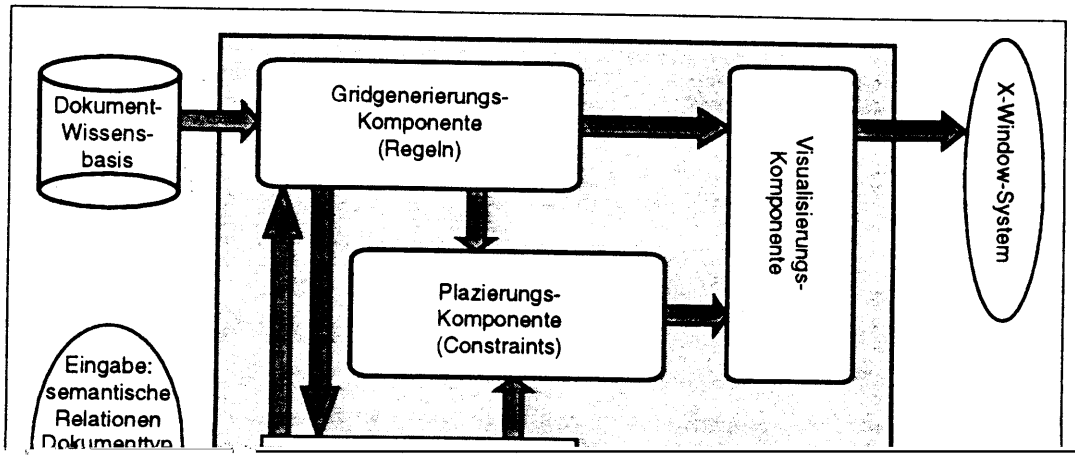


Abbildung 3: Der Layout-Manager in WIP

LayLab muß dazu die von den media-spezifischen Generatoren erzeugten Text- und Graphikboxen entsprechend der vom Präsentationsplaner spezifizierten semantischen Relationen auf einem Dokument zu arrangieren, d.h. es sind Größe und Position der individuellen Layoutelemente zu bestimmen.

Für dieses als Prototyp konzipierte System wurde zunächst die Schnittstelle zum Präsentationsplaner durch eine Testumgebung ersetzt. Dabei handelt es sich um eine menüorientierte Oberfläche (vgl. Kap. 4), die eine direkte Eingabe von Layoutobjekten und semantischen Relationen zwischen diesen ermöglicht, wie sie von den verschiedenen Generatoren geliefert werden.

Die eingegebenen Relationen werden in einem internen *Objektverwalter* abgelegt, der allen

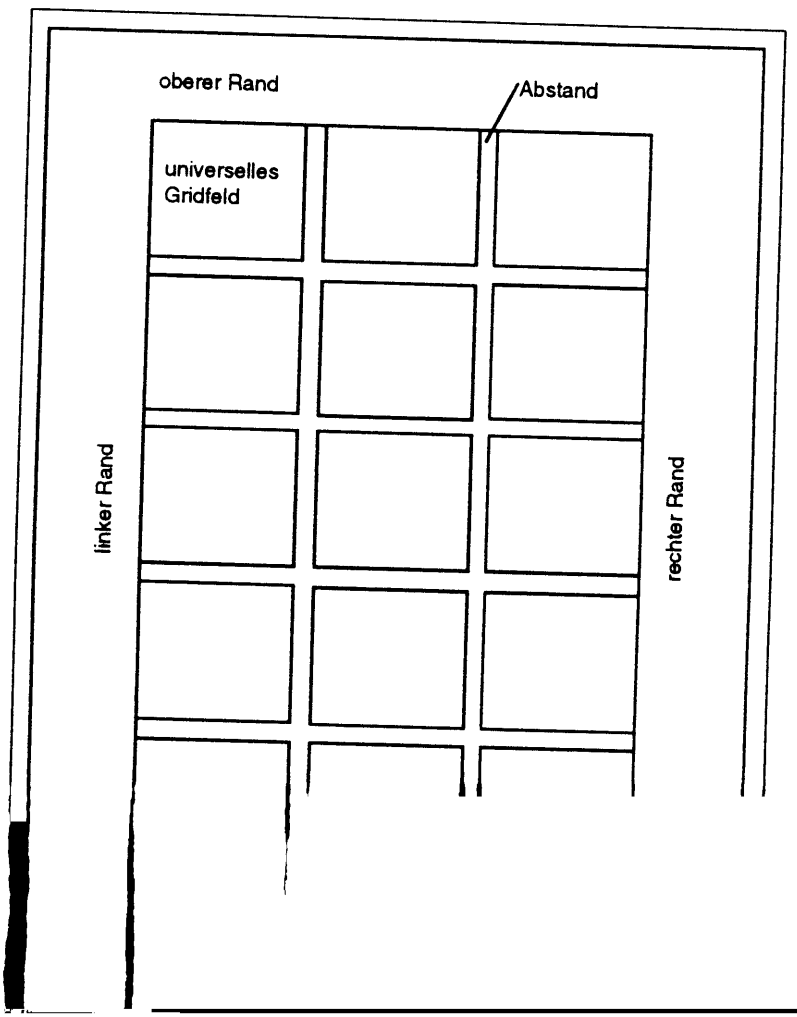

Dokument-
Wissens-
basis
Gridgenerierungs-
Komponente
(Regeln)
Visualisierungs-
Komponente
X-Window-System
Plazierungs-
Komponente
(Constraints)
Eingabe:
semantische
Relationen

trum gerecht werden, visuellen Zusammenhalt garantieren und einen gewissen Anspruch auf Dauerhaftigkeit Rechnung tragen." [Stankowski & Duschek 89]

2.1 Das Konzept des Design-Grids

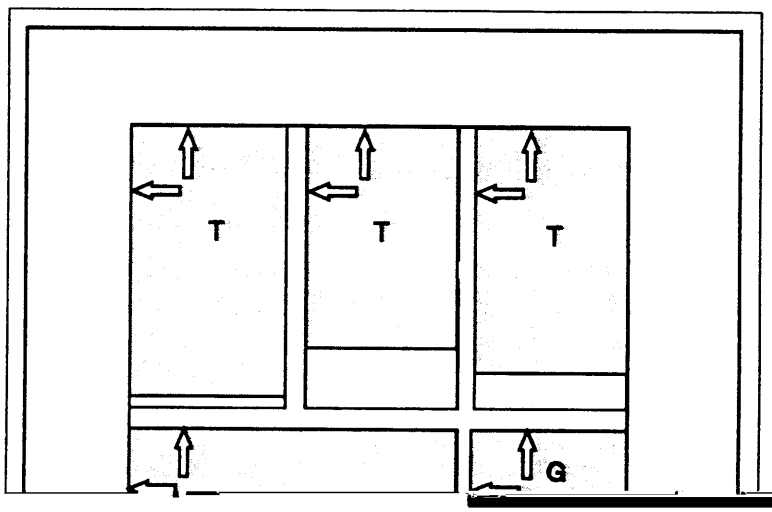
Um einerseits die einheitliche und funktionale Gestaltung eines mehrseitigen Dokuments zu erleichern und andererseits den Suchraum für die Positionierung der Layoutobjekte (Graphiken und Texte) erheblich einzuschränken, haben wir uns für die Verwendung eines typographischen Rasters entschieden. Ähnliche Ansätze werden auch im System *GRIDS* [Feiner 88] und in den Arbeiten von Beach [Beach & Stone 83, Beach 85] und Friedell [Friedell 84] verfolgt.

Die Grundlagen des Rastersystems sind bereits seit den zwanziger Jahren durch einige Vertreter der sachlich-funktionellen Typographie im Bestreben nach einem Ordnungssystem in der visuellen Kommunikation entwickelt worden.



wickelt [Müller-Brockmann 81]. Es wurde geschaffen als Arbeitsinstrument zum sicheren und schnellen Konzipieren, Organisieren und Gestalten von visuellen Problemlösungen. Abbildung 3 zeigt ein Beispiel für ein typisches Grid.

Definition (nach [Müller-Brockmann 81]): Ein *typographisches Raster (engl. Grid)* besteht aus einer Menge von horizontalen und vertikalen Linien, die eine zweidimensionale Fläche gitterförmig in kleinere Felder zumeist gleicher Größe (universelle Gridfelder) unterteilt. Diese Felder werden getrennt durch jeweils gleiche Abstände in horizontaler wie in vertikaler Richtung. Umgeben wird das Raster von einem Rand.



che von Rasterfeldern. Paßt eine Graphik nicht auf Anhieb, so wird versucht, diese durch Zoomen oder Schneiden in das Raster einzupassen (vgl. Abb. 6).

2.2 Wahl der Granularität des Rasters

Durch die Wahl bzw. die Konstruktion des Rasters werden bereits vor der eigentlichen Plazierung der Layoutobjekte wichtige designrelevante Richtlinien für die Gestaltung des Dokuments aufgestellt.

