Comparing the Usability of Grid-bag and Constraint-based Layouts

Clemens Zeidler, Johannes Müller, Christof Lutteroth, Gerald Weber
Department of Computer Science
University of Auckland
Private Bag 92019, Auckland
New Zealand
{czei002, jmue933}@aucklanduni.ac.nz and {lutteroth, gerald}@cs.auckland.ac.nz

ABSTRACT

While the usability of GUI design methods has been studied in general, the usability of layout specification methods is largely unexplored.

In this paper we provide an empirical comparison of two popular GUI layout models, grid-bag layout and constraint-based layout. While the grid-bag layout is a powerful layout model, the constraint-based layout is able to generate even more general and flexible layout configurations. We performed a controlled experiment with postgraduate students of Computer Science and Software Engineering, measuring efficiency, accuracy and preference for typical layout specification and editing tasks.

The results show significant differences between both layout models: the initial specification of GUIs is faster with a grid-bag layout whereas editing of existing complex layouts is faster and more accurate with a constraint-based layout. The study shows that constraint-based layout, although it may seem more complicated at first glance, can compete with and in some cases even outperform more conventional techniques in terms of their usability.

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: Evaluation/methodology

General Terms

Experimentation, Measurement, Performance

Keywords

Layout, API, Usability, Empiric Evaluation

1. INTRODUCTION

Developing and maintaining Graphical User Interfaces (GUIs) is complex and time consuming, therefore it is important that developers have tools that are easy to use. In early Graphical User Interface (GUI) toolkits the developer had to position widgets manually at fixed positions with fixed widths and heights. This was not only tedious but also inflexible. It was difficult to create resizeable GUIs that look good at different layout sizes, and also layout editing was complicated because generally many widgets had to be repositioned after a change, e.g. after a new widget was inserted. This also caused usability problems for end-users: GUIs were less likely to be resizable and more likely to contain layout errors.

Layout managers are a solution to these problems as they automate the layout of widgets in a GUI. Anything that can be placed into a layout is called a *layout item*. Editing a layout specification can be done on a higher abstraction level which makes it unnecessary to handle low level details such as moving layout items around. Layout managers can support various layout models, e.g. the 1-dimensional group layout model[1], the grid-bag layout model [23] or the constraint-based layout model [14], to mention some of the most popular ones.

Note that the existence of GUI Builders does not make the study of layout models less relevant. Although there are graphical GUI builders [15] to support the design of GUIs, many developers still prefer to specify a layout programmatically. This may have various reasons, apart from the personal developer preferences, for instance: no suitable GUI builder for the used toolkit or platform is available, used layout items are not supported by the GUI builder or layout specifications are changing at runtime. Furthermore, GUI builders are still using certain layout models and it is usually necessary for the developer to understand these models when using a GUI builder.

A frequently applied type of layout model is the grid-bag layout model. In this layout model layout items are organized in the cells of a grid [14]. This approach allows the definition of a large class of layouts by aligning items in the grid. However, this simplicity precludes the specification of layouts with more complex dependencies between widgets. For example, it is not possible to specify the size of a layout item as a multiple of another item's size.

Such dependencies can be specified with a constraint-based layout by applying linear constraints. The class of possible grid-bag layouts is a subset of the class of possible constraint-based layouts [21], therefore the constraint-based layout is the more powerful layout model.

Constraint-based layout models have been studied for quite a while in the research community [1, 11, 14, 18, 22] and recently attracted a lot of attention because of a newly introduced constraint-based layout model in the Cocoa API of Apple's Mac OS X[2].

OZCHI '12, November 26-30, 2012, Melbourne, Victoria, Australia

[1] See for example: Qt – a cross-platform application and UI framework, 2012 `http://qt.nokia.com/products/`

[2] Cocoa Auto Layout Guide, 2012 `http://developer.apple.com`

However, even though the constraint-based layout model is more powerful than the grid-bag layout model, this does not mean it is more usable. So far it is unclear whether either of them has any advantages in terms of usability. Hence, in this paper we study the question how the constraint-based and the grid-bag layout models differ in terms of usability. Specifically, we study the following research questions:

- How does the usability of the grid-bag and the constraint-based layout models differ for specifying layouts?
- How does the usability of the grid-bag and the constraint-based layout models differ for editing layouts?

Here we analyze just the bare layout models and do not look at nested layouts.

To answer these questions we conducted a controlled experiment with postgraduate students of Computer Science and Software Engineering. To operationalize the abstract term *usability* for our study, we investigate it as the efficiency and accuracy with which a task can be completed and the preferences users have [10]. The efficiency is measured as task completion time, the accuracy as the number of errors made, and the preference with standardized items on a Likert-scale. The experiment was paper-based to abstract from concrete API implementations and directly compare the underlying layout models. Participants performed several specification and editing tasks on layouts.

From our user evaluation we found that the grid-bag layout model is more efficient than the constraint-based layout model when specifying a layout from scratch. When it comes to editing an existing layout that is reasonably complex, the constraint-based layout model is significantly faster. With regard to the accuracy, the constraint-based layout model results in less errors when specifying and editing layouts. This becomes significant when editing more complex layouts. Overall, the participants preferred the constraint-based layout model, especially for editing a layout.

Section 2 gives an overview over the grid-bag layout model and the constraint-based layout model. Furthermore, this section describes the complexity for editing layouts using these layout models. Related work is discussed in Section 3. Section 4 describes the methodology of our study. The results are presented in Section 5. Section 6 discusses the results and analyses threats to validity. The conclusions are summarized in Section 7.

