Automatic Generation of User Interface Layouts for Alternative Screen Orientations

Clemens Zeidler[1(✉)], Gerald Weber[1], Wolfgang Stuerzlinger[2], and Christof Lutteroth[3]

[1] University of Auckland, Auckland, New Zealand
{clemens.zeidler,g.weber}@auckland.ac.nz
[2] School of Interactive Arts and Technology (SIAT), Simon Fraser University, Vancouver, Canada
w.s@sfu.ca
[3] University of Bath, Bath, UK
c.lutteroth@bath.ac.uk

Abstract. Creating multiple layout alternatives for graphical user interfaces to accommodate different screen orientations for mobile devices is labor intensive. Here, we investigate how such layout alternatives can be generated automatically from an initial layout. Providing good layout alternatives can inspire developers in their design work and support them to create adaptive layouts. We performed an analysis of layout alternatives in existing apps and identified common real-world layout transformation patterns. Based on these patterns we developed a prototype that generates landscape and portrait layout alternatives for an initial layout. In general, there is a very large number of possibilities of how widgets can be rearranged. For this reason we developed a classification method to identify and evaluate "good" layout alternatives automatically. From this set of "good" layout alternatives, designers can choose suitable layouts for their applications. In a questionnaire study we verified that our method generates layout alternatives that appear well structured and are easy to use.

Keywords: Automatic layout · Layout design · Screen rotation · Device independence

1 Introduction

Today, mobile apps need to support different screen orientations, i.e., landscape and portrait, and need to have reasonable layouts for both orientations. To support this need, mobile platform user interface (UI) development frameworks enable developers to specify flexible UIs, which can adapt to different conditions through two means: UI designers can use *layout managers* to resize a UI automatically to the available screen space, and UI designers can specify *layout alternatives* for different screen orientations.

R. Bernhaupt et al. (Eds.): INTERACT 2017, Part I, LNCS 10513, pp. 13–35, 2017.
DOI: 10.1007/978-3-319-67744-6_2

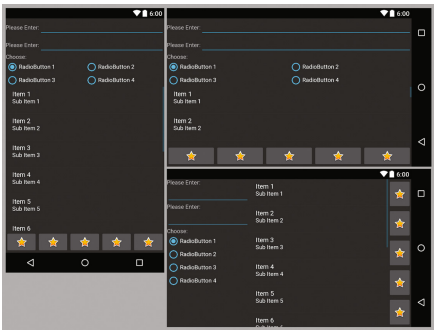

Fig. 1. Left: portrait layout. Top right: original layout in landscape mode; space is not optimally used. Bottom right: layout adapted to landscape mode.

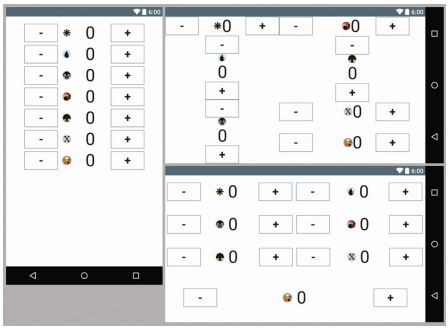

Fig. 2. There are many possible layout alternatives for a given portrait layout. Top right: an example of a reasonable good alternative. Bottom right: an example for a bad alternative.

Layout managers adapt the size of each UI widget to the available space, but in general preserve the overall arrangement of the widgets. An example of this is shown in Fig. 1: the left parts shows a UI designed for portrait orientation, and at the top-right, the same UI is shown in landscape. As this UI's layout manager keeps relative positioning intact, the landscape UI still has the same overall layout as the original design on the left. Yet, due to the different aspect ratio between portrait and landscape orientation, the UI at the top-right does not make optimal use of the available space: the interface is stretched horizontally while the list view displays only a few items.

Some layout managers, such as a flow layout, can make a layout more adaptive to different screen sizes. However, such adaptive layout managers may not always be powerful enough or can lead to undesirable results. For example, a flow layout may break a row of widgets at an undesired position. Also, multiple flow layouts in the same layout may result in undefined behavior since multiple solutions are possible. To avoid this, many UI designers today manually specify separate layout alternatives for certain screen conditions, e.g., for landscape. In Fig. 1, such a layout is shown at the bottom-right: widgets are rearranged to make better use of the available space. Yet, manually creating alternative layouts is laborious and error prone. Regardless of the used layout manager, the developer has to consider the layout carefully for each screen condition. Automatic generation of layout alternatives would not only make it easier for developers to adapt their layouts to alternative screen orientations, but also support their creativity by providing new layout ideas they may not have envisioned as an alternative.

The number of possible layout alternatives grows exponentially with the number of widgets in the layout, i.e., in a complex layout there is a huge number of ways widgets can be rearranged. This makes it difficult to analyze all possible options and, more importantly, it is not clear how to identify "good" layout

alternatives from the huge set of possible alternatives. For example, Fig. 2 shows two possible landscape alternatives for a relatively simple portrait layout; while intuitively the bottom alternative is better than the top one, it is not directly clear how to automatically detect this. Here, we develop a novel classification method to identify "good" alternatives.

To address the question of how layout alternatives should be generated, we analyzed 815 existing mobile apps to identify common layout transformation patterns. Based on the findings we define transformation rules between layouts and apply these transformation rules systematically to a given layout to create appropriate alternatives. The generated alternatives are then ranked using a novel objective function, which is then used to provide a shortlist of suggestions to a GUI designer. In our survey of existing apps, we identified that landscape alternatives for portrait layouts contain the most changes, which motivated us to tackle this transformation first. In many cases, this transformation is the more appropriate option for the design process, as the default orientation of a mobile UI is portrait and the landscape alternative is typically added after [1,2]. However, our prototype also supports the generation of portrait alternatives for a landscape layout. In a questionnaire study, we found that our method is able to generate and identify landscape layout alternatives that appear to be well structured, easy to use, and that the automatic generation of layout alternatives can support GUI design.

Our contributions are:

1. An empirical study summarizing the use of layout alternatives in existing apps.
2. An automatic method to generate landscape and portrait layout alternatives.
3. An objective function to identify a set of "good" layout alternative candidates.
4. An evaluation of the approach.

2 Related Work

A variety of design guidelines and UI design patterns exist for mobile devices [3–5]. Nilsson [6] summarized design patterns for mobile UIs; some of these patterns focus on utilizing screen space, including the switch between portrait and landscape mode. Scarr et al. [7] compared the performance of three adaptation techniques for displaying a set of icons – scaling, scrolling and reflow – and illustrated the usability benefits of preserving spatial layout consistency. However, the mentioned work does not provide guidelines or patterns to adapt layouts to different screen orientations or conditions.

Adaptive layout classes can adapt a GUI to different screen sizes [8]. This technique is also used for web-based document layouts [9,10]. However, as discussed earlier, such adaptive layout classes are not always sufficient for more drastic aspect ratio changes where widgets need to be rearranged.

UI generators generate UIs automatically [11–14], and are typically based on UI models which are specified by designers, such as task models, abstract UI models and concrete UIs [15]. In this approach layout templates are filled with

widgets from a specification to form a concrete layout. This has also been used to generate multiple user interfaces from a single specification [16–18]. Similarly, documents with dynamic content can be rendered using multiple templates, selected based on constraints [19], or web pages can be transformed into layout templates [20]. Lee et al. [21] provide predefined layout examples and generate the final web document from a selected example. With this approach, the output layout is predefined by a designer while in our approach the layout is generated dynamically.