So wird durch das Grid der Satzspiegel des Dokuments festgelegt und die Randbereiche bestimmt. Zum anderen werden durch das Raster die Möglichkeiten für die Plazierung der Layoutobjekte auf eine endliche Anyahl von Koordinaten eingeschränkt. Damit hat die Auswahl des Grids einen entscheidenden Einfluß auf das Layout des endgültigen Dokuments.

Die Aufteilung in Spalten (Satzspiegel) beeinflußt nicht nur stark den Lesefluß des Benutzers, sondern sie ist auch mitprägend für den Charakter eines Dokuments. Desweiteren läßt sich durch eine geschickte Konstruktion des Grids erreichen, daß die Leerräume zwischen den einzelnen Objekten, die bei der Anwendung eines Rasters unvermeidlich sind, auf ein Minimum beschränkt werden können.

Aus diesem Grund kann nicht einfach ein beliebiges festes Raster für alle Dokumente und Inhalte verwendet werden, sondern für jedes neue Dokument und jede Menge von Layoutobjekten ist ein individuell erzeugtes Raster nötig.

Die Aufgaben, die sich daraus für die Erstellung eines Grids ergeben, lassen sich wie folgt explizieren:

- Festlegung des Satzspiegels entsprechend der Funktion des Dokuments
- Berechnungdes universellen Gridfeldes
- Anpassen der Graphiken
- Formatieren der Texte
- Berücksichtigung der dokumentspezifischen Generierungsparameter (Dokumenttyp, Benutzer, etc.)
- Bestimmung der Randbereiche und der Abstände zwischen den Layoutobjekten

Durch die Granularität wird die Flexibilität der Plazierung bestimmt. Je feiner das Raster, desto höher ist die Variablität der Plazierung.

Die Anforderungen an ein gutes Grid sind also im einzelnen die Sicherstellung von Uniformität, Kohärenz, Konsistenz, Sachlichkeit, Tranzparenz und Funktionalität für das Dokument.

2.3 Die Konstruktion des Grids

Zunächst allgemein zur Arbeitsweise des Algorithmus. Wie bereits erwähnt arbeitet der Algorithmus in mehreren Stufen (vgl. Abb. 7):

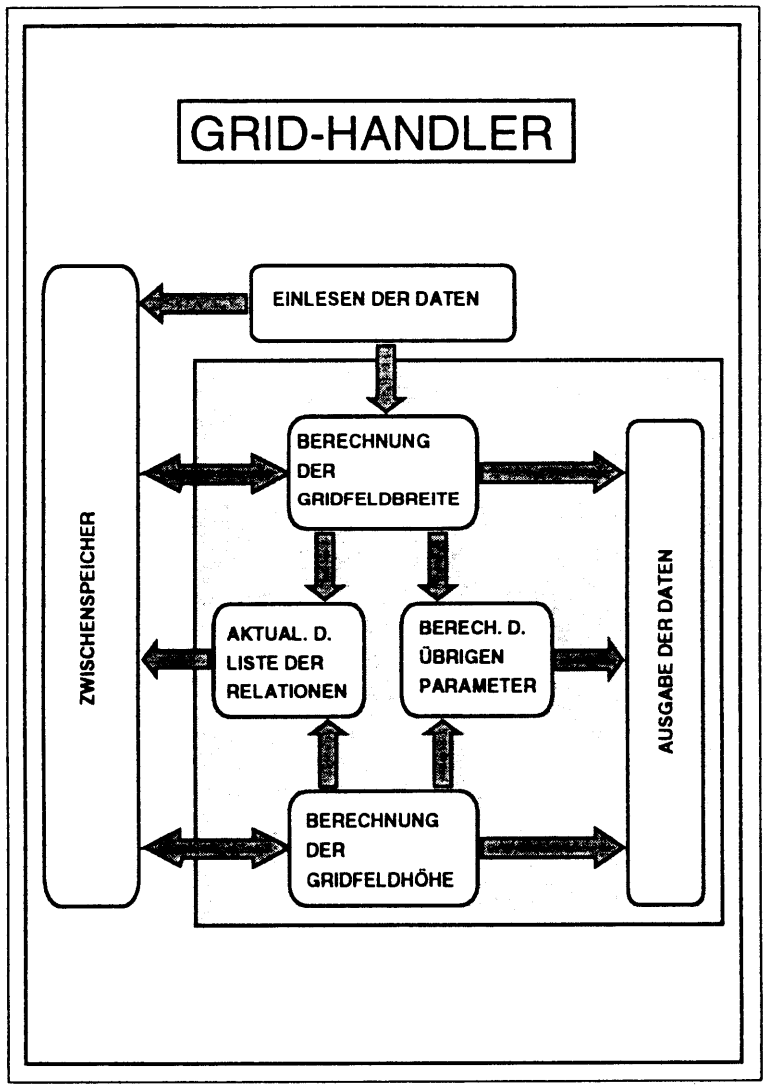


Abbildung 7: Architektur der Gridgenerierungs-Komponente

Als erstes wird die Breite des universellen Gridfeldes berechnet. Dazu wird in Abhänigkeit von den Dokumentdaten eine Liste möglicher Spaltenbreiten (= Gridfeldbreiten) berechnet. Dabei ist es nicht nötig, daß das Grid die Dokumentseite (abzüglich der Mindesträander des gewählten Dokumenttyps) bis an die Ränder ausfüllt. Es besteht ein gewisser Spielraum, der eine bessere Anpassung an die Objekte ermöglicht.

Für die Berechnung der Breite spielen zunächst nur die Graphiken eine Rolle. Für jede der möglichen Spaltenbreiten wird nun überprüft, welche Graphiken, auch gezoomt oder geschnitten, in diese Breite passen. Für die anderen werden die

Diese Summe läßt sich interpretieren als Maß für die Qualität der jeweiligen Spaltenbreite. Von diesen bewerteten Breiten wird diejenige mit der besten Bewertung als Breite des universellen Gridfdeldes ausgewählt.

Dann wird die Liste der Layoutobjekte aktualisiert. Das bedeutet, daß für die Texte die Höhe berechnet wird, indem sie in der aktuellen Spaltenbreite umgebrochen werden. Diese Höhe wird dann zusammen mit der Breite in die jeweiligen Objektslots eingetragen. Die Liste der Objekte besitzt damit eine konsistente Form, da nun die Slots für Höhe und Breite bei allen Objekten korrekt belegt sind (vorher nur Graphiken).

Anschließend wird die Höhe des universellen Gridfeldes bestimmt. Dabei muß die Höhe immer ganze Vielfache der Zeilenhöhe betragen. Die Höhe ergibt sich nun dadurch, daß die Höhe des niedrigsten Objekts aus der Liste der Layoutobjekte bestimmt wird und diese auf ganze Vielfache der Zeilenhöhe abgerundet wird.

Als letztes werden noch weitere Daten berechnet. Das ist zum einen der Abstand der Gridfelder voneinander. Zum anderen gibt der Algorithmus aber auch den Abstand des gesamten Rasters vom linken und vom oberen Rand des Dokuments an, der durch Ausbalancieren der Rasters innerhalb der defaultmäßigen Ränder erfolgt. Schließlich wird noch

Aufzoomen: Teste: `(abstand/objekt-breite)` $\leq$ `obere_Zoomgrenze`
Abzoomen: Teste: `(abstand/objekt-breite)` $\geq$ `untere_Zoomgrenze`

* Ist sie durch Schneiden (bis zu erlaubter Schnitt-Grenze) einzupassen? Teste: abstand $\leq$ zulässiger_Abschnitt (Die berechneten Zoomfaktoren bzw. Abschnitte werden zwischengespeichert) Ist auch dadurch keine Paßgenauigkeit für die Graphik zu erreichen, wird
* der Fehler für diese Graphik berechnet (Abstand zur nächsten Spaltengrenze) (mod objekt-breite spalten-breite) Für jede getestete Spaltenbreite werden die einzelnen Fehler, die die Graphiken verursachen würden, aufsummiert. Dieser Wert (Gesamtfehler) ist ein Maß für die 'Güte' dieser Spaltenbreite.

- Auswahl der Spaltenbreite mit dem geringsten Gesamtfehler als Breite des universellen Gridfeldes.

- Aktualisierung der Liste der Layoutobjekte
  - Die Zoomfaktoren und Abschnitte der Objekte werden entsprechend der berechneten Gridfeld-Breite im Objekt-Verwalter aktualisiert.
  - Die Texte werden entsprechend der Spaltenbreite umgebrochen und die Werte für die Höhe in die Objekte eingetragen.

- Berechnung der Höhe des universellen Gridfeldes
  - Suchen des Layoutobjektes (Text oder Graphik) mit der geringsten Höhe.
  - Abrunden auf das nächste ganze Vielfache der Zeilenhöhe. Dieser Wert ist die Höhe des universellen Gridfeldes.