2. GRID-BAG AND CONSTRAINT-BASED LAYOUTS

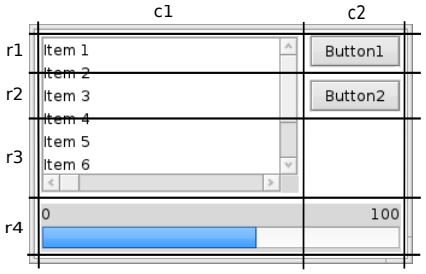
In the following section we discuss how a layout can be specified using the grid-bag layout model and the constraint-based layout model. During the design process it is sometimes necessary to edit an already existing layout specification. This can, for example, happen if an additional button has to be inserted into a layout or the size of an item needs to be adapted. Editing a layout can have a different complexity for different layout models. To quantify this *edit complexity* we use the following simple definition. The edit complexity of a layout is the number of layout items that have to be updated during an edit operation for one single change.

2.1 Grid-Bag Layout

One of the most prominent layout models is the grid-bag layout model. This layout model is for example used in HTML tables[3] and almost all available GUI toolkits support this layout model.

In a grid-bag layout a layout item can be placed in a 2-dimensional grid. Beside the row and the column that specifies where an item is placed, a *row-span* and *column-span* specifies how many rows and columns a layout item occupies. Using row- and column-spans for one item makes it possible to create complex layouts. Furthermore, it is possible to create a link between items not directly adjacent, e.g. by placing them in the same row or column. Figure 1 shows an example for a grid-bag layout. Here all layout items occupy one or more cells in the grid.

The grid-bag layout can be tuned by giving the rows and columns weightings, for specifying which row and column should use more space compared to the other rows and columns.



	Row	Column	Row-Span	Colum-Span
Item 1 Item 2	r1	c1	3	1
Button1	r1	c2	1	1
Button2	r2	c2	1	1
0 100	r4	c1	1	2

Figure 1: Layout using grid-bag layout specifications.

Editing a Grid-bag Layout.

Whenever an edit operation causes the change of the row and column number of a grid-bag layout, other items in the layout might be affected and need to be updated. There are two cases that make this update necessary. First, items to the right or further down from the inserted layout item get a higher row or column value. Secondly, the row- or column-span of items which are intersecting in the vertical or horizontal direction have to be updated. Figure 2 illustrates these two cases, i.e. after the insertion of Button 2 the column number of Button 1 changes from 1 to 2 and the column-span of the progress bar changes from 1 to 2. The simple case of an insertion into an empty cell does not change the row and column number and so does not effect any other layout items.

2.2 Constraint-Based Layout

In the constraint-based layout model, the layout specifications are described by constraints using linear equalities and inequalities. Constraint-based layouts have attracted much attention in research and industry in recent years [1, 17, 14]. In the following the nomenclature of the constraint-based layout model ALM (Auckland Layout Model) [14] is used. However, in general any constraint-based layout model such as the Java SpringLayout,[4] is suitable for the further usability study. It only has to provide an easy way to connect borders of layout items together.

In ALM, the variables in constraints are called *tabstops*, which represent horizontal or vertical boundaries of layout items. Other frequently used names for the same concept are aligners, snap lines,

[3]D. Raggett. RFC1942: HTML Tables, 1996.

[4]Spring Layout API documentation, 2012 `http://docs.oracle.com/javase`

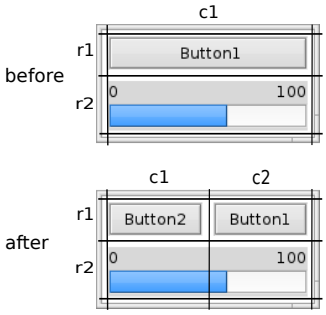
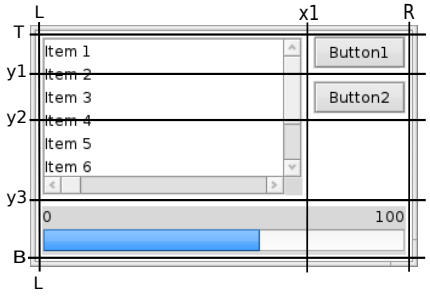


Figure 2: Grid-bag layout: Button 2 is inserted left beside Button 1. The existing Button 1 is moved from c1 to c2 and the column-span of the progress bar changes from 1 to 2.

guides, or anchor lines. The edges of each layout item are connected to two horizontal (top and bottom) and two vertical (left and right) tabstops. Through this, items can be aligned by connecting them to the same tabstops. The rectangular layout area, i.e. panel or window, has four tabstops that are connected to the layout borders to facilitate alignment with the layout boundaries.

A simple constraint-based layout is shown in Figure 3. For example, to make sure that the two buttons stay on the right of the list view the left side of the buttons share the same tabstop $x1$.



	Left	Top	Right	Bottom
Item 1 Item 2	L	T	x1	y3
Button1	x1	T	R	y1
Button2	x1	y1	R	y2
0 100	L	y3	R	B

Figure 3: Layout using constraint-based layout specifications.

Editing a Constraint-Based Layout.

While changes in a grid-bag layout can effect items anywhere in the layout, in a constraint-based layout changes are more local. This means only layout items which have to be connected to a newly added tabstop have to be updated. In the example in Figure 4 a list box is added between a text view and three buttons. Here, a new tabstop $x2$ is connected to the right of the text view and only this item has to be updated. The three buttons on the right do not need to be updated. From this example, one can see how many layout items have to be updated depends on where the new tabstop is inserted, i.e. if $x2$ had been inserted on the right side of the list view and $x1$ stayed at the right side of the text view all three buttons would need to be updated.