In our work we use an objective function to find an optimal layout alternative for the given screen conditions. This technique has also been used to improve the usability and aesthetics of UIs created in a sketching tool [22] as well as for automatic document formatting [23]. Sears [24] proposed an objective function for UI layouts which accumulates the expected time required to perform common tasks. Similarly, SUPPLE [25,26] adapts layouts to user's motor and vision capabilities by choosing optimal widgets. Our objective function is based on design patterns and designed to find "good" layouts using layout information *only* available at design time. The designer is then able to select an alternative layout from a list of possibly "good" layouts. SUPPLE also optimizes layout containers, e.g., the orientation of a linear layout, to make a layout fit to different screen sizes. Compared to SUPPLE, we perform more complex changes to the layout topology by re-grouping widgets and recursively applying a set of layout transformations.

Layout graph grammars, a.k.a. spatial graph grammars, have been proposed as a formal method for layout transformation. Brandenburg [27] described how context-free graph grammars can be extended with layout specifications to visualize a graph. Others extended this work and showed how context-sensitive graph grammars with layout specifications and actions can be used to describe certain adaptations of web pages [28,29]. However, transformations still have to be defined manually.

Web pages often need to be adapted for viewing on mobile devices. Roudaki et al. [30] summarized various approaches for this purpose. Qiu et al. [31] compared methods for adapting and rendering web pages on small display devices by breaking them down into a structural overview and detail views for content. Florins and Vanderdonckt [32] proposed rules for graceful degradation of UIs when they are used on more constrained devices. Yu and Kong [33] compared small-screen UI design patterns for news websites. In our work we target layout transformations for screen orientation changes where the screen area stays the same.

Personalized UIs have been proposed as a way to adapt UIs dynamically to different users and situations. Hinz et al. [34] proposed to mine personalized content adaptation rules from user interactions for the optimization of web content presentation on mobile devices. The Framework for Agile Media Experiences (FAME) [35] permits developers to define logical page models that are dynamically filled with content deemed appropriate for a certain user while satisfying given constraints. However, these approaches use predefined layout specifications to render the UI.

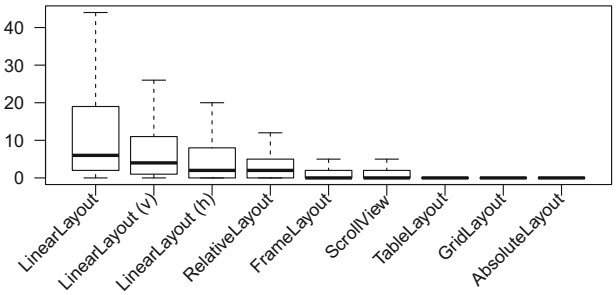

Fig. 3. Boxplot of number of layouts per application.

UI builders are development tools for specifying resizable UI layouts. Constraint-based layouts, such as Auto Layout[1], are often used to make a layout resizable [8,36–38]. Hottelier et al. [39] proposed a technique which allows designers to generalize and specialize constraint-based layout specifications using direct manipulation. Xu et al. proposed methods that make it easier for designers to beautify UIs by inferring layout constraints [40] and to arrange graphic elements with commands [41] based on the way they are visually grouped. The Auckland Layout Editor [42] maps simple drag&drop operations to linear layout constraints. However, little help is available when existing layouts need to be adapted to new requirements.

3 Analysis of Existing Layout Alternatives

To get an overview of current practices in layout adaptation, we surveyed existing layouts in real software projects, and analyzed how such layouts are adapted to different screen conditions. Also, we were interested in identifying commonly used adaptation patterns. To answer these questions we automatically analyzed the source code of all 815 (in 2014) available open source Android apps from the F-Droid project[2]. Android uses a standardized way to specify layout alternatives for different screen conditions in XML files, which provided the data for our study. Layouts can also be specified directly in Java code, but in our data set only 2% of the layouts were instantiated that way.

Usage of Layout Models. Our survey identifies layout models that are frequently used in Android layout specifications. This enables us to get a better grasp on the types of layout arrangements that need to be transformed in practice. Android offers many of the common layout models, such as: LinearLayout for arranging widgets vertically in a column or horizontally in a row; RelativeLayout, which is a form of constraint-based layout; FrameLayout, which usually contains only a single widget; and GridLayout, which arranges widgets in a grid. Figure 3 shows

[1] developer.apple.com/design/adaptivity.
[2] f-droid.org.

a boxplot of the usage frequencies for different layout models in the apps. LinearLayout is the most common model, and vertical LinearLayouts are somewhat more common than horizontal ones. The second most frequent class of layouts is RelativeLayouts, followed by the simple FrameLayout. Other models, such as TableLayout and GridLayout, are relatively uncommon. These results are consistent with the findings of Shirazi et al. [43]. ScrollView is not a layout model, but is usually used to make a layout resizable when screen estate is scarce.

Analysis of Layout Transformations. To analyze the relevance of layout alternatives, we surveyed how frequently developers specify such alternatives for their apps. Furthermore, we analyzed how much each layout alternative differs from the corresponding default layout. To measure this, we looked at the widgets that we were able to uniquely identify (by their ID) in a layout alternative as well as in the corresponding default layout. We counted how many of these widgets are at a different structural position ($\Delta_{position}$) in a layout alternative compared to the default. Widgets were considered to be at different positions if their paths to the root of the layout hierarchy, as given in the XML file, differed.

Among the 815 apps with 11,110 layouts we found 365 (3.2%) alternatives for landscape layouts and 397 (3.5%) alternatives for different layout sizes (247 (2.2%) alone for large[3] layout sizes). The low percentage of layout alternatives alone already supports the need for automatic layout alternative generation. We found that the layout alternatives for landscape contain on average more widgets at different positions than the alternatives for different sizes ($\Delta_{position} = 6.5$ vs. $\Delta_{position} = 3.6$).

To understand how a layout specification is typically adapted to different screen sizes and orientation, we manually analyzed 99 of the layout alternatives in detail (63 landscape and 36 other resizing alternatives). In particular, we selected those layout alternatives in which the widget positions changed significantly, i.e., with $\Delta_{position} \geq 10$, plus 15 randomly selected layouts with a smaller difference. Since we did not know what layout adaption techniques we would encounter, we first identified layout adaptation categories by manually analyzing the layouts. In a second step another researcher classified the layouts into these categories. We identified the following categories of transformations in the layouts:

1. **Appearance**: changes in fonts, text labels, image content, spacing or insets, widget size (but not arrangement), etc.
2. **Widgets**: widgets were added or removed.
3. **Horizontal Flow**: a row of widgets was broken into several adjacent rows, or adjacent rows were joined.
4. **Vertical Flow**: a column of widgets was broken into several adjacent columns, or adjacent columns were joined.
5. **Pivot**: a horizontal linear layout was transformed into a vertical one, or vice versa.
6. **Scrolling**: a scrollable container was inserted to deal with overflowing content.

[3] developer.android.com/guide/practices/screens_support.html.

