Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-94-15

Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces

Winfried H. Graf, Stefan Neurohr

May 1994

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, 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, SEMA Group, and Siemens. 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
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

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.

Dr. Dr. D. Ruland
Director

Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces

Winfried H. Graf, Stefan Neurohr

DFKI-RR-94-15

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-9400).

ISSN 0946-008X

Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces

Winfried H. Graf and Stefan Neurohr

German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
Email: {graf,neurohr}@dfki.uni-sb.de

Abstract

In the expanding field of visual applications, layout design and graphical editing tasks are crucial points. In this paper, we address the incorporation of AI aspects in the visual software design process and the automatic layout and beautification of informational graphics such as visual programs and chart diagrams. Since layout in dynamic settings frequently requires a direct manipulation responsiveness, an incremental redesign of the generated visual material is necessary. Following our previous work on constraint-based multimodal design, we show how powerful constraint processing techniques, such as constraint hierarchies and dynamic constraint satisfaction, can be applied to visual programming environments in order to maintain graphical style and consistency for a meaningful layout. We describe *InLay*, a system for constraint-based presenting and editing visual programs. Finally, we will have a short look at some extensions with regard to advanced interaction and visualization techniques.

Keywords: multimedia layout, constraint-based visualization, visual programming, incremental beautification, dynamic constraint satisfaction

Contents

1 Introduction

There is an expanding field of applications which have to communicate information visually, such as the design of graphical software, informational charts, and multimedia interfaces [Catarci et al. 92, Maybury 93, André et al. 93a]. While the infrastructure for accessing, processing and displaying vast amounts of information is developing rapidly, we still need adequate technology that enables an efficient and profitable availability of this information. But existing presentation and design tools are growing more and more cumbersome to use, as the complexity of the application grows, such as in visual programming. That means, users need sophisticated mechanisms which communicate specific information when they need it and in the right format to perform certain tasks. In this respect, advanced visual interfaces should be able to generate interactive multimedia presentations in order to account for flexibility, effectiveness, reactivity, and consistency [André et al. 93b]. Particularly, meaningful layout and visual editing facilities can be seen as integral parts of the next generation's design and presentation environments.

One essential shortcoming of current visual programming environments is their lack of providing adequate tools for maintaining graphical style, legibility, and consistency of dynamic presentations. Traditional graph layout algorithms do not have knowledge about how the information is really used, since they have no explicit representation of the display objects and their causal relationships. Frequently, program visualization is just a mapping from program code to visual items.

As visual presentations are intended to convey specific information to their users, visual interfaces are limited in their power and usefulness when they are fixed to syntactic features only. Up to now program visualization was restricted to a *structural visualization* based on the program's data structures or underlying computational models, e.g., using Nassi-Shneiderman diagrams. Conceptual information for visualizing domain concepts mostly cannot automatically be inferred from the corresponding program data. Instead, we propose a *conceptual visualization* by reflecting interface semantics about the logical and rhetorical structure of the presentation content that has been stated by semantic and pragmatic relations. By formalizing the intent of a presentation as a presentation goal and allowing the specification of design parameters, we can tailor a presentation with one and the same communicative intent to different people and situations (cf. [Seligmann & Feiner 91, Wahlster et al. 93]).

The primary contributions of this article are twofold: we address both, the visual design process (visual programming) and the layout of generated multimedia information (program visualization). Obviously, the presentation goals in both domains overlap. By the example of *InLay* (**In**teractive **Lay**out Laboratory), we describe enhancements to constraint-based visualization to extend the capabilities of graphical interfaces. We have developed constraint techniques that support the generation of dynamic visual presentations that automatically maintain a set of visibility constraints as the user edits the design or modifies the viewing parameters.

Following our previous work on constraint-based graphical layout [Graf 92, Graf 93], we show how advanced constraint processing techniques, such as prioritizing constraints, incremental and dynamic constraint satisfaction, can be widely applied to interactive visual environments in order to maintain graphical style and visibility constraints for a meaningful presentation.

In this paper, we do not present a new layout algorithm but a declarative representation formalism for graphical design knowledge, including knowledge about layout, typography, dynamics, and interaction, to enhance the presentation and interpretation of visual information. Conceptually, our goal is to go beyond traditional presentation techniques with providing a general framework for automated layout and graphical editing tasks in visual programming environments, such as exploring, monitoring, and presenting information. Ultimately, we suggest some novel presentation techniques to produce the most effective view of the information using animated layout, abstracting, focussing, etc. to quickly process large amounts of visual information in order to exploit humans' visual capabilities.

2 An Example

To illustrate the functionality of the system, let's have a short look at a small example taken from a visual programming environment, as it may be used in CASE tools.

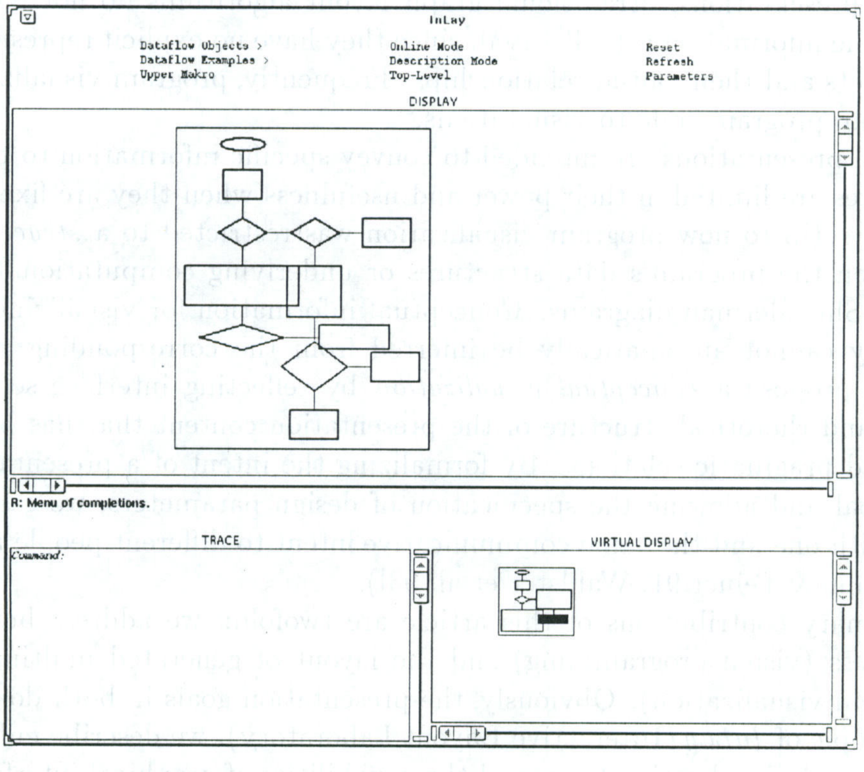

Figure 1: Visual Programming Environment

The following snapshots of a system run will demonstrate how *InLay* supports the design process of visual programs by incrementally beautifying the display after visual editing operations, such as adding, dragging, and re-sizing display elements. Fig. 1 gives an impression of a typical, inconsistent and incoherent display as it frequently occurs in programming interfaces without editing and layout support. The so-called *virtual display* in the lower right corner of the display shows a facility to zoom into a deeper level of the hierarchical information structure. Because of the lack of space, we will not treat the handling of design parameters in this example.

Fig. 2 shows another display area that has already been beautified by *InLay* exploiting stylistic features and applying visibility constraints to an incrementally designed visual program.