To specify the position of a layout item uniquely, the layout item has to be connected to at least one horizontal and one vertical tabstop which are directly or indirectly connected to a layout border tabstop. Indirectly connected to a layout border tabstop means that there are some constraints that set a relative position between the tabstop and a border tabstop, which is necessary to specify the layout in a unique way. For example, if a layout item is not connected to any layout border tabstop, it is not clear where the item should be positioned within the layout; all positions are valid.

A layout specification is well defined if all layout items have to be directly or indirectly connected to at least one horizontal and one vertical layout border tabstop. Well-definedness limits the edit complexity of a constraint-based layout. Usually a designer places a layout item either beside another layout item or between two layout items, or adjacent to a window border. In this way at least one horizontal and one vertical border of the layout item is connected to an existing tabstop, e.g. if a new button should be aligned to the top-right of another layout item, it is connected to the top and right tabstop of this item. Figure 4 shows an example where only a single tabstop ($x2$) has to be inserted. This limits the edit complexity to the number of layout items, which have to be connected to a maximum of two new tabstops.

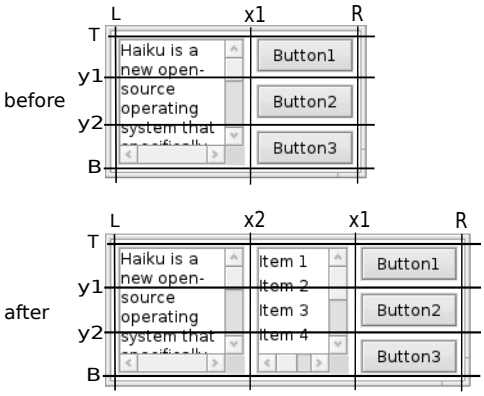


Figure 4: Constraint-based layout: The list view is inserted between the text view on the left and the three buttons on the right side. After the insertion, only the right of the text view has to be connected to the new tabstop $x2$.

3. RELATED WORK

In our study we are interested in the differences of the usability of two layout models: the grid-bag layout model and the constraint-based layout model. An often applied research method in Software Engineering to identify such effects are controlled experiments [2]. In a systematic literature survey Sjøberg et al. identified a total of 1.9 % of published papers in leading Software Engineering venues that conducted controlled experiments [19]. However, to the best of our knowledge none of them investigates GUI layout.

Within the HCI community, a research stream of API usability evaluation is emerging, which argues that for programmers the API of frameworks, Software Development Kits (SDKs) and libraries are a user interface to computers. It has therefore to be designed according to some usability criteria [16, 4, 6]. Usability studies for APIs were, for example, conducted in the domain of Service Oriented Architectures [3], or specifically for the use of language

constructs such as names [5], patterns [7] or the content of API documentation [12].

There is also some work in the program comprehension community related to API usability. Hou and Li conducted a case study about problems programmers have with the use of the Java Swing API [8]. They conducted an empirical evaluation of newsgroup posts and identified a set of API obstacles.

In the domain of GUI APIs little research has been done about usability so far. An experimental study about the usability of notations for XAML and Windows Forms was conducted by Kosar et al. [13]. Their experiment mainly focused on the understanding of the notations of the APIs and their usability. They did not analyze the usability of layout models of both frameworks. Their study also differs from our study in the methodology. In our experiments, participants actively specify and change layouts. In their study, participants were only ask to evaluate existing GUI specifications in XAML and Windows Forms. Hence they only measured accuracy but not efficiency as we do.

No work exists that directly compares the usability of layout models. Even though industry begins to adapt the constraint-based layout model from an empirical point of view, it is still unclear whether this model is more usable than well established models such as the grid-bag layout model.

4. METHODOLOGY

We first conducted a pilot study to better understand the effects the constraint-based layout model has on usability. With this experience we formulated concrete research hypotheses, for which we designed a main study.

4.1 Pilot Study

Based on our own experience with the constraint-based and grid-bag layout models, we expected that specifying or editing layouts with a constraint-based layout is faster and less error prone. When specifying a layout item in a grid-bag layout, the correct row and column-span has to be counted. We expected that if the row- or column-span of a layout item is high this should take more time and should be less accurate. Moreover, when editing a constraint-based layout, just the surrounding tabstops have to be considered, whereas in the grid-bag layout the whole grid has to be checked for changes (see Section 2).

To investigate this assumption we designed a pilot study that was primarily focused on layout specification and only secondarily on layout editing. The participants had to perform 12 tasks once for the grid-bag layout model and once for the constraint-based layout model. Half of these tasks where training tasks similar to the main tasks. From the six main tasks, four tasks were to specify a layout and two tasks were to edit two of the previously specified layouts. The tasks were quite simple with only a few layout items to be specified, and only marginal changes in the layout specification were required when editing a layout. The evaluation was paper-based to reduce programming-related influences such as the factor of different programming skills, and to focus the study on the different underlying concepts of both layout models.

However, after a few participants performed the tasks it became clear that the usability of aconstraint-based layout is not better with respect to all usability parameters, contrary to our initial assumption. First, the measured task completion times were clearly in favor of a grid-bag layout. Secondly, it became clear that larger row- or column-span values are not a real problem when specifying a layout item; in most cases the values seemed to be trivial to determine.

4.2 Hypotheses

Based on the experience gained in the pilot study, we were able to formulate research hypotheses about the differences between both layout models, as potential answers to our research questions. The hypotheses are as following:

H1 The grid-bag layout model is faster for layout specification.