- Berechnung weiterer Daten
  - Berechnung der Anzahl der Gridfelder in X- und Y-Richtung.
  - Ausbalancieren des berechneten Grids auf der Dokumentseite. Die Differenz zwischen den minimalen Randbereichen (Default für Dokumenttyp) und den neuen Abständen wird entsprechend der prototypischen Verhältnissen für diesen Dokumenttyp als neuer Randbereich vorgesehen.
  - Für jedes Layoutobjekt wird die Anzahl der Gridfelder, die es belegt, berechnet (relative Größe) und zusätzlich im Objekt-Verwalter abgelegt.

- Übergabe der Daten
  - Datenübergabe an die Plazierungs-Komponente:
    Übergeben wird die Anzahl der Gridfelder in X- und Y-Richtung. Zusammen mit der relativen Größe der Objekte ist damit die Plazierung möglich.

- Datenübergabe an die Visualisierungs-Komponente:
  Übergeben werden alle zur visuellen Darstellung des Rasters benötigten Daten:
  * die Breite und Höhe des universellen Gridfeldes
  * der Abstand zwischen jeweils zwei Gridfeldern
  * die Anzahl der Gridfelder in X- und Y-Richtung
  * die Randbereiche des Rasters (angegeben durch die Abstände vom linken- bzw. oberen Rand der Dokumentseite (x-offset, y-offset).

2.4 Die Dokumentwissensbasis

LayLab soll in der Lage sein, die verschiedensten Arten von Dokumenten zu gestalten. Dazu ist es nötig, Wissen bereitzustellen über die verschiedenen Arten von Dokumenten. Dies geschieht in der Dokumentwissensbasis.

Benötigt werden vor allem Defaultwerte über die Größe der Dokumentseite, die für diesen Dokumenttyp üblichen Randbereiche und die Anzahl der Spalten. Bei einigen dieser Größen sind mehrere Wahlmöglichkeiten vorgesehen, z.B. sind für den Dokumenttyp 'Zeitung' fünf, sechs oder sieben Spalten zulässig.

Daneben wurde, um eine noch größere Flexibilität zu gewährleisten, die Möglichkeit vorgesehen, gewisse Toleranzen für die Randbereiche zu erlauben. Es fällt dadurch leichter, ein Grid zu konstruieren, daß die geforderten Bedinungen (s. 2.2) optimal erfüllt.

Da eine Typographie-Komponente bislang noch nicht implementiert ist, wird diese von der Dokumentwissensbasis simuliert. Das bedeutet, daß die Dokumentwissensbasis zusätzlich zu dem eigentlichen Dokumentwissen typographische Informationen enthält über Art und Größe der für dieses Dokument üblicherweise verwendeten Schrift, sowie über den Zeilenabstand und die durchschnittliche Schriftbreite.

Die bereits definierten Dokumentypen sind bisher:

- Bedienungsanleitung im Querformat
- Bedienungsanleitung im Hochformat
- Zeitung
- Memo und
- Overheadfolien.

Diese Definitionen lassen sich betrachten als allgemeine Prototypen (Konzepte) des jeweiligen Dokumentyps. Sie enthalten noch einige Freiheitsgrade. Möchte nun ein Benutzer

einen restriktiveren Dokumenttyp generieren, so läßt sich dieser als Instanz eines der bereits bestehenden Konzepte definieren, falls dies sinnvoll ist.

2.5 Erweiterungsmöglichkeiten

Neben den schon angesprochen Erweiterungsmöglichkeiten der Dokumentwissensbasis durch Hinzufügen von neuen Dokumenttypen gibt es vor allem noch Verbesserungsmöglichkeiten im Bereich der Handhabung von Typografiedaten.

Da im Moment noch keine Typografiekomponete für das WIP-System existiert, erfolgt die Festlegung von Font und Schriftgröße für ein Dokument durch die Defaultwerte aus der Dokument-Wissensbasis.

Wenn das auch bei verschiedenen Anwendungen durchaus erwünscht ist (für eine Zeitung z.B. ist es wichtig, daß sie für den Leser einen hohen Wiedererkennungswert hat, also immer in der gleichen Schrift erscheint und stets ein ähnliches Layout besitzt), so ist dieses Verfahren doch im Einzelfall oft zu statisch.

Ebenso findet im Moment die Berechnung des Zeilenumbruchs ausschließlich aufrund von statistischen Werten statt, also ohne die Struktur des Textes zu berücksichtigen. Auch hier ließe sich durch Zusammenarbeit mit einer Typografiekomponete eine bessere Lösung finden.

3 Die Constraint-basierte Plazierungskomponente

Die Aufgabe der Plazierungskomponente ist es die Layoutobjekte, möglicherweise verschiedenener Modalität, mittels semantisch-pragmatischer Relationen, welche von einem Präsentationsplaner vorgegeben werden, auf dem von der Gridgenerierungskomponente erzeugten Raster anzuordnen (vgl. Kap. 2). Dabei werden Constraints sowohl zur Repräsentation der Beziehungen zwischen Layoutobjekten, als auch als Berechnungsmodell zur Plazierung der einzelnen Objekte im Dokument verwendet.

3.1 Repräsentation der verschiedenmodalen Objekte

Die Ausgabe der einzelnen Generatoren in WIP sind von unterschiedlicher Modalität. Um diese verschiedenartigen Objekte anordnen zu können, mu eine Abbildung gefunden werden, die eine homogene Handhabung aller Objekte gestattet. Dabei ist darauf zu achten, da geometrische Ausmae, sowie inhaltsabhängiges Wissen erhalten bleibt. Hierzu werden die Objekte auf Layoutobjekte abgebildet, die durch Typ, Größe, x/y-Koordinate und Relationen zu anderen Objekten eindeutig beschrieben werden.

Da die äußere Form eines Layoutobjektes wird durch seine geometrischen Ausmaße bestimmt wird, kann es durch ein umschreibendes Rechteck modelliert werden, was durch die Koordinate der linken oberen Ecke, der Höhe und der Breite bestimmt ist. Dadurch erhält man die Abbildung *frame*:

$$frame : OBJECTS \rightarrow FRAMES,$$

wobei OBJECTS die Menge der Objekte, FRAMES $\subseteq$ NxNxNxN.
Für $o \epsilon OBJECTS$ gilt:

$$\begin{aligned} pr_1(frames(o)) &= x - coordinate(o) \\ pr_2(frames(o)) &= y - coordinate(o) \\ pr_3(frames(o)) &= higth(o) \\ pr_4(frames(o)) &= width(o), \end{aligned}$$

mit der naheliegenden Definition der Funktionen *x-coordinate, y-coordinate, higth* und *width* und pr_i, mit $i \epsilon \{1,2,3,4\}$, die Projektionsfunktion ist.

Nach Abbilden der Objekte auf Rahmen können diese unter Verwendung der Objetktypen, der Generierungsparameter und den Relationen mittels Constraints plaziert werden.

3.2 Der Repäsentationsformalismus für semantisch-pragmatische Beziehungen

Semantisch-pragmatische Relationen sind Aussagen über lokale, topologische Beziehungen zwischen Objekten. Sie lassen im allgemeinsten Fall Alternativen in der Umsetzung zu, so daß durch sie implizierte Aussagen durch verschiedene topologische Anordnungen erreicht werden können. Aus diesem Grund wird ein deklarativer Formalismus benötigt, der Beziehungen zwischen Layoutobjekten unterstützt und Alternativen in der Auswahl zuläßt. Eine natürliche Art eines solchen Repräsentationsformalismus sind Constraints. Es gibt in der Literatur verschiedene Konzepte von Constraint-Solvern, wie z.B. in den Systemen ThingLab [Borning 81] oder CHIP [Van Hentenryck 89]. Im Fall der pragmatisch-semantischen Beziehungen handelt es sich um arithmetische Berechnungen zwischen den verschiedenen, durch eine Relation verbundener Layoutobjekten. Hierfür ist ein Constraints-Solvers der Value-Inference-Klasse geeignet, da mittels diesem Beziehungen feste Werte mittels einem Propagierungsmechanismus durch ein Constraint-Netzwerk weitergeleitet werden können.

