PC・スマートフォン・タブレット対応のアプリケーション作成を支援する GUI ビルダー

A GUI Builder for Supporting the Creation of Applications for PCs, Smartphones, and Tablets

江幡　真吾
Shingo Ebata
法政大学情報科学部ディジタルメディア学科
E-mail: shingo.ebata.8d@stu.hosei.ac.jp

Abstract

As smartphones and tablets become more and more popular around the world, it is becoming more and more common to create multi-platform applications for PCs, smartphones, and tablets. Existing GUI builders make it easy to create simple applications for devices with different aspect ratios. However, when creating complex GUI layouts where the positions or sizes of widgets are not the same, or where multiple widgets are aligned on the same row, existing GUI builders are limited in automatically supporting devices with different aspect ratios. If the automatically created layout is not appropriate, the developer needs to rearrange it by him/herself, which is time-consuming and burdensome. This paper proposes a GUI builder that supports the placement of widgets by solving constraints and setting placement order and widget priority. It determines how to deal with overflowing widgets when changing to layouts with different aspect ratios. To evaluate this GUI builder, the paper compares the placement of widgets in various layout patterns. The paper also presents the result of a user study in which the participants specified parameters for the positions and sizes of the widgets by considering placement order and widget priority by themselves.

1. はじめに

現在，スマートフォンやタブレットが世界中で普及し，PC・スマートフォン・タブレットに対応するマルチプラットフォームのアプリケーションを作成することが多くなっている．主に作成されているアプリケーションはシンプルなものが多く，既存の GUI ビルダーを利用することによって簡単にアスペクト比の異なる端末に対応したアプリケーションを作成できる．しかし，ウィジェットの位置やサイズが揃っていなかったり，複数のウィジェットが同じ行に並んでいたりするような複雑な GUI レイアウトを作成する場合，既存の GUI ビルダーではアスペクト比が異なる端末への自動対応に限界がある．このため，自動で作成されたレイアウトが適切でない場合は作成者自身が再配置しなければならない．再配置する作業は作成者の意図で自由に GUI レイアウトを決定できる一方，時間がかかり作成者の負担が大きい．

これまでにも，アスペクト比が異なるレイアウトを自動的に作成する GUI ビルダーに関する研究や技術が存在する．例えば，汎用的でシンプルなレイアウトを作成する場合に利用される Auto Layout [1]や ORC Editor [2]である．しかし，これらで複雑なレイアウトをアスペクト比の異なるレイアウトに変更する際，作成者の意図に関係なく自動的に配置順が決まるため，意図しない順序でウィジェットが配置される可能性がある．その場合，理想のレイアウトを作成するためには，アスペクト比毎に作成者がウィジェットを再配置しなければならない．

本研究では，PC 向けの GUI を基に，様々なアスペクト比や画面の大きさに対応する GUI の作成を容易化する GUI ビルダーを提案する．本研究の GUI ビルダーを使用することによって，複雑な GUI レイアウトでも異なるアスペクト比毎に再度 GUI レイアウトを作成する必要がなくなる．また，制約充足問題を内部的に解き，配置順と優先順位を用いることによって，PC 上の複雑な GUI レイアウトから，異なる画面の大きさに対応する作成者の意図に沿ったシンプルな GUI レイアウトを作成できる．

本研究の GUI ビルダーを評価するために，PC で作成した様々なレイアウトパターンと，自動で作成されたタブレットやスマートフォンの画面におけるウィジェットの配置を比較した．また，ウィジェットの位置や大きさのパラメータを指定し，配置順と優先順位を被験者自身で考え，エミュレータ上で提示したレイアウトになるかどうか確認する被験者実験を行った．レイアウト作成完了までにかかった時間を記録し，実験後にアンケートを行った．実験の結果，タブレットやスマートフォンの GUI レイアウトは，シンプルかつ作成者自身の要望に合わせて作成できるが，GUI ビルダーの使い勝手は十分でなく，現状の機能だけでは実用的なアプリケーションの作成は難しいことが分かった．

2. 関連研究

Apple [1]は，Mac・iPhone・iPad などの画面サイズに応じて自動的にレイアウトを作成できる Auto Layout を提案した．Auto Layout にはウィジェットの配置を制約で指定