H2 The constraint-based layout model is faster for layout editing.

H3 The constraint-based layout model is less error-prone for layout specification.

H4 The constraint-based layout model is less error-prone for layout editing.

H5 The constraint-based layout model is perceived as being easier to use.

To test these hypotheses we refined the design of our pilot study for the main study.

4.3 Main Study

In our pilot study we noticed that our tasks were too easy and focused too much on layout specification to reliably discriminate between both layout models. Therefore, for our main study we increased the layout complexity of all tasks and included more layout editing tasks.

The experiment had a within-subject design [9], which means that each participant had to work with the constraint-based layout and the grid-bag layout. Each participant had to conduct the same three tasks using a constraint-based layout and a grid-bag layout.

For each layout model, first some background was given and then the technique was explained. The participants then had to perform a training task in which they were allowed to ask clarification questions. The explanations of the experimenter were supported by a printed one page how-to, which remained as a reference for the participant during the whole experiment.

Afterwards two tasks followed for which the task completion time was measured by an experimenter. The experiment closed with a post questionnaire. To minimize order bias, the participants alternately started either with the constraint-based layout or the grid-bag layout. The design of the study is depicted in Figure 5.

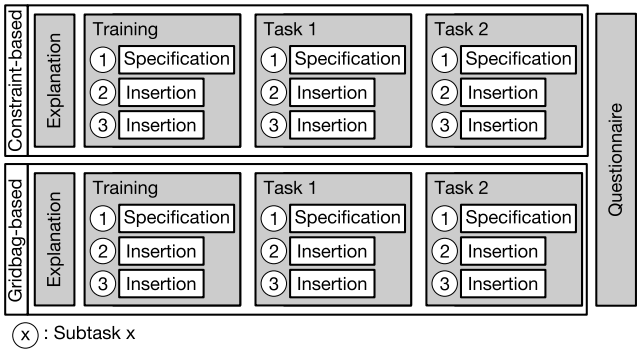


Figure 5: Sketch experimental design of the main study

The tasks were subdivided into three subtasks. In the following we refer to a certain subtask as Task.X.Y, where X is the number of the main task and Y the number of the subtask. The first subtask of each task was to specify a given layout with the means of the respective layout model. The layout was presented as a printed

screenshot. The specification had to be done on a sheet of paper with a table for all layout elements, similar to the tables shown in Figures 1 and 3. The layout items in the table were displayed in an iconized form to make it easy for the participants to identify them.

The following two subtasks were about editing the initial layout specification according to changes of the layout presented in a new screenshot. In the table the new layout item had an extra row which was then uncovered. In this way the participant was able to specify the new item and update the specification of the existing items in the same table. This mimics a real design process where a developer has done the initial layout specification and then edits this specification. To distinguish between the specifications of the different editing subtasks the participants used pens in different colors for each subtask. For each subtask the time that the participant needed to specify or edit a layout was measured.

The participants were asked to complete each subtask in two separate steps. First they had to become familiar with the layout they had to specify. That was done by sketching and labeling either, in the case of grid-bag layout, a suitable grid into the given screenshot or, in case of constraint-based layout, the required tabs. In the second step they were asked to derive relevant parameters (e.g. the rows and columns of the layout items) from the layout and write them into a provided table.

In the following we describe the experimental material (tasks and questionnaire) and how we conducted the experiment.

4.3.1 *Tasks and Questionnaire*

The training task was similar to the main tasks. It was designed to be sufficiently complex and to cover all interesting pitfalls the participants could run into while doing the main tasks, e.g. the training layout contained an empty area where it might be unclear where a tabstop or a row or a column should be inserted (see Figure 6 (a)).

Task 1 (Figure 6 (b)) was designed as a relatively simple task. The first insertion was a text view in the middle and the second insertion was a "Button" between two existing buttons. For the first insertion subtask, the edit complexity for the constraint-based layout was two or three, depending on where the new tabstop was added (see Section 2.2). For the second inserting subtask the edit complexity was two. The edit complexity for the grid-bag layout was four for the first insertion subtask and five for the second insertion subtask.

Task 2 (Figure 6 (c)) was more complex than the previous one. It mimicked a window of a chat application. The editing subtasks were designed to be part of a possible development process. The first edit subtask was the insertion of a "video call" button beside an existing "audio call" button. This subtask had an edit complexity of one for the constraint-based layout and an edit complexity of six for the grid-bag layout. In the second insertion subtask, another button had to be added with an edit complexity of two for the constraint-based layout and of eight for the grid-bag layout respectively. This means that the second task, while still relative simple, was more complex than the first task.

The questionnaire contained three demographics questions (gender, age, occupation), eleven 5-point Likert-scale questions with predefined answers and one open-ended question. The Likert-scale questions were:

- Q1 I often use computers in my everyday life.
- Q2 I understood the task and was able to perform it.
- Q3 It was easy to specify and edit the layouts using grid-bag layout.
- Q4 It was easy to specify and edit the layouts using constraint-based layout.