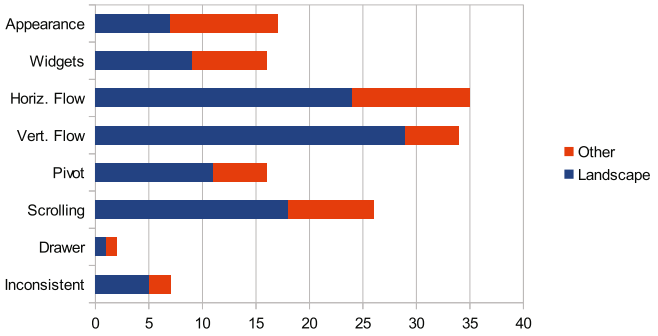

Fig. 4. Histogram of the identified transformation categories found in landscape layout alternatives and in alternatives for different screen sizes.

7. **Drawer**: widgets were moved to a hidden "drawer" container to make room.
8. **Inconsistent**: clear misalignment of widgets or unnecessary changes in widget order.

Figure 4 shows the distribution of categories for layout alternatives. Note that each alternative can be in multiple categories, e.g., a linear layout may have been pivoted as well as split into multiple columns. As the vertical space is reduced in landscape, the most common adaptation is unsurprisingly vertical flow. Scrolling is very frequent; it offers a simple – if not always user friendly – solution to content overflow. Overall, the results indicate that transformations of linear layouts, such as flow across columns and rows, as well as pivoting are frequently used for creating a landscape alternative.

4 Automatic Generation of Layouts

Building on the findings and identified transformation patterns from the previous section, we developed a tool that generates layout alternatives for a given layout. This is a challenging problem since the solution space grows exponentially with the number of changes applied to the initial layout. Moreover, defining an objective function that can identify "good" layout alternatives is demanding: the "goodness" of a layout is a complex construct that incorporates aesthetic and semantic criteria, which are hard to define (or even recognize) in a way that can be easily automated. As identified in the previous section, landscape layout alternatives contain the most layout changes. For this reason we first target layout transformations between portrait and landscape orientation. To address the complexity of the solution space, we focus on the most common transformations for landscape alternatives, i.e., flow and pivot (Fig. 4).

We treat the generation of good landscape alternatives as a constrained discrete optimization problem. We define a set of layout transformations and, starting with the initial layout, search the solution space by applying transformations repeatedly. To limit the search, we define and apply an objective function to

prioritize transformations that appear more promising. The objective function can only estimate the "goodness" of a layout as perceived by a UI designer, therefore the layout with the best objective value is not necessarily the best choice. We rank the generated layout alternatives by their objective value and allow UI designers to quickly browse and edit the top-ranked ones.

4.1 Describing Layout Transformations with Tiling Algebra

To describe layout specifications and their transformations concisely at a high abstraction level, we use an algebraic description called a *tiling algebra* [44]. Every layout is denoted by an algebraic term. Terms can be defined recursively as follows, based on two associative tiling operators | and /:

1. **Atoms**: The smallest terms of the algebra are individual widgets (i.e., layouts containing only a single widget). They are given capitalized names, e.g., *Button* or B.
2. **Horizontal Tiling** |: If a and b are terms representing layouts, then the term $a|b$ represents a horizontal layout in which layout a is arranged to the left of layout b.
3. **Vertical Tiling** /: If a and b are terms representing layouts, then the term a/b represents a vertical layout in which layout a is arranged on top of layout b.

Then, a horizontal linear layout with three widgets A, B and C is written as $A|B|C$.

Fragments: Terms in the tiling algebra can nest analogously to layouts, and we call the nested sub-layouts in a term *fragments*. This is similar to binary space partitioning layouts [45]. For example, $l_1 = (A|B)/(C|D)$ describes that a layout $l_2 = A|B$ is above a layout $l_3 = C|D$. l_2 and l_3 are fragments of l_1.

Named Tabstops: To describe more complex layouts, such as table and grid layouts, we introduce named tabstops, i.e., named grid lines between fragments. A named tabstop can occur multiple times in a fragment which allows us to describe alignments within a fragment. For example, the fragment of a simple 2×2 grid can be described using a named tabstop i: $(A\underset{i}{|}B)/(C\underset{i}{|}D)$. This means the tabstop i occurs in the first and the second row of the grid.

Levels: Terms can be described by abstract syntax trees (ASTs), where each inner node is a tiling operator and the leaf nodes are widgets. We call the fragments that are formed at a depth n of an AST, i.e., the fragments combined at the same nesting depth n, the nth *level* of the term. For example, consider the layout $l_1 = (A|B)/(C|D|(E/F))$: the first level contains only l_1 itself; the second one contains $A|B$ and $C|D|(E/F)$; the third level contains C, D and $E|F$; and the fourth one contains E and F.

The tiling algebra is well suited to describe LinearLayouts, FrameLayouts, TableLayouts and many GridLayouts. Transforming RelativeLayouts automatically to a tiling algebra is not always possible and non-trivial. Some complex,

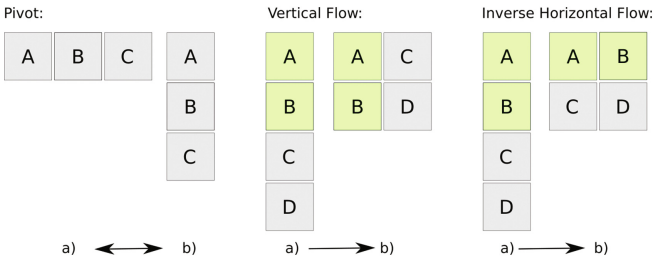

Fig. 5. Examples for the transformation rules.

interleaved grid layouts cannot be described using a single fragment and named tabstops. However, for the layouts analyzed in our survey, we were able to describe all GridLayouts and manually replace RelativeLayouts by layout classes that can be described using tiling algebra.

4.2 Transformation Rules

Based on the most common transformations identified in our survey of existing landscape alternatives, we define three basic transformation rules on fragments for the transformation from portrait to landscape: pivot, vertical flow and inverse horizontal flow. We define the inverse transformation rules for the transformation from landscape to portrait layouts analogously. Figure 5 visualizes the transformation rules.

We use the right-arrow symbol $\rightarrow$ to denote transformations from one layout to another. Note that the variables in the rules (e.g., a) represent fragments, i.e., they are not necessarily widgets but can possibly be broken down further.

Transformation rules for portrait to landscape:

Pivot: The pivot transformation changes the orientation of a fragment, i.e., the $|$ operator becomes $/$, and vice versa. For example, $I/C/L \rightarrow I|C|L$. The pivot transformation generalizes to table layouts, e.g., $(A \underset{i}{|} B)/(C \underset{i}{|} D) \rightarrow (A \underset{i}{/} B)|(C \underset{i}{/} D)$, i.e., the grid is transposed.

Vertical Flow: The vertical flow transformation breaks a column, i.e., vertical LinearLayout, into two adjacent ones:

$$a_1/\ldots/a_n \rightarrow (a_1/\ldots/a_k)|(a_{k+1}/\ldots/a_n).$$

The vertical flow transformation takes multiple break positions into account. For example, $I/C/L$ can be transformed to $(I|C)/L$ or alternatively to $I|(C/L)$.

Inverse Horizontal Flow: This transformation splits a column into multiple rows:

$$a_1/\ldots/a_n \rightarrow (a_1 \underset{m}{|} \ldots \underset{n}{|} a_i)/\ldots/(a_k \underset{m}{|} \ldots \underset{n}{|} a_n).$$

For example, $A/B/C/D/E \rightarrow (A \underset{m}{|} B)/(C \underset{m}{|} D)/E$. Similar to the vertical flow transformation multiple possibilities how rows are merged are taken into account, e.g.,
$A/B/C/D \rightarrow (A \underset{m}{|} B)/(C \underset{m}{|} D)$ or $(A|B|C)/D$.

