Improved Formulation of Flexible Widget Layout

Takuto Yanagida* and Hidetoshi Nonaka*
*Graduate School of Information Science and Technology
Hokkaido University, Sapporo 060-0814, Japan
Tel: +81-11-706-6861, Fax: +81-11-706-6861
E-mail: {takty, nonaka}@main.ist.hokudai.ac.jp

***Abstract*—We propose an improvement of our previous work, a formulation of the flexible widget layout (FWL) problem as a fuzzy constraint satisfaction problem (FCSP). Automatic widget layout is an important challenge for the graphical user interfaces (GUIs) generation. In the field of model-based user interface design, the layout is more complicated because to select widgets is needed. FWL is GUI generations with deciding which widgets are used. In this paper, we improve our previous work so that the formulation coincides more strictly with FCSP.**

I. INTRODUCTION

Automatic widget layout is one of the most important challenges for the dynamic generation of graphical user interfaces (GUIs). Especially, in the model-based user interface (UI) design, widget layout is more complicated. In the field, a system generates GUIs based on *logical descriptions*, which specify *UI functions* instead of widgets. Thus, a system must *select widgets* before placing them.

Automatic GUI generation from logical descriptions requires both deciding which widgets are used and completing the layout immediately especially when a system does this at run time. We call the layout satisfying the conditions, *the flexible widget layout* (FWL), and we call the problem of determining a layout which fulfills conditions of widgets, *the FWL problem*.

In our previous work [2, 4], we have proposed a formulation of FWL problem as a *fuzzy constraint satisfaction problem* (FCSP) and a method for solving the problem. In our formulation, *fuzzy constraints* express subjective constraint conditions involved with usability, sensitivity, etc. Moreover, our method can perform the layout in a practical time.

In this paper, we improve our previous work so that the formulation coincides more strictly with FCSP. In the first formulation, its domains are not statically decided, but dynamically change when searching solutions. Therefore, binary constraints checking these domains are in fact under the influence of more than two variables. In this paper, this problem is resolved using the *binarization* of n-ary constraints.

II. FLEXIBLE WIDGET LAYOUT PROBLEM

The FWL problem is a solution search problem for finding better combinations of widgets. Each widget is selected from *a widget candidate set*, which contains widgets representing the same UI function, but having different *size* and *desirability*. UI functions are modeled as *selection act model*, where they are represented as *selection acts* [1, 3]. A set of UI elements is expressed as $U = U_\mathrm{S} \cup U_\mathrm{G} \cup U_\mathrm{D}$, where U_S, U_G, and U_D denotes respectively selection, group, and description elements.

The UI elements are represented as widgets $W = W_\mathrm{N} \cup W_\mathrm{C}$, which are divided into normal widgets and container widgets. As normal widgets, we use eight widely-used widgets for representing selection elements; and *caption label* and *abbr. label* for description elements. As container widgets, we use *vertical array*, *horizontal array*, and *tab pages* for representing group elements; and *left labeling* and *top labeling* for the positioning of description elements. We defined the desirability $\alpha \in [0,1]$ corresponding to types of widgets; we order the desirability in terms of the usability of the widgets.

The UI elements are mapped to corresponding widget sets. Selection elements and description elements are mapped to a set of normal widget candidates $W_i \subset W_\mathrm{N}$, and group elements and positioning of description elements are mapped to a set of container widget candidates $W_i \subset W_\mathrm{C}$. A UI element is represented with widget $w \in W_i$ chosen from its candidate set. Each widget candidate $w \in W_i$ has a unique minimum size $ms_w = \langle ms.width_w, ms.height_w \rangle$, which is defined by a corresponding UI element.

III. FORMULATION

In the formulation, variable $x_i \in X$ corresponds to widget candidate set W_i and the value assigned in it expresses a selected candidate from the set. X is divided into X_N and X_C, which express the variable sets for the normal and container widget candidates respectively. Widget candidate sets of selections, groups, descriptions, and the positioning of the descriptions, are expressed with the variables.