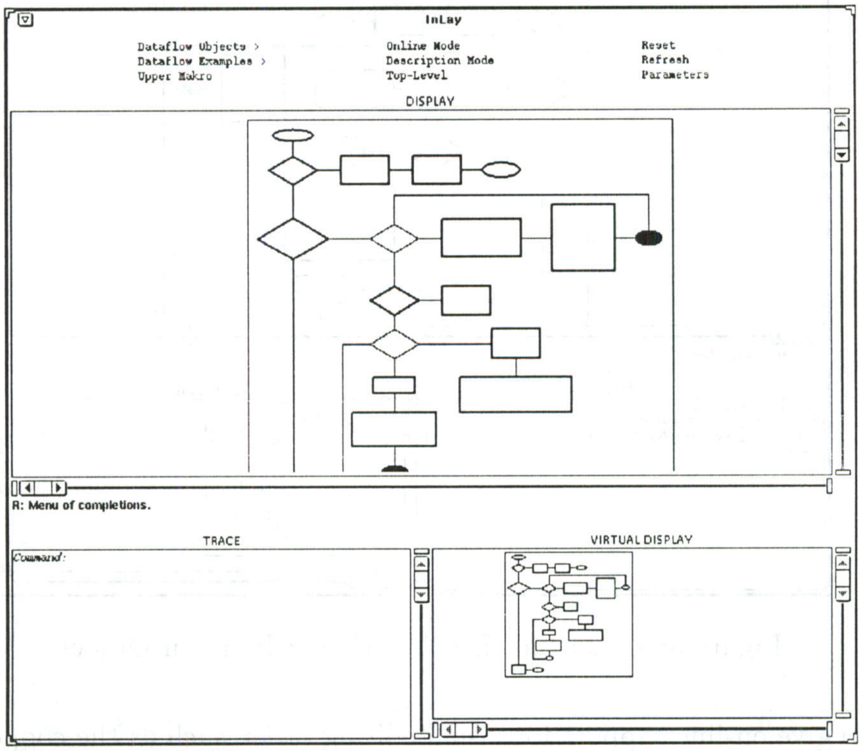

Figure 2: Display after User Interaction

One essential problem in visual programming is concerned with the handling of global conflicts between not directly related visual items, after inserting a new program construct into the dataflow graph (cf. section 4.4). *InLay* can detect such global conflicts and backtrack to select an alternative, consistent design that is shown in Fig. 3.

3 Design Issues and Overview

As with many other interesting AI design problems, geometric search and optimization problems are immanent to many applications of intelligent graphical and multimedia interfaces. An important class of such problems in visual presentation generation involves computing positions, dimensions, and topological orientations of multimedia entities that satisfy a set of visibility constraints. Visual program environments using typical depictions such as dataflow diagrams and charts, graphs, and trees, are essentially characterized by its structural, temporal, conceptual, and presentation properties. Here, most design systems suffer from problems such as interactivity and responsiveness, local/global consistency and coherence, expressiveness and effectiveness, processing semantic and context information as well as handling weak, dynamic, and temporal relations.

One of our main goals concerns the exploitation of common-sense knowledge about program visualization for presentation beautification and the adaptation of layouts on the

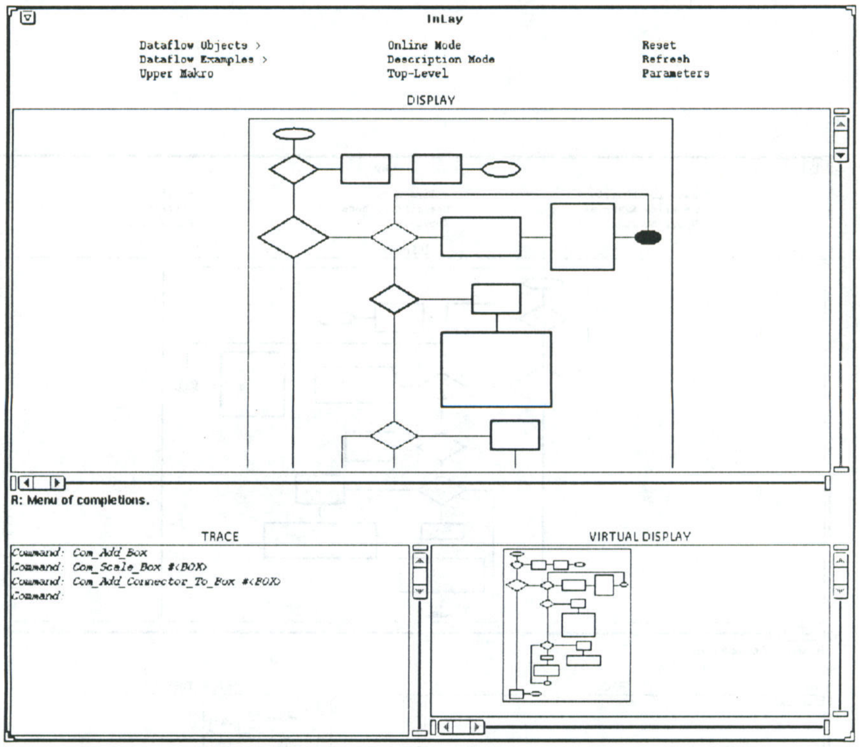

Figure 3: Consistent Display after Adding an Object

fly in order to give on-line support for visual editing tasks, such as the cooperative design of software diagrams. Our central guiding design rationales to overcome these problems are communicative intent, interactivity, adaptivity, and incrementality.

Intent and Layout A visual presentation is designed to fullfill a specific communicative intent such as the visualization of an object's state, location, properties (e.g., underlying program code), dynamics, or its relations to other objects. Visual clarity of a presentation can be achieved by using a visual organization of information that emphasizes the underlying logical and rhetorical structure of the information to be presented as well as context information. Therefore, the intent of a presentation that can be specified as a communicative goal should partial be communicated through its layout.

While most of the previous work on mutlimedia presentation design only addresses quantitative or syntactical aspects, we also focus on the intent of the presentation in order to generate syntactically as well as semantically correct presentations. In *InLay*, we deal with page layout as a rhetorical force, influencing the intentional and attentional state of the user. Therefore, the user is requested to state only interface semantics, not syntax. We view the layout itself as an important carrier of meaning.

It is a central claim of this paper, that only a deeper treatment of visual constraints in information presentation addresses the ergonomic aspects of layout and so can reduce the semantic gap between user and computer. In this respect, semantic-pragmatic information between visual elements communicating presentation and exchange acts as well as program constructs, e.g., *if-then-else*, *case*, *while* operations, can represented by means of so-called *rhetorical relations*, such as 'sequence', 'graphics-text', 'contrast', 'elaboration', 'comply-with-request', that are reflected by graphical constraints specifying topological arrangement facilities for visualization. The user can graphically state information about semantical relationships between visual elements which should be reflected by the layout. These relations can also be specified by a content planner [André & Rist 93] or directly derived from the application code.

Adaptivity and Interactivity In a number of situations, it is an important feature to be able to produce customized layouts of visual items with minimal effort. Especially, in dynamic display environments, displays must be flexible enough to accommodate varying numbers and sizes of visual objects. The quest for user interaction is based on the fact that is impossible to anticipate all needs and requirements of each potential user in an infinite number of presentation situations. Therefore, systems using dynamic graphical presentations, such as in visual programming, have to adjust their design in response to user interaction in order to achieve an expressive and effective output with high coherence.

Since frequently software visualization systems are limited in the breath and extensibility of their displays as they do not allow user-defined displays we have developed a set of techniques for the generation of highly adaptive interfaces that support user-controlled design of graphical presentations and customize a presentation to the communicative situation and a specific user. We approximate the fact that communication is always situated by making all decision processes sensitive to design parameters such as user's layout preferences, presentation type, presentation intent, output medium, and resource limitations. Design parameters can affect how information is displayed, e.g., the text in a node can be scaled with the size of the node or remain fixed.