The transformation rules for landscape to portrait are:

Pivot: Same as for portrait to landscape.

Inverse Vertical Flow: The inverse vertical flow merges two columns:

$$(a_1/\ldots/a_k)|(a_{k+1}/\ldots/a_n) \rightarrow a_1/\ldots/a_n.$$

Horizontal Flow: The horizontal flow merges multiple rows into a column:

$$(a_1|\ldots|a_i)/\ldots/(a_k|\ldots|a_n) \rightarrow a_1/\ldots/a_n.$$

4.3 Grouping of Fragments

In order to transform a layout meaningfully, it is useful to group logically related widgets together. Logically related widgets are sometimes already grouped in a layout specification, but this is not always the case. For example, consider a UI layout in which three labels L are logically paired with editable text fields E: the correct grouping may already be encoded in the layout specification with a separate nested layout for each of the pairs, i.e., $l_1 = (L/E)/(L/E)/(L/E)$, but the layout may also simply be defined in a single layout as $l_2 = L/E/L/E/L/E$. l_1 is easier to transform meaningfully, as related widgets stay together when applying the previously defined transformations. For example, a vertical flow transformation may split a pair of label and text field: $l_2 \rightarrow (L/E/L)|(E/L/E)$. However, a similar transformation on the groups of l_2 would leave the pairs intact: $l_1 \rightarrow ((L/E)/(L/E))|(L/E)$.

Our grouping approach is inspired by Gestalt principles [46]: it forms groups according to repeated patterns of fragments (principle of similarity) to make sure that the fragments in each group are transformed together (principle of common fate). To group the fragments in a linear layout, we (1) identify the longest repeated consecutive pattern of fragments in the layout, and (2) group all occurrences of the pattern into new fragments, i.e., linear sub-layouts. Step (1) is equivalent to the well-known longest repeated substring problem, which can be solved in linear time. For example, to group a layout $l_3 = a/L/L/E/L/E/L/L/E/L/E/b$ the algorithm (1) identifies $l_4 = (L/L/E/L/E)$ as the longest repeating consecutive pattern, and (2) groups the occurrences of the pattern so that $l_3 \rightarrow a/l_4/l_4/b$. The grouping process can be recursively repeated, i.e., we can look for groups in the fragments a, l_4 and b, which results in $l_4 \rightarrow L/l_5/l_5$ with $l_5 = (L/E)$. Sometimes there are multiple ways how items can be grouped in which case the grouping algorithm returns multiple solutions, e.g., $L|E|L|E|L$ can be grouped as $(L|E)|(L|E)|L$ or $L|(E|L)|(E|L)$.

It is possible that the grouping produced by this approach is wrong, i.e., does not match the logical relations in the UI. Therefore, this grouping is only considered as a possible fragment when applying transformations. By itself, the grouping does not change a layout. However, after grouping a fragment, transformations on that fragment are more likely to respect the aforementioned Gestalt principles.

4.4 Objective Function

We define an objective function $f(l)$ to estimate the quality of a layout specification l. Smaller values of l indicate better layouts. The function is a weighted sum:

$$\begin{aligned} f(l) = {} & w_{min} \cdot t_{min}(l) + w_{pref} \cdot t_{pref}(l) + w_{ratio} \cdot t_{ratio}(l) \\ & + w_{nTra} \cdot t_{nTra}(l) + w_{sym} \cdot t_{sym}(l) + w_{level} \cdot t_{level}(l) \end{aligned}$$

with t being quality terms that are calculated for a layout, and w being constant weights which are chosen empirically. This sum can be extended to incorporate additional criteria. In the following we briefly summarize the definitions of the quality terms and the heuristics behind them. The first three quality terms correspond directly to commonly-used penalty functions, e.g., in constraint-based UI layouts [8,47]. These terms take the minimum and preferred widget and layout sizes into account. Here, the preferred size is the natural size of a widget or a layout, i.e., the size a widget or a layout would obtain if their are no other constraints, such as the window size, forcing it to a different size. The other quality terms are motivated by research into UI aesthetics and usability [7,48,49]. We focused on terms that are well-motivated by previous work, but acknowledge that more terms, e.g., to account for alignment, could be added to the objective function.

Minimum Size: "A layout that can be shrunk to a compact size is likely to have a good structure [8,47]."

$$t_{min} = \frac{min_w^2 + min_h^2}{screen_w^2 + screen_h^2}$$

min_w, min_h, $screen_w$ and $screen_h$ are the width and height of the minimum and the screen size. The divisor normalizes the value to the screen size. If the layout in its minimum size does not fit on the screen, we assign t_{min} a very large value.

Preferred Size: "If the widgets of a layout have a size close to their preferred size, the space is well used [8,47]."

$$t_{pref} = \frac{\sum_i ((pref_{w,i} - size_{w,i})^2 + (pref_{h,i} - size_{h,i})^2)}{screen_w^2 + screen_h^2}$$

$pref_i$ and $size_i$ are the preferred and the actual size of each widget.

Aspect Ratio: "A good layout in its preferred size has an aspect ratio close to that of the target screen [8,47,50]."

$$t_{ratio} = |ratio_{pref} - ratio_{screen}|/ratio_{screen}$$

We constrain $t_{ratio} \leq 1$ to limit the impact of this term.

Number of Transformations: "A smaller number of transformations is preferable as this preserves the spatial consistency of the UI [7]." t_{nTra} is the number of transformations which was applied to derive the layout from the initial one. We constrain $t_{nTra} \leq 5$ to limit the impact of this term.

Symmetry: "Layouts with a high symmetry are preferable [48,49]." We chose a simple, screen-independent definition of symmetry based on the AST of a layout fragment: a fragment is symmetric if it consists of topologically equivalent parts. That is, layouts are symmetric if all the sub-layouts they contain are structurally equivalent and simple widgets are symmetric by definition. This can be understood intuitively through examples: $(A/B)|(C/D)|(E/F)$ is symmetric because A/B, C/D and E/F are structurally equivalent (vertical tilings of two widgets). The fragment $(A|B)/(C|D)/E$ is not symmetric because E is a single widget and not a horizontal tiling of two widgets as the other sub-fragments. Based on this notion,

$$t_{sym}(l) = 1 - \frac{s(l)}{widgetCount(l)^2 \cdot nLevels(l)}$$

with s defined recursively on a fragment $l = a_1/\ldots/a_n$ or $l = a_1|\ldots|a_n$ as

$$s(l) = \begin{cases} nWidgets(l)^2 \cdot nLevels(l) & \text{, if } l \text{ symmetric} \\ \sum_{i=1..n} s(a_i) & \text{, otherwise} \end{cases}$$

$s(l)$ measures the degree of symmetry in l: if l is symmetric, the value returned is higher with the size of l, i.e., with its number of widgets $nWidgets$ and levels $nLevels$. If l itself is not symmetric, the same logic is applied to its sub-layouts, summing up the value of each symmetric sub-layout found. The divisor ensures that $0 \leq t_{sym}(l) \leq 1$.