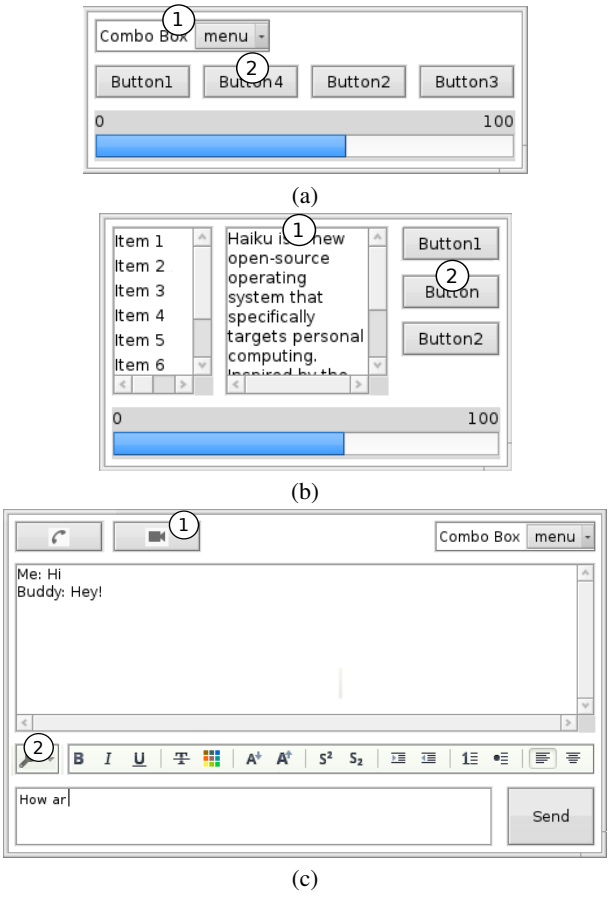


Figure 6: Task layouts after the second insertion subtask. The first and second inserted layout items are marked with a "1" and "2" respectively. (a) Training Task (b) Task 1 (c) Task 2

- Q5 I have experience with designing user interfaces.
- Q6 I have used or known grid-bag layout before, e.g. HTML tables.
- Q7 I have used or known constraint-based layout before.
- Q8 As a programmer I would like to use constraint-based layout in the future.
- Q9 As a programmer I would like to use grid-bag layout in the future.
- Q10 Specifying and editing a layout using constraint-based layout was difficult.
- Q11 Specifying and editing a layout using grid-bag layout was difficult.

Each experimental session took approximately one hour and took place in a lab where we closed door and blinds to minimize disturbance from external sources. We had two experimenters. To minimize the experimenter bias we decided to do the first four experiments with both experimenters. One of them was the main experimenter who interacts with the participant. The other one observes the main experimenter and also measured the time.

5. RESULTS

We recruited 12 participants who were students of Computer Science at least at Masters level. The sample contains three female and nine male participants. Figure 7 summarizes the experience level of the participants. All participants were heavy computer users and

most of them had experience with GUI design. Most of them were familiar with grid-bag layout, whereas more than a half of the sample were not familiar with constraint-based layout. They all understood the tasks they were asked to do.

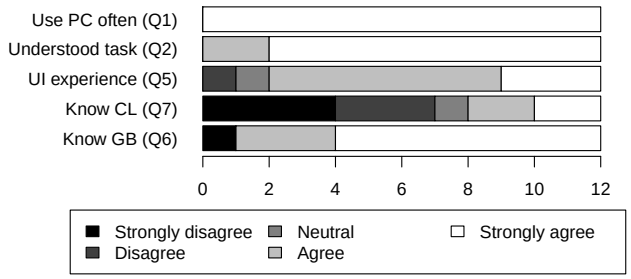


Figure 7: Frequencies of experience related answers

5.1 Efficiency

To compare the efficiency for the specification and editing tasks, we calculate the completion time difference

$$\Delta Time = t_{constraint} - t_{grid}$$

for each participant. It turned out that all the completion times of all tasks were likely not normally distributed. Hence, we applied a Wilcoxon rank sum test to calculate the significance of our hypotheses. Table 1 depicts the mean of $\Delta Time$, the standard deviation σ of $\Delta Time$ and the Wilcoxon test statistic p_{wrs} value.

Table 1: Test statistics for H1 and H2

	$\overline{\Delta Time}$	σ	p_{wrs}
Task 1.1 (H1)	12.50 *	22.97	0.05
Task 1.2 (H2)	-5.42	19.25	0.24
Task 1.3 (H2)	-8.25	36.65	0.47
Task 2.1 (H1)	16.50 **	16.00	<0.01
Task 2.2 (H2)	-25.00 **	25.28	<0.01
Task 2.3 (H2)	-37.00 **	28.46	<0.01

Our first hypothesis (**H1**) is that specifying layouts with the grid-bag layout model is faster than with the constraint-based layout model. Figure 8 depicts the boxplots of the task completion times for the specification tasks (CL means constraint-based layout and GB means grid-bag layout). For both tasks the task completion

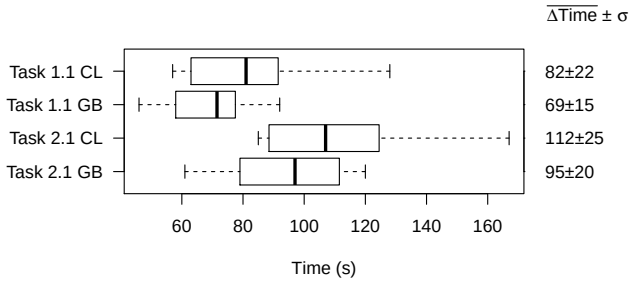


Figure 8: Boxplots of the task completion times for the specification tasks

time was higher for the constraint-based layout model. From the resulting p_{wrs}-values for Tasks 1.1 and 2.1 (Table 1) we can confirm the **H1** hypothesis. These results match our experience from the pilot study.

