June 20, 2018

Optimizer for user-interface layout computations

John Hoford Hoford

Nicolas Roard

Optimizer for user-interface layout computations

ABSTRACT

User-interface (UI) developers use design tools such as layout managers to lay out UI elements. A good design tool enables developers to create efficient user interfaces, while itself being efficient to use. Current design tools sometimes produce designs that have inefficient user interfaces, e.g., inflexible (hard-to-change) layouts, or user interfaces with performance problems.

This disclosure provides techniques that improve the efficiency of the computations underlying the UI design tool. Additionally, the techniques provide direct, visual feedback attributing computational cost to specific sub-areas of the screen, enabling developers to optimize the user interface.

KEYWORDS

- user interface
- UI design
- layout manager
- constrained layout
- widgets
- design tool
- positioning rules

BACKGROUND

User-interface (UI) developers use computer-aided design tools, e.g., layout managers, to lay out UI elements. A good layout manager tool enables developers to create efficient user interfaces, while itself being efficient to use. Performance problems in user interfaces are

important for UI developers to address since an ill-performing UI can negatively affect user adoption and/or retention of an application. This is particularly true for user interfaces for mobile systems and applications.

Modern user interfaces comprise widgets positioned by a layout manager. Layout managers typically follow two approaches:

- relatively simple positioning rules, with complex layouts achieved by nesting multiple layout managers; or
- relatively complex and powerful rules, with complex layouts achieved by a single layout manager.

The first approach is easier to implement but can lead to inflexibility or hard-to-change layouts. The second approach is more flexible. However, both approaches can lead to performance or efficiency problems when designing complex user interfaces.

Some layout managers, typically of the second approach, enable developers to position widgets using constraints expressed in terms of a set of linear equations. These constraints are solved via linear programming techniques, e.g., simplex. An issue with linear-constraints based layout managers is that the resulting matrix can become large, even if sparse. While powerful, such layout managers run into performance issues, e.g., when dealing with complex situations such as when the user interface includes a large number of widgets.

DESCRIPTION

This disclosure describes an efficient approach to the computations underlying the layout manager. The computational approach described herein enables attribution of computational cost to specific sub-areas of the screen. Such attribution is provided as visual feedback, e.g., via a heat map, thereby enabling developers to make suitable changes to optimize the user interface.

A key observation is that linear-constraints based layout managers do not expose the underlying linear equations directly. Rather, a high-level grammar of constraints is presented to developers using, e.g., a visual editor. These constraints are often unambiguous, such that the power of a linear equation solver is unnecessary; instead, per techniques of this disclosure, direct resolution is achieved.

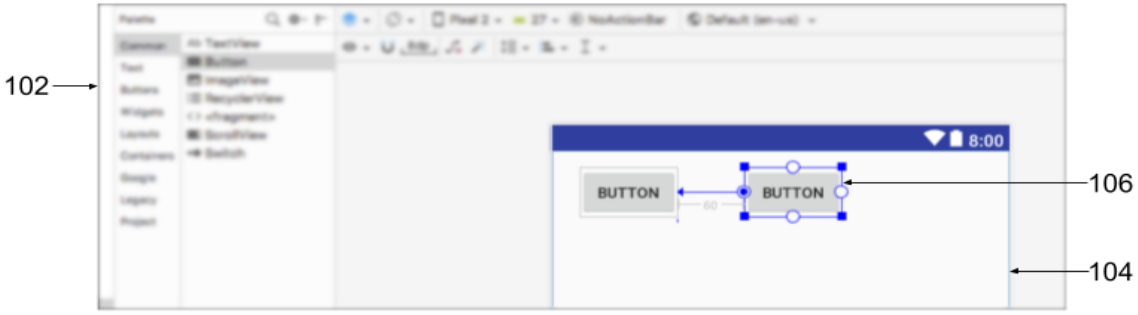

Fig. 1: Example of a layout manager

Fig. 1 illustrates an example of a layout manager (102) used by UI developers to develop user interfaces. A simulated version of the user interface (104) is presented within the visual editor of the layout manager. This enables developers to create and experiment with designs such as placing or moving a button (106).

