Design by Example*

A User-Centred Approach for the
Specification of Document Layout

Anne Brüggemann-Klein† and Stefan Hermann‡

February 15, 1997

Abstract

DESIGNER is a layout specification system for generically coded documents. It is based on the paradigm *design by example* and the methodology of using *layout objects* to provide graphic artists with an ergonomically sound user interface for their trade.
In this paper we introduce layout objects and present DESIGNER's graphical user interface using some simple examples to explain the main ideas of our system.

1 Introduction

Despite the economic pull of the market (Text processing is the raison d'être for many personal computers) and despite longstanding research efforts at laboratories and universities, very few fundamentals of document processing have yet been established. One paradigm that seems to be generally accepted though is generic coding, embodied by the SGML standard [5] as a method to separate the document content from any meta information required by a specific processing application, such as formatter code. This makes it possible to separate the document content from its graphical presentation, thus enabling the creation of several different presentations from a single source document, for example one for online viewing, one for printing, and one for archiving.

SGML, however, only solves the problem of the syntactic representation of generically coded documents. The problem of providing generically coded documents with graphical semantics, that is, to define their graphical presentation, remains largely unsolved. This is the problem we address with our paradigm *design by example* and with our system DESIGNER.

Current tools used to specify layout for generically coded documents are based on style sheet mechanisms. They use a methodology that defines the layout for generic elements by a large number of parameters. The use of this methodology is problematically because of different reasons. Mainly because this is a low-level approach that often causes problems through mutual impact, constraints or side effects between different parts of layout specifications. It may be compared to the use of early programming languages (e.g., assembler).

An actual trend in specifying layout is the use of high-level objects. These are predefined components realizing complex formatting tasks. A small number of these objects suffices to specify the layout of most documents. A parallel development in the area of computer languages was the

*This work is sponsored by the Deutsche Forschungsgemeinschaft, grant number BR 1309/2-1.
†brueggem@informatik.tu-muenchen.de
‡hermann@informatik.tu-muenchen.de

modularization of programs by using procedures and libraries. Concerning to the specification part this trend was started by the new DSSSL [7] standard. A suitable formatter model was introduced by Murata and Hayashi [9].

Our system DESIGNER follows this trend and introduces high-level *layout objects*. They are used in our system in two different aspects. The first one is described in the rest of this paper and deals with the use of layout objects in the task of creating design specifications. The second aspect is the use of layout objects to realize a new kind of formatter. We realized layout objects as objects in a software technical sense. This allowed us to implement a mighty formatter being more modular than other formatters in a short time. This second aspect is not considered in this paper any more. Section 2 describes in detail the development of the layout objects used in our system.

Besides a new design specification methodology DESIGNER offers two more innovations.

Firstly, we have developed a new paradigm for layout specifications, which we call *design by example*. The graphic designer who specifies a layout works on a sample document, assigning layout objects to specific elements of the document. In the background, a system's component generalizes rules from these sample assignments that can then be applied to other documents of a similar type (of the same DTD in SGML lingo). Generalizing rules from examples is a challenging task if the layout style of an element is to be context dependent; that is, if the layout style varies with the structural context of the element it is applied to.

We are working on a module that generates a suitable set of sample documents from an SGML DTD. Our next step is to define a language for context descriptions and to devise a method for generalizing context descriptions from examples.

By building the Designer system on the paradigm *design by example*, we hope to better support graphic designers in specifying the layout for whole classes of complex documents. Details concerning this paradigm are described in another paper.

Secondly, we have implemented a graphical user interface for DESIGNER. We represent the sample document on which the graphic artist is working on the screen as a tree that can locally collapse and expand, depending on the user's focus of attention. Layout objects can be picked up from a palette in order to apply them to generic elements. The system implements the drag & drop technique. Our goal is to provide graphic artists with an ergonomically sound user interface for their trade. A description of the graphical user interface is given in Section 3.

2 A new design specification methodology: *layout objects*

We assume that the authors of design specifications are graphic designers skilled in the task of applying typographic rules to documents, but not qualified to deal with complex programming languages. Taking into consideration the points we discussed in the introduction we may therefore state that without new paradigms and tools it is a very complex task for designers to create design specifications.