Beside the task completion time for layout specification, we are also interested in the effect of the constraint-based layout model on the task completion time for editing tasks. Our hypothesis **H2** is that the constraint-based layout model allow faster layout editing. However, from the measured completion times shown in Figure 9 this is not quite clear. Task 1.2 seems to have been completed

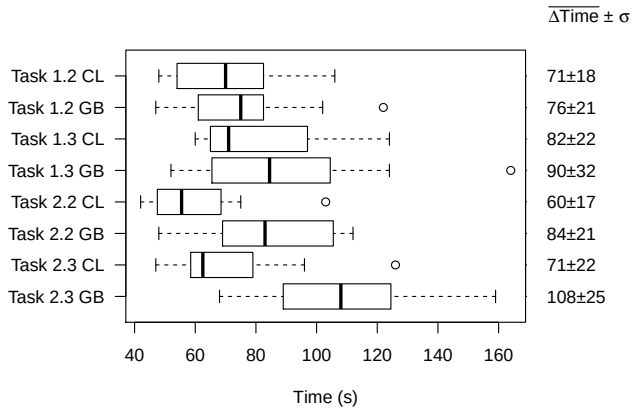


Figure 9: Boxplots of the task completion times for the editing tasks

faster with the grid-bag layout model, but all other tasks seem to have been completed faster when using the constraint-based layout model. Table 1 shows the results for editing tasks. The tests show that Tasks 2.2 and 2.3 were completed faster with the constraint-based layout model on a 0.1 % significance level. The results for Task 1.2 and 1.3 are less clear, here the null hypothesis can only be rejected with less statistical significance. The more complex Tasks 2.2 and 2.3 support **H2** whereas this is not so clear for the easier Tasks 1.2 and 1.3.

5.2 Accuracy

The accuracy is another important factor which indicates how usable a layout model is. For both, the specification and the editing task, we counted each item that has not been specified correctly as one error. This means multiple wrong values for a single layout item, e.g. wrong row and wrong column span, are counted just as one error. For the editing subtasks, values specified wrongly in the preceding specification or editing task are not counted as mistakes for the current subtask.

The accuracy is high if the number of layout specification errors E in a certain task is low. We measured the accuracy for the specification and the editing tasks to test **H3** and **H4**. Similar to the efficiency, the difference between the error rate of the constraint-based layout model and the grid-bag layout model $\Delta E = E_{constraint} - E_{grid}$ is calculate and a Wilcoxon rank sum test is applied (Table 2).

Hypothesis **H3** says that when specifying layouts using the constraint-based layout model the error rate is lower. This hypothesis can be tested using the observations from Task 1.1 and 2.1. The boxplots in Figure 10 give no clear picture. The specification subtask of Task 1 contains more errors for the constraint-based layout, but that changes for Task 2 where the grid-bag layout model seems to generate more errors. From Table 2 it can be seen that the p_{wrs}-value of 0.07 is quite low but still not significant. Therefore, we

Table 2: Test statistics for H3 and H4

	$\overline{\Delta E}$	σ	p_{wrs}
Task 1.1 (H3)	0.08	0.67	0.72
Task 1.2 (H4)	-0.42	0.79	0.09
Task 1.3 (H4)	-0.25	0.97	0.20
Task 2.1 (H3)	-0.25	0.45	0.07
Task 2.2 (H4)	-0.33	0.65	0.09
Task 2.3 (H4)	-1.08 *	1.68	0.03

cannot significantly confirm **H3** but the data indicates that the hypothesis is true.

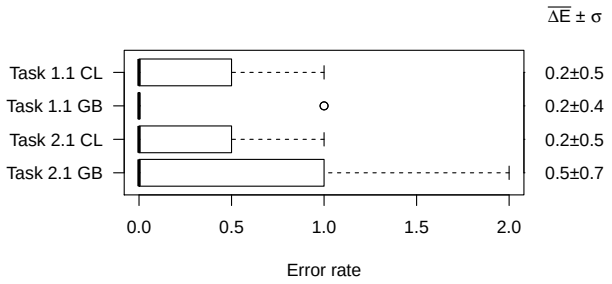


Figure 10: Boxplots of the error rates for the specification tasks

Hypothesis **H4** claims that the error rate is lower when editing a constraint-based layout than a grid-bag layout. We test this hypothesis with the observed errors of the editing tasks (Figure 11). Here, the boxplots give a rather clear picture. The error rate seems generally higher for the grid-bag layout model. However, conducted statistical tests in Table 2 confirm this assumption only for the most complex editing Task 2.3. Here we can reject the null hypothesis to a 5 % significance level and can accept **H4**.

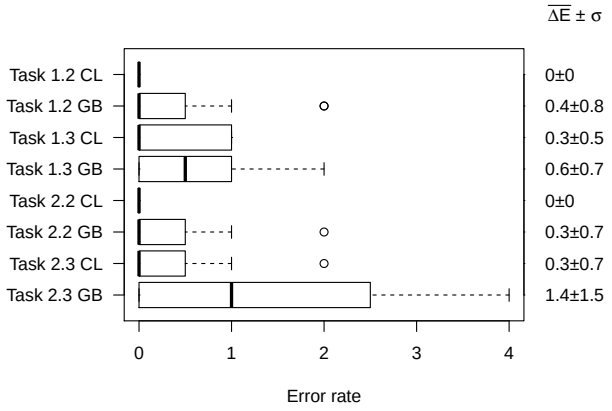


Figure 11: Boxplots of the error rates for the editing tasks

5.3 Preference

With hypotheses **H1** – **H4** we studied the actual behavior of the participants. With hypothesis **H5** we try to understand their subjective impression of the usability of the constraint-based layout model. For that we asked several questions for which the frequencies of the answers are depicted in Figure 12.