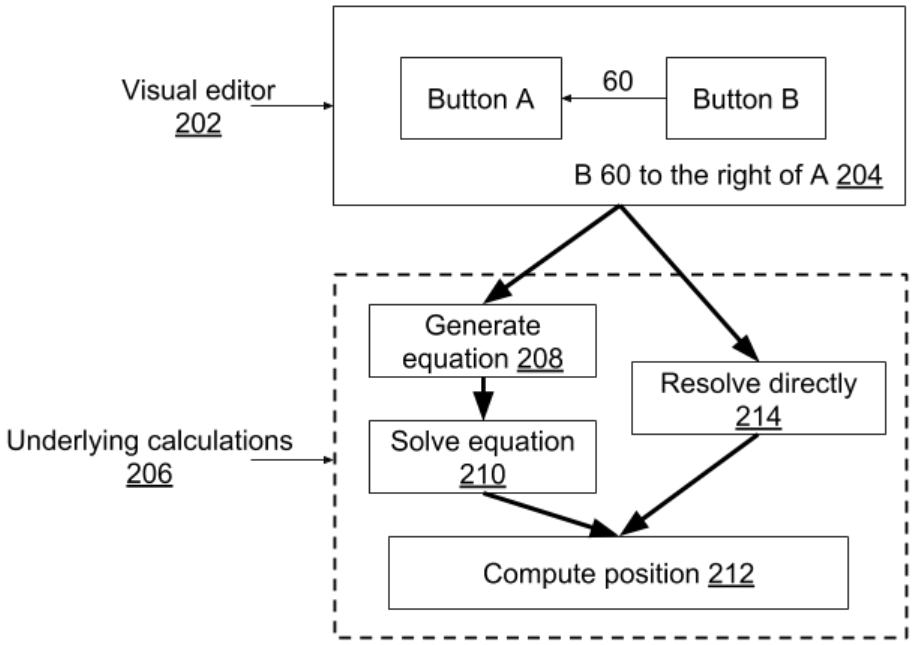

Fig. 2: The visual editor of a layout manager, and calculations underlying a UI design

Fig. 2 illustrates the visual editor (202) of a layout manager, and the calculations underlying a design (206). In the illustrated example, a user of the visual editor makes a design change, e.g., “move button B 60 units to the right of button A” (204). The underlying calculations comprise generating a set of linear equations (208), solving the equations (210), and computing a resulting position (212) of the UI element, e.g., button B. Per techniques of this disclosure, certain design changes are resolved directly (214) without the need to perform the steps of equation generation and solving.

In many situations, the direct resolution pass, as described herein, results in a substantial reduction in the number of variables and equations sent to the equation solver. This improves the overall efficiency of the layout manager, in turn resulting in better performance of the user-interface as seen by the end-user. Per the techniques described herein, ambiguous situations are still solved by the equation solver, thereby resulting in a fast layout manager that allows complex situations to be expressed.

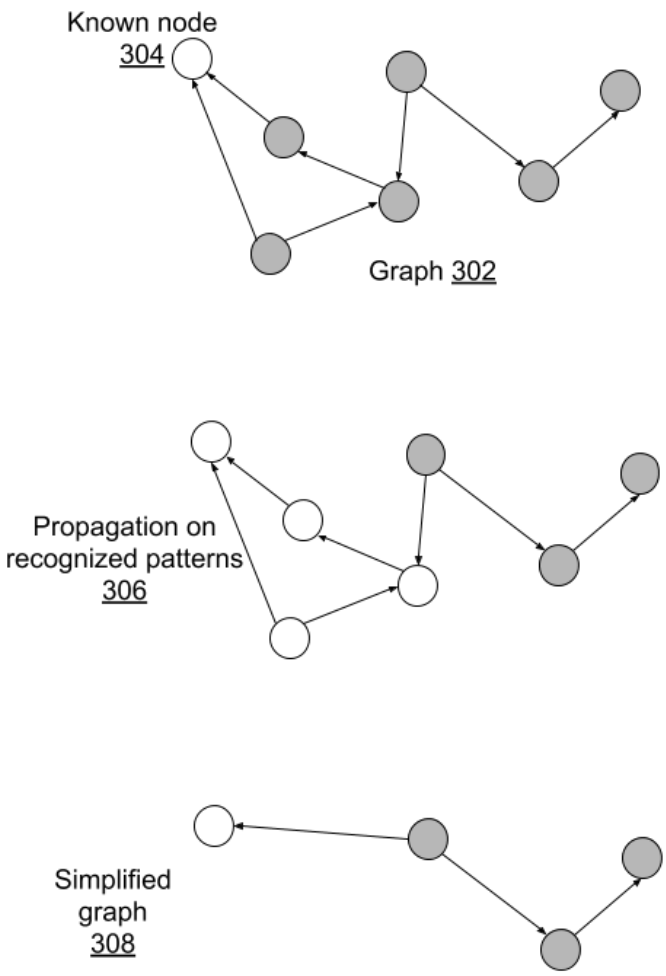

Fig. 3: Representing widgets as a graph of interconnected nodes, and simplifying graph by direct resolution

Fig. 3 illustrates the mechanism of direct resolution. A widget is associated with each node, such that the set of widgets forms a graph (302) of interconnected nodes. Each node, e.g., left, top, right, bottom, baseline, etc., can be connected to any other node. In this manner, the set of widgets is described as a dependency graph of interconnected relationships between nodes. Attributes of the links between the nodes include the types of constraints and costs of the dependency. In graph 302, one or more nodes are known nodes (304), also referred to as resolved nodes. Resolved nodes are illustrated in white, while unresolved nodes are illustrated in grey.

