Scout: Mixed-Initiative Exploration of Design Variations through High-Level Design Constraints

Amanda Swearngin, Amy J. Ko, James Fogarty
Paul G. Allen School, The Information School
University of Washington
Seattle, WA, 98195
amaswea@cs.uw.edu, ajko@uw.edu, jfogarty@cs.uw.edu

ABSTRACT

Although the exploration of variations is a key part of interface design, current processes for creating variations are mostly manual. We present Scout, a system that helps designers explore many variations rapidly through mixed-initiative interaction with high-level constraints and design feedback. Past constraint-based layout systems use low-level spatial constraints and mostly produce only a single design. Scout advances upon these systems by introducing high-level constraints based on design concepts (e.g. emphasis). With Scout, we have formalized several high-level constraints into their corresponding low-level spatial constraints to enable rapidly generating many designs through constraint solving and program synthesis.

INTRODUCTION

Variations are key in interface design. When a designer creates multiple design variations in parallel, it results in higher quality design outcomes [7] and more diverse solutions [3]. When designers compare these alternatives, they provide stronger critiques and make better decisions [6, 16]. However, designers face several barriers in creating many, high quality variations. First, it is hard to come up with a completely new idea and avoid fixation [10]. Second, creating a design requires knowledge of design and usability principles. It can be challenging, especially for novices, to follow accepted design principles with every variation. Finally, creating variations is still a manual process that requires low-level resizing, restyling, and relocating interface elements to produce design prototypes.

To aid designers in creating and exploring interface variations, we introduce Scout, a mixed-initiative system that helps designers explore many layout variations rapidly. A designer using Scout expresses their desired interface elements and constraints between them, and Scout generates design variations satisfying these constraints.

Scout uses techniques from program synthesis and constraint solving to automatically generate variations satisfying design constraints. There is a rich history of the use of constraints for generating interface layout and graphic designs [2, 8, 11, 18, 19, 20]. However, these tools primarily create one solution at a time, and do not allow the user to control specific attributes of the variations [15]. With Scout, the designer can explore the entire space of potential solutions, and

Figure 1. The workflow of Scout’s interface. Designers add elements to Scout’s interface through the widgets library (A). Designers create constraints in a tree (B). Designers explore designs satisfying those constraints, give feedback (D) that appears as an annotation in the tree (C), and curate saved and discarded designs in the designs exploration area (E). Designers can export a design to refine in their design prototyping software (F).
can narrow down the potential solutions to those that satisfy their design constraints. Additionally, in constraint-based layout systems, interface properties are mostly through low-level spatial constraints. For example, in Apple’s AutoLayout [9], constraints are expressed as mathematical equations like “Element1.Leading = Element2.Trailing + 8.0” which states that the margin between Element1 and Element2 must be 8 pixels. Such constraints are unintuitive and tedious to maintain for designers. The ALE [20] automatically infers some low-level constraints, however, still only outputs a single solution.

In contrast, Scout lets designers specify high-level constraints (e.g., “The header should be the most visually salient element”) based on usability and visual design principles. Scout translates high-level constraints into low level spatial constraints, and automatically generates designs satisfying them. Designers define high-level constraints through Scout’s constraint language which lets designers group related elements, and annotate emphasized and deemphasized elements. Designers explore and refine the designs through feedback constraints.

**CONSTRAINT LANGUAGE**

Scout allows designers to express high-level constraints based on design principles. The goal is for designers to organize their interface in a principled way, emphasize important elements, and explore the space of possible designs through feedback.

**Proximity, Repetition, and Emphasis**

A key principle of interface design is to have a clear, organized hierarchy [13]. The Structure Principle [4] states that interfaces should keep related things together and unrelated things separate, which is motivated by Gestalt theory [12]. Scout lets designers create proximity groups of related elements. Scout ensures that these elements appear as a visually distinct group in design variations (e.g. Figure 1).

A key usability principle is that elements should appear in the order they are used for a task [14]. Scout lets designers specify that the order is important or unimportant for a proximity group. Scout maintains the elements in reading or task order in each variation. A designer can also specify that an element should appear first (e.g., a header) or last (e.g., a footer), and Scout will only vary the positions of the unconstrained elements in design variations.

