The DesignComposer: Context-Based Automated Layout for the Internet

Alexander Kröner

German Research Center for Artificial Intelligence (DFKI) GmbH
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
kroener@dfki.de
http://www.dfki.de/~kroener

Abstract

Layout plays crucial role in readability and interpretation of presentations. This holds also for online presentations, where automated layout generation is needed to deal with dynamic presentation contents and changing presentation contexts. In this paper, we present the DesignComposer prototype, an automated layout generator for web pages. It uses different AI approaches and a declarative representation of knowledge about design and presentation context to gain maximum flexibility in the generation process as well as an easy access to the layout knowledge.

Introduction

Having a look on today's information society, the huge amount of information offered by different media, such as magazines, TV or the Internet makes it more and more difficult for information providers to gain a viewer's attention. If a presentation is bad to read or difficult to understand, he will ignore it soon and address another offer.

Therefore it is of great importance to make presentations appealing, a task which requires comprehensive knowledge about content selection and layout design. For static presentations with a fixed content and a single output environment this task can be solved by human authors and layout designers.

Online presentations however are often partially or fully dynamic. They are displayed in environments differing in used hard- and software, and their content may be generated during display time from a knowledge source like a database.

Using a static layout under these conditions may result in layout deficiencies which have a negative effect on readability and understanding of the offered information. Some of them are:

- bad positioning: if key objects are on bad positions or are not on the visible screen, the viewer is forced to interact with the presentation. For example he has to search using scrolling. If he does not, he may oversee important parts of the presentation.
- unsuited style: a bad coloring or sizing generally makes layout more difficult to read and therefor the presented information more difficult to access.

Such problems can be avoided by the adaptation of layout to the current presentation context. Doing this manually is not only a cumbersome and expensive task, but even impossible in applications with a dynamic presentation content or when layout should be personalized, an upcoming service which is requested by information providers for their customers.

Thus simple pattern techniques have become a common way to realize a simple form of an adaptive automated layout. But these enable only a restricted usage of a human designers knowledge, what restricts the complexity – and thus the quality – of the produced layout. Furthermore, the presentation contexts complexity makes it nearly impossible to take each possible case into consideration for layout. And finally, since none or little search is done, the pattern approach cannot react on "unexpected" contexts in an appropriate way, what reduces the quality of result again.

To overcome these problems, you need a more powerful toolkit. In this paper we will propose an approach for the automated layout generation of web pages which is based on the integration of complex layout styles, a context model and a mix of several approaches for the processing of that knowledge.

Approach

Basically, layout is a visualization of presentation elements which include content fragments as well as semantically and pragmatically annotations. Layout can be subdivided into layout objects which represent some information, like a piece of text or an image, and are displayed in a layout style. Such a style may include requirements for:

- Topology: shape and location of a layout object, typically connected to some validation criteria like non-overlapping.
- Composition: dependencies between layout objects. It expresses object groupings and more complex connections like navigation structures.
- Form: properties like color describing the "look" of a layout object.

The layout generation process has to map presentation elements to sets of layout objects without violating their individual style requirements.

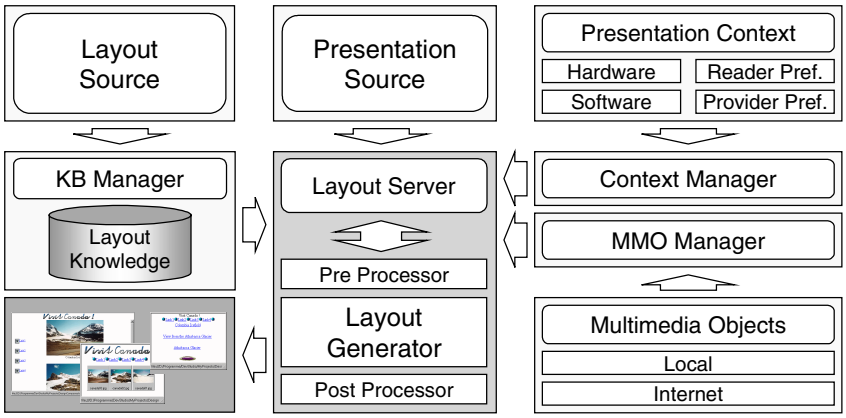

Fig. 1: Abstract architecture of the DesignComposer

Different approaches are known for automated layout generation, such as dynamic programming (e.g., [Pla81]), relational grammars (e.g., [WW94]), rule-based systems (e.g., [Fei88]), genetic algorithms (e.g., [KMS94]) and especially constraint processing techniques (cf. [Gra97]).