The values of domains are tuples, which are calculated from the bottom to the top of the tree structure of the variables. The domain of $x_i \in X_\mathrm{N}$ is a set of the tuples as follows:

$$D_i(\in D_\mathrm{N}) = \{\langle w, ms_w \rangle | w \in W_i \subset W_\mathrm{N}\},$$

where $ms_w = \langle ms.width_w, ms.height_w \rangle$ is the minimum size of w. The minimum size is defined by the type of widget, its item size, and the item height. The domain of $x_i \in X_\mathrm{C}$ is a set of the tuples as follows:

$$D_i(\in D_\mathrm{C}) = \big\{\langle w, M, ms_{w,M} \rangle \;\; w \in W_i \subset W_\mathrm{C},\\ M \in D_{\mathrm{child}\ (i,1)} \times \cdots \times D_{\mathrm{child}\ (i,\mathrm{cn}_i)}, \mathrm{checksize}\big(W_i, ms_{w,M}\big)\big\},$$

where M is a combination of values of child widget candidates, $\mathrm{child}(i,j)$ is the function for obtaining the index of jth child of W_i, cn_i is the number of children of W_i, and

Function 1 checksize($W_{i,j}, ms$)

if W_i is root **then**
  if $ms.width \leq given\_width$ **and** $ms.height \leq given\_height$ **then**
    return true
  else
    return false
  end if
else
  $ms' \leftarrow \mathrm{ems}'(W_i, ms)$
  return checksize(W_i, ms')
end if.

checksize(W_i, ms) is the function which checks whether the combination of its parameters is available or not with estimated minimum size (ems) (Function 1). By this function, the domains for container variables are pruned when constructing a FCSP. In the function, $given\_width$ and $given\_height$ are given as the size of the client area of the dialog box, and function ems′ is defined as follows:

$$\mathrm{ems}'\left(W_i \subset W_{\mathrm{C}}, ms_{w \in W_{i,j}}\right) = \min_{w \in W_i}\left(ms''_{w,\{\mathrm{ems}(W_{i,1}),\ldots,ms_w,\ldots,\mathrm{ems}(W_{i,\mathrm{cn}(i)})\}}\right),$$

$$\mathrm{ems}(W_i \subset W_{\mathrm{C}}) = \min_{w \in W_i}\left(ms'_{w,\{\mathrm{ems}(W_{i,1}),\ldots,\mathrm{ems}(W_{i,\mathrm{cn}(i)})\}}\right),$$

$$\mathrm{ems}(W_i \subset W_{\mathrm{N}}) = \min_{w \in W_i}(ms_w) = \langle \min_{w \in W_i}(ms.width_w), \min_{w \in W_i}(ms.height_w)\rangle.$$

The container widgets have different sizes of child widgets; thus, the sizes of tuples of their domains are also different. The minimum size of vertical array (VA), horizontal array (HA), and tab pages (TP) is calculated with the minimum sizes of its child widgets ($ms_{w_{i,j}} = \langle ms.width_{w_{i,j}}, ms.height_{w_{i,j}}\rangle$) as follows (where gaps of children and tabs space are omitted):

$$ms_{\mathrm{VA} \in W_i} = \langle \max_j\left(ms.width_{w_{i,j}}\right), \sum_j ms.height_{w_{i,j}}\rangle,$$

$$ms_{\mathrm{HA} \in W_i} = \langle \sum_j ms.width_{w_{i,j}}, \max_j\left(ms.height_{w_{i,j}}\right)\rangle,$$

$$ms_{\mathrm{TP} \in W_i} = \langle \max_j\left(ms.width_{w_{i,j}}\right), \max_j\left(ms.height_{w_{i,j}}\right)\rangle.$$