Supervisor: Prof. Hiroshi Hosobe

できる機能や，アスペクト比が異なるレイアウトが自動的に作成された後に手直しをしたりできる機能がある．

Yue ら [2]は OR 制約を用いたアダプティブ GUI レイアウトを作成する ORC Editor を提案した．ORC Editor には，画面サイズが縮小する際に表示しないウィジェットを選ぶ機能などがある．

3. 提案手法

本節では提案手法について述べる．

3.1. 制約充足問題によるウィジェット配置

本手法では，ウィジェットの配置を決定するために制約充足問題を解決する．制約充足問題とは，制約と呼ばれる変数に関する関係の集合が与えられたとき，それに含まれる変数の値を上手く定めることによって，全ての制約を充足する問題を言う．

自動的にアスペクト比が異なるレイアウトを作成する際，ウィジェットが画面からはみ出してしまうことがある．このため，不等式による制約を複数組み合わせて充足できるかどうかを調べる．具体的には，ウィジェットの幅と高さをそれぞれ Widget_W，Widget_H，ウィジェットを配置可能な領域の幅と高さをそれぞれ，Area_W，Area_H として，以下の C1, C2 の制約が充足可能かどうか調べる．

$$C1: Widget\_W < Area\_W$$
$$C2: Widget\_H < Area\_H$$

さらに，以下の 4 つの論理式で場合分けをする．

$$C1 \wedge C2,\ C1 \wedge \neg C2,\ \neg C1 \wedge C2,\ \neg C1 \wedge \neg C2$$

C1∧C2はウィジェットの幅と高さがそれぞれ配置可能な領域に収まっている場合，C1∧¬C2はウィジェットの幅は配置可能な領域に収まっているが高さは収まっていない場合，¬C1∧C2はウィジェットの幅は配置可能な領域に収まっていないが高さは収まっている場合，¬C∧¬C2はウィジェットの幅も高さも配置可能な領域に収まっていない場合を表している．

3.2. 配置順と優先順位

本手法では，ウィジェット毎に配置順と優先順位を設定する．アプリケーション作成者がウィジェットを配置して PC の GUI を作成した後，ウィジェットの配置順と優先順位をつけることによって，自動で様々なアスペクト比に対応した GUI が作成され，新たに GUI を作成する必要をなくなる．

配置順を設定することによって，複数のウィジェットの y 座標が重なる場合，番号順に左上からウィジェットを並べる．図 1 に配置順の使用例を示す．

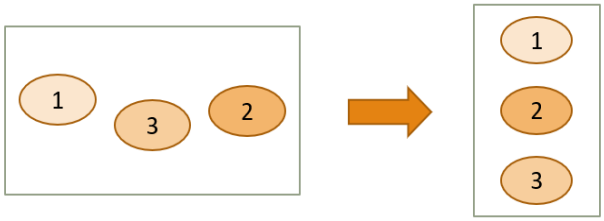

図 1　配置順の使用例

優先順位を設定することによって，画面内にウィジェットが全て入りきらない場合，優先順位の高いウィジェットが入るようにする．また，優先順位が低く画面内に入りきらないウィジェットを非表示にする．図 2 に優先順位の使用例を示す．

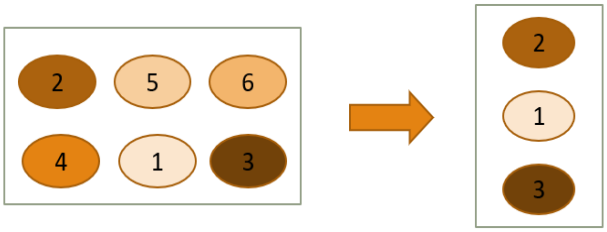

図 2　優先順位の使用例

配置順と優先順位を組み合わせることもでき，画面に入りきらなかった優先順位が最も低いウィジェットを非表示にし，それ以外は配置順の小さい順にウィジェットを配置する．図 3 に赤い数字を配置順，青い数字を優先順位として組み合わせた場合の例を示す．