Visual Editing Graphical editing of large and complicated visual presentations are laborious to produce by hand. Here, it is unacceptable to spend more time on the visualization of the application information to be presented than the application algorithm itself. For example, in a program visualization context, the user doesn't want to waste time with layout specifications. On the other hand, fully automated layout design frequently leads to suboptimal results. So, there comes an increasing need to involve the user in the graphical design process. It has been proven as an effective procedure, in the first phase of the design process, to allow the specification of rough layouts by the user, which can be incrementally beautified by the system following certain design constraints in later stages.

In order to make the editing and layout process of graphical presentations more efficient, we allow the user to create an interface through *pre-editing* of layout sketches, which can automatically be beautified by the system. Moreover, we facilitate the user to tailor the interface to his needs by *post-editing* automatically laid out presentations, using editable graphical histories or changing default layout schemata interactively. The user will be allowed to edit display objects using operations such as panning, dragging, re-sizing, zooming into, and modifying viewing parameters. Then the system supports smooth, incremental changes between successive displays.

Incrementality As a significant change in the application requires a complete redesign of the interface, we encourage an incremental update of the interface to improve its quality and efficiency. Therefore, in dynamic graphical environments there is a need for incrementality, that means the immediate realization of parts of a stepwise provided input, e.g., the beautification of lines in a flowchart and positions of display objects. As we will not rely on pre-defined links between pre-stored multimedia information items, presentation design will be done at runtime in order to decrease the response time and react more promptly to the application. As the user browses or edits a dynamic graphical presentation, the system updates visibility changes smoothly to avoid visual discontinuities. All views of the dataflow chart are redesigned simultaneously in realtime to maintain a set of visibility constraints automatically as the viewing specification changes, ensuring that specific objects remain visible.

4 Constraint-Based Visualization

As has been shown in previous work by [Borning & Duisberg 86] among others, many complex visual interface operations and transformations can efficiently be facilitated using constraint processing techniques. For example, the beautification of large graphical networks can be automatically arranged and lines in flow charts made consistent within a certain tolerance interval. The declarative semantics of constraint languages allows one to specify graphical objects while avoiding extraneous concerns about the realization of the visualization algorithms. Moreover, automatically applying constraints to multimedia displays allows a better control of the design space and facilitates a redesign of a generated presentation on the fly.

To address these features, we propose a clean separation between the application and the visualization domain, which implies the description and editing of the algorithm, the presentation, and relationships between the two domains, respectively. This decoupling of the visualization from the application code results in easy modification and elegant specification capabilities and facilitates special visualization effects (see section 5). In the following sections, we will give more details on the fundamentals of *InLay's* underlying constraint system.

4.1 Representation of Visual Objects

In *InLay*, all visual objects are described by its bounding rectangles. The size of an object depends on the application data and their format, e.g., a text object containing program code is formatted automatically by a constraint-based typography component [Graf 92]. Each visual object is generated as a subclass of class *BOX*, which manages the structure of the constraint network, and inherits variables and methods from it. Furthermore, classes can inherit from multiple superclasses, e.g., *Display Node* delivers information about the underlying constraint graph, which enables *InLay* to handle *Procedures*.

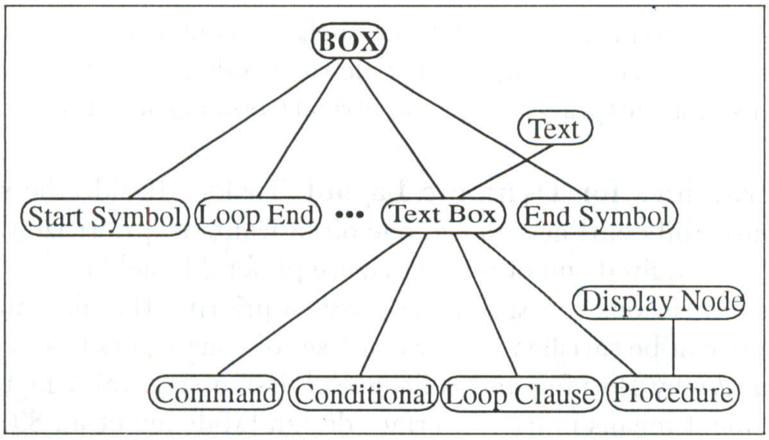

Figure 4: Class Structure for Visual Programs

Visual objects are defined by its state and behavior that is specified as a set of constraints. New instances of visual objects can be created by instantiating the specific prototype class. For example, a box is represented by variables for: x-value, y-value, width, height, and distances to connected objects. We use a dataflow implementation of constraints as in [Maloney et al. 89]. This concept allows one to easily enrich the system with other visual language constructs and adapt it to further domains (e.g., multimedia desktops). A part of the class structure for visual programming elements is shown in Fig. 4.

4.2 The Constraint Language

In the context of visual programming environments, we use constraints extensively to maintain visibility and local/global consistency between application data and its graphical depictions as well as among multiple views of data, e.g., after dragging, zooming, or iconifying display objects, state animation events, and specify application- and user-specific requirements. Furthermore, graphical constraints map the communicative intent to graphical style describing how visual objects should be arranged. Here, a thorough elaboration of a constraint-oriented approach enables both the designer and the user of a system to naturally express the meaning of the intended information such that the computer can maintain even the semantics of the implementation.

Relations between multimedia objects can be classified as *semantic-pragmatic*, *geometrical/topological*, and *temporal*. Semantic-pragmatic constraints can be compiled into graphical constraints that represent aesthetical knowledge about perceptual criteria concerning the organization of visual elements, such as the sequential ordering (horizontal vs. vertical layout), alignment, grouping, symmetry, or similarity. Geometrical and topological constraints refer to absolute and relative constraints. Temporal relations are used in the case of animated presentations to represent temporal, spatial information (e.g., stating a constraint while the mouse button is held down).

Primitive constraints represent elementary local relations, e.g., 'beside', 'connect', or 'under', expressing basic geometric relations. These constraints are specified by sets of mathematical equations (e.g., two objects that are constrained to touch at specific points) or by sets of inequalities (e.g., one object is constrained to lie inside another). The primitive constraints can be aggregated to more complex *compound constraints*, specifying the visualization of semantic-pragmatic relations such as 'contrast' or 'sequence'. Furthermore, the underlying constraint language is able to encode aesthetical knowledge in order to express certain semantic/pragmatic, geometrical/topological, and temporal relations.

Constraint Hierarchies for Dynamic Layout Tasks Beside the semantic classification of local constraints outlined above, one often wants to prioritize the constraints in those which must be required and others which are preferably held and could be relaxed in the worst case. If the various constraints are given a priority, the most important and restrictive constraints can be satisfied first. Such a set of constraints has been introduced as a *constraint hierarchy* by Borning and colleagues.[1] Using constraint hierarchies has been proved as a convenient means in user interface design [Maloney et al. 89, Myers et al. 90] and the *LayLab* layout system [Graf 93] for declaring relative desires. We distinguish between *obligatory*, *preferred*, and *default constraints*. The latter state default values, which remain fixed unless the corresponding constraint is removed by a stronger one.

Dynamic Constraints and Pointer Variables In graphical synthesis tasks like display layout, constraints frequently have only local effects, i.e., the set of variables that is relevant to a solution, dynamically changes as presentations are incrementally generated by a presentation system or by user actions during problem solving. So, we have to distinguish between *static constraints* that are related to a fixed set of variables and *dynamic constraints* that are generated on the fly. In this case, the problem with incremental constraint solvers is that they can reason upon a changing set of variables and constraints, but cannot themselves handle the activity of a variable. A typical form of dynamic constraints in graphical environments concerns those in which the number of layout elements belonging to one relation is not known a priori, such as a sequence of connected display objects or subtrees of a dataflow graph.