Based on these assumptions we may formulate our goals in the project *Design by Example* as follows: We want to develop and implement a new design specification methodology for designers

that enables them to work in a way they can easily apply their knowledge while defining even complex design specifications for generically coded documents.

In order to do this our next steps will be to investigate on how a designer might contemplate a document with the aim of getting a formalism for design specifications suitable for our target group. These steps are strongly influenced by the DSSSL standard and the work on a formatter hierarchy for structured documents by Murata and Hayashi [9].

To begin with in Figure 1 we introduce the logical structure of the document instance (belonging to the DTD given in the same figure) that guides us through the development of our methodology.

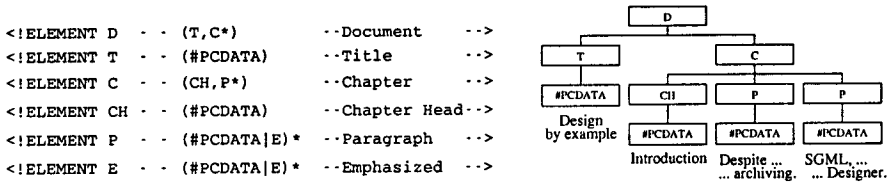

Figure 1: DTD fragment and the logical structure of a sample document

On the left side Figure 2 shows a formatted version of our sample document. On the right side of the same figure we propose an obvious sequence of "mental objects[1]" a reader recognizes (besides the semantic contents of the text components) while reading the document.

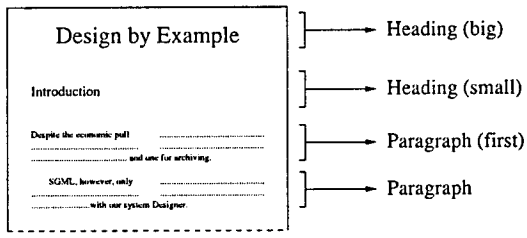

Figure 2: A formatted document and an interpretation of its formatting semantics

To reach this interpretation the reader might have used practical knowledge as an interpretation base as specified in following table that shows the correlations between the used formatting and the mental objects recognized by the user.

used formatting	interpreted mental objects
Paragraph (centred) Glyphs (18pt)	Heading (big) consisting of Characters
Paragraph (left, no indent) Glyphs (16pt)	Heading (small) consisting of Characters
Paragraph (flushed, no indent) Glyphs (12pt)	Text (first Paragraph) consisting of Characters
Paragraph (flushed, indent) Glyphs (12pt)	Text consisting of Characters

[1]We define mental objects as chunks one learns to recognize while learning to read that represent text components (e.g., headings, paragraphs, lists)

If we analyze the used formatting in the light of the DSSSL standard we may regard the formatted document as a sequence of objects, in our example as a sequence of attributed paragraph objects, each with a set of glyph objects for children as shown in Figure 3.

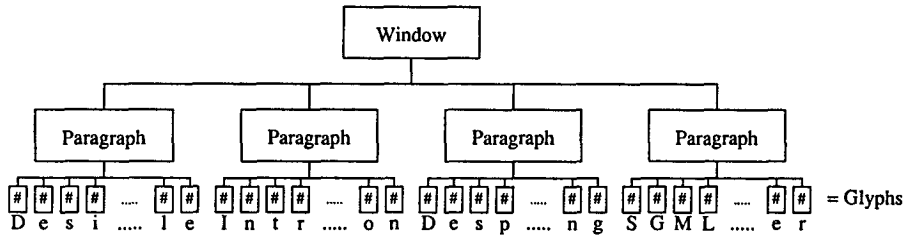

Figure 3: The document, consisting of a hierarchy of objects

In our methodology we call these objects *layout objects.* Layout objects are closely related to the mental objects we recognize while reading documents (e.g., characters, paragraphs, lists). The difference between them is caused by a process of *generalization.* This generalization is an analysis of potential new layout objects with the aim of minimizing the number of offered layout objects. During this analysis we examine whether or not a potential new layout object can be realized by using an already existing and more general one through setting its parameters accordingly to realize the desired effect or an already existing layout object may be generalized by the potential new one. E.g., a "heading" layout object should be generalized to a paragraph layout object with special set attributes (e.g., centred, no first line indent).