If an implementation is kept general enough, each of these should be able to solve all the problems of dynamic layout. Unfortunately, proceeding this way leads for real-world applications to the following problems:

Efficiency: the renouncement of specific solving strategies can cause efficiency problems. For example, specific algorithms on constraint basis are known for the solution of topological problems, which are typically more efficient than a general constraint solver.

Representation: layout knowledge has to be easy to access by human layout designers. A knowledge representation language can help here, but using a single approach makes knowledge specification sometimes difficult: for example constraints provide a good means of expressing topological conditions, but are unsuited (or at least unhandy) for the specification of solving strategies.

Restricting the layout problem to a specific domain like dictionaries or manuals can help to avoid these difficulties. But this is no suitable solution for automated layout of web pages, which do not have a common topic.

We decided to create our new layout generator, the DesignComposer, based on a combination of different approaches, especially constraints, rules and pattern techniques. Based on experience with real-world and experimental layout generation software, we tried to get the most appropriate toolkit for each of the tasks associated with layout creation.

The resulting mapping between task and approach for layout styles is shown in Tab. 1. It is only of a heuristically nature, and it is not explicit, because sometimes you may express knowledge in several ways. Human layout designers should use it as a guide for the specification of layout knowledge to guaranty an effective usage of the DesignComposer, but they are not limited in trying out other combinations.

Task	**Approach**
Control	Rules
Composition	Rules
Style	Constraints & Patterns
Topology	Constraints

Tab. 1: The mapping between task and approaches

The approach is accompanied by a context model build up from information about the presentation provider and the presentation reader. It supports simple conflict resolution strategies between concurrent context data to provide processing of partial invalid and multiple data definitions. The context information may be used in each stage of the generation process to do decisions about layout.

Architecture

The DesignComposer's architecture is shown in an abstract way in Fig. 1. Main input is a markup presentation, typically an XML or HTML script, generated automatically by some external software, or manually by a human author. It is forwarded to the layout server module. Further input is layout knowledge specified by a human layout designer which is stored in a layout knowledge base.

The layout server module controls layout tasks currently under progress. It may access an individual knowledge base and context description for each given layout task.

Layout tasks are accomplished in three steps: First, pre processing takes place, where the presentation input and layout knowledge base may be prepared for processing. Then layout is created by the generator. After finishing this task, the result is post processed to optimize and to beautify it.

The current result may be written out (currently as a set of HTML files) at any time, and it is stored by the layout server to give external software the opportunity to gain additional information about the result from the DesignComposer's internal structures.

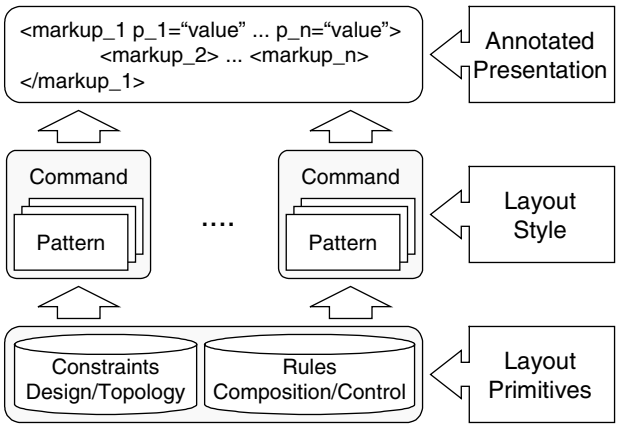

Fig. 2: External layout representation

Layout Representation

We distinguish between two categories for the representation of layout knowledge: the external representation visible to presentation authors and layout designers and the internal representation used by the layout generator for the processing of the external layout knowledge.

External Representation

The first step to a flexible layout knowledge representation is the separation of layout knowledge from the presentation content. The knowledge is bundled into a layout style, what brings the following advantages:

- Support of document classes: one style can be applied to different presentations – a presentation can be shown in different styles
- The author is freed from the often cumbersome and difficult layout task
- Maintenance of layout knowledge is simplified

Using the approach discussed before, our style definition has to be able to represent different kinds of layout knowledge, especially constraints and rules. Our goal was to map these *layout primitives* to the presentation, keeping the presentation content free from layout knowledge.

Explicit references from the content to layout primitives would require knowledge about their meaning for layout. Thus we decided to introduce so-called *design commands*, interface structures used to map semantic and pragmatic aspects of a presentation onto layout primitives stored in the layout knowledge base. This is done on the one side by connecting a design command with a markup, which should be used as a semantic or pragmatic annotation by the presentations author, and on the other side by connecting the design command with a set of *design patterns*, each of them describing a unique composition, topology and design for the command by a set of layout primitives.