Transformation Level: "A transformation close to the root of the fragment hierarchy is better as this preserves the spatial consistency of larger sub-layouts [7]." t_{level} is the deepest level that has been changed by the transformations that derived the layout from the initial one. For example, consider an initial layout $l_1 = (A/(B/C))/(D/E)$ with three levels. If we transform $l_1 \rightarrow l_2 = (A/(B/C))|(D/E)$ (pivot of the first/on level 1) and then $l_2 \rightarrow l_3 = (A|(B/C))|(D/E)$ (pivot of/at level 2), then $t_{level}(l_3) = \max(1, 2) = 2$. We constrain $t_{level} \leq 5$ to limit the impact of this term.

4.5 Empirical Optimization of Parameters

The weights of the objective function are parameters of the layout generation approach. They should be chosen so that the objective function correlates with

the "goodness" of a layout. Initially, we had no data about the "goodness" of landscape layout alternatives, i.e., no ground truth. Thus, we had to optimize the weights based on our own subjective ratings of "goodness", facing the following two challenges:

1. **Precision**: It is hard for an individual to tell precisely how "good" a layout alternative should be rated.
2. **Local Optimization**: Adjusting the weights to create a more appropriate objective value for a specific layout can reduce the objective value for other layouts.

We addressed the precision challenge as follows: while it is hard to get precise ratings, it is usually possible for an experienced individual to recognize layout alternatives that are clearly unsuitable ("bad" layouts) and layout alternatives that could be suitable ("good" layouts). Using our automatic layout transformation approach, we generated a variety of landscape alternatives for a number of portrait layouts and then selected a set of "good" and "bad" layouts from them, based on subjective judgment. The weights were chosen in a way that maximizes the discriminatory power of the objective function, i.e., so that the "good" layouts had low and the "bad" layouts high objective values.

To address the challenge of local optimization, we used a form of linear regression based on quadratic programming. For each possible pair of a "good" landscape layout alternative l_{good} and a "bad" alternative l_{bad} for a given portrait layout, a soft constraint is added to a linear constraint system: $f(l_{bad}) - f(l_{good}) \geq 1$. This difference should be at least 1 for every such pair, so the layouts can be discriminated. Note that we do not add constraints for every possible pair of good and bad layout alternatives. We only add constraints for every possible pair derived from the same layout, as the layout generation approach compares only such alternatives with each other.

Overall, this yields a set of weights for the objective function. In the following table, the *man* row gives the values for the initial manual process and the *opt* row gives the results for the subsequent constraint optimization.

	w_{min}	w_{pref}	w_{ratio}	w_{nTra}	w_{sym}	w_{level}
man	0.2	10	0.4	0.04	1	0.2
opt	2.8	10	0.2	0.4	2.5	0.8

The values are scaled to $w_{pref} = 10$. Note that the weights of different objective function terms are not directly comparable as the terms are not normalized to each other.

4.6 Search Strategy

In order to find suitable landscape layouts, starting with the given layout, we search the space of possible layouts by repeatedly applying the transformation

rules. The solution space grows exponentially with the number of transformation steps: at each step, the number of transformations that can be applied is roughly linear to the layout size. The pivot rule can be applied to any fragment in a layout; the vertical flow and inverse horizontal flow rules can be applied to any vertical fragment, usually in more than one way; and grouping can be applied recursively on fragments containing repetitions.

As a result, a brute force search is not practical in many cases. To search the solution space more efficiently, we perform transformations on intermediate layouts with a good objective value. That is, we generally try to optimize a layout already estimated to be comparatively good before looking at worse layouts. Furthermore, in order to preserve symmetry in a layout (one of the quality terms of the objective function), we use a heuristic that favors a transformation to be consistently applied to a whole level of the layout. That is, we first apply the same transformation to all fragments of a chosen level: if the fragments on that level are similarly structured, i.e., if there is symmetry between them, then applying the same transformation to each of them is likely to preserve that symmetry. For example, applying the pivot transformation to the entire (symmetric) second level of $(A/B)|(A/B)$ yields a symmetric layout $(A|B)|(A|B)$.

However, using this search strategy, we found that calculating a sufficient number of good layout alternatives still takes up to several minutes. This is not acceptable for a tool that supports the interactive design process of a developer. A performance analysis revealed that most time was spent in the relatively complex calculation of the minimum and actual layout size needed for the objective function evaluation. To solve this problem we split the objective function into a part that is quick to calculate and a second that is slower: $f = f_{fast} + f_{slow}$ with $f_{fast} = w_{nTra} \cdot t_{nTra} + w_{sym} \cdot t_{sym} + w_{level} \cdot t_{level}$ and $f_{slow} = w_{min} \cdot t_{min} + w_{pref} \cdot t_{pref} + w_{ratio} \cdot t_{ratio}$. We found that there is a good correlation between f and f_{fast}, i.e., for the analyzed layouts we found that the six best layouts chosen by f were among the 23 best layouts chosen by f_{fast}. This allows us to perform a coarse search using f_{fast} followed by a detailed search on the found layouts using the full objective function f. With this approach we can perform the search for good layout alternatives in under a second on a i5-4300U CPU at 1.90 GHz.

5 Implementation

We implemented the layout alternative generation approach as a plugin for the Android Studio development environment, the official development environment for Android, which has an integrated design editor for XML layout files[4]. With our plugin the user can generate landscape and portrait layout alternatives from a layout XML file. As shown in Fig. 6, the user can quickly browse the top-ranked layout alternatives and edit a chosen alternative if desired. Internally, the plugin transforms the input into a constraint-based layout [8,47] in order to calculate the size properties required for the objective function.

[4] The source code is available on: gitlab.com/czeidler/layoutalternatives.

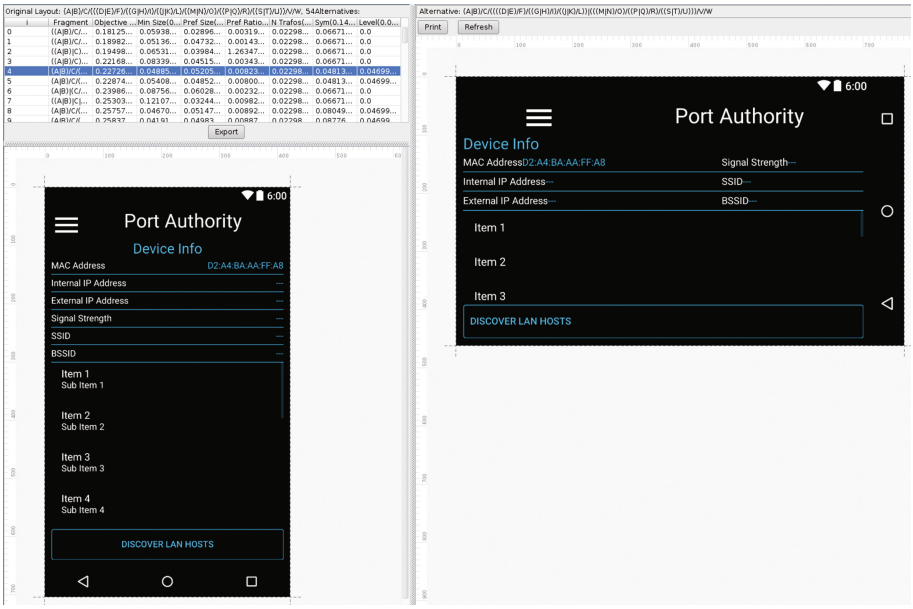

Fig. 6. Screenshot of the implementation: a list of the layout alternatives with the best objective values (top-left), the original portrait layout (bottom-left), and the selected landscape layout alternative (right).