Consequently, each layout object specifies a parameterized visual layout effect that could realize, depending on its attributation, multiple different mental objects.

We are implementing a suite of layout objects that are suitable for online publishing of documents in hypertext form. The suite comprises objects for scrollable windows, hypertext linking, paragraph formatting, formatting of mathematical formulae etc. Obviously, by adding different types of objects, for example page-breaking objects, we can support linear, printable documents as well. Therefore, in contrast to the style sheet approach, DESIGNER facilitates true cross-media publishing.

3 Graphical user interface of DESIGNER: the module STYLER

In this section we present the graphical implementation of our methodology of using layout objects to specify the layout of generic elements with the help of sample design specifications.

This means that a graphic designer who specifies a layout for a class of documents does so by working on sample documents belonging to a DTD, rather than directly on a DTD. This implies that in our system the design specification process is executed on occurrences of generic elements, not on generic elements as they are defined in the DTD which means that we are capable of specifying context dependent design rules.

Initially we name some requirements a graphical implementation suited for our target group, graphic designers, should comply with:

- It should minimize the need for users to deal with programming languages,
- nonetheless being universal and expressive, it should allow users to specify the layout they want and
- be comfortable in the sense that a desired layout may be specified as fast and easy as possible, not being influenced by mutual impact, constraints or side effects of other parts of the specification.

However, the graphical user interface has to be as complex as necessary to be adequate for the task of specifying also complex design specifications.

3.1 STYLER's main window

We already mentioned that we want to specify layout by working on sample documents. This arises the question of how to offer these sample documents to a user in a graphical user interface.

Evaluating different display alternatives and considering the fact that we want to express context dependent design rules we chose a tree representation of the sample documents. This representation reflects the logical structure in a space saving and compact way. Figure 4 shows STYLER's main window with our sample document (introduced in Figure 1) displayed on the right side. On the left side of the main window is a palette with available layout objects. Their usage for creating a design specification is explained in the following.

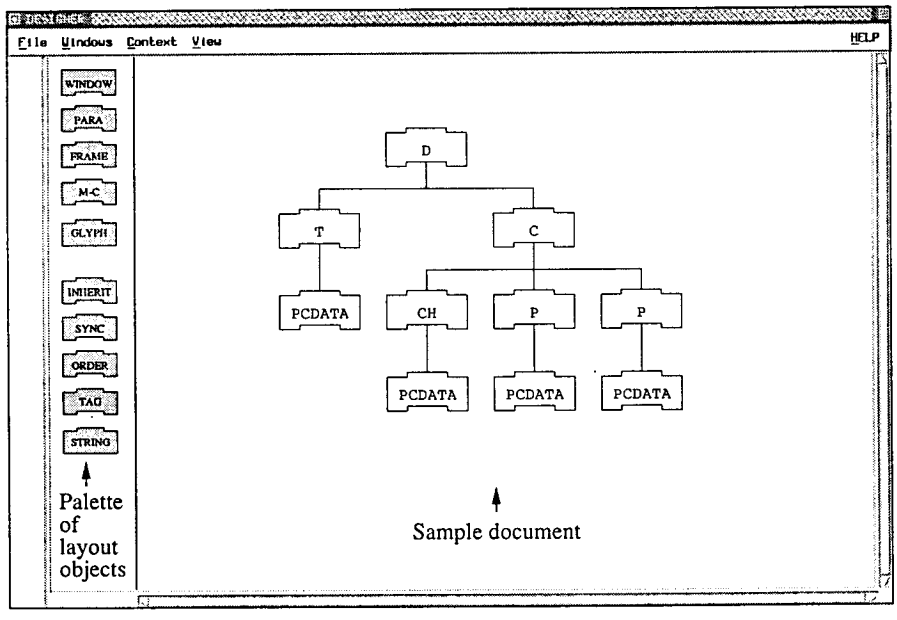

Figure 4: Main window, layout object palette and display of a sample document

3.2 Simple design specifications

There are four basic design action types that were analyzed as sufficient to express universal design specifications [3]:

1. Applying parameterized formatting tasks.
2. Automatic numbering.
3. Generation of boilerplate and derived text.
4. Rearrangement of elements.