This representation model offers different levels of complexity to a user, which require an increasing amount of experience in dealing with layout knowledge (see Fig. 2). On the topmost level, the author of a presentation has to deal with simple markups. Going deeper into the representation, layout design knowledge is required at several levels of abstraction. First, one may create and modify design commands build from existing design patterns. Then the pattern level follows consisting in sets of layout primitives. On the bottom level, primitives can be specified, what should be done by experienced users.

Layout knowledge expressed in the *design command language* DCL. Basically, DCL is a Java-like and therefor procedural language with several extensions for the expression of patterns, layout primitives and markup references. It is organized by a knowledge base control. Heart of this module is the DCL parser, which compiles DCL scripts into a knowledge base.

Internal Representation

The layout generator maps the layout knowledge stored in the knowledge base onto the internal object model.

Each element of this model is connected to a layout handler, a hard coded module which describes the specific layout behavior of the connected objects. One can distinguish between atomic and composed objects. Atomic objects are objects which cannot be divided into smaller objects by the layout generator such as pictures or words. Usually their layout is more of a static nature, but atoms which dynamic layout like scalable images are – an appropriate layout handler assumed – also supported by the object model. Composed objects are build up from other composed objects and atoms; they group these objects to logical units. Such *containers* carry their own local layout knowledge base to do assertions about subordinate objects.

Thus a hierarchy of layout objects is formed which provides several advantages like local constraint scopes stored in containers or inheritance of layout constraints.

For reasons of efficiency, the internal object model is completely hard coded. Its object oriented structure makes it easy to add further extensions in the future, what allows the integration of new layout handlers which can solve problems the current handlers are not able to deal with.

Layout Generation

Generating a layout based on the knowledge representation model described above means – simply spoken – the translation of a markup script into an instance of the internal layout object model. This is done at several levels (see Fig. 3).

On the top level, a design command is selected. The next step is the selection of a design pattern by the design constraint solver. At the pattern level, the current layout result may be modified by the change of existing and the addition of new material, what is done by processing the patterns composition description step by step.

The composition description mainly consists in references to other design commands. Thus processing may

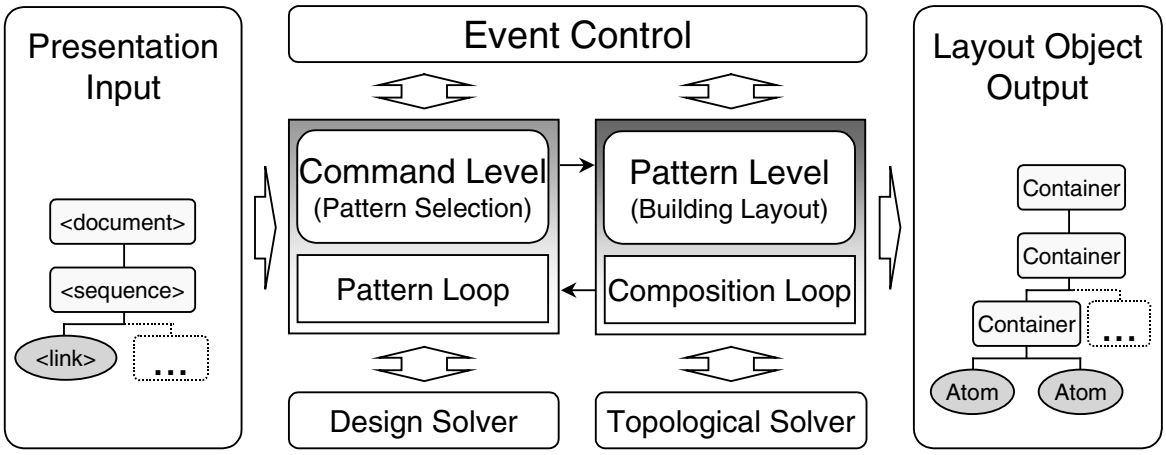

Fig. 3: Layout processing

change back to the top level or, if no further decomposition is possible, directly execute the current layout handler to get an atomic layout object.

Each change in the current result is reported to the topological constraint solver, which checks the validity of constraints which are part of the result and the pattern currently being executed.

The layout generation process is accompanied by an event controller, a demon-like module which compares the current generation state with known patterns. If such a pattern is identified – a scope is full, a sequence is broken etc. – an event is generated and the event handling module is started, which modifies the current result if required.