A typical form of dynamic constraints in interactive graphical environments concerns those in which the number of layout elements belonging to one relation is not known a priori, such as the objects in a subtree of the dataflow graph. Therefore, the constraint language must be able to handle *indirect references* [Vander Zanden et al. 91]. Indirection allows the specification of layouts, independently of the number of layout objects using pointer variables that facilitate the programmer to specify constraints like procedures in imperative languages.

In *InLay* dynamic distance constraints do not work on single variables, but on a dynamic list structure of variables which represent parts of the required subtree. Here, variables can reference to dynamic sets of required global topological information. Consequently, dynamic constraints are able to encode global relations instead of the local relations of default constraints. For example, in dataflow graphs the constraint *dynamic-distance-vertical* maintains relations between all objects to the right and the distance to the object below. The system manages these objects in a list, which is dynamically updated after each user or system actions by computing the trace to the root of the dataflow tree and adding or deleting the concerned variables in the lists of relevant nodes. This concept reduces the size and number of constraints, that must be dynamically created and deleted, which is a time consuming task.

[1] Previous papers [Freeman-Benson et al. 90, Borning et al. 92] have provided an analysis of the theoretical background of constraint hierarchies.

While static constraints are easily encoded in Lisp S-expressions, dynamic constraints are represented by Lambda expressions which enable constraints having a procedural style to model dynamic application behavior. If pointer variables point to nil, the constraints maintain default values.

4.3 Constraint Satisfaction

We have implemented an incremental constraint hierarchy solver which is capable of processing the introduced dynamic and prioritized constraints. This solver is based on *DeltaBlue* [Freeman-Benson et al. 90], an efficient local propagation algorithm that generates plans which can be reused repeatedly to solve the same constraint without calling the constraint solver again. As *DeltaBlue* is structure-based, a *lazy evaluation* technique can efficiently be applied to handle indirect references with respect to specific constraints such as connection constraints. This means, a constraint with indirect reference is evaluated when all its input variables of the dynamic list are evaluated. Therefore, the solver can handle large and complex constraint networks very efficiently.

4.4 From Local to Global Visual Consistency

In *InLay* we try to enforce local consistency conditions in order to simplify the subsequent computation of a globally coherent model of visual objects.

Consistency and Graphical Style As we described before, the system is able to translate interface semantics into constraint representations exploiting visibility constraints to maintain a consistent presentation and graphical constraints to meet certain aesthetical criteria. We regard a dataflow chart as consistent, if there are no overlappings and line crossings as well as all formatting requirements are fulfilled. First, we will consider local consistency issues only. For instance, the consistent sequentional arrangement of two visual program commands A and B is determined using the following constraint network:

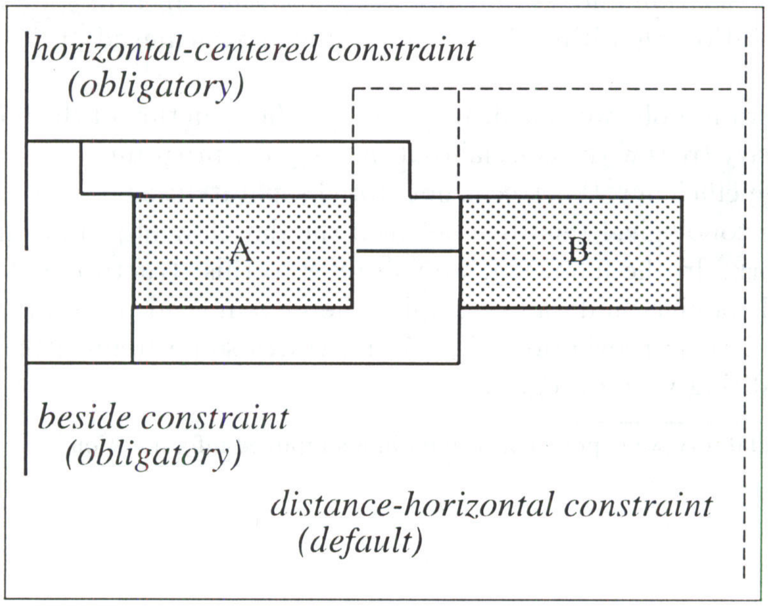

Figure 5: Horizontally Centered Placing

Fig. 5 illustrates four different kinds of constraints:

- The topological constraint *horizontal-centered*, which states a centered placing in the horizontal direction:

  `B.y = A.y - 0.5*A.h + 0.5*B.h` [2]

- The topological constraint *beside*, which guarantees the non-overlapping of two objects:

  `B.x = A.x + A.w + A.dist-hor`

- The metrical constraint *distance-horizontal*, which specifies the distance between the two objects:

  `A.dist-hor = min{o.5*A.w, 0.5*B.w}`

- In addition, the system adds an edge between the two boxes, which is constrained by a so-called *connection constraint*. Starting and ending point of the edge are fixed to the center of the opposite sides (not shown in the graphics).

In general, these constraints guarantee the local consistency of two visual objects as well as their semantical connection that is reflected by their relative orientation. Moreover, the graphical style can be maintained globally using default distance and connection constraints.

Reaching Global Consistency with Conflict Avoiding Constraints Since visibility constraints as described above only guarantee locally consistent presentations, visual programs frequently suffer from global conflicts between not directly related objects, e.g., overlapping conflicts and line crossings after dragging one object as shown in Fig. 6. For a global conflict handling global knowledge about the topology of the whole dataflow chart is necessary. In general, there are two possible strategies to deal with global conflicts: a posteriori *conflict solution* and a priori *conflict prevention*. In *InLay*, we have tried a rule-based conflict solution algorithm first, but we have not achieved an acceptable runtime efficiency.

In visual program tools, we benefit from the specific structure of the flowcharts that can be treated as binary trees with a special root, namely the program's starting symbol. With respect to storage efficiency, this tree is not stored separately, as parts of it can directly be derived from the topological relations between the dataflow objects from the underlying constraint network. In the following, we will refer to this structure as the *dataflow tree*. In order to avoid conflicts in dataflow graphs, the system needs information about typical conflict cases and their prevention. Fig. 7 illustrates some frequently occuring conflict patterns when editing visual programs.