3.3 Constraint-basierte Verarbeitung von semantisch-pragmatischen Relationen

Semantisch-pragmatische Beziehungen unterteilen sich in drei verschiedene Ebenen. Die untere Ebene, in der sogenannte primitive Beziehungen verarbeitet werden, behandeln grundlegende Verbindungen zwischen Variablen der Layoutobjekte. Im Constraint-Formalismus sind sie als sogenannte *Primitve Constraints* anzusprechen. Auf diesen primitiven Beziehungen aufbauend, gibt es gruppierende Beziehungen, die mehrere primitive und auch gruppierende Beziehungen zu einer Beziehung zusammenfassen. Mittels diesem Aggregierungsmechanismus ist es möglich beliebig komplexe Beziehungen aus bereits definierten Beziehungen aufzubauen. Der Constraint-Formalismus unterstützt diese Form von Beziehungen durch *Compound-Constraints.* Eine besonderer Fall tritt auf, wenn Relationen verarbeitet werden, die keine genau definierte Anzahl von Objekte umfassen, also in ihrer Form dynamisch sind. Hierbei ist es zur Laufzeit notwendig entsprechende Constraints aufzubauen, die passend zur Anzahl der Objekte primitive und gruppierende Constraints miteinander verbinden. Solche Relationen werden im Constraint-Formalismus durch Dynamische Constraints verarbeitet.

3.3.1 Primitive Beziehungen

Primitive Beziehungen betrachten die einfachsten Abhängigkeiten zwischen zu plazierenden Objekten. Sie betreffen die geometrischen Verhältnisse zwischen zwei Objekten. Die

Relationen, die diese Beziehungen repräsentieren sind beispielsweise 'BESIDE', 'UNDER' und 'EQUAL'. Sie lassen sich mit Hilfe von Constraints in einfacher Weise realisieren.

Beispielsweise wird das Constraint 'CONNECT' wie folgt definiert:

```
Name :        'CONNECT
Methods : '(  (- ( - X2 Breite) DIST)
              (- ( - X2 X1) DIST)
              (+ ( + X1 Breite) DIST)
              (- ( - X2 X1) Breite) ),
```

wobei das erste Element der Methodenliste *X1*, das zweite *Breite*, das dritte *X2* und das vierte *DIST* berechnet. Durch dieses Constraint wird die Relation

$$X2 = X1 + Breite + DIST$$

ausgedrückt.

Die Bedeutung des Constraint 'CONNECT'

plexeren hin zu ereichen.

Ein Beispiel für ein einfaches Compound-Constraint ist 'CONTRAST'. Ausgegangen wird davon, daß ein Kontrast nur zwischen zwei Bilder bestehen kann
Die pragmatisch-semantische Relation 'CONTRAST' drückt eine inhaltlich gegensätzliche, graphische Aussage aus. Die üblichen Darstellungsformen sind, daß die Graphiken horizontal oder vertikal angeordnet werden. Dabei ist darauf zu achten, daß kein anderes Objekt zwischen beide plaziert wird. Weiterhin ist wichtig, daß beide Objekte auf einer gemeinsamen Linie angeordnet werden und daß beide Objekte sich nicht überlappen.

Die vereinfachte Struktur des Constraints ist dann die folgende:

```
Name :        'CONTRAST
Methods : (   ( (BESIDE (x G1) (Breite G1) (x G2) )
              (EQUAL (y G1) (y G2))
              ( (UNDER (y G1) (Höhe G1) (y G2))
              (EQUAL (x G1) (x G2)) )
```

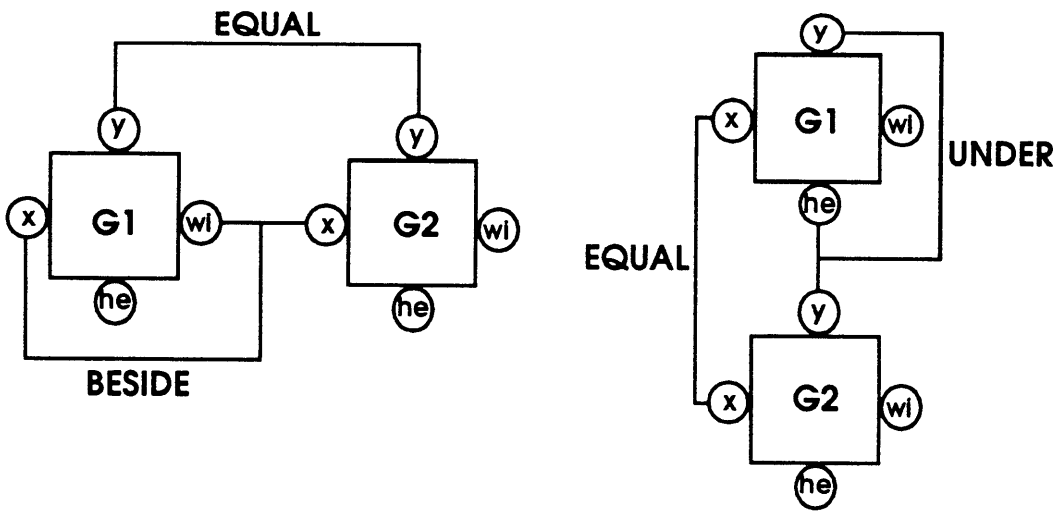


Abbildung 8: Alternative Constraint-Netze für die Relation *CONTRAST*

Die Semantik des Constraints bewirkt eine horizontale bzw. im zweiten Fall eine vertikale Alignierung der Objekte an die durch das erste Objekt vorgegebene Linie. Die Alternative wird hierbei dann angewandt, wenn die erste Möglichkeit nicht erfüllt werden kann. Zusäatzlich gewährleisten die Constraints, daß die Layoutobjekte sich nicht überlappen.

Das Compound-Constraint wird durch die Constraint-Variablen der Layoutobjekte parametrisiert, aufgerufen, wobei die Reihenfolge der Objekte und der einzelenen Parameter in der Parameterliste signifikant ist. Die Parameterliste unterteilt sich in jeweils Gruppen

zu vier Parametern, wobei der erste die x-Variable, der zweite die y-Variable, der dritte die Höhen-Variable und der vierte die Breiten-Variable übernimmt.
In der Definition eines Constraints sind solche Parameter durch Ziffern, entsprechend ihrer Stelle in der Parameterliste und einem vorangehenden '?' gekennzeichnet.

3.3.3 Dynamische Beziehungen

Ein Problem, wenn Relationen mit Constraints ausgedrückt werden sollen, die eine beliebige Anzahl von Objekten betrifft. Dies tritt beispielsweise bei der 'SEQUENCE'-Relation auf. A priori ist nicht feststellbar, wieviel Objekte eine Sequenz bilden. Von daher ist es notwendig zur Laufzeit ein Constraint mit der passenden Struktur zu generieren, was der Anzahl der Objekte entspricht. Dieses neugenerierte Constraint wird mit einem Index, der die Anzahl der signifikanten Objekte angibt, in einer Constraint-Basis abgelegt und kann alsdann genutzt werden. Tritt der Fall auf, daß ein Constraint benötigt wird, welches noch nicht existiert, wird eine Generierungskomponente benutzt, die für eine Reihe von dynamischen Constraints ein neues Constraint erzeugt.

Ein Beispiel hierfür ist die Relation 'SEQUENCE'. Sie stellt eine Beziehung zwischen einer Reihe von drei Graphik- und Textobjekten her. Es gibt eine übliche Art eine Sequenz darzustellen und eine alternative. Mittels Constraints kann dies wie folgt ausgedrückt werden:

```
Name : 'SEQUENCE
Variables : G1, G2, G3, T1, T2, T3
Methods : (
        (DefDynRelForm :Typ 'SEQUENCE
        :MethodForm '(
        ;;; Diese Methoden ordnen die Graphiken und Texte vertikal an
        ((CONNECT ^6 ^2 ^3)
        (EQUAL ^5 ^1)
        (EQUAL ^2 ^ ^2)
        (CONNECT ^^1 ^1 ^4)
        (CONNECT $1 ^1 ^4)
        (IF-LESS-EQUAL $2 $1 MAXX)
        (CONNECT $1 ^5 ^8)
        (IF-LESS-EQUAL $2 $1 MAXX ))
        ;;; Diese Methoden ordnen die Graphiken und Texte vertikal an
        ((CONNECT ^5 ^1 ^4)
        (EQUAL ^6 ^2)
        (EQUAL ^1 ^^1)
        (CONNECT ^ ^2 ^2 ^3 ))))
        )
```

Es sind hierbei zwei alternative Methoden für die Plazierungsangegeben.
Die ersten sechs Constraints, welche in der Methodenliste aufgeführt werden, repräsentieren das horizontale Ausrichten der Sequenz. Horizontal bedeutet, daß Graphiken horizontal über den dazugehörenden Texten angeordnet werden.
Die Alternative hierzu ist das vertikale Ausrichten der Graphiken mit den dazugehörenden Texten.

