API for ORCSolver: An Efficient Solver for Adaptive GUI Layout with OR-Constraints

Yue Jiang[1] **Wolfgang Stuerzlinger**[2] **Matthias Zwicker**[1] **Christof Lutteroth**[3]
[1]Department of Computer Science, University of Maryland, College Park, MD, USA
[2]School of Interactive Arts + Technology (SIAT), Simon Fraser University, Vancouver, BC, Canada
[3]Department of Computer Science, University of Bath, Bath, UK
{yuejiang, zwicker}@cs.umd.edu w.s@sfu.ca c.lutteroth@bath.ac.uk

ABSTRACT
We provide an application programming interface (API) to formalize GUI layouts that can be solved at interactive rates by our solver and also enable users to efficiently create layouts. Our API mainly contains the classes for *ORCLayout* (Abstract), *ORCWidget*, *Pivot*, *ORCColumn*, *ORCRow*, *Flow*, *HorizontalFlow*, *VerticalFlow*, *FlowAroundFix*. We only include public classes, methods, and parameters in this document.

In addition, we also provide some examples to show how to specify layouts with API.

CLASS ORCLAYOUT [ABSTRACT CLASS]
An abstract class to define all the sub-layouts

@param name: the name of the sub-layout

@param parent: the parent sub-layout of this sub-layout

@param weight: the weight (priority) of the sub-layout (default: 1)

set_weight(float weight)
Sets the weight of the sub-layout which represents the priority of the sub-layout.

@param weight: the weight (priority) of the widget

constraint_spec() [Abstract Method]
Constructs the constraint system including optimization variables, constraints, and objective functions for this layout.

solve()
Solves the constraint systems for sub-layouts recursively. We perform the branch-and-bound algorithm through recursive calls, so in the end, we only need to call $root.solve()$ to perform the algorithm, where *root* is the root of the branch-and-bound tree.

CHI'20, April 25–30, 2020, Honolulu, HI, USA
ACM ISBN 978-1-4503-6708-0/20/04. . . $15.00
DOI: https://doi.org/10.1145/3313831.XXXXXXX

get_best()
Gets the best solution for the layout after $solve()$ is called.

@return best_leaf: the leaf node containing the best solution in the branch-and-bound tree

@return best_leaf_result: the best solution for the layout

@return best_leaf_loss: the loss value of the best solution for the layout

CLASS ORCWIDGET (EXTENDS ORCLAYOUT)
Defines a widget or a sub-layout if the sub-layout only contains one widget.

@param name: the name of the sub-layout

@param width_and_height: the min/pref/max width and height of the widget

@param parent: the parent sub-layout of this sub-layout

@param optional: True if the widget is optional in the layout

set_optional()
Sets the widget to be optional, which can be removed if there is not enough space for it.

CLASS PIVOT (EXTENDS ORCLAYOUT)
Specifies an OR constraint that may flip the orientation of a layout from horizontal to vertical, or vice versa. Meanwhile, it also switches between horizontal and vertical flows in the sub-layouts.

@param name: the name of the sub-layout

@param parent: the parent sub-layout of this sub-layout

@param window_width: window width (default: None)

@param window_height: window height (default: None)

set_layout(column_or_row)
If the input is column, then column has higher priority than row when the losses are the same.

@param column_or_row: the layout Pivot wants to switch

CLASS ORCCOLUMN (EXTENDS ORCLAYOUT)

Constructs a *Column* layout dividing the whole window into different areas as a column. A *Column* is a rectangular vertical arrangement of sub-layouts.

@param name: the name of the sub-layout

@param parent: the parent sub-layout of this sub-layout

@param window_width: window width (default: None)

@param window_height: window height (default: None)

define_sublayouts(sublayouts)

Defines the sub-layouts contained in the column.

@param sublayouts: the set of sub-layouts contained in the column

CLASS ORCROW (EXTENDS ORCLAYOUT)

Constructs a *Row* layout dividing the whole window into different areas as a row. A *Row* is a rectangular horizontal arrangement of sub-layouts.

@param name: the name of the sub-layout

@param parent: the parent sub-layout of this sub-layout

@param window_width: window width (default: None)

@param window_height: window height (default: None)

define_sublayouts(sublayouts)

Defines the sub-layouts contained in the row.