---

[2] To improve readability we express constraints in a simplified infix notation

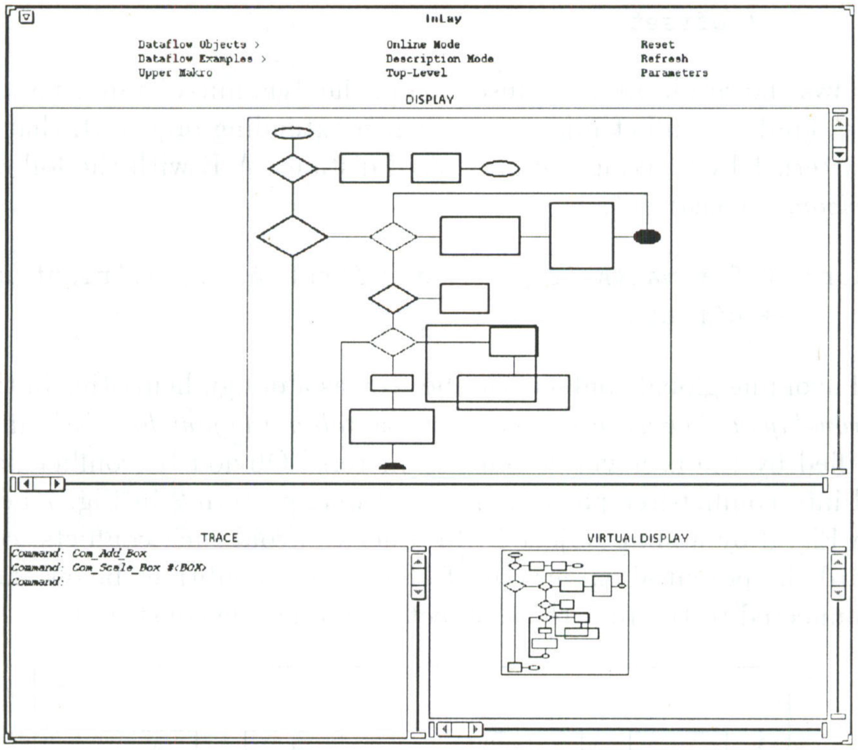

Figure 6: Global Inconsistencies in Visual Programs

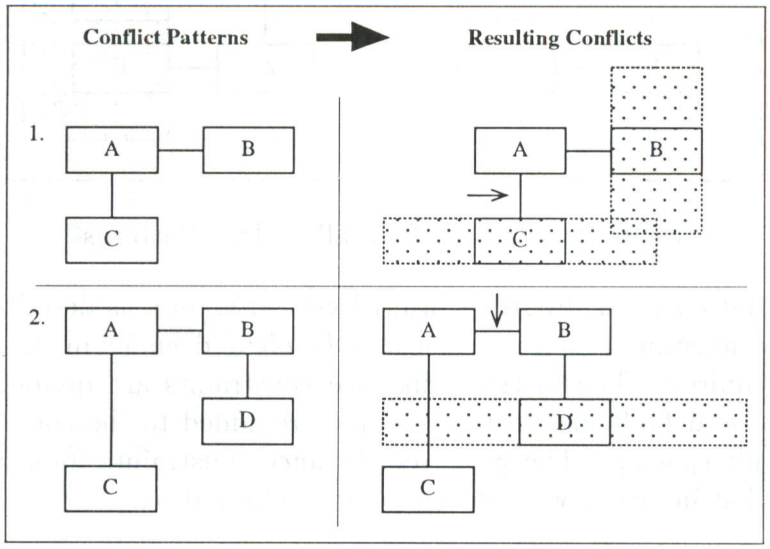

Figure 7: Conflict Patterns in a Dataflow Graph

A main conflict results from the arrangement of objects as in pattern 1. This kind of conflict can be solved by relaxing one of the distances A-C or A-B. In the case distance A-C is relaxed, the vertical distance must be equal to the maximum y extension of all objects which are leaves in the subtree to the right plus some offset. This can be expressed by the following *dynamic-distance-vertical* constraint:

```
A.dist-vert = max{ A.↑right.first.x + A.↑right.first.h,...,
                   A.↑right.last.x + A.↑right.last.h}
              + offset
```

Pattern 2 shows the result after the insertion of the dynamic-distance-vertical constraint. Here, another kind of conflict might occur when extending object D, that can be solved similar to pattern 1 by relaxing the horizontal distance A-B with the following *dynamic-distance-horizontal* constraint:

```
A.dist-hor = 0.5 * max{A.↑right-under.first.w,..., A.↑right-under.last.w}
             + offset
```

This form of avoiding global conflicts can be seen as a design heuristic. In this respect, we speak of a *row-layout heuristic* in case 1 and a *column-layout heuristic* in case 2, which can be specified by the user via design parameters. Obviously, conflict patterns can be transformed into conflict-free pattern. For example, pattern 2 in Fig. 7 becomes conflict pattern 3 in Fig. 8 by adding object E. In order to avoid such conflicts, constraints also have to regard the potential extensions of all objects in subtrees below the referred one, which are connected to the root (here object C) via beside constraints.

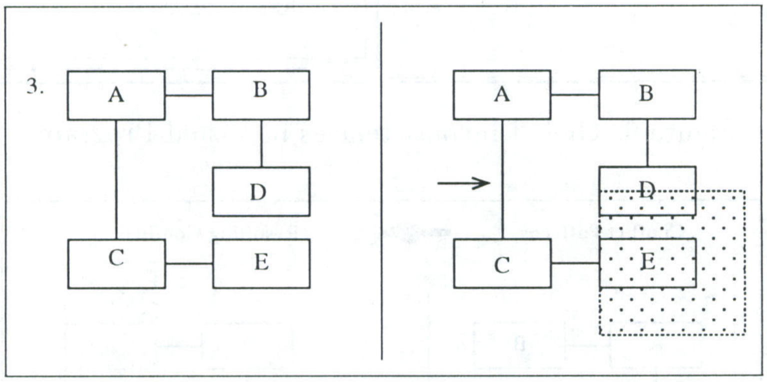

Figure 8: Change of Dataflow Tree Patterns

We take this into account by using prioritized constraints as described above. Here, topological and connecting constraints, such as *beside* and *under* in the examples above, are defined as required. The metrical distance constraints are divided in default and preferred ones. The default distance constraints are added to the constraint network to express the default spacing. The preferred distance constraints for conflict prevention have a strength that increases with the content of information.

5 Advanced Visualization Techniques

Visual program layout and program visualization make a number of useful contributions for all kinds of graphical editing tasks. Furthermore, innovative visualization and interaction techniques for dynamic displays can generate new insights, e.g., algorithm animation facilities [Brown 88] and 3D information retrieval [Mackinlay 92], and can be seen as a new quality of communication media. The performance of our constraint solver allowed some realtime extensions with regard to an intelligent use of the following techniques:

- Animated layout of 2D graphics
- Abstracting presentation parts
- Focussing of active display objects
- Editing graphical histories
- Generation of meta presentation parts
- Visualization of information structures

For example, the user is facilitated to zoom into deeper level of a hierarchical information structure (e.g., textual program code), to move currently processed objects into the focus, or to view context information on virtual displays.

6 Related Work

As graphics infrastructure becomes more and more sophisticated, rapidly expanding activities have entered the area between visual interfaces and AI systems (e.g., [Maybury 93, Sullivan & Tyler 91, Catarci et al. 92]), including work on graphical layout, graphics generation, visual programming, multimedia user interfaces, as well as constraint-based reasoning. Since the physical format and layout of a presentation often conveys the structure, intention, and significance of the underlying information and therefore plays an important role in presentation coherency, automatic layout facilities are included increasingly in presentation systems and multimedia interfaces. Recent approaches in automated multimedia/multimodal layout investigate the use of rule-based [Feiner 88], constraint-based [Graf 92] and case-based reasoning [MacNeil 90] methods for representing graphical design knowledge.

While much research has concentrated on the automated visualization of informational graphics [Mackinlay 85], the knowledge-based layout and beautification of graphically presented relational information and visual programs have nearly been explored. Marks [Marks 90] investigated the encoding of arc-node diagrams in his *ANDD* system that grouped nodes sharing common graphical values (e.g., shape, color, size) to re-inforce perception of graphical properties. It was guided by so-called"pragmatic directives", that specify which interconnection patterns should be made visible by the layout. One weakness of the system is the realization of generated perceptual organization with the expensive layout algorithm. Beside a rule-based realization, a genetic [Kosak et al. 91] and a constraint-driven algorithm [Dengler et al. 93] have been implemented. Analogously, Roth and Mattis [Roth & Mattis 91] sorted chart objects and tree nodes to support search.

More recently a new class of systems, which try to overcome general layout problems using genetic algorithms, e.g., *GALAPAGOS* [Masui 92], has become popular. These approaches do not solve constraints directly, but modify candidate solutions with random values and approximate an optimal solution by iteration. Since they allow the generation of suboptimal solutions, they are rather robust. Usually genetic algorithms suffer from the problem that they are too slow, not reliable and some kinds of constraints are hard to specify. Moreover, genetic algorithms can not support conceptual visualization, as they have no design knowledge about the presentation, and they do not allow modifications at runtime as well as specification of users' preferences.