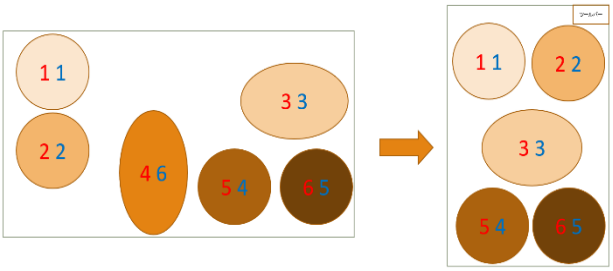

図 3　配置順と優先順位の組み合わせの使用例

3.3. アルゴリズム

図 4 に提案手法のアルゴリズムを示す．全体的な流れとしては，PC で配置したウィジェットのリストを List_p に格納し，List_p の要素を配置順の昇順で並び替えた後，ウィジェットの幅や高さとウィジェットが配置できる幅や高さを参照して，C1, C2 の制約が充足可能かどうかを判定する．画面の左上から順番にウィジェットを配置するため，あるウィジェットが前のウィジェットと同じ行に配置できる場合は，前のウィジェットの右側に配置されるよう，あるウィジェットの x, y 座標を設定する．配置できない場合は，同じ行内にあるウィジェットの中で，高さの値が大きいウィジェットをその行の高さとして，次の行の左側に配置されるよう，あるウィジェットの x, y 座標を設定する．Android 上で使用するウィジェットを List_a に格納していき，配置可能領域内にウィジェットが収まらず，全てのウィジェットが格納できなかった場合は優先順位の低いウィジェットを List_p から List_e に移し，List_p のウィジェットが全て List_a に格納されるまで初めからやり直す．

```
1   List_p ← 空のリスト # PC で使用したウィジェット
2   List_a ← 空のリスト # Android 上で使用するウィジェット
3   List_e ← 空のリスト # List_p から削除したウィジェット
4   Window_W ← ターゲットのウィンドウの幅
5   Window_H ← ターゲットのウィンドウの高さ
6   while (List_a と List_p の要素数が一致していない)
7     List_a ← 空のリスト
8     List_p をウィジェットの配置順の昇順で並び替える
9     Area_W ← 画面の幅
10    Area_H ← 画面の高さ
11    for (i ← 0 から List_p の要素数 – 1)
12      C1 ← ウィジェットの幅が Area_W 未満であるかどうか
13      if (C1 が充足可能)
14        C2 ← ウィジェットの高さが Area_H 未満であるかどうか
15        if (C2 が充足可能)
16          p ← copy(List_p[i])
17          p.x ← Window_W – Area_W
18          p.y ← Area_H – p.height
19          List_a に p を追加する
20          Area_W ← Area_W – p.width
21        else
22          n ← List_p 内の優先順位の数字が高い要素の位置
23          List_e に List_p[n]を追加する
24          List_p から n 番目の要素を削除する
25          List_a の要素をすべて削除する
26      else
27        Max_H ← 行内のウィジェットうち高さが最大のもの
28        Area_H ← Area_H – Max_H
29        Area_W ← Window_W
30        C2 ← ウィジェットの高さが Area_H 未満であるかどうか
31        if (C2 が充足可能)
32          p ← copy(list_p[i])
33          p.x ← Window_W – Area_W
34          p.y ← Area_H - p.height
35          List_a に p を追加する
36          Area_W ← Area_W – p.width
37        else
38          n ← List_p 内の優先順位の数字が高い要素の位置
39          List_e に List_p[n]を追加する
40          List_p から n 番目の要素を削除する
41          List_a の要素をすべて削除する
```

図 4　提案手法のアルゴリズム

4. 実装

本研究の GUI ビルダーを Python で実装した．GUI ライブラリにはマルチプラットフォーム対応の GUI 作成に特化した Kivy [3]を使用した．制約充足には SMT ソルバー Z3 [4]を使用した．

Android アプリケーションを実行する際は Kivy Lancher を利用する．エミュレータ上の設定から画面サイズや解像度を変更することによって，スマートフォンやタブレットなどの画面サイズに対応できる．