Many interfaces also include repeating patterns of elements (e.g., a card list). Scout allows designers to apply a repeat group constraint which ensures that the layout of groups of elements is kept consistent across repetitions. Scout automatically infers and suggests to the designer when this constraint can be applied to repeating patterns of elements.

Emphasis is a concept in graphic design [17] that has been adapted to interface design. Emphasis design principles state that interfaces should have a main focal point to let the user know what to do next [1]. In Scout, designers can emphasize and de-emphasize important and unimportant elements. Designers can only emphasize one element or group of elements to provide a main focal point. Scout will then emphasize the element by increasing its size, and moving it towards the center of the design which increases visual prominence [17].

**Feedback on Designs**

In Scout, designers explore and refine the set of designs through feedback constraints. Scout allows several types of feedback including global (e.g. “Use a 50 pixel layout grid”), relational (e.g. “Keep the logo above the header.”), element (e.g. “Keep this element here.”), and arrangement specific (e.g. “Keep the layout of these elements horizontal.”).

**SCOUT USER INTERFACE WALKTHROUGH**

Lucy is a UX designer, and is exploring designs for a Hollywood Walking Tours app. Lucy feels stuck, so she decides to use Scout to help her explore design variations.

Lucy first imports her desired interface components into Scout from her agency’s component library (Figure 1, A). Lucy adds her elements to Scout’s interface through the constraints tree (Figure 1, B) including a “Continue” button, a header, and a few labels and tour images. She creates a proximity group for the set of tour images and labels. Scout suggests that she apply the “Repeat Group” constraint to the proximity group to ensure the layout of the icon and label pairs is kept consistent in the variations.

Lucy views her designs through the design exploration area (Figure 1, E). Lucy saves a few designs she likes which appear in the saved designs panel, and “throws away” several designs into a discarded designs panel. Lucy prefers the designs that use a 5 pixel layout grid, so she uses the feedback menu to tell Scout to keep the layout grid at 5 pixels. Her feedback appears as an annotation on the constraints tree (Figure 1, C). Now, Lucy can request more designs exploring variations of designs with a 5 pixel layout grid. Scout has marked several designs as invalid by highlighting them with red diagonal stripes. Lucy hovers her mouse over the invalid designs and Scout highlights the conflicting annotations in the constraints tree.

**IMPLEMENTATION**

The interface elements, high-level constraints, and design feedback the designer creates in the Scout interface are translated into a set of formalized constraints in a design synthesis engine that uses Z3 [5] with a custom search algorithm. Scout produces a set of x and y coordinates, height, and width for each element as output which is displayed as a design canvas in the Scout interface. Within Scout, we have formalized basic design constraints (e.g. non-overlapping, stay-in-bounds) and high-level constraints (e.g., emphasis and proximity grouping). To produce good designs, we have formalized some graphic design principles (e.g. alignment, balance, and symmetry), and use them in a cost function to rank the designs. To give the designer a more spatially diverse set of designs, we apply a distance metric and randomized ordering of variable assignments in the constraint solver.

Currently, Scout can explore layout variations that change the positions and sizes of elements to provide proximity grouping, emphasis, and repeat groups. In the future, we plan to formalize more design principles into the solver, develop a more sophisticated cost function to select high-quality designs, and vary more properties of design (e.g., fonts, color schemes) to diversify the variations.
REFERENCES

https://www.interaction-design.org/literature/article/emphasis-setting-up-the-focal-point-of-your-design


http://dx.doi.org/10.1007/978-3-642-31991-4_4

http://dx.doi.org/10.1007/978-3-642-21643-5_8

http://dx.doi.org/10.1145/1294211.1294253

https://developer.apple.com/design/adaptivity/

http://dx.doi.org/10.1016/0142-694X(91)90093-F

http://dx.doi.org/10.1145/169659.169528

http://dx.doi.org/10.1007/BF00442382


http://dx.doi.org/10.1145/97243.97281

http://dx.doi.org/10.1145/2702123.2702149

http://dx.doi.org/10.1145/1124772.1124960


http://dx.doi.org/10.1145/2642918.2647398

http://dx.doi.org/10.1145/108844.109905

http://dx.doi.org/10.1145/2501988.2502007