6 Limitations

In the following we discuss limitations of our approach. In our analysis of existing layouts (Sect. 3) we focused on Android and it is not clear if our results can be generalized to other platforms, e.g., iOS. However, the main Android layout classes are similar to those on other mobile platforms, and mobile design patterns are comparable across platforms [5]. With a fairly large number of analyzed layouts we believe we captured a representative picture of existing transformation patterns and that our results are transferable to different platforms.

Currently, our prototype does not handle dynamic layouts, e.g., layouts that change on user interaction. While it is possible that our prototype could handle multiple intermediate steps of a dynamic layout, such a feature is currently difficult to implement since dynamic behavior is not specified in the layout XML files.

To limit the solution space we focused on a small set of transformations (Sect. 4). This makes it sometimes difficult or even impossible to generate some rarer layout alternatives, e.g., multiple transformations may have to be applied to achieve a certain result. To solve this problem more specialized transformations need to be introduced.

7 Evaluation

We evaluated how well our objective function is able to identify "good" layout alternatives using a questionnaire study. Based on our analysis of existing layout alternatives in real-world apps, we selected 15 portrait layouts with a landscape alternative that changed significantly, covering different app genres and UI styles. Then we generated a ranking of landscape layout alternatives with the best objective values for each of the 15 portrait layouts. Finally, the quality of the resulting landscape layouts was assessed. In a survey we asked participants to rate the layout alternatives.

7.1 Pilot Validation

Before conducting the questionnaire study, we tried to validate our approach through the following hypothesis:

H1 Layout alternatives comparable to the original landscape layouts from the app developer are ranked highly by the objective function.

To answer H1 we searched among the generated layout alternatives for layouts that are comparable to the original landscape layouts. We judged a layout alternative as comparable if it follows the general layout topology of the original, i.e., if the correct layout transformations had been applied. Here, minor details that our transformations cannot achieve, e.g., font size changes or widget alignment, were ignored. For all 15 layouts we were able to identify layout alternatives that were comparable to the original landscape layout. Figure 7 shows that most of these layout alternatives were ranked among the top three candidates. That is, they were among the top three generated layouts according to their objective values, and therefore near the top of the list of alternatives in our prototype. This supports H1, i.e., that the prototype generates the expected layout and that the layout is ranked highly by the objective function.

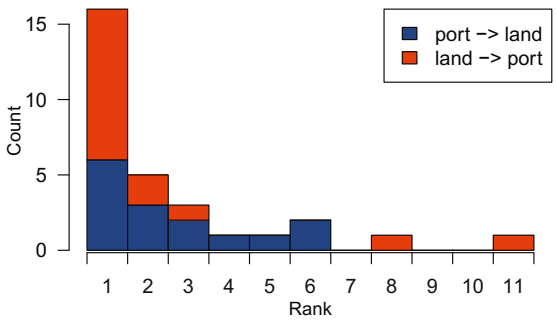

Fig. 7. Automatic ranking of the layout alternatives which are comparable to the original landscape and portrait layouts created by the app developer. Our objective function ranks the original layout alternatives highly.

For further validation we performed the inverse test, i.e., generated portrait layout alternatives from the generated landscape alternatives. We were able to transform all landscape layouts back to their original portrait ones. Furthermore, the layout alternative matching the original portrait layout was also ranked highly (Fig. 7). Interestingly, the portrait alternative matching the original portrait layout were generally ranked higher than the landscape alternative matching the original landscape layout. An explanation for this is that the transformation rules for landscape to portrait are simpler than the inverse rules. For example, while there are generally multiple ways to break a column into two columns using the vertical flow transformation, there is only one way to merge two columns.

7.2 Methodology

In our questionnaire study we aimed to verify the following hypotheses:

H2 The layout alternatives that are ranked highly contain acceptable layouts.
H3 Such layouts are useful as a starting point for designing landscape alternatives.

We used a web questionnaire to evaluate the six best layout alternatives, i.e., those with the smallest objective values, generated for each of the 15 portrait layouts. Note that the six best layout alternatives always contained the *original landscape alternative* (see Fig. 7). After questions about demographics and previous experience with UI design, the 15 portrait layouts and their generated landscape alternatives were shown in randomized order. For each portrait layout, a participant was first asked to rate the portrait layout itself and then each of the landscape layout alternatives in randomized order. To facilitate comparisons, we showed each layout alternative next to the original portrait layout. For each portrait layout and each layout alternative we asked the following 5-point Likert-scale questions (which are loosely based on [51]), ranging from strongly disagree to strongly agree:

Q1 The portrait/landscape layout appears well structured.
Q2 The portrait/landscape layout appears easy to use.
Q3 There are problems with this portrait/landscape layout.

After each of the 15 layouts, i.e., the portrait layout and its six landscape alternatives, we asked the following questions, again using the 5-point Likert scale:

Q4 Some of the proposed layout alternatives were as I expected them to be.
Q5 The proposed layout alternatives were missing one or more layouts that I would have expected.
Q6 The proposed layout alternatives contained good layouts that were unexpected.
Q7 The proposed layout alternatives gave me a good overview of how the layout can be transformed to landscape.

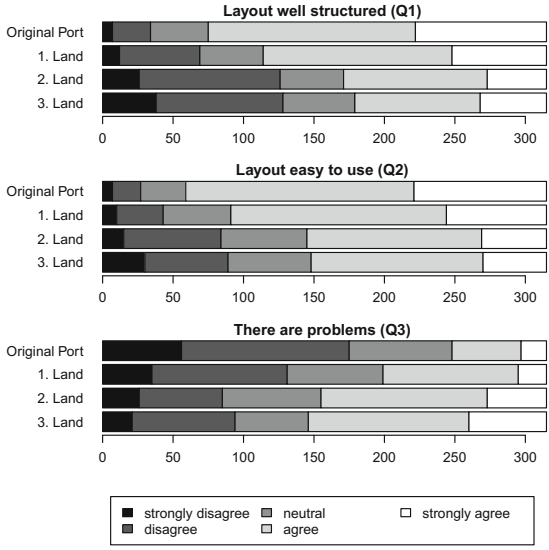

Fig. 8. Answers to Q1, Q2, Q3 for the original portrait and the three best automatically ranked alternatives. For 15 layouts, 21 participants gave $21 \cdot 15 = 315$ answers for the original layout and its alternatives.

Q8 The proposed layout alternatives contained a good starting point for the design of a landscape layout alternative.

At the end of the questionnaire the participants were asked to answer the following 5-point Likert-scale questions:

Q9 As a layout designer I would like to use a tool that proposes layout alternatives for different screen conditions, e.g., landscape alternative for a given portrait layout.

Q10 I do not think a tool that proposes layout alternatives would be useful for layout designers.

Finally, we gave participants the option to leave free-text comments.

7.3 Results and Discussion

21 participants (9 female, 12 male, average age 31) responded to our ad and filled the questionnaire. According to the responses, 13 had UI design experience, seven were familiar with UI layout design tools, and eight had designed layout alternatives for different screen conditions. The participants were a mix of professionals working in IT, designers, and computer science students. In total, $15 \times 7 \times 21 = 2205$ quality ratings for $15 \times 7 = 105$ layouts were submitted (15 apps with $(6 + 1)$ layouts each).