In this section we will explain how layout objects are used for specifying the first one, applying parameterized formatting tasks to instances of generic elements. While doing so we introduce the terms *design rules*, *context* and *inheritance*.

3.2.1 Design rules, layout objects and their attributes

A design rule specifies the assignment of layout objects to a generic element and therefore specifies its formatting. In order to define design rules layout objects can be picked up from the palette and be applied to generic elements (drag & drop).

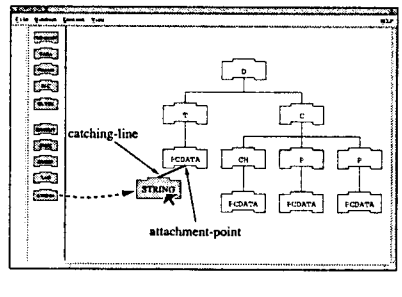

Figure 5: Dragging a layout object

To give visual feedback as to where a user could place a layout object (*attachment points*), generic elements are drawing *catching lines* between themselves and the dragged layout object (Figure 5) to indicate an acceptance. As we will see later, layout objects, too, may accept layout objects and therefore may draw catching lines.

In the most simple case a design rule assigns only one layout object to a generic element. Figure 6 shows on the left side such a simple assignment, expressing that the generic element P should be formatted like a paragraph. Additionally, one may set attributes (in a special window) to influence the actual appearance of the paragraph, such as the first line indent.

To express complex formatting a design rule may assign any number of layout objects to a generic element. E.g., the right rule shown in Figure 6 specifies that there should be an additional frame around the paragraph. A design rule is to be read bottom up, meaning an upper layout object encapsulates a lower one and thus defines an *in_a* relation.

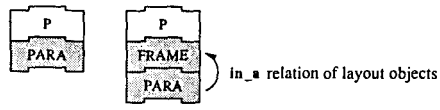

Figure 6: Simple design rules

To specify the formatting of a PCDATA element we may use a special *string object* that is not directly specifying a formatting but realizing a *case statement*. For every case a user distinguishes (the case conditions are entered in a special window) as well as for a default case he may specify a set of layout objects just as in a normal rule. Figure 7 gives an example of such a string object with two rules.

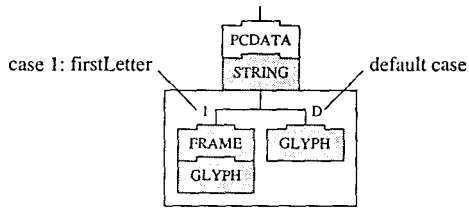

Figure 7: String object with two rules

In order to format the PCDATA element all its characters are processed one after another by evaluating the case statement and applying the given rule of the first matching case condition. Conditions may be of the kind "first character", "every second character" and so on.

In Figure 7 the left rule specifies that a matching character should be formatted by producing a glyph that has a frame around itself. The default rule determines that all other characters are realized by producing just a glyph.

Figure 8 shows an example that specifies the complete formatting for a paragraph and its characters.