5. 実験

5.1. 動作実験

本研究の GUI ビルダーが様々なレイアウトパターンに対応しているかどうか調べる．ウィジェットの幅や高さを分かりやすくするため，全てボタンでレイアウトを作成する．パターンは，配置順が変わる単純なパターン(図 5)，優先順位に基づき画面サイズによって表示されないウィジェットを含むパターン(図 6)，多数の同じようなウィジェットを並べたパターン(図 7)，ウィジェットを不規則に配置したパターン(図 8)に分けられる．図中のボタンのラベルで O は配置順，P は優先順位を示す．図 5〜図 8 で，左上は PC 上で作成したレイアウト(画面サイズ 1280×720)，左下はエミュレータ上でタブレット表示をしたレイアウト(画面サイズ 1024×600)，右はエミュレータ上でスマートフォン表示をしたレイアウト(画面サイズ 720×1280)である．結果は全て意図の通りであり，画面サイズ変更後のウィジェットの位置が画面サイズに応じて変わっている．ただし，同じ行にあるウィジェットどうしの高さは中央揃えではないため，図 8 の場合は，ウィジェットが意図の通りに配置されているものの，全体的にまとまっているようには見えない．

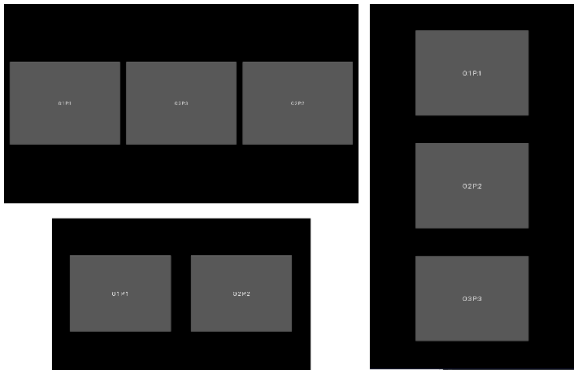

図 5　配置順が変わる単純なパターン

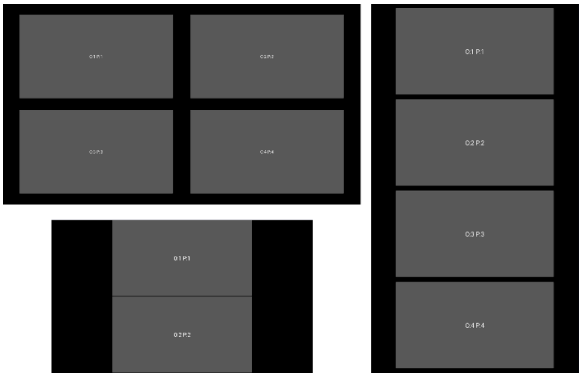

図 6　優先順位に基づき画面サイズによって表示されないウィジェットを含むパターン

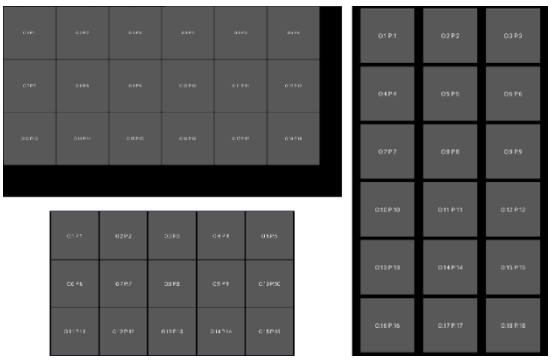

図 7　多数の同じようなウィジェットを並べたパターン

図 8　ウィジェットを不規則に配置したパターン

5.2. 被験者実験

GUI ビルダーの使い勝手に関する被験者実験を行った．被験者は大学生 5 人(平均年齢 21.2 歳，男性 3 名，女性 2 名)である．最初に口頭で本研究の GUI ビルダーの使い方を説明する．次にウィジェットの位置や大きさのパラメータを指定し，配置順と優先順位を被験者自身に考えてもらう．説明終了後，GUI レイアウトを作成し始めた時点から時間を測定する．図 9 のように左側のレイアウトを PC 上で作成してから，Android エミュレータ上で右側のレイアウトになった時点で実験を終了する．さらにアンケートを行い，GUI に関するプログラミング経験があるか，難しかった点，良かった点，改善できる点，使用した感想を調査する．