To validate H2 we consider the ratings of the best generated layout alternatives in comparison to that of the original portrait layout. In Fig. 8 we can

see that layout alternatives ranked higher by the objective function generally also got better ratings. The first and second best alternatives received a positive rating by more than half of the participants. The third alternative received a borderline result. The fact that two layouts received a positive rating by the majority can be interpreted as a form of resilience of the tool output; we can expect more than one proposed layout to be acceptable by many. Overall, these results are a good support for hypothesis H2.

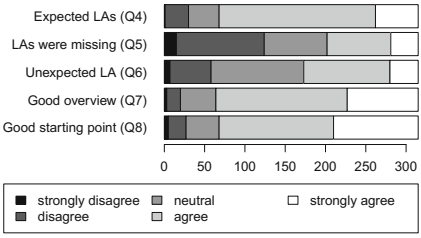

Fig. 9. Combined answers to Q7-Q11 for all layouts.

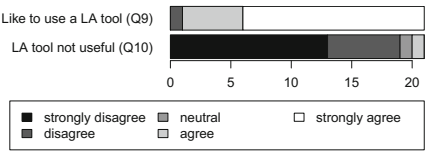

Fig. 10. Questions Q9-Q10 at the end of the questionnaire.

To validate H3 we looked in particular at the answers to the summary questions about the layout alternatives for each of the portrait layouts (Fig. 9). Q8 addresses H3 most directly and was answered strongly in the affirmative, supporting acceptance of H3. Q7 addresses more the added value of the whole set of layout alternatives and the positive result strengthens H3 further. For Q5, agreement signals potential shortcomings of the tool, but less than half the responses agree, so this does not seem to be a limiting factor. For Q6, a majority of positive answers is not necessarily expected, and the positive responses indicate benefits for at least some of the users. If a sizable fraction of users finds unexpected layouts of reasonable quality, this would mean that the tool can provide a useful service designing new layouts through stimulating creativity. In that sense, Q6 adds an important additional dimension to the evaluation of H3. Q4 complements Q6 to some degree and addresses whether the tool is able to obtain reasonable layouts automatically. The answers show that the tool appears predictable to the users in that it delivers at least some expected layouts. We can therefore accept H3 in two ways: first, the tool eases the UI design work as seen in the answers to Q4 and Q8, and second, the tool might also provide new ideas, as indicated in the answers to Q6. Figure 10 shows that there was a strong agreement among the participants that a tool that proposes layout alternatives would generally be useful. This was also backed by comments from the participants. For example: "A tool that automatically displays alternatives for layout designs is extremely useful." or "Some of the layouts were really good, especially ones that are aligned symmetrically."

Study Limitations. One threat to validity of the questionnaire results is acquiescence bias, which could lead to alternatives being rated overly positive.

We addressed this threat by asking for three quality measures (Q1-Q3), which allowed participants to express a differentiated judgment. Furthermore, agreement to Q3 signals a negative judgment. The fact that higher ranked alternatives got better ratings even though the order in the survey was randomized (Fig. 8) speaks against a strong acquiescence bias. Similarly, a variety of questions (Q4-Q8) was used to investigate H3 and to obtain a differentiated understanding of possible benefits.

Another threat to validity is a possible selection bias in the layouts chosen for the user study. The population of layouts is necessarily a very heterogeneous space, and we tried to mitigate such a bias by basing our selection of layouts on the layout survey described earlier, which looked at a larger set of apps and layouts.

A minor issue is that for technical reasons, some custom widgets in layouts were not rendered correctly, i.e., only as gray boxes containing the widget name. We told the participants to expect this and judge the layouts imagining that the gray boxes were appropriate widgets. Still, this might have had a slight negative effect on ratings.

8 Conclusion

In this paper we investigated how layout alternatives for alternative screen orientations can be generated automatically. To identify suitable layout transformation patterns we analyzed layouts from 815 existing Android apps. Based on this analysis, we designed a set of rules to transform layouts and presented a new way to computationally estimate the quality of a transformed layout. Using this quality measure as objective function, we automatically generated landscape and portrait layout alternatives for a given layout. Finally, we validated the utility of our implementation and whether the objective function is able to identify "good" layouts in a questionnaire study. We found that layout alternatives ranked highly by the objective function were rated well by the participants. Generated layout alternatives appeared to be well structured and easy to use. Moreover, our method generated good layouts that were not anticipated by the participants, which means our prototype can actively help developers to design new GUIs.

In future work, we plan to identify other transformation rules, e.g., rules which apply visual appearance changes or change the layout more drastically. Yet, to improve the accuracy of the corresponding objective function, we need to simultaneously investigate new quality terms, such as a term that encapsulates additional Gestalt principles. Also, we plan to improve the quality of the layout alternatives by learning from the alternative choices designers made for their layouts while using our system. Finally, transformations between different screen sizes, e.g., tablet to phone, should be supported.

References

1. Morris, J.: Android User Interface Development: Beginner's Guide. Packt Publishing, Birmingham (2011)
2. Sahami Shirazi, A., Henze, N., Dingler, T., Kunze, K., Schmidt, A.: Upright or sideways?: analysis of smartphone postures in the wild. In: Proceedings of 15th International Conference on HCI with Mobile Devices and Services, pp. 362–371 (2013)
3. Adipat, B., Zhang, D.: Interface design for mobile applications. In: AMCIS 2005 Proceedings, p. 494 (2005)
4. Tidwell, J.: Designing Interfaces: Patterns for Effective Interaction Design. O'Reilly Media Inc., Sebastopol (2010)
5. Neil, T.: Mobile Design Pattern Gallery: UI Patterns for Smartphone Apps. O'Reilly Media Inc., Sebastopol (2014)
6. Nilsson, E.G.: Design patterns for user interface for mobile applications. Adv. Eng. Softw. **40**(12), 1318–1328 (2009). Designing, modelling and implementing interactive systems
7. Scarr, J., Cockburn, A., Gutwin, C., Malacria, S.: Testing the robustness and performance of spatially consistent interfaces. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 3139–3148 (2013)
8. Zeidler, C., Lutteroth, C., Weber, G.: Constraint solving for beautiful user interfaces: how solving strategies support layout aesthetics. In: Proceedings of the 13th International Conference of the NZ Chapter of the ACM's Special Interest Group on HCI, CHINZ 2012, pp. 72–79 (2012)
9. Frain, B.: Responsive web design with HTML5 and CSS3 (2012)
10. Nebeling, M., Matulic, F., Streit, L., Norrie, M.C.: Adaptive layout template for effective web content presentation in large-screen contexts. In: Proceedings of the 11th ACM Symposium on Document Engineering, DocEng 2011, pp. 219–228 (2011)
11. Raneburger, D., Popp, R., Vanderdonckt, J.: An automated layout approach for model-driven wimp-ui generation. In: Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2012, pp. 91–100 (2012)
12. Yanagida, T., Nonaka, H., Kurihara, M.: Personalizing graphical user interfaces on flexible widget layout. In: Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2009, pp. 255–264 (2009)
13. Kim, W.C., Foley, J.D.: Providing high-level control and expert assistance in the user interface presentation design. In: Proceedings of the INTERACT 1993 and CHI 1993 Conference on Human Factors in Computing Systems, pp. 430–437 (1993)
14. Talton, J., Yang, L., Kumar, R., Lim, M., Goodman, N., Měch, R.: Learning design patterns with bayesian grammar induction. In: Proceedings of the 25th ACM Symposium on User Interface Software and Technology, pp. 63–74 (2012)
15. Paterno, F., Santoro, C.: One model, many interfaces. In: Kolski, C., Vanderdonckt, J. (eds.) Computer-Aided Design of User interfaces III, pp. 143–154. Springer, Dordrecht (2002). doi:10.1007/978-94-010-0421-3_13
16. Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R., Pignol, M.: Generating remote control interfaces for complex appliances. In: Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology, UIST 2002, pp. 161–170 (2002)