Ein *dynamisches Constraint* wird in der gleichen Weise wie ein Compound-Constraint parametrisiert. Da die Definition eines dynamischen Constraints zur Laufzeit zu einem normalen Compound-Constraint expandiert werden muß, werden nicht die üblichen '?'-Parameter verwendet, sondern '^'-Parameter. Die Definition eines dynamischen Constraints beschreibt die Beziehungen von Untereinheiten der semantisch-pragmatischen Relation. Im Beispiel der Sequenz besteht diese aus beliebig vielen Grphik-Text-Blöcken. Die '^'-Parameter werden zur Laufzeit zu '?'-Parametern expandiert, wobei sie entsprechend der Nummer der Untereinheit der sie zugehören, einen Offset bekommen (bei einer Sequenz beträgt dieser Offset 8). Da diese Untereinheiten Verbindungen zu anderen Untereinheiten haben, müssen diese ebenfalls per Constraint ausgedrückt werden. Dies geschieht mittels '^^'-Parameter, die die gleiche Bedeutung wie '^'-Parameter haben, nur

das sie sich auf die Untereinheit vor der aktuellen beziehen.

Neben diesen Constraints, die Beziehungen zwischen Layoutobjekten repräsentieren, gibt es eine Reihe von Kontroll-Constraints (z.B. 'IF-LESS-EQUAL'). Solche Constraints

- Rahmen dürfen sich nicht überlappen
- Der Anschluß neuer Rahmengruppen muß effizient sein.
- Aggregierung zu beliebigen Strukturen

Dadurch, daß ein Rahmen mit den Rahmen der anderen Gruppe verbunden wird, können sich diese Rahmen niemals überlappen, da alle weiteren Gruppen wiederum durch deren umfassenden Rahmen repräsentiert werden. Der zweite Punkt betrifft den Fall, daß zu einer Gruppe von Objekten etwas hinzugefügt werden soll. Der umfassende Rahmen verhindert, daß eine Suche stattfinden muß, um das Maximum in einer Dimension, an der die beiden Gruppen zusammenkommen, zu finden. Der letzte Punkt drückt die Hierarchisierung von Gruppen aus. Es ist mit solchen Rahmen möglich beliebige höhere Strukturen zu bilden und in das Constraint-Netz einzufügen ohne sich um die Einzelteile der Gruppe zu kümmern. Veränderungen an den Variablenbelegungen der Rahmen-Variablen propagieren sich bis zu den untersten Rahmen fort.

Das Bild zeigt einen umfassenden Rahmen eines Constraint-Netz 'SEQUENCE'. Die Variablen des umfassenden Rahmens sind durch Constraints mit den Maxima bzw. Minima der gesamten Gruppe verbunden. Die Bestimmung der Maxima bzw. Minima wird bei der Instanziierung der Gruppe bestimmt und über 'EQUAL'-Constraints mit den Variablen des Rahmens verbunden.

3.5 Beschreibung des inkrementellen Constraint-Solvers

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:

1. 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 starte vom stärksten verbleibenden Constraint der Hierarchie.
2. Sind alle Methoden des Constraints blockiert, so markiere das Constraint als geblockt und betrachte es nicht weiter. Fahre in der Hierarchie weiter fort.
3. Sind die beiden ersten Punkte nicht erfüllt, so mache mit dem Constraint nicht und fahre ebenfalls in der Hierarchie fort.

Die Weiterentwicklung dieser einfachen Constraint Solving Strategie sind inkrementelle Solver, wie sie u.a. von A. Borning et al. im Rahmen des ThingLab-Projektes bei Xerox PARC entwickelt wurden (vgl. [Borning 81, Freeman-Benson et al. 90]). Zugrunde liegt hier ein inkrementeller Constraint Solver, der durch die Verwendung von Gewichtungen der Constraints die Möglichkeit der Hierarchisierung von Constraint beinhaltet. Durch diese Wichtungen können Constraints ihrer Stärke nach unterschieden werden und so Lösungen von Constraint-Netzen berechnet werden, ohne daß alle Constraints erfüllt worden sind. A priori wird zwischen unbedingten und bedingten Constraints unterschieden. Effizienzsteigernd wirkt sich hierbei aus, daß zu jeder Variablenbelegung Informationen über das Constraint und seine Wichtung bei der Variablen gespeichert werden. 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 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.

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	—

Die graphische Darstellung dieser Hierarchie ist in Abbildung 9 dargestellt. Die Pfeile repräsentieren Constraints, wobei an den Kanten die Wichtungen eingetragen sind. Gestrichelt dargestellte Constraints sind nicht erfüllte Constraints.

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

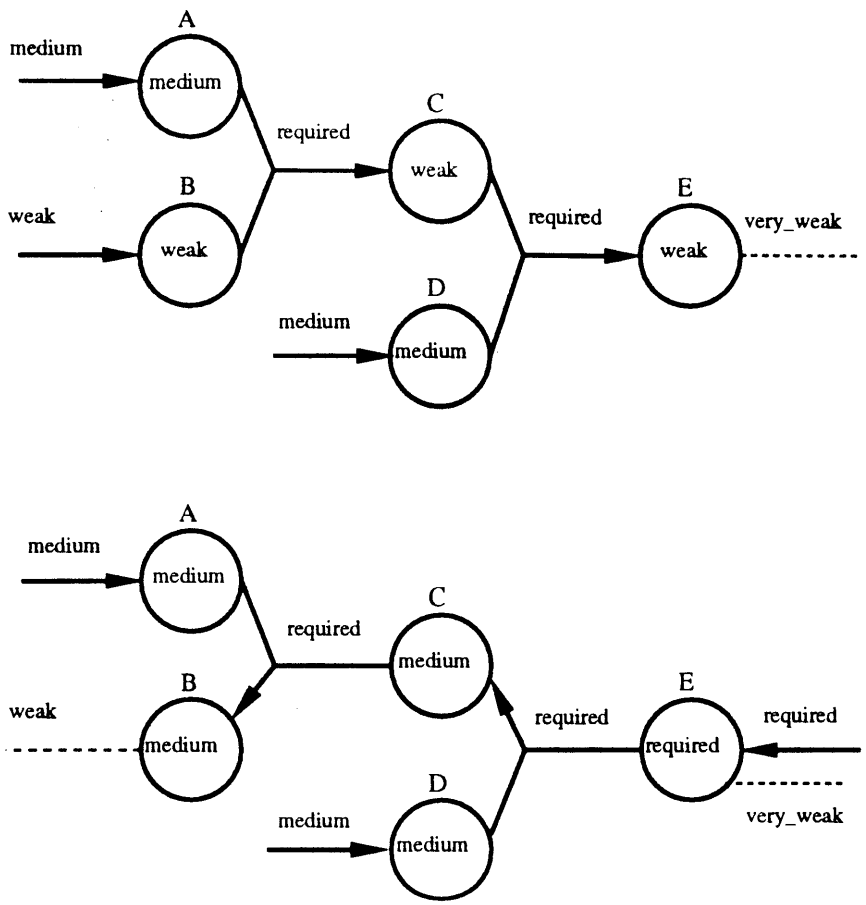


Abbildung 9: Constraint-Netz mit fünf Constraints

Definition : 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.

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

Wird im obigen Beispiel das Constraint mit der Wichtung *required* in das Netz an die Variable E angelegt, so findet der beschriebene Algorithmus statt, der prüft, 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. Im folgenden beginnt die Propagierung, welche eine Neuberechnung des Constraint-

Netzes solange weiterführt, bis entweder keine Constraints mehr vorhanden sind, die noch nicht ausgeführt wurden, oder die Constraints zu schwach sind um neue Belegungen zu berechnen.

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 für die Belegung der Variable C verwendet, da sie von den Variablen des Constraints 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.

Es gibt zwei Schnittstellenfunktionen um ein Constraint-Netz anzusprechen. Die erste Funktion *InsertConstraint* fügt ein neues Constraint ein, die Funktion *DeleteConstraint* entfernt eins aus dem Netz. Ebenfalls zur Verwaltung [redacted]

Mittels dieser Anweisung wird eine Additionsverbindung zwischen den Variablen A, B und C vorgenommen. Entsprechend den bereits bestehenden Wichtungen der Variablen wird entweder A, B oder C entsprechend den Methoden neu belegt. Im Falle,daß alle Variablen stärker als das Constraint gewichtet sind, z.B. mit *required*, wird das Constraint geblockt und in eine Warteschlange geschoben, bis die Ursache für seine Blockierung nicht mehr besteht.
Als Rückgabe liefert die Funktion die Kennung des Constraints, sodaß im Falle einer Rücknahme die Funktion *DeleteConstraint* mit der Kennung als Parameter aufgerufen werden kann. Beim Löschen des Constraints wird das Constraint aus dem Netz entfernt, die Belegung der durch das Constraint bestimmten Variable zurückgenommen, nachgesehen, ob ein anderes Constraint vorhanden ist, welches jetzt anwendbar ist und die Propagierung angestoßen, falls Änderungen auftreten.