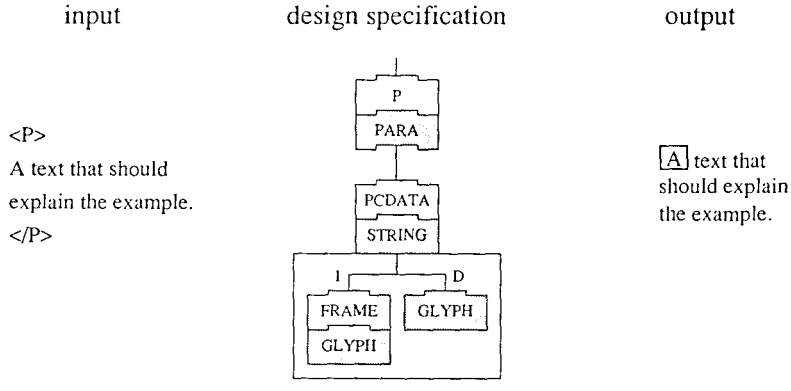

Figure 8: Complete design specification for a paragraph and its characters

To demonstrate the expressive strength of using layout objects for creating design specifications we explain the specification shown in Figure 9. It shows that we may even specify that a generic

element P formatted as a paragraph may have a generic element emph for a child also formatted as a paragraph and consequently specifies the formatting that is shown in Figure 9 on the right side.

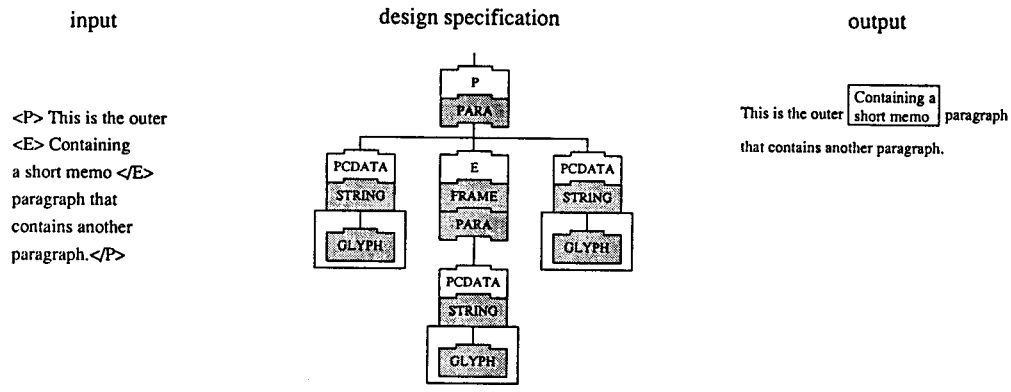

Figure 9: A paragraph layout object containing another paragraph layout object

3.2.2 Context

There is the need for context dependent design rules. E.g., all but the paragraphs immediately preceded by a heading should be indented. To suffice this requirement we offer the possibility to define *contexts* for design rules. A design rule with a specified context is applied only to those instances of generic elements that match the specified context.

To specify a context for a design rule the user marks the generic element a context should be declared for and selects STYLER's *context mode*. Now the user just marks all elements that should build the context by clicking on them. Figure 10 shows this for a paragraph preceded by a heading.

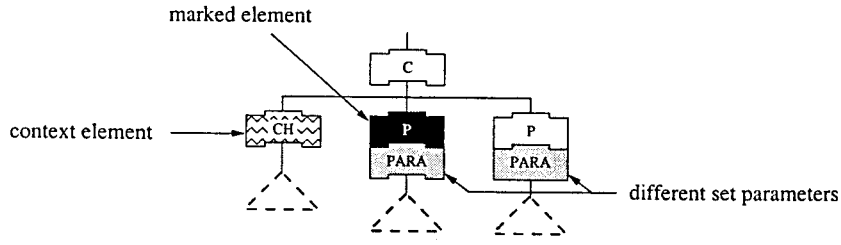

Figure 10: Context for a design rule

The user may specify a design rule for each case a particular formatting is needed. In the example given in Figure 10 the user may simply specify a special setting for the first line indent parameter.

To express a context like "an emphasized text in an emphasized text (where the first emphasized element is not directly father of the second)" we additionally offer the operators ? (matches any element) and * (matches any number of any element) known as wildcards from file systems. These operators may be specified with the help of context sensitive menus.

In defining a context a user may also make references to the attributes of generic elements.

If there are several design rules matching to the instance of a generic element the rule with the "most specific" context (simplified the context with the most selected elements) is chosen. Therefore a design rule with no specified context could be considered as a default rule.