The minimum sizes of a left labeling (LL) and a top labeling (TL) are calculated based on the size of their description widget ($ms_{w_{i,\mathrm{D}}}$) and the minimum sizes of their one child widget ($ms_{w_{i,\mathrm{C}}}$) as follows (where gaps are omitted):

$$ms_{\mathrm{LL} \in W_i} = \langle ms.width_{w_{i,\mathrm{D}}} + ms.width_{w_{i,\mathrm{C}}}, \max\left(ms.height_{w_{i,\mathrm{D}}}, ms.height_{w_{i,\mathrm{C}}}\right)\rangle,$$

$$ms_{\mathrm{TL} \in W_i} = \langle \max\left(ms.width_{w_{i,\mathrm{D}}}, ms.width_{w_{i,\mathrm{C}}}\right), ms.height_{w_{i,\mathrm{D}}} + ms.height_{w_{i,\mathrm{C}}}\rangle.$$

When the estimated minimum sizes of container widgets are calculated, the above equations are also used, but, instead of actual widget sizes, the estimated minimum sizes of child widgets are used recursively there.

In this formulation, crisp and fuzzy constraints are used accordingly. Each variable is connected by a unary constraint for representing the desirability, and two variables of a container and one of its child elements are connected by a binary constraint for representing a parental relationship. Unary constraint $c_k \in C_{\mathrm{D}}$ denotes the desirability of the value of its scope x_{k_1} as their satisfaction degrees. If the scope of c_k is $S_k = \{x_{k_1}\}$ and the value of x_{k_1} is $v\left(\in D_{k_1}\right) = \langle w, \ldots\rangle, w \in W_{k_1}$, the satisfaction degree of c_k is calculated as follows:

$$c_k(v)(\in C_{\mathrm{D}}) = des(w),$$

where des is the projection from widgets to their desirability α. A binary constraint $c_k \in C_{\mathrm{P}}$ denotes whether the assignments of the variables of its scope correspond with each other. Each value is a tuple of a combination of widget and its minimum size, and thus, this constraint accords this combination with the actual combination of its child widgets. If the scope of c_k is $S_k = \{x_{k_1}, x_{k_2}\}$, the value of x_{k_1} is $v_p\left(\in D_{k_1}\right) = \langle w, M, ms_w\rangle$, and the value of x_{k_2} is $v_c \in D_{k_2}$, the satisfaction degree of $c_k(v_p, v_c)$ is calculated as follows:

$$c_k(v_{\mathrm{p}}, v_{\mathrm{c}})(\in C_{\mathrm{P}}) = \begin{cases} 1 & \text{if } v_{\mathrm{c}} = M[\mathrm{childindex}(x_{k_1}, x_{k_2})] \\ 0 & \text{otherwise} \end{cases}.$$

where childindex(x_1, x_2) is the projection from pairs of variables to the index of the widget candidates (corresponding to $x_2 \in X$) as a child of the parent widget candidates (corresponding to $x_1 \in X_{\mathrm{C}}$).

IV. Conclusions

In this paper, we have proposed an improved formulation of the flexible widget layout problem (FWL). The formulation presented here coincides more strictly with the FCSP framework, and thus, the possibility of extending this work with other techniques for FCSP is increased. The widget selection before doing layout is general; it is not specific for model-based GUI generations, because GUI designers also need to select widgets when they do layouts by hand. Although the constraints in the problem are limited to parental compositions and widget desirability, this limitation is posed in the current implementation, but not in our approach itself.

References

[1] Yanagida, T., Nonaka, H.: Architecture for migratory adaptive user interfaces. In: Proceedings of CIT 2008, pp. 450–455. IEEE, Sydney, Australia (2008).

[2] Yanagida, T., Nonaka, H.: Flexible widget layout with fuzzy constraint satisfaction. In: Proceedings of SMCia/08, pp. 387–392. IEEE, Muroran, Japan (2008).

[3] Yanagida, T., Nonaka, H.: Interaction description with service-specific meanings. In: Proceedings of CITSA 2008, pp. 185–188. IIIS, Orlando, FL, USA (2008).

[4] Yanagida, T., Yasuhiro, S., Nonaka, H.: Flexible widget layout based on fuzzy constraint satisfaction. Journal of Japan Society for Fuzzy Theory and Intelligent Informatics 20 (6), pp. 840-849, (2008).