In previous work several dedicated algorithms for drawing graphs have been proposed, a comprehensive survey gives the annotated bibliography by [Di Battista et al. 93]. Most traditional algorithms exploit heuristics to overcome the complexity, but most of them are not flexible enough and do not allow to add user preferences. Instead, for the diagram server *ALF* [Di Battista et al. 90] a new approach in building a tailorable and extensible automatic layout facility for the production of nice drawings has been suggested. This tool selects from a spectrum of special-purpose algorithms the best suitable one for a specific application and then allows the assembling of new algorithms from existing ones. From this broad field we should further remark a new way by [Böhringer & Paulisch 90] to achieve stability in the layout of cyclic graphs and the system *Converge* [Sistare 90] for 3D geometric modeling. Another approach in interactive graph layout proposes a novel methodology for viewing large graphs [Henry & Hudson 91]. Its basic concept is to allow the user to interactively navigate through large graphs and learn about them in concise sections of appropriate size.

Moreover, visual programming and graphical editing tasks have been addressed by programming- and constraint-by-example techniques to enhance the interactive specification and editing of graphical presentations [Myers 91, Kurlander 93]. An automatic beautifier for line drawings and illustrations that makes use of automatic constraint generation as part of a 2D graphics editor is described in [Pavlidis & Wyk 85]. In this approach, vertices are re-drawn precisely following certain constraints, such as nearly adjacent or coincident lines, that are inferred from an initial sketch which are imposed on the beautified version. But most of the drawing programs are limited to syntactical constraints.

Our work on advanced visualization and interaction techniques has mainly benefited from the influential work by Mackinlay et al. at Xerox PARC on the *Perspective Wall* [Mackinlay et al. 91] and the *Information Visualizer* [Mackinlay 92]. A similar approach for viewing and interacting with large layouts on limited displays in order to show local details and global context in one view is detailed in [Sarkar et al. 93]. Here, orthogonal and polygonal algorithms have been used for stretching a so-called rubbersheet, but they suffer from discontinuies at the boundaries and scaling/dimensioning problems respectively.

Up to now only rudimentary work has been done in the areas "animated layout" and "layout of presentations including animation". Research in this fields was mainly concerned with topics like *animation of programs* and *visual programming* (e.g., [Brown 88, Chang 90, Duisberg 90]). Here, numerous visualization systems for producing diagrams automatically from program code as well as generating static graphical displays of data structures have been delevoped and allow for editing the underlying program. But most of them propose less effective canonical displays which are less effective and difficult to create for linked and nested structures. Examples of a program visualization systems build *GELO* [Duby et al. 89] and its 3D extension *PLUM* [Reiss 93], general-purpose packages to visualize information about programs. Here, the layout of linked hierarchical objects, is described via constraints. *GELO* includes predefined data views and allows the graphical specification of topological constraints by the user. Essential shortcomings of *GELO* are that its displays are not aesthetically pleasing and the graphics can not be tailored to the user's needs. Another interesting approach carried out in the context of *Pictorial Janus* for visualizing object-oriented programming systems, addresses a declarative formalism for the definition of graphical layout [Haarslev & Möller 92]. Apparently, no work has been done on applying principles of graphic design to visualizing data structures.

7 Implementation

A first prototype of the system *InLay*, a tool for incremantal, constraint-based editing of visual presenations, has been implemented using Allegro Common Lisp/CLOS for object-oriented interface programming and the Common Lisp Interface Manager (CLIM), Release 2.0, an X window compatible Lisp-based window programming interface that provides a layered set of portable services for constructing user interfaces. The incremental constraint solver which is based on the *DeltaBlue* [Freeman-Benson et al. 90] algorithm is O(cN) in the number of affected constraints. The system is considered to be experimental work in progress. So, for this first protoype version, we have deliberately limited our application domains to visual programs and multimedia displays. We have further concentrated only on a small set of design heuristics with wide applicability.

8 Conclusion

We have described the constraint-based interactive display manager *InLay* that automatically handles all aspects of display layout and editing in visual presentation environments as one extension of the multimedia layout mangager *LayLab* [Graf 93]. The approach detailed in this paper has already been approved for the automatic layout control of the multimodal/multimedia presentation system *WIP* [André et al. 93a, Wahlster et al. 93] and its interactive extensions. A further application will be concerned with support of the semi-automated 3D graphics editor *AWI* [Rist et al. 94].

Beside visual programming environments there arise numerous potential application domains that suffer from visual design and consistency problems, such as the broad field of intelligent multimedia interfaces, CASE tools, with particular utilities for program animation, debugging, process monitoring, on-line help (instruction, tutoring), and documentation, as well as different kinds of network diagram designers, CSCW, and virtual realities.

References

[André & Rist 93] E. **André** and T. **Rist**. *The Design of Illustrated Documents as a Planning Task.* In: Maybury [Maybury 93].

[André et al. 93a] E. **André**, W. **Finkler**, W. **Graf**, T. **Rist**, A. **Schauder**, and W. **Wahlster**. *WIP: The Automatic Synthesis of Multimodal Presentations.* In: Maybury [Maybury 93], pp. 75–93.

[André et al. 93b] E. **André**, W. **Graf**, J. **Heinsohn**, B. **Nebel**, H.-J. **Profitlich**, T. **Rist**, and W. **Wahlster**. *PPP – Personalized Plan-Based Presenter.* DFKI Document D-93-05, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Saarbrücken, Germany, 1993.

[Böhringer & Paulisch 90] K.-F. **Böhringer** and F. Newberry **Paulisch**. *Using Constraints to Achieve Stability in Automatic Graph Layout Algorithms.* In: Proceedings of the CHI'89 (Human Factors in Computing Systems), pp. 43–51, Seattle, WA, 1990.

[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. 92] A. **Borning**, B. **Freeman-Benson**, and M. **Wilson**. *Constraint Hierarchies.* LISP and Symbolic Computation: An International Journal, 5(3):223–270, 1992.

[Brown 88] M. H. **Brown**. *Algorithm Animation.* ACM Distinguished Dissertations. Cambridge, MA: MIT Press, 1988.

[Catarci et al. 92] T. **Catarci**, M. F. **Costabile**, and S. **Levialdi** (eds.). *Advanced Visual Interfaces, Proceedings of the International Workshop AVI '92.* World Scientific Series in Computer Science - Vol. 36. Singapore: World Scientific Press, 1992.

[Chang 90] S.-K. **Chang**. *Visual Languages and Visual Programmimg.* New York, NY: Plenum Press, 1990.

[Dengler et al. 93] E. **Dengler**, M. **Friedell**, and J. **Marks**. *Constraint-Driven Diagram Layout.* In: Proceedings of the 1993 IEEE Symposium on Visual Languages, pp. 330–335, Bergen, Norway, 1993.

[Di Battista et al. 90] G. **Di Battista**, A. **Gianmarco**, G. **Santucci**, R. **Tamassia**, and I. G. **Tollis**. *The Architecture of Diagram Server.* In: Proceedings of the 1990 IEEE Workshop on Visual Languages, pp. 60–65, Skokie, IL, 1990.

[Di Battista et al. 93] G. **Di Battista**, P. **Eades**, R. **Tamassia**, and I. G. **Tollis**. Algorithms for Drawing Graphs: an Annotated Bibliography, 1993.

[Duby et al. 89] C. **Duby**, S. **Meyer**, and S. P. **Reiss**. *Using GELO to visualize Software Systems.* In: Proceedings of the UIST'89 (ACM SIGGRAPH Symp. on User Interface Software and Technology), pp. 149–157, Williamsburg, VA, 1989.

[Duisberg 90] R. **Duisberg**. *Visual Programming of Program Visualizations - A Gestural Interface for Animating Algorithms.* In: T. Ichikawa, E. Jungert, and R. Korfhage (eds.), Visual Languages and Applications, pp. 161–173. New York, NY: Plenum Press, 1990.