3.2.3 Inheritance, attributes and variables

We already mentioned that attributes influence the actual appearance of layout objects. The most simple way to set a value for an attribute is to specify the value explicitly (e.g., first line indent = 1cm) in a special window.

Another way to set values for attributes is inheritance. As layout objects are ordered in a tree inheritance for attribute values may base on the usual inheritance mechanisms know from programming languages.

With the help of *inheritance objects* a user may define *variables* and set values for them. If an attribute for a layout object is not set explicitly it gets the current value of the variable with the same name making it possible to set values for whole sub trees of a design specification.

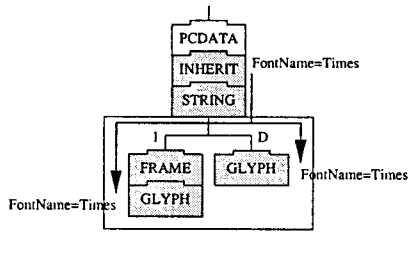

Figure 11: Inheritance for attribute values

To set the values for attributes and variables a designer may use mathematical terms including variables.

A special variable that is set automatically by a PCDATA element is `text` whereby a following string object (that has an attribute called `text`) may automatically get the PCDATA element's text.

3.3 Complex design specifications

Up to now we only introduced objects that use the document content and keep the linear sequence of generic elements. But these objects do not suffice to express all of the basic design action types we introduced.

3.3.1 Boilerplate text and other content

In order to define boilerplate text and other content (e.g., graphics) layout objects offer additional attachment points to the already introduced ones. Figure 12 shows that these are on the left and on the right side of objects. All layout objects may be attached to these attachment points.

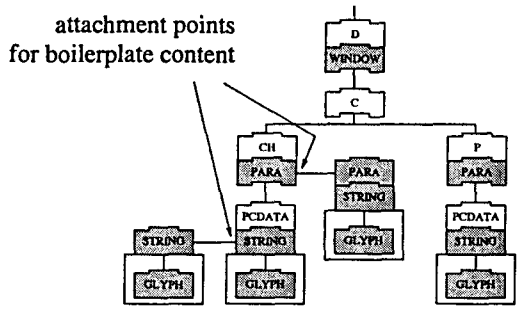

Figure 12: Boilerplate text

Automatic numbering may be realized through a combined use of inheritance and string objects.

3.3.2 Flows, generated content

To solve the problem of generating a table of contents (generated text) from the given document we use a *flow* concept. A flow is an ordered sequence of layout objects.

As an extention to the current design rule model each instance of a generic element may produce sets of layout objects for multiple flows (a generic element may have context dependent design rules for every flow).

To get an order between different flows we offer a *synchronisation object* that accepts multiple flows and produces a single flow by ordering the accepted flows.

Figure 13 shows a design specification for a heading that produces layout objects for the table of contents flow (**C**) and the normal text flow (**T**). These two flows are ordered by a synchronisation object at the top of the document tree before they are printed.

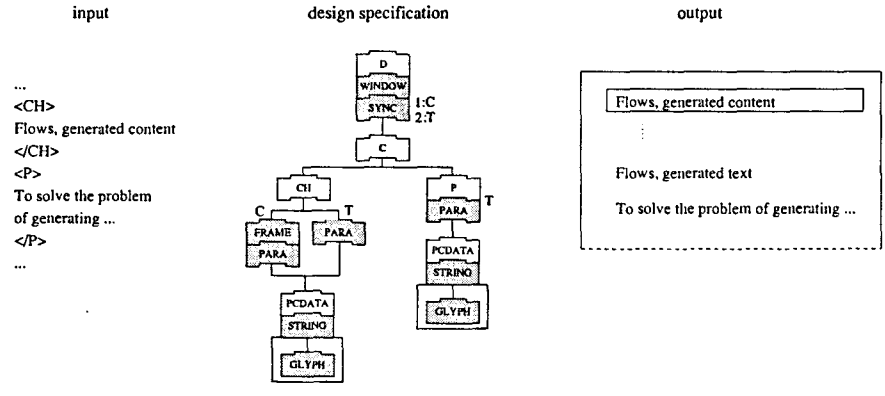

Figure 13: Table of contents flow (C) and normal text flow (T), synchronized

3.3.3 Ordering

So far we explained how to specify the generation of content. Still missing is the possibility to specify a sorting mechanism on layout objects.