4 Die Visualisierungs-Komponente

4.1 Überblick

Die Aufgabe der Visualisierungskomponente realisiert die Bildschirmausgabe der Gridberechnung und des Gesamtlayouts. Darüberhinaus bietet sie die Möglichkeit den aktuellen Stand des Layoutprozesses an Hand eines sog. *Previews* anzusehen. Dieses Modul wurde auf der Basis der auf der Symbolics unter Genera 8.0 zur Verfügung stehenden zu X-Windows kompatiblen Fensterumgebung realisiert (vgl. [Symbolics 90]. Dazu wurden folgende Funktionen implementiert:

- StartPreview
- Make-Grid
- Insert-Frame
- Delete-Frame
- Move-Frame

Die StartPreview Funktion dient dem Öffnen und Schließen eines Bildschirmfensters, zur Eingabe der Liste der Relationen, der Liste der Layoutobjekte sowie des Dokumenttyps. Die Make-Grid Funktion dient der Darstellung des universellen Grids auf dem Fenster. Die Insert-Frame Funktion hat die Aufgabe, die angeordneten Objekte auf dem Grid darzustellen. Die Objekte werden dazu in einem Array als eine Liste verwaltet. Auf diese Liste kann durch den Objektnamen zugegriffen werden. Die Delete-Frame Funktion löscht ein Objekt aus dem Grid und entfernt die genannte Objekt- Datenstruktur aus der Liste der Objekte. Die Move-Frame Funktion bewegt ein Objekt von einer Position zu einer neuen Position, wobei der Wert der alten Position in der Objekt-Datenstruktur durch die Werte der neuen Position ersetzt wird.

4.2 Das X-Window System

An dieser Stelle möchten wir eine kurze Einführung in die Funktionalität des *X-Window Systems* geben. Einen Überblick über X-Windows findet sich in [Scheifler & Gettys 86, McCormack & Asente 88, Jones 88]. Das X-System wurde seit 1984 am Massachusetts Institute of Technology (MIT) entwickelt und basiert auf dem Client-Server-Modell. Die zentrale Instanz, der X-Server, kommuniziert dabei mit mehreren Anwendungen, den sog. Clients. Zu den wesentlichen Aufgaben des X-Servers gehört die Verwaltung der Geräte eines Arbeitsplatzes (Display), zu denen neben der Tastatur und einer Maus auch

ein oder mehrere Bildschirme gehören. Der X-Server verwaltet die graphischen Ressourcen, wie die Bildschirmfläche in Form von rechteckigen Bildschirmfenstern (Windows), virtuelle Ausgabebereich im Speicherbereich des X-Servers (Pixmaps), Farbtabellen und Farbtabelleneinträge oder Bildschirm-Schriftarten (Fonts).

Das X-System ermöglicht die Bildung von Hierarchien sich überlappender Bildschirmfenster. Die Ausgabe eines Fensters wird automatisch an den Fenstergrenzen und an den Grenzen sich eventuell überlappender Fenster 'geklippt' (abgeschnitten). Bezugspunkt für die Ausgabe ist die linke obere Ecke des Fenster, so daß die Ausgabe unabhängig von der Position des Fensters auf dem Bildschirm erfolgen kann.

X-System bietet graphische Ausgabeprimitive wie Punkte, Linien, Rechtecke, Kreisbogen, Füllflachen und Rasterbildflächen an, deren Darstellung durch die Angabe von Graphikattributen in weiten Bereichen beeinflußt werden kann.

Eine Anwendung mit besonderer Aufgabe ist der Window-Manager. Er ermöglicht es dem Benutzer, die Lage und Größe des Fensters zu verändern. Dazu stellt er die entsprechenden Interaktionsmöglichkeiten zur Verfügung, beispielweise im Form von sensitiven Bereichen innerhalb eines Rahmens um ein Fenster. Er bietet üblicherweise auch die Möglichkeit, momentan nicht benötigte Fenster zu Piktogrammen (Icons) zu schrumpfen. Gleichzeitig kann ein Window-Manager auch eine Strategie für das Layout der Fenster und Icons realisieren, z.B kann er verhindern, daß sich Fenster der obersten Hierarchiestufe überlappen oder größer als die Bildschirmfläche werden.

4.3 Die implementierten Funktionen

4.3.1 StartPreview

Die Funktion wird durch (StartPreview) aufgerufen. Nach dem Aufruf der Funktion wird ein Fenster geöffnet. Das Fenster besteht aus vier sog. "Panes":

- Title-Pane
- Menu-Pane
- Graphic-Pane
- Description-Pane.

Zur Pane-Konfiguration siehe *defflavor PreviewFrame* im Anhang. Das Title-Pane stellt die Überschrift des Fensters dar und enthält den Schriftzug "Preview". Das Graphic-Pane ist ein statisches Fenster und bildet das universelle Grid und die Objekte ab. Das Description-Pane ist ein dynamisches Fenster und zeigt Informationen über die einzelnen

Objekte an. Das Menu-Pane dient der Eingabe der Liste der Relationen, der Liste der Objekte, sowie des Dokumenttyps. Es enthält vier maussensitive Komponenten:

- User-Menu
- Refresh
- Description
- Exit.

Refresh löscht das universelle Grid und die Objekte aus dem Graphic Pane, wobei das Fenster aktiv bleibt. Die Liste der Relationen, die Liste der Objekte und der Dokumenttyp werden wieder auf nil gesetzt. Description aktiviert das Description Pane. Exit deaktiviert das Fenster und setzt die Liste der Relationen, die Liste der Objekte und den Dokumenttyp ebenfalls auf nil.

Das User-Menu dient der Auswahl einer der folgenden Dokumenttypen:

- Bedienungsanleitung im Hochformat
- Bedienungsanleitung im Querformat
- Zeitung
- Memo
- OverHeadProjektor Folien für normalen Betrachtungsabstand
- OverHeadProjektor Folien für großen Betrachtungsabstand
- Bildschirm

Das Menu kann durch Exit verlassen werden.

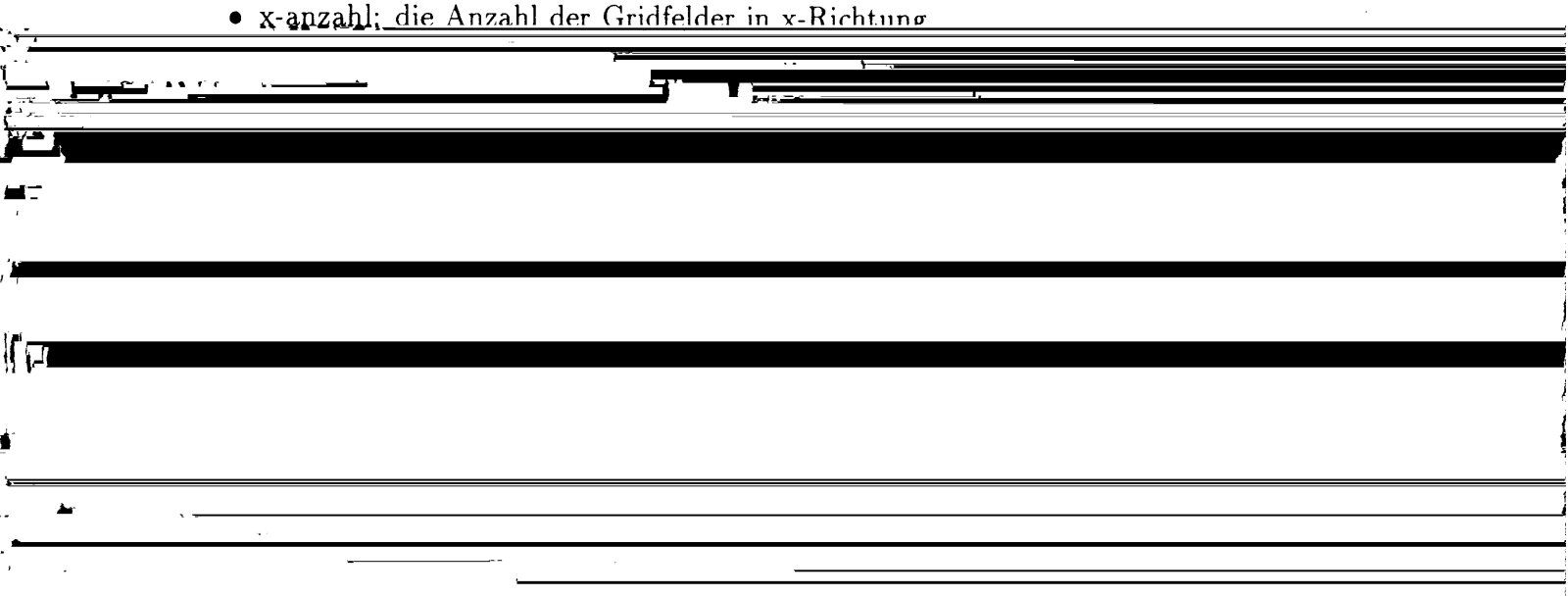
Wenn ein Dokumenttyp ausgewählt wird, öffnet sich ein Fenster, das die Auswahl aus folgenden Relationen erlaubt:

- Sequenz
- Kontrast
- Text-Graphik-Block

Nachdem der Relationstyp ausgewählt ist, werden zwei Menüfenster (nicht maussensitiv) geöffnet, die dem Benutzer Eingaben von Information über Graphiken und Texte erlauben. Die Eingaben erfolgen über die Tastatur.