[Feiner 88] S. **Feiner**. *A Grid-Based Approach to Automating Display Layout.* In: Proceedings of the Graphics Interface '88, pp. 192–197. Los Altos, CA: Morgan Kaufmann, June 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.

[Graf 92] W. H. **Graf**. *Constraint-Based Graphical Layout of Multimodal Presentations.* In: Catarci et al. [Catarci et al. 92], pp. 365–385. Also DFKI Research Report RR-92-15.

[Graf 93] W. H. **Graf**. *LayLab: A Constraint-Based Layout Manager for Multimedia Presentations.* In: Salvendy and Smith [Salvendy & Smith 93], pp. 446–451. Also DFKI Research Report RR-93-41.

[Haarslev & Möller 92] V. **Haarslev** and R. **Möller**. *Visualization and Graphical Layout in Object-Oriented Systems.* Journal of Visual Languages and Computing, (3):1–23, 1992.

[Henry & Hudson 91] T. R. **Henry** and S. E. **Hudson**. *Interactive Graph Layout.* In: Proceedings of the UIST'91 (ACM SIGGRAPH Symp. on User Interface Software and Technology), pp. 55–64, Hilton Head, SC, 1991.

[Kosak et al. 91] C. **Kosak**, J. **Marks**, and S. **Shieber**. *A Parallel genetic algorithm for network-diagram layout.* In: Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 458–465. Los Altos, CA: Morgan Kaufmann, 1991.

[Kurlander 93] D. **Kurlander**. *Reducing Repetition in Graphical Editing.* In: Salvendy and Smith [Salvendy & Smith 93], pp. 409–414.

[Mackinlay et al. 91] J. D. **Mackinlay**, G. G. **Robertson**, and S. K. **Card**. *The Perspective Wall: Detail and Context Smoothly Integrated.* In: Proceedings of the CHI'91 (Human Factors in Computing Systems), pp. 173–179, New Orleans, LA. 1991.

[Mackinlay 85] J.D. **Mackinlay**. *Automatic Design of Graphical Presentations.* PhD thesis, Dept. of Computer Science, Stanford University, Stanford, CA, 1985.

[Mackinlay 92] J. **Mackinlay**. *The Information Visualizer: A 3D User Interface for Information Retrieval.* In: Catarci et al. [Catarci et al. 92].

[MacNeil 90] R. **MacNeil**. *Adaptive Perspectives: Case-based Reasoning with TYRO, the Graphics Designer's Apprentice.* In: Proceedings of the 1990 IEEE Workshop on Visual Languages, pp. 138–142, Skokie, IL, 1990.

[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 (ACM Conference on Object-Oriented Programming Systems, Languages, and Applications), pp. 381–388, October 1989.

[Marks 90] J. **Marks**. *A Syntax and Semantics for Network Diagrams.* In: Proceedings of the 1990 IEEE Workshop on Visual Languages, pp. 104–110, Skokie, IL, 1990.

[Masui 92] T. **Masui**. *Graphic Object Layout with Interactive Genetic Algorithms.* In: Proceedings of the 1992 IEEE Workshop on Visual Languages, Seattle, WA, 1992.

[Maybury 93] M. **Maybury** (ed.). *Intelligent Multimedia Interfaces.* Menlo Park, CA: AAAI Press, 1993.

[Myers et al. 90] B. **Myers**, D. **Guise**, R. B. **Dannenberg**, B. T. **Vander Zanden**, D. **Kosbie**, P. **Marchal**, and E. **Pervin**. *Comprehensive Support for Graphical, Highly-Interactive User Interface: The Garnet User Interface Development Environment.* IEEE Computer, 23(11):71–85, November 1990.

[Myers 91] B.A. **Myers**. *Using AI Techniques to Create User Interfaces by Example.* In: Sullivan and Tyler [Sullivan & Tyler 91], pp. 385–402.

[Pavlidis & Wyk 85] T. **Pavlidis** and C. Van **Wyk**. *An Automatic Beautifier for Drawings and Illustrations.* Computer Graphics, 19(3):225–234, 1985.

[Reiss 93] S. P. **Reiss**. *A Framework for Abstract 3D Visualization.* In: Proceedings of the 1993 IEEE Symposium on Visual Languages, pp. 108–115, Bergen, Norway, 1993.

[Rist et al. 94] T. **Rist**, T. **Krüger**, G. **Schneider**, and D. **Zimmermann**. *AWI: A Workbench for Semi-Automated Illustration Design.* To appear in AVI'94, 1994.

[Roth & Mattis 91] S. F. **Roth** and J. **Mattis**. *Automating the Presentation of Information.* In: Proceedings of the IEEE Conference on AI Applications, Miami Beach, FL, 1991.

[Salvendy & Smith 93] G. **Salvendy** and M. J. **Smith** (eds.). *Human-Computer Interaction: Software and Hardware Interfaces.* Amsterdam: Elsevier, 1993. Proceedings of HCI International'93 (5th International Conference on Human-Computer Interaction jointly with 9th Symposium on Human Interface (Japan)).

[Sarkar et al. 93] M. **Sarkar**, S. S. **Snibbe**, O. J. **Tversky**, and S. P. **Reiss**. *Stretching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens.* In: Proceedings of the UIST'93 (ACM SIGGRAPH Symp. on User Interface Software and Technology), pp. 81–92, Atlanta, GA, 1993.

[Seligmann & Feiner 91] D. **Seligmann** and S. **Feiner**. *Automated Generation of Intent-Based 3D Illustrations.* Computer Graphics, 25(3), July 1991.

[Sistare 90] S. **Sistare**. *A Graphical Editor for Constraint-Based Geometric Modeling.* PhD thesis, Department of Computer Science, Harvard University, 1990.

[Sullivan & Tyler 91] J. **Sullivan** and S. **Tyler** (eds.). *Intelligent User Interfaces.* Frontier Series. New York, NY: ACM Press, 1991.

[Vander Zanden et al. 91] B. T. **Vander Zanden**, B. A. **Myers**, D. **Guise**, and P. **Szekely**. *The Importance of Pointer Variables in Constraint Models.* In: Proceedings of the UIST'91 (ACM SIGGRAPH Symp. on User Interface Software and Technology), pp. 155–164, Hilton Head, SC, 1991.

[Wahlster et al. 93] W. **Wahlster**, E. **André**, W. **Finkler**, H.-J. **Profitlich**, and T. **Rist**. *Plan-based Integration of Natural Language and Graphics Generation.* Artificial Intelligence, Special Issue on Natural Language Processing, 63, 1993.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
67608 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder per anonymem ftp von ftp.dfki.uni-kl.de (131.246.241.100) unter pub/Publications bezogen werden.

Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or via anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) under pub/Publications.