In Figure 14 we introduce *order objects* enabling a designer to specify an order on layout objects of a certain flow. The order object reorders the layout objects in the flow it accepts according to the *tag object* every layout object directly connected to an order object should contain for a child. A tag object has an attribute **order** (of type string) that is used by the order object to calculate the sequence of the to be ordered layout objects.

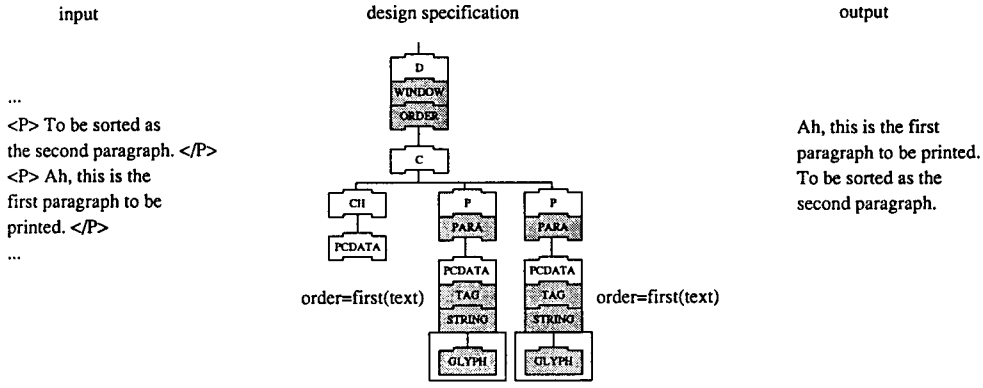

Figure 14: Ordering of layout objects

3.3.4 Constraining of layout objects

One more layout object accepting multiple flows is the *multicolumn* layout object having a variable number of columns with each of it accepting a single flow. For such a layout object a designer might want to specify the constraint that two layout objects in two different columns should be placed side by side.

To explain how such constraints between layout objects are expressed we discuss the example given in Figure 15.

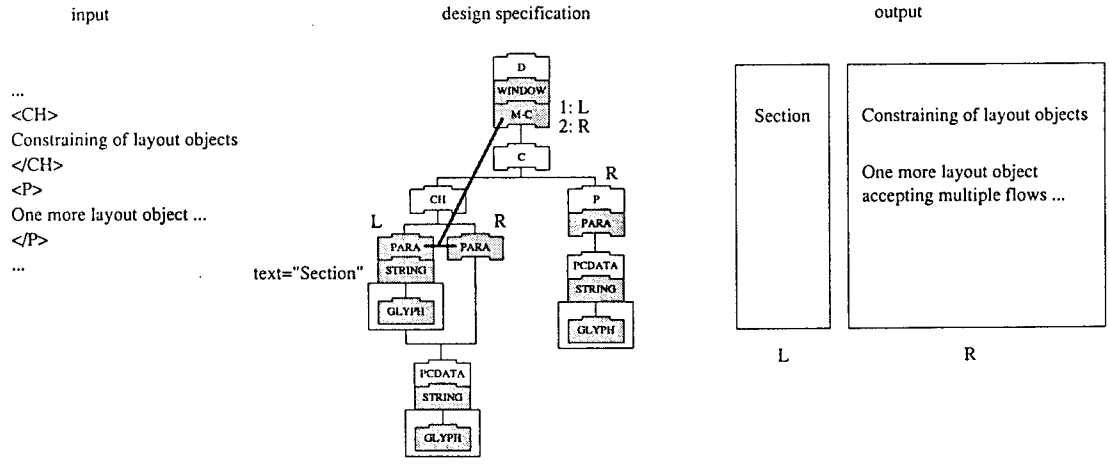

Figure 15: Constraining of layout objects

To express constraints between two layout objects to a common parent layout object the designer switches to the *constraint mode*. Now the designer first clicks on the common parent layout object

that should resolve the constraint. Secondly the designer has to draw a *constraint line* between the two constrained layout objects.