@param sublayouts: the set of sub-layouts contained in the row

CLASS FLOW [ABSTRACT CLASS] (EXTENDS ORCLAYOUT)

An abstract class to Construct a flow sub-layout.

@param name: the name of the sub-layout

@param widget_list: widgets to flow in this sub-layout

@param parent: the parent sub-layout of this sub-layout

@param balanced: True if we want the flow to be balanced.

@param fixed_boundary: True if all the boundaries are fixed for this flow sub-layout. (default: False)

@param boundary_distance: when fixed_boundary is True, if the flow is horizontal, then the distance is the distance between top boundary and bottom boundary, and if the flow is vertical, then the distance is the distance between left boundary and right boundary. (default: None)

connect_to_flow(other_flow)

Sets the connected flow the current flow connects to.

@param other_flow: the flow which the current flow connects to

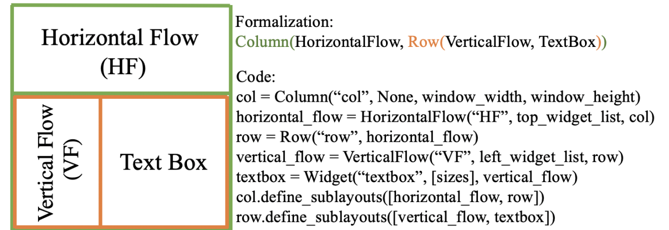


Figure 1. The formalization and corresponding code for a *Column* layout which divides the window into a horizontal flow and a *Row* layout containing a vertical flow and a text box.

solve() [Method overriding]

Solves the constraint systems for sub-layouts recursively. We perform the branch-and-bound algorithm through recursive calls. We override the method in the class *ORCLayout* to allow creating branches in the branch-and-bound tree.

CLASS HORIZONTALFLOW (EXTENDS FLOW)

Defines a horizontal flow sub-layout which uses heuristics in our ORCSolver to solve the horizontal flow sub-layout.

CLASS VERTICALFLOW (EXTENDS FLOW)

Defines a vertical flow sub-layout which uses heuristics in our ORCSolver to solve the vertical flow sub-layout.

CLASS FLOWAROUNDFIX (EXTENDS FLOW)

Defines a flow sub-layout where all the widgets flows around a fixed area. It uses heuristics in our ORCSolver to solve the flow sub-layout.

@param name: the name of the sub-layout

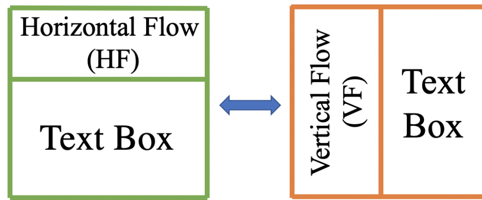
@param widget_list: widgets to flow in this sub-layout

@param parent: the parent sub-layout of this sub-layout

EXAMPLES

The layout in Figure 1 can be formalized as *Column(HorizontalFlow, Row(VerticalFlow, TextBox))*. The layout uses a *Column* layout dividing the window into a horizontal flow and the lower part which is a *Row* layout containing a vertical flow and a text box.

The layout in Figure 2 can be formalized as *Pivot(Column(HorizontalFlow, TextBox))*. The *Pivot* pattern can turn horizontal arrangements into vertical ones. The layout uses a *Column* with a horizontal flow, *e.g.*, of toolbar widgets. The flow is normally placed above the text box, *i.e.*, toolbar at the top. Because of the Pivot, the layout solver can turn the *Column* into a *Row* and at the same time, turn the horizontal flow into a vertical one, *i.e.*, toolbar on the left. Given this layout formalization, the solver can break the layout down into two alternatives, *i.e.*,



Formalization:
Pivot(Column(HorizontalFlow, TextBox))

Code:

```
pivot = Pivot("p", None, window_width, window_height)
col = Column("col", pivot)
horizontal_flow = HorizontalFlow("HF", top_widget_list, col)
textbox = Widget("textbox", [sizes], horizontal _flow)
pivot.set_layout(column)
col.define_sublayouts([horizontal_flow, textbox])
```

Figure 2. The formalization and corresponding code for a *Pivot* layout which can switch between top toolbar and left toolbar.

Column(HorizontalFlow, TextBox) OR *Row(VerticalFlow, TextBox)*.