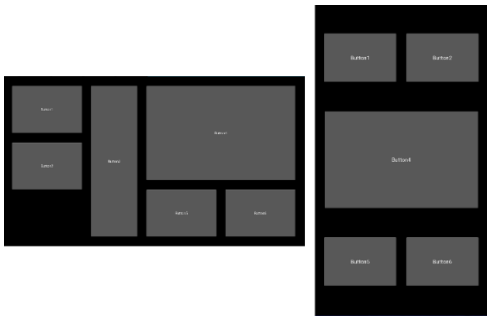

図 9　被験者実験で作成させた GUI レイアウト

実験の結果，初めからレイアウトを提示していたことと，優先順位の選択肢が 1 つではなかったことから，レイアウト作成にかかった時間は被験者によって大きい差はなかった(図 10)．しかし，被験者によって，配置順と優先順位の意味や，本研究の GUI ビルダー・Android エミュレータの操作を素早く理解できたかによって小さい差が生じていた．

難しかった点として「ウィジェットの座標が直感的に理解できなかった」などの意見があった．良かった点として「Android エミュレータ上でのレイアウトが自動で真ん中揃えになっていて良かった」などの意見があった．改善できる点として「間違えた時にやり直しがきくようにした方がいい」などの意見があった．使った感想は「悪い部分を改善して実用的なアプリケーションが作成できるようになったら使ってみたい」などがあった．

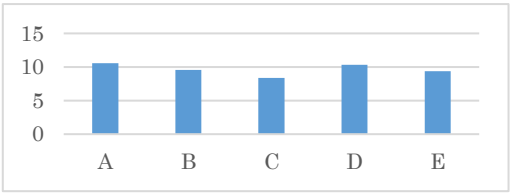

図 10　5 人の被験者(A, B, C, D, E)のレイアウト作成にかかった時間(分)

6. 議論

被験者実験のアンケート調査から，GUI ビルダーを直感的に使用できないことや，配置したウィジェットを作成者が消去できない問題点があったことが分かる．直感的に使用できるようにするためには，ドラッグアンドドロップでウィジェットを配置できるようにしたり，ウィジェットのサイズを柔軟に変更できるような機能を実装したりする必要がある．提案手法をより有用にするためには，図 8 のようなパターンに対して，ウィジェットの配置を縦横それぞれ左揃え・中央揃え・右揃えに設定することによって，より GUI ビルダー使用者の意図に沿ったレイアウトを自動で作成できるように機能を改善し，使用可能なスペースを最大限有効活用してまとまったレイアウトにするための仕組みを実現する必要がある．

現状の GUI ビルダーは，実用的なアプリケーションを作成するのに十分な機能を提供できていない．また，ウィジェットの配置を縦横それぞれ左揃え・中央揃え・右揃えに設定することによって，より GUI ビルダー使用者の意図に沿ったレイアウトを自動で作成できるようにする必要がある．

7. おわりに

本研究では，PC 向けの GUI を基に，様々なアスペクト比や画面の大きさに対応する GUI の作成を容易化する GUI ビルダーを提案した．制約充足問題を内部的に解き，配置順と優先順位を用いることによってウィジェットの自動配置を実現した．

問題点は，配置順と優先順位を利用する場合，スマートフォンやタブレット上でウィジェットを複雑に配置するような GUI を作成できない点である．今後の課題は，より複雑な GUI を自動で作成できる仕組みを考案することである．

文　献

[1] Apple Inc., *Auto Layout Guide*, 2011.

[2] Y. Jiang, R. Du, C. Lutteroth and W. Stuerzlinger, "ORC Layout: Adaptive GUI Layout with OR-Constraints," *Proc. CHI*, no. 413, pp. 1-12, 2019.

[3] M. Virbel, T. E. Hansen and O. Lobunets, "Kivy - A Framework for Rapid Creation of Innovative User Interfaces," *Proc. Mensch & Computer*, pp. 69-73, 2011.

[4] L. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," *Proc. TACAS*, pp. 337-340, 2008.