The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-93-09
Philipp Hanschke, Jörg Würtz:
Satisfiability of the Smallest Binary Program
8 pages

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf: Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-11
Bernhard Nebel, Hans-Jürgen Bürckert:
Reasoning about Temporal Relations:
A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-12
Pierre Sablayrolles: A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-13
Franz Baader, Karl Schlechta:
A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-14
Joachim Niehren, Andreas Podelski, Ralf Treinen: Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster: PLUS - Plan-based User Support Final Project Report
33 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz: Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-17
Rolf Backofen:
Regular Path Expressions in Feature Logic
37 pages

RR-93-18
Klaus Schild: Terminological Cycles and the Propositional μ-Calculus
32 pages

RR-93-20
Franz Baader, Bernhard Hollunder:
Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-22
Manfred Meyer, Jörg Müller:
Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy:
Comparative Study of Connectionist Simulators
20 pages

RR-93-24
Rainer Hoch, Andreas Dengel:
Document Highlighting —
Message Classification in Printed Business Letters
17 pages

RR-93-25
Klaus Fischer, Norbert Kuhn: A DAI Approach to Modeling the Transportation Domain
93 pages

RR-93-26
Jörg P. Müller, Markus Pischel: The Agent Architecture InteRRaP: Concept and Application
99 pages

RR-93-27
Hans-Ulrich Krieger:
Derivation Without Lexical Rules
33 pages

RR-93-28
Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker: Feature-Based Allomorphy
8 pages

RR-93-29
Armin Laux: Representing Belief in Multi-Agent Worlds viaTerminological Logics
35 pages

RR-93-30
Stephen P. Spackman, Elizabeth A. Hinkelman:
Corporate Agents
14 pages

RR-93-31
Elizabeth A. Hinkelman, Stephen P. Spackman:
Abductive Speech Act Recognition, Corporate Agents and the COSMA System
34 pages

RR-93-32
David R. Traum, Elizabeth A. Hinkelman:
Conversation Acts in Task-Oriented Spoken Dialogue
28 pages

RR-93-33
Bernhard Nebel, Jana Koehler:
Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-34
Wolfgang Wahlster:
Verbmobil Translation of Face-To-Face Dialogs
10 pages

RR-93-35
Harold Boley, François Bry, Ulrich Geske (Eds.):
Neuere Entwicklungen der deklarativen KI-Programmierung — *Proceedings*
150 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

RR-93-36
Michael M. Richter, Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Gabriele Schmidt: Von IDA bis IMCOD: Expertensysteme im CIM-Umfeld
13 Seiten

RR-93-38
Stephan Baumann: Document Recognition of Printed Scores and Transformation into MIDI
24 pages

RR-93-40
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf:
Queries, Rules and Definitions as Epistemic Statements in Concept Languages
23 pages

RR-93-41
Winfried H. Graf: LAYLAB: A Constraint-Based Layout Manager for Multimedia Presentations
9 pages

RR-93-42
Hubert Comon, Ralf Treinen:
The First-Order Theory of Lexicographic Path Orderings is Undecidable
9 pages

RR-93-43
M. Bauer, G. Paul: Logic-based Plan Recognition for Intelligent Help Systems
15 pages

RR-93-44
Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, Martin Staudt: Subsumption between Queries to Object-Oriented Databases
36 pages

RR-93-45
Rainer Hoch: On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition
21 pages

RR-93-46
Philipp Hanschke: A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning
81 pages

RR-93-48
Franz Baader, Martin Buchheit, Bernhard Hollunder:
Cardinality Restrictions on Concepts
20 pages

RR-94-01
Elisabeth André, Thomas Rist:
Multimedia Presentations:
The Support of Passive and Active Viewing
15 pages

RR-94-02
Elisabeth André, Thomas Rist:
Von Textgeneratoren zu Intellimedia-Präsentationssystemen
22 Seiten

RR-94-03
Gert Smolka:
A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 pages

RR-94-05
Franz Schmalhofer, J.Stuart Aitken, Lyle E. Bourne jr.:
Beyond the Knowledge Level: Descriptions of Rational Behavior for Sharing and Reuse
81 pages

RR-94-06
Dietmar Dengler:
An Adaptive Deductive Planning System
17 pages

RR-94-07
Harold Boley: Finite Domains and Exclusions as First-Class Citizens
25 pages

RR-94-08
Otto Kühn, Björn Höfling: Conserving Corporate Knowledge for Crankshaft Design
17 pages

RR-94-10
Knut Hinkelmann, Helge Hintze:
Computing Cost Estimates for Proof Strategies
22 pages

RR-94-11
Knut Hinkelmann: A Consequence Finding Approach for Feature Recognition in CAPP
18 pages

RR-94-12
Hubert Comon, Ralf Treinen:
Ordering Constraints on Trees
34 pages

RR-94-13
Jana Koehler: Planning from Second Principles —A Logic-based Approach
49 pages

RR-94-14
Harold Boley, Ulrich Buhrmann, Christof Kremer:
Towards a Sharable Knowledge Base on Recyclable Plastics
14 pages

RR-94-15
Winfried H. Graf, Stefan Neurohr: Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces
20 pages

RR-94-16
Gert Smolka: A Foundation for Higher-order Concurrent Constraint Programming
26 pages

DFKI Technical Memos

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben:
The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

TM-93-01
Otto Kühn, Andreas Birk: Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta:
Conlfict Resolving Negotiation for COoperative Schedule Management
21 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer:
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

TM-93-04
Hans-Günther Hein:
Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

TM-93-05
Michael Sintek: Indexing PROLOG Procedures into DAGs by Heuristic Classification
64 pages

TM-94-01
Rainer Bleisinger, Klaus-Peter Gores:
Text Skimming as a Part in Paper Document Understanding
14 pages

TM-94-02
Rainer Bleisinger, Berthold Kröll:
Representation of Non-Convex Time Intervals and Propagation of Non-Convex Relations
11 pages

DFKI Documents

D-93-07
Klaus-Peter Gores, Rainer Bleisinger:
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-08
Thomas Kieninger, Rainer Hoch:
Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

D-93-09
Hans-Ulrich Krieger, Ulrich Schäfer:
TDL ExtraLight User's Guide
35 pages

D-93-10
Elizabeth Hinkelman, Markus Vonerden, Christoph Jung: Natural Language Software Registry
(Second Edition)
174 pages

D-93-11
Knut Hinkelmann, Armin Laux (Eds.):
DFKI Workshop on Knowledge Representation Techniques — Proceedings
88 pages

D-93-12
Harold Boley, Klaus Elsbernd, Michael Herfert, Michael Sintek, Werner Stein:
RELFUN Guide: Programming with Relations and Functions Made Easy
86 pages

D-93-14
Manfred Meyer (Ed.): Constraint Processing – Proceedings of the International Workshop at CSAM'93, July 20-21, 1993
264 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-15
Robert Laux:
Untersuchung maschineller Lernverfahren und heuristischer Methoden im Hinblick auf deren Kombination zur Unterstützung eines Chart-Parsers
86 Seiten

D-93-16
Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Gabriele Schmidt: Design & KI
74 Seiten

D-93-20
Bernhard Herbig:
Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus
97 Seiten

D-93-21
Dennis Drollinger:
Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken
53 Seiten

D-93-22
Andreas Abecker:
Implementierung graphischer Benutzungsoberflächen mit Tcl/Tk und Common Lisp
44 Seiten

D-93-24
Brigitte Krenn, Martin Volk:
DiTo-Datenbank: Datendokumentation zu Funktionsverbgefügen und Relativsätzen
66 Seiten

D-93-25
Hans-Jürgen Bürckert, Werner Nutt (Eds.):
Modeling Epistemic Propositions
118 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-26
Frank Peters: Unterstützung des Experten bei der Formalisierung von Textwissen
INFOCOM:
Eine interaktive Formalisierungskomponente
58 Seiten

D-93-27
Rolf Backofen, Hans-Ulrich Krieger, Stephen P. Spackman, Hans Uszkoreit (Eds.):
Report of theEAGLES Workshop on Implemented Formalisms at DFKI, Saarbrücken
110 pages

D-94-01
Josua Boon (Ed.):
DFKI-Publications: The First Four Years
1990 - 1993
75 pages

D-94-02
Markus Steffens: Wissenserhebung und Analyse zum Entwicklungsprozeß eines Druckbehälters aus Faserverbundstoff
90 pages

D-94-06
Ulrich Buhrmann:
Erstellung einer deklarativen Wissensbasis über recyclingrelevante Materialien
117 pages

D-94-08
Harald Feibel: IGLOO 1.0 - Eine grafikunterstützte Beweisentwicklungsumgebung
58 Seiten

Using Graphical Style and Visibility
Constraints for a Meaningful Layout in Visual Programming Interfaces

Winfried H. Graf, Stefan Neurohr

RR-94-15
Research Report