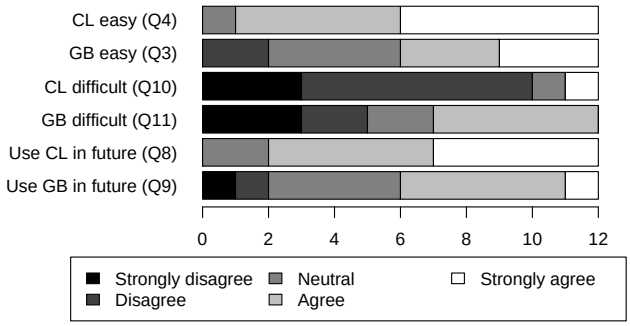


Figure 12: Frequencies of answers to impression questions

To test **H5** we included two question pairs ("is easy" and "is difficult") into the questionnaire. The results of a Wilcoxon rank sum test are depicted in Table 3. Here L is the value from the Likert scale which goes from -2 for strongly disagree to 2 for strongly agree. According to these results, for the first question pair, we can reject the null hypothesis on a significance level of 1 % and for the second question pair on 5 % significance level. These results are also backed up by many comments from the participants who said that editing a constraint-based layout is much easier and faster than editing a grid-bag layout.

Table 3: Test statistics for H5

	$\overline{\Delta L}$	σ	p_{wrs}
Q3 - Q4	-0.83 **	0.94	0.01
Q11 - Q10	0.67 *	1.15	0.04

6. DISCUSSION

For the effectiveness the data shows that specification is faster for the grid-bag layout model. For Task 1.1 it was on average 12.5s and for Task 2.1 it was on average 16.5s faster (Table 1). There are two possible explanation for that behavior. First, when specifying a constraint-based layout for all layout items, the four surrounding tabstops have to been found. Similar to that, a layout item in a grid-bag layout is specified by the four values column, row, row-span and column-span. However, in a grid-bag layout items often have a small row- and column-span which makes it very easy to determine the span values, e.g. a layout item in a single cell has a row- and column-span of one. Secondly, the grid-bag layout model is more familiar to the developers (Figure 7) and thus they need less time to perform this task.

However, for the editing tasks the picture changes. For the easier tasks the constraint-based layout was slightly, but not significantly, faster. When the tasks became more complex (Task 2.2 and 2.3) editing a layout was clearly faster in the constraint-based layout model. The main explanation for that observation is that the edit complexity for the constraint-based layout remained almost constant whereas the edit complexity for the grid-bag layout increased. This is mainly because changes in a constraint-based layout are local: changing a layout item only affects the surrounding layout items, while for the grid-bag layout in the worst case the whole layout has to be updated (Section 2).

The measured accuracy showed a similar behavior. For Task 1 the results are not clearly in favor of the constraint-based layout

model, but for the more complex Task 2 the error rates for specification and editing are dropping considerable. For Task 2.3 the error rate was significantly lower. Again, the main reason we identified is that the edit complexity is smaller for a constraint-based layout than for a grid-bag layout.

The objectively measured data for efficiency and accuracy matches the subjective impressions the respondents reported in the questionnaire. The participants perceived the constraint-based layout model as easier to use. This is interesting because most of the participants had known the grid-bag layout model before. Even in a short training session the participants became comfortable with the new layout model and experienced its advantages.

The results of the experiment suggest that the constraint-based layout model can compete with well-established layout models such as the grid-bag layout model, and can outperform them for more complex editing tasks. These interpretations have to be seen in the light of some possible threats to their validity.

Internal Validity

Since we applied a within-subject experimental design, we could face an order bias. To minimize this possible threat, we alternated the order in which the layout models were used.

Another threat could be that because we only used two main tasks for the evaluation not all interesting layout configurations are covered. Since the experiments took already about an hour we were not able to introduce another task. Since both tasks show the same pattern (specification is slow with constraint-based, editing is fast) we assume that the tasks capture the general effects and are hence adequate to study the differences in usability of both layout models. Furthermore, we designed the subtasks in such a way that they became successively more complex, hence covering a range of different complexities.

Many of the participants already knew the grid-bag layout model. That could have introduced an advantage for the grid-bag layout model. We minimized this threat by abstracting from a concrete API implementation to a paper-based experiment and with an extensive training phase.

A final internal threat could be social desirability bias which could stem from the fact that some participants knew about our work in the field of constraint-based layout models. They could therefore have been positively biased in the questionnaire answers about the constraint-based layout model. We minimized this threat by including these questions twice but with other formulations in the questionnaire. Even though this measure can help reducing the bias, we have to be aware of this problem in the interpretation of the results.

External Validity

The study poses some threats to its external validity. First, it can be an issue that our abstraction from concrete API implementations did not reflect the usability of a real-world API. However, the objective of our study was to understand the differences in the usability of the layout models. A specialization to a concrete implementation is not intended and may in fact endanger the generalizability of the results. Nevertheless, it can be assumed that differences in usability of layout models translate to similar differences in usability of well designed APIs that implement these layout models.

Another threat is the selection of the sample which consisted solely of students. However, we only selected students at least at master's level. For them we can assume sufficient knowledge in Computer Science to consider them as software developers. This assumption is also supported by a recent study about the representativeness of students for professional software developers [20].

Finally, the task design is an issue. The tasks were rather small compared to real-world GUI layouts.However, we designed the tasks in such a way that they reflect typical problems of real-world development situations (e.g. empty cells in a layout or introducing new layout items). Layouts in practice may replicate these problems several times, but the cognitive process to handle them is the same. We believe therefore that we can extrapolate our results to such more complex cases.