If the processing fails, search can be done on each level: layout handlers may modify objects, patterns may be changed and so on. The large search space formed this way is searched through with guided backtracking: based on simple heuristics the system tries to identify the object responsible for a given problem to start search at a point where success is most probably. Additional heuristics provide an efficient handling for well known problem situations.

Example

In the following, we present a simple example illustrating the variety of layouts we may generate from a given input.

Let's assume we want to generate a layout for a presentation consisting of images in dependency of a given browser size. The presentation input could look as follows:

```
<document name="Visit Canada !">
<chapter name="Banff">
<sequence>
<describe target="Canada01.jpg"
   desc="The Bow Falls near Banff."/>
<describe target="Canada02.jpg"
   desc="The Vermillion Lakes."/>
<describe target="Canada03.jpg"
   desc="Banff's main road."/>
</sequence>
</chapter>
</document>
```

The markups used in the presentation refer to design commands specified in DCL. Let's have a closer look at the design command for the markup <describe>. The markup requests the layout for a description using a set of arguments, here an image file and a description text.

```
design describe(string target, string desc) {
   pattern full:
      constraints = hCenter;
      { <image src=target/><text src=desc/> }
   pattern icon:
      container = LinkObj(target);
      { <image src=getIconFile(target)/> }
   pattern textual:
      container = LinkObj(target);
      { <text src=desc/> }
}
```

Three patterns are defined for <describe>. The first one, *full*, shows the image and the describing text. The second one, *icon*, tries to display an icon which is linked to the image but suppresses the textual description, and the last one, *textual*, shows only the description, also linked to the original image.

Varying the output size will produce layouts differing in design, composition and topology (see Fig. 4 for three results). They are spread over multiple pages connected by an automatically generated navigation structure. A more detailed description, further examples and a demo are available on the DesignComposer homepage:

http://www.dfki.de/~kroener/DC.htm

Fig. 4: Different layouts for one presentation

Conclusion and Further Work

In this paper, we have briefly discussed the prototype of a new layout generation system, the DesignComposer, especially designed for, but not restricted to the layout of web pages. The generation process considers the layout context, contains different AI techniques, and relies on a flexible and easy to maintain object model which enables high flexibility.

The approach includes also a highly sophisticated declarative representation of layout knowledge, which provides both presentation authors and layout designers an easy access to layout knowledge at different access levels.

The DesignComposer prototype offers several options for extensions. Currently, a learning module is being integrated into the layout server. Our goal is to use design knowledge from existing results for new results based on similar input parameters. A first implementation of this module already exists and has produced pronouncing results.

Another point is the need for standardization of layout structures. Based on the World Wide Web Consortium's Document Object Model specification, we will change our object model to simplify the access by external software.

Furthermore, a graphical user interface is under construction which will simplify presentation construction and layout knowledge specification.

Finally, the DesignComposer has to be integrated into other systems to show its abilities in the daily use. Under consideration are the AiA presentation system (see [AGM+96] and [AMR98]) and the information retrieval system IMAP, conducted in collaboration with Sarit Kraus from Bar-Ilan university.

References

[AGM+96] André, E.; Graf, W. H.; Müller, J.; Profitlich, H.-J.; Rist, T.; Wahlster, W. 1996. AiA: Adaptive Communication Assistant for Effective Infobahn Access. Technical report, German Research Center for Artificial Intelligence (DFKI) GmbH, Saarbrücken, Germany.

[AMR98] André, E.; Müller, J.; Rist, T. 1998. WebPersona: A Life-Like Presentation Agent for the World-Wide Web. *Knowledge-Based Systems* 11(1): 25-36.

[Fei88] Feiner, S. 1988. A Grid-Based Approach to Automating Display Layout. In *Proceedings of the Graphics Interface 1988*, 192-197. Los Altos, CA: Morgan Kaufmann.

[Gra97] Graf, W. H. 1997. Constraint-Based Graphical Layout of Multimodal Presentations. In M. T. Maybury and W. Wahlster (Publ.), *Readings in Intelligent User Interfaces*. Los Altos, CA: Morgan Kaufmann.

[KMS94] Kosak, C.; Marks, J.; and Shieber, S. 1991. A parallel genetic algorithm for network-diagram layout. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 458-465. Los Altos, CA: Morgan Kaufmann.

[Pla81] Plass, M. F. 1981. Optimal Pagination Techniques for Automatic Typesetting Systems. Ph.D. diss., Dept. Of Computer Science, Stanford Univ.

[WW94] Weitzman, L., and Wittenburg, K. 1994. Automatic Generation of Multimedia Documents Using Relational Grammars. In *Proceedings of the ACM Multimedia 1994*, 443-451. San Francisco, CA.