17. Nichols, J., Rothrock, B., Chau, D.H., Myers, B.A.: Huddle: automatically generating interfaces for systems of multiple connected appliances. In: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology, UIST 2006, pp. 279–288 (2006)
18. Nichols, J., Chau, D.H., Myers, B.A.: Demonstrating the viability of automatically generated user interfaces. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2007, pp. 1283–1292 (2007)
19. Schrier, E., Dontcheva, M., Jacobs, C., Wade, G., Salesin, D.: Adaptive layout for dynamically aggregated documents. In: Proceedings of the 13th International Conference on Intelligent User Interfaces, IUI 2008, pp. 99–108 (2008)
20. Kumar, R., Talton, J.O., Ahmad, S., Klemmer, S.R.: Bricolage: example-based retargeting for web design. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2011, pp. 2197–2206 (2011)
21. Lee, B., Srivastava, S., Kumar, R., Brafman, R., Klemmer, S.R.: Designing with interactive example galleries. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2010, pp. 2257–2266 (2010)
22. Todi, K., Weir, D., Oulasvirta, A.: Sketchplore: sketch and explore with a layout optimiser. In: Proceedings of the 2016 ACM Conference on Designing Interactive Systems, pp. 543–555 (2016)
23. Hurst, N., Li, W., Marriott, K.: Review of automatic document formatting. In: Proceedings of the 9th ACM Symposium on Document Engineering, pp. 99–108 (2009)
24. Sears, A.: Layout appropriateness: a metric for evaluating user interface widget layout. IEEE Trans. Softw. Eng. **19**(7), 707–719 (1993)
25. Gajos, K., Weld, D.S.: SUPPLE: automatically generating user interfaces. In: Proceedings of the 9th International Conference on Intelligent User Interfaces, IUI 2004, pp. 93–100 (2004)
26. Gajos, K.Z., Wobbrock, J.O., Weld, D.S.: Automatically generating user interfaces adapted to users' motor and vision capabilities. In: Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology, pp. 231–240 (2007)
27. Brandenburg, F.J.: Designing graph drawings by layout graph grammars. In: Tamassia, R., Tollis, I.G. (eds.) GD 1994. LNCS, vol. 894, pp. 416–427. Springer, Heidelberg (1995). doi:10.1007/3-540-58950-3_395
28. Zhang, K., Kong, J., Qiu, M., Song, G.L.: Multimedia layout adaptation through grammatical specifications. Multimedia Syst. **10**(3), 245–260 (2005)
29. Kong, J., Zhang, K., Zeng, X.: Spatial graph grammars for graphical user interfaces. ACM Trans. Comput.-Hum. Interact. **13**(2), 268–307 (2006)
30. Roudaki, A., Kong, J., Yu, N.: A classification of web browsing on mobile devices. J. Vis. Lang. Comput. **26**, 82–98 (2015)
31. Qiu, M.K., Zhang, K., Huang, M.: An empirical study of web interface design on small display devices. In: Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2004, pp. 29–35 (2004)
32. Florins, M., Vanderdonckt, J.: Graceful degradation of user interfaces as a design method for multiplatform systems. In: IUI 2004, vol. 4, pp. 140–147 (2004)
33. Yu, N., Kong, J.: User experience with web browsing on small screens: experimental investigations of mobile-page interface design and homepage design for news websites. Inf. Sci. **330**, 427–443 (2016). sI: Visual Info Communication
34. Hinz, M., Fiala, Z., Wehner, F.: Personalization-based optimization of web interfaces for mobile devices. In: Brewster, S., Dunlop, M. (eds.) Mobile HCI 2004. LNCS, vol. 3160, pp. 204–215. Springer, Heidelberg (2004). doi:10.1007/978-3-540-28637-0_18

35. Lempel, R., Barenboim, R., Bortnikov, E., Golbandi, N., Kagian, A., Katzir, L., Makabee, H., Roy, S., Somekh, O.: Hierarchical composable optimization of web pages. In: Proceedings of the 21st International Conference on World Wide Web, pp. 53–62 (2012)
36. Zeidler, C., Müller, J., Lutteroth, C., Weber, G.: Comparing the usability of grid-bag and constraint-based layouts. In: Proceedings of the 24th Australian Computer-Human Interaction Conference, OzCHI 2012, pp. 674–682 (2012)
37. Lok, S., Feiner, S.: A survey of automated layout techniques for information presentations. Proc. SmartGraphics **2001**, 61–68 (2001)
38. Borning, A., Marriott, K., Stuckey, P., Xiao, Y.: Solving linear arithmetic constraints for user interface applications. In: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology, UIST 1997, pp. 87–96 (1997)
39. Hottelier, T., Bodik, R., Ryokai, K.: Programming by manipulation for layout. In: Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology, UIST 2014, pp. 231–241 (2014)
40. Xu, P., Fu, H., Igarashi, T., Tai, C.L.: Global beautification of layouts with interactive ambiguity resolution. In: Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology, UIST 2014, pp. 243–252 (2014)
41. Xu, P., Fu, H., Tai, C.L., Igarashi, T.: Gaca: group-aware command-based arrangement of graphic elements. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, pp. 2787–2795 (2015)
42. Zeidler, C., Lutteroth, C., Sturzlinger, W., Weber, G.: The Auckland Layout Editor: an improved GUI layout specification process. In: Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology, pp. 343–352 (2013)
43. Sahami Shirazi, A., Henze, N., Schmidt, A., Goldberg, R., Schmidt, B., Schmauder, H.: Insights into layout patterns of mobile user interfaces by an automatic analysis of Android apps. In: Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 275–284 (2013)
44. Zeidler, C., Weber, G., Gavryushkin, A., Lutteroth, C.: Tiling algebra for constraint-based layout editing. J. Log. Algebr. Methods Program. **89**, 67–94 (2017)
45. Hertzog, P.: Binary space partitioning layouts to help build better information dashboards. In: Proceedings of the 20th International Conference on Intelligent User Interfaces, IUI 2015, pp. 138–147 (2015)
46. Köhler, W.: Gestalt Psychology: An Introduction to New Concepts in Modern Psychology. Liveright Publishing Corporation, New York (1947)
47. Lutteroth, C., Strandh, R., Weber, G.: Domain specific high-level constraints for user interface layout. Constraints **13**(3), 307–342 (2008)
48. Lavie, T., Tractinsky, N.: Assessing dimensions of perceived visual aesthetics of web sites. Int. J. Hum. Comput. Stud. **60**(3), 269–298 (2004)
49. Ngo, D.C.L., Teo, L.S., Byrne, J.G.: Modelling interface aesthetics. Inf. Sci. **152**, 25–46 (2003)
50. Nebeling, M., Matulic, F., Norrie, M.C.: Metrics for the evaluation of news site content layout in large-screen contexts. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2011, pp. 1511–1520 (2011)
51. Moshagen, M., Thielsch, M.T.: Facets of visual aesthetics. Int. J. Hum. Comput. Stud. **68**(10), 689–709 (2010)