Die Menüs werden verlassen, sobald der Benutzer "Exitäuswählt. Nach dem Verlassen des User-Menus werden die Eingabedaten, die Liste der Relationen, die Liste der Objekte, sowie der Dokumenttyp an die Gridgenerierungskomponente übergeben. Es werden die Parameter des Grids berechnet, sowie die Liste der Relationen ergänzt und in einen konsistenten Zustand gebracht. Die Funktion wird aufgerufen durch (berechne-grid). Die Gridgenerierungskomponente bearbeitet die Eingabedaten und gibt eine Liste aus, die folgende Parameter enthält: (breite hoehe abstand x-anzahl y-anzahl x y)

- breite: die Breite und
- höhe: die Höhe eines Gridfeldes
- abstand: der Abstand zwischen zwei Gridfeldern
- x-anzahl: die Anzahl der Gridfelder in x-Richtung



- x: die Koordinaten der
- y: linken oberen Ecke des Grids

Diese Liste wird dann an die Make-Grid Funktion übergeben.

4.3.2 Make-Grid

Die Aufgabe dieser Funktion ist die Darstellung des Grids auf dem "Graphic Pane".

Die Funktion wird wie folgt aufgerufen: (make-grid (breite hoehe abstand x-anzahl y-anzahl x y))

Nach der Parameterübergabe werden die Breite und die Höhe aller Felder addiert. Der Wert der Gesamtbreite und der Gesamthöhe des Grides wird benötigt, um festzustellen, ob die Breite bzw. die Höhe des Fensters überschritten wird. Wenn dies der Fall ist, erfolgt eine Fehlermeldung.

das in einer oder in beiden Richtungen größer ist als das Fenster, darzustellen. Es erfolgt in diesem Falle eine Fehlermeldung.

Der Aufbau des Grids geschieht durch mehrere Aufrufe der X-Window Funktion :draw-line x1 y1 x2 y2

Es ist zu beachten, daß das Koordinatensystem des Fensters seinen Ursprung in der linken oberen Ecke hat.

4.3.3 Insert-Frame

Die Aufgabe dieser Funktion ist die Darstellung der Objekte auf dem Grid. Die Funktion wird wie folgt aufgerufen: (insert-objekt name x y breite höhe)

- name: der Name des Objektes.
- x, y: Koordinaten, an denen das Objekt plaziert werden soll.
- breite: die Breite des Objektes.
- höhe: die Höhe des Objektes.

Nach der Parameterübergabe wird überpüft, ob das Objekt den Dokumentrahmen überschreitet oder nicht. Wenn das Objekt den Dokumentrahmen überschreitet, wird es zurückgewiesen.

Wenn das Objekt akzeptiert wird, dann wird es in die Liste der Objekt- Datenstrukturen eingefügt.

Die Objekt-Datenstruktur hat die Form (defstruct gframe name x y breite höhe)

diese entspricht den Insert-Frame-Parametern.

Die Liste der Objekt-Datenstruktur wird durch einen Array realisiert (defvar matrix-frame (make-array 10 :fill-pointer 0))

Das Objekt wird durch folgenden Befehl :draw-rectangle width height x y dargestellt.

4.3.4 Delete-Frame

Diese Funktion hat die Aufgabe ein Objekt aus dem Grid zu löschen und seine Datenstruktur aus der Liste der Objekt-Datenstrukturen zu entfernen.

Wenn ein Objekt gelöscht wird, wird sein Objekt-Name aus der Liste der Objekt- Datenstrukturen herausgesucht. Die Such-Funktion ist eine lineare Suche; sie vergleicht den zu löschenden Objekt-Namen mit den Objekt-Namen in der Liste der Objekt- Datenstruktur.

Ist das gesuchte Objekt gefunden, makiert die Such-Funktion die Stelle, an der sich das gesuchte Objekt befindet.

Das Feld, in dem sich das gesuchte Objekt befindet, wird durch das letzte Element des Arrays überschrieben. Dieses wird dann aus dem Array entfernt. Wenn das gesuchte Objekt das letzte Element der Liste ist, wird es direkt aus der Liste entfernt.

Die Delete-Frame Funktion wird aufgerufen durch: (delete-frame name)

4.3.5 Move-Frame

Diese Funktion hat die Aufgabe, ein Objekt von einem Gridfeld zu anderem zu bewegen.

Wenn ein Objekt von einem Feld zu anderem Feld bewegt wird, wird zuerst der Objekt-Name in der Liste der Objekt-Datenstrukturen durch eine Such-Funktion gesucht. Das Suchverfahren ist analog zum Suchverfahren der Delete-Frame-Funktion.

Ist das gesuchte Objekt gefunden, wird der Wert der Position des gefundenen Objektes durch den Wert der neuen Position ersetzt. Das Objekt wird dann aus dem Grid gelöscht, und die Funktion stellt das Objekt an der neuen Position dar.

Die Move-Frame Funktion wird wie folgt aufgerufen. (move-frame name x-new y-new)

Die folgenden Abbildungen zeigen Hardcopies der Oberflche des LayLab-Systems bei einem Beispiellauf zur Erzeugung eines Layouts für eine Bedienungsanleitung zu einem technischen Gerät.