4 Conclusions and future work

In this paper we have presented a new design specification system for generically coded documents. The main advantage of DESIGNER is that in contrast to current systems DESIGNER uses high level semantic layout objects for specifying the semantic and layout of generic elements. Additionally DESIGNER offers a graphical user interface.

In order to get a user-centred system we have implemented and evaluated a first prototype which is currently improved in a second implementation. The given figures have been adapted from this second implementation.

We have recognized that the use of layout objects allows the creation of design specifications current formatters can not cope with. We have therefore implemented our own formatter that can currently deal with most parts of our design specifications. The formatter is based on the ideas introduced by Murata and Hayashi [9].

Future research on DESIGNER will focus on our paradigm *design by example*: generalizing design rules from sample assignments. At present a designer has to specify design rules and its contexts in order to create a design specification. In a future version of DESIGNER we want to include a module that runs in the background and generalizes rules from sample assignments. This is a challenging task if the layout of an element is to be context dependent.

This work is sponsored by the Deutsche Forschungsgemeinschaft, grant number BR 1309/2-1.

References

[1] W. Appelt. Dokumentaustausch in Offenen Systemen. Einführung in die ISO-Norm 8613: Office Document Architecture (ODA) and Interchange Format. Springer, Berlin, 1990.

[2] F. Arnold, 1997. Personal communication.

[3] A. Brüggemann-Klein. Formal Models in document Processing. Habilitationsschrift vorgelegt an der Mathematischen Fakultät der Albert-Ludwigs-Universität zu Freiburg i. Br., 1993.

[4] J. Clark. Source code for SP, a SGML parser. Different tutorials about DSSSL. URL http://www.jclark.com, 1996.

[5] C. F. Goldfarb. The SGML Handbook. Clarendon Press, Oxford, 1990.

[6] ISO 8613. Information Processing - Office Document Architecture (ODA) and Interchange Format. International Organization for Standardization, Geneva, 1989.

[7] ISO/IEC DIS 10179.2. Information Technology - Processing languages - Document Style Semantics and Specification Language (DSSSL). International Organization for Standardization, Geneva, 1996.

[8] E. V. Munson. A new presentation language for structured documents. In A. Brown, A. Brüggemann-Klein & A. Feng, editors, EP96, Proceedings of the Sixth International Conference on Electronic Publishing, Document Manipulation and Typography, pages 125-138, September, 1996 (Journal Special Issue, Volume 8, Issues 2 and 3 of the International Journal Electronic Publishing - Origination, Dissemination and Design, edited by D. F. Brailsford and R. K. Furuta.).

[9] M. Murata and K. Hayashi. Formatter Hierarchy for Structured Documents. In C. Vanoirbeek & G. Coray, editors, EP92, The Cambridge Series on Electronic Publishing, pages 77-94, Cambridge, 1992. Cambridge University Press.

[10] D. E. Knuth. The TEXbook. Addison-Wesley, Reading, MA, 1986.

[11] V. Quint, translated by E. Munson. The languages of Grif. URL ftp://ftp.inrialpes.fr/pub/opera/logiciels/languages.ps.gz, 1997.

[12] W. Rieger. SGML für die Praxis. Springer, Berlin Heidelberg, 1995.

[13] C. Roisin and I. Vatton. Merging Logical and Physical Structures in Documents. Electronic Publishing: Origination, Dissemination, and Design (EPODD) EP '94. Fifth International Conference on Electronic Publishing, Document Manipulation, and Typography, Darmstadt, Germany, 13-15 April 1994. 6/4 (December 1993), pages 327-337. URL ftp://ftp.imag.fr/pub/OPERA/doc/EP94.ps.gz

[14] H. W. Lie and B. Bos. Cascading Style Sheets, level 1. URL http://www.w3.org/pub/WWW/TR/PR-CSS1, 1996.