7. CONCLUSION

Two of the most versatile layout models are the grid-bag layout model and the constraint-based layout model. While the grid-bag layout is quite popular, the constraint-based layout is lesser known although it is the more general and flexible layout model. We conducted an experimental study comparing the usability of these two layout models. The usability was measured considering the factors efficiency, accuracy and preference. These factors where measured in various tasks, which included the specification of new layouts and the editing of existing layouts with different layout and edit complexities.

We found that it is faster to specify new layouts with the grid-bag layout model. However, when it comes to editing the previously specified layouts, the constraint-based layout model outperforms the grid-bag layout model. Especially for more complex layouts, constraint-based layout is significantly faster and has a significantly higher accuracy. From the questionnaire, the participants preferred the constraint-based layout and, consistent with the experiment, found it much easier for editing a layout.

The results of this study are quite promising and are clearly in favor of the more powerful constraint-based layout model. A next step would be to test the usability on widely used API implementations. For that purpose, even more complex programming tasks could be designed.

Our empirical study substantiates the effort of the industry to implement constraint-based layout models in their GUI APIs (e.g. Cocoa's Auto Layout or Java's SpringLayout). This model has clear advantages over common layout models, and it is worth to give it a try. Only time will tell if the constraint-based layout model will be widely adopted — at least it has the potential.

8. REFERENCES

[1] Greg J. Badros, Alan Borning, and Peter J. Stuckey. The Cassowary linear arithmetic constraint solving algorithm. *ACM Transactions on Computer-human Interaction*, 8:267–306, 2001.

[2] Victor R. Basili, Forrest Shull, and Filippo Lanubile. Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering*, 25:456–473, 1999.

[3] Jack K. Beaton, Brad A. Myers, Jeffrey Stylos, Sae Young (Sophie) Jeong, and Yingyu (Clare) Xie. Usability evaluation for enterprise soa apis. In *Proceedings of the 2nd international workshop on Systems development in SOA environments*, SDSOA '08, pages 29–34, New York, NY, USA, 2008. ACM.

[4] Steven Clarke. Usability of software tools impacts developer efficiency. *Dr. Dobbs*, 2004.

[5] J.M. Daughtry. The style and substance of api names. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*, pages 259 –260, sept. 2010.

[6] John M. Daughtry, Umer Farooq, Brad A. Myers, and Jeffrey Stylos. Api usability: report on special interest group at chi.

SIGSOFT Software Engineering Notes, 34(4):27–29, July 2009.

[7] Brian Ellis, Jeffrey Stylos, and Brad Myers. The factory pattern in api design: A usability evaluation. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 302–312, Washington, DC, USA, 2007. IEEE Computer Society.

[8] Daqing Hou and Lin Li. Obstacles in using frameworks and apis: An exploratory study of programmers' newsgroup discussions. In *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ICPC '11, pages 91–100, Washington, DC, USA, 2011. IEEE Computer Society.

[9] D.C. Howell. *Statistical Methods for Psychology*. Psy 613 Qualitative Research and Analysis in Psychology. Cengage Learning, 2012.

[10] ISO 9241-11:1998. *Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability*. ISO, Geneva, Switzerland.

[11] Noreen Jamil, Johannes Müller, Christof Lutteroth, and Gerald Weber. Extending linear relaxation for user interface layout. In *IEEE 24rd International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, 2012. to appear.

[12] Andrew J. Ko and Yann Riche. The role of conceptual knowledge in api usability. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 173–176, 2011.

[13] Tomaž Kosar, Nuno Oliveira, Marjan Mernik, Maria João Varanda Pereira, Matej Crepinsek, Daniela Carneiro da Cruz, and Pedro Rangel Henriques. Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems*, 7(2):247–264, 2010.

[14] Christof Lutteroth, Robert Strandh, and Gerald Weber. Domain specific High-Level constraints for user interface layout. *Constraints*, 13(3), 2008.

[15] Brad Myers, Scott E. Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Transactions on Computer-human Interaction*, 7:3–28, March 2000.

[16] Mary Beth Rosson. Human factors in programming and software development. *ACM Computing Surveys*, 28(1):193–195, March 1996.

[17] Veit Schwartze, Sebastian Feuerstack, and Sahin Albayrak. Behavior-sensitive user interfaces for smart environments. In Vincent Duffy, editor, *Digital Human Modeling*, volume 5620 of *Lecture Notes in Computer Science*, pages 305–314. Springer Berlin / Heidelberg, 2009.

[18] Adriano Scoditti and Wolfgang Stuerzlinger. A new layout method for graphical user interfaces. In *Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference*, pages 642–647. IEEE, 2009.

[19] Dag I. K. Sjøberg, Jo Erskine Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, Nils-Kristian Liborg, and Anette C. Rekdal. A Survey of Controlled Experiments in Software Engineering. *IEEE Transactions on Software Engineering*, 31:733–753, 2005.

[20] Mikael Svahnberg, Aybüke Aurum, and Claes Wohlin. Using students as subjects - an empirical evaluation. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, pages 288–290, New York, NY, USA, 2008. ACM.

[21] Gerald Weber. A Reduction of Grid-Bag Layout to Auckland Layout. In *Australian Software Engineering Conference*, pages 67–74, 2010.

[22] Clemens Zeidler, Christof Lutteroth, and Gerald Weber. 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 Human-Computer Interaction*, CHINZ '12, pages 72–79, New York, NY, USA, 2012. ACM.

[23] John Zukowski. *Java AWT reference*. O'Reilly & Associates, Inc., 1997.