The direct resolution mechanism traverses graph 302, and recognizes patterns of relationships that result in non-ambiguous positioning, e.g.,

- basic offset from a resolved node;
- guidelines positioning on parent when dimension is known;
- centered and bias positioning between two already resolved nodes;
- chain/ group positioning between two already resolved nodes; etc.

Once such patterns are recognized, direct resolution collapses them to resolve the positioning of the nodes. A resolved node in the graph propagates to its dependents (306), potentially enabling them to be resolved as well. This results in a simplified relationship graph (308) that is easier to solve. In many situations the graph collapses in its entirety eliminating the need to run the equation solver. In this manner, the layout manager accrues performance gain.

A widget is typically associated with a specific area of the screen. Thus, the techniques of this disclosure enable providing a visual representation to UI developers of the computational cost of implementing the UI design in an end-user application. This informs UI developers about areas in the layouts that are amenable to further optimization, thereby supporting developer-driven optimization.

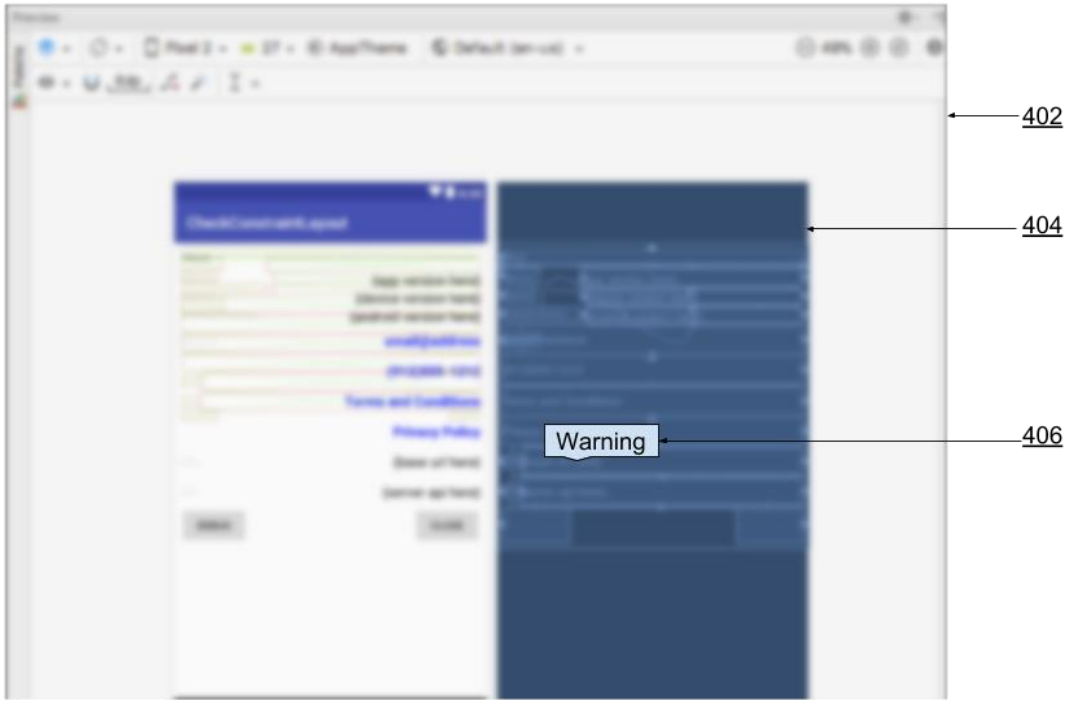

Fig. 4: Visual representation of computational hot-spots within a user-interface design

As illustrated in Fig. 4, the visual representation of the computational costs of a UI can be surfaced in the form of warnings of hotspots (406) within a user interface (404) that is under design using a layout manager (402). Alternately, the visual representation could be in the form of a heat map of relative costs of the constraints at different portions of the screen of a device that renders the user interface.

CONCLUSION

User-interface (UI) developers use design tools such as layout managers to lay out UI elements. A good design tool enables developers to create efficient user interfaces, while itself being efficient to use. Current design tools sometimes produce designs that have inefficient user interfaces, e.g., inflexible (hard-to-change) layouts, or user interfaces with performance problems. This disclosure provides techniques that improve the efficiency of the computations underlying the UI design tool. Additionally, the techniques provide direct, visual feedback

attributing computational cost to specific sub-areas of the screen, enabling developers to optimize the user interface.