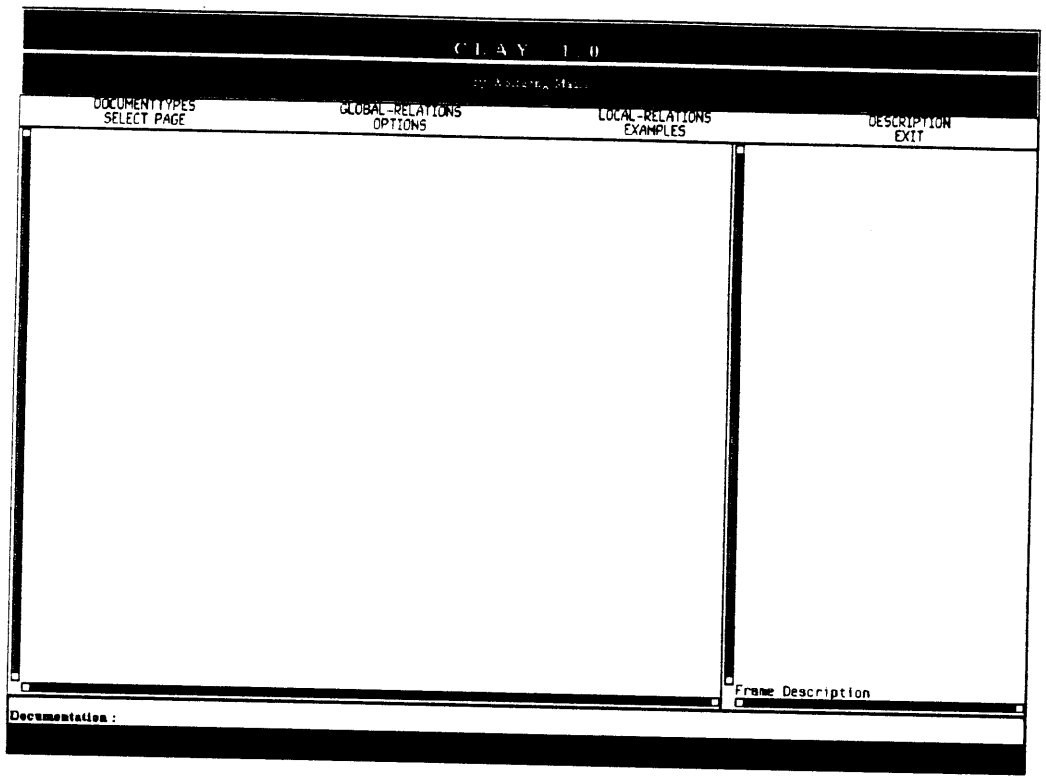


Abbildung 10: Die Oberfläche des LayLab-Prototypen

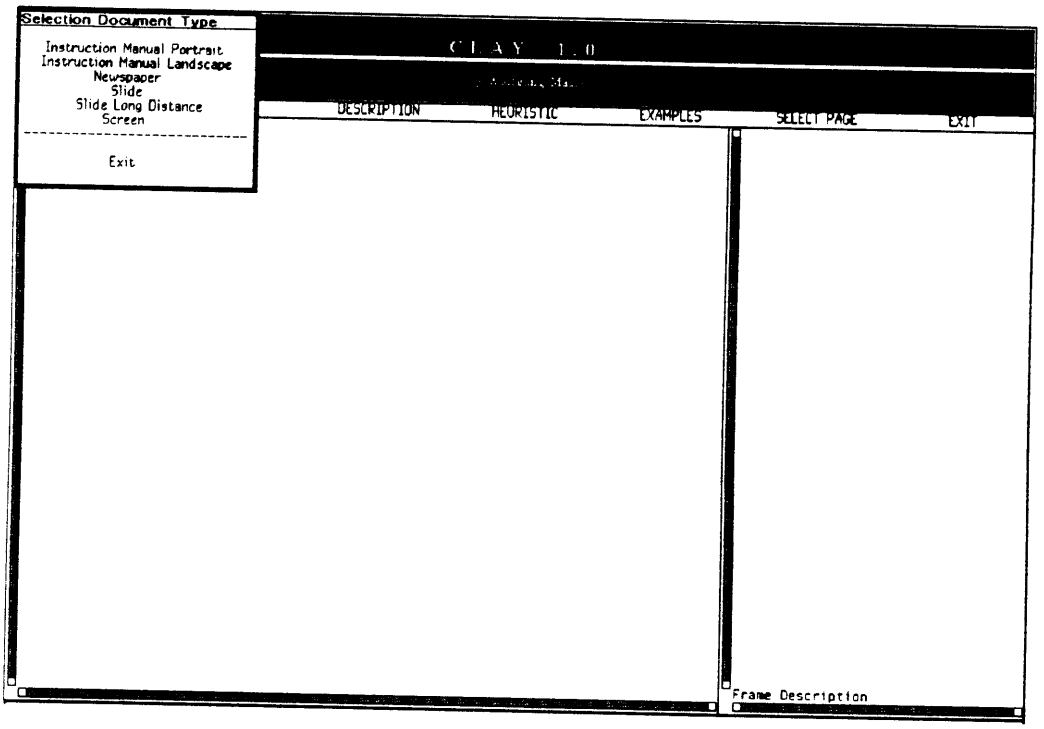


Abbildung 11: Eingabe des Dokumenttyps über Pop-up-Menüs

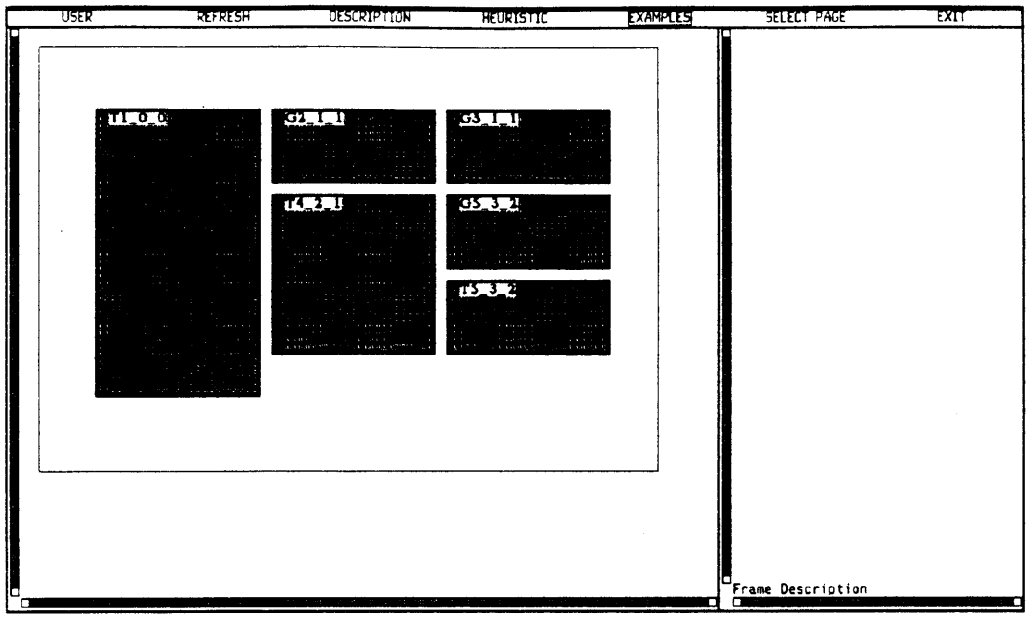


Abbildung 12: Ein von LayLab generiertes Beispiellayout für eine Bedienungsanleitung

Literatur

[André & Rist 90a] André, E., Rist, T.: Towards a Plan-Based Synthesis of Illustrated Documents. In: Proc. of 9th ECAI (1990) 25-30

[André & Rist 90b] André, E., Rist, T.: Synthesizing Illustrated Documents: A Plan-Based Approach. In: Proc. of InfoJapan '90, Vol. 2 (1990) 163-170

[Arnheim 69] Arnheim R., Hrsg.: Visual Thinking. London: Faber and Faber (1969)

[Beach & Stone 83] Beach R.J., Stone M.: Graphical Style: Towards High Quality Illustrations. ACM Computer Graphics, Vol. 17, No. 3 (1983)

[Beach 85] Beach, R.J.: Setting Tables and Illustrations with Style. Xerox PARC Technical Report CSL-85-3 (1985)

[Borning 81] Borning, A.: The Programming Language Aspect of ThingLab: a Constraint-oriented Simulation Laboratory. ACM Transaction on Progamming Languages and Systems, 3(4): 353-387 (1981)

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

[Feiner 88] Feiner, S.: A Grid-Based Approach to Automating Display Layout. In: Proc. of Graphics Interface 88, Palo Alto, CA: Morgan Kaufmann (1988) 192-197

[Freeman-Benson et al. 90] Freeman-Benson, B., Maloney, J., and Borning, A.: An Incremental Constraint Solver. Communications of the ACM, Vol. 33, No. 1 (1990)

[Friedell 84] Friedell, M.: Automatic Synthesis of Graphical Object Descriptions. Computer Graphics, 18(3): 53-62 (1984)

[Graf 90] Graf, W.: Spezielle Aspekte des automatischen Layout-Designs bei der koordinierten Generierung von multimodalen Dokumenten. GI-Workshop "Multimediale

[Jones 88] Jones, O.: Introduction to the X-Window System. Englewood Cliffs, NJ: Presentice-Hall (1988)

[Maaß 92] Maaß, W.: Constraint-basierte Plazierung in multimodalen Dokumenten am Beispiel des Layout Managers in WIP. Master's thesis, Dept of Computer Science, University of Saarbrücken (1992)

[McCormack & Asente 88] McCormack, J., Asente, P.: An overwiew to the X Toolkit. In Proc. ACM SIGGRAPH Symp. on User Interface Software, Banft Alberta, Ca (1988) 46-55

[Miller 87] Miller, G.A.: The magical number seven, plus or minus two. Scientological Review, 56 (1987)

[Müller-Brockmann 81] Müller-Brockmann, J.: Grid Systems in Graphic Design. Stuttgart: Hatje (1981)

[Pickering 89] Pickering, R.: DTP auf einen Blick, Grundlagen von Layout und Design. Neuss: Hilchner-Verlag Daten und Medien (1989)

[Scheifler & Gettys 86] Scheifler, R.W., Gettys, J.: The X Window System. ACM Trans. on Graphics 5 (2), 79-109 (1986)

[Stankowski & Duschek 89] Stankowski, A., Duschek, K.: Visuelle Kommunikation. Berlin: Dietrich Reimer (1989)

[Symbolics 90] Symbolics User Manuel for Genera 8.0 No. 10, Programming the User Interface (1990)

[Van Hentenryck 89] Van Henntenryck, P.: Constraint Satisfaction in Logic Programming. MIT Press, Cambridge, MA (1989)

[Wahlster et al 89] Wahlster, W., André, E., Hecking, M., Rist, T.: WIP: Knowledge-based Presentation of Information. German Research Center for Artificial Intelligence, DFKI Research Report (1989)

[Wahlster et al. 91] Wahlster, W., André, E., Graf, W., Rist, T.: Designing Illustrated Texts: How Language Production Is Influenced by Graphics Generation. In: Proc. 5th Conf. of the European Chapter of the ACL (1991) 8-14

[Wahlster et al. 92] Wahlster, W., André, E., Bandyopadhyay, S., Graf, W., Rist, T.: WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation. In: Ortony, A., Slack, J., Stock, O. (Hrsg.), Computational Theories of Communication and their Applications, Berlin: Springer-Verlag (1992)

[Willows & Houghton 87] Willows D., Houghton H., Hrsg.: The Psychology of Illustration, Vol. 1, 2. Berlin: Springer-Verlag (1987)

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

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

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

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-91-08
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

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-91-07
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"

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

LAYLAB: Ein System zur automatischen Plazierung von Text-Bild-Kombinationen in multimodalen Dokumenten

Wolfgang Maaß, Thomas Schiffmann, Dudung Soetopo, Winfried Graf

D-92-03
Document