Flexible Widget Layout with Fuzzy Constraint Satisfaction

Takuto Yanagida, Hidetoshi Nonaka
Graduate School of Information Science and Technology
Hokkaido University
Sapporo 060–0814, Japan
Email: {takty, nonaka}@main.ist.hokudai.ac.jp

***Abstract*—We propose a new solution for the flexible widget layout (FWL) problem, formulating this problem as a fuzzy constraint satisfaction problem (FCSP) in the field of artificial intelligence. Widget layout performed by computers is one of the most important challenges for automatic generation of graphical user interfaces (GUIs). In the field of model-based user interface design, the widget layout is more complicated because the process of selecting widgets is needed. The FWL, we named, is the automatic GUI generation, which requires both (1) deciding which widget types and their alignments are used and (2) completing the layout in a certain time especially when the system generates them in run time. Our system automatically selects appropriate widgets and lays them out in certain rectangles. We formulate the desirability of the selection straightforward as fuzzy constraints; therefore, we can utilize existing technique of FCSP for the FWL. We divide the layout process into three phases, and using an existing FCSP algorithm, we realize the layout in a short time enough not to keep users of the generated GUI waiting.**

I. Introduction

Widget layout performed by computers is one of the most important challenges [1] for automatic generation of graphical user interfaces (GUIs). We mean *widget layout* as the process of determining the positions and the sizes of widgets, such as list boxes, radio buttons, and panels for grouping them. The layout has a significant impact on the usability of applications and services based on GUIs, and it determines the ease of tasks which can be accomplished on them.

In the field of model-based user interface (UI) design [2], [3], the widget layout is more complicated because the process of selecting widgets is needed. Many researches in the field proposed the systems which generate GUIs automatically from *logical descriptions* through layout processes. The systems realize the diversity of generated UIs with the logical descriptions, which specify the common UI functions independently of specific devices and platforms, for example [4], [5]. Namely, the descriptions have no specification of widgets; hence, the systems need to select widgets according to the UI functions before they place widgets. However, the researches handle mainly how to realize various devices and platforms rather than how to execute layout. The ubiquitous interactor [6] shows the results of GUI generations from logical descriptions, where we can see their system does layouts, but the authors do not mention it. The personal universal controller [7] was proposed for remote controlling various appliances with PDAs. A rule-based layout algorithm is explained, but it is dependent on specific application domain and does not handle the widget selection corresponding to the same functions. In XWeb [8], both the widget selection and layout are mentioned, but the proposed system does not handle their combination.

The automatic GUI generation from logical descriptions requires both (1) deciding which widget types and their alignments are used and (2) completing the layout in a certain time especially when the system generates them in run time. We call it *flexible widget layout* (FWL). The systems for the FWL search the combination of widgets and their alignments selecting from their candidates. This feature can expand the possibilities of layout because a system can select small widgets with less usability for small screens, or large widgets with enough usability for large screens. However, when the generation processes are performed in run time of service use, the systems need to finish the layout in real time. Existing *layout managers* may seem to be solutions. In these days, many GUIs are developed with toolkit such as Swing [9], Windows Forms [10], and Qt [11] including some layout managers. The layout managers perform in run time, but they only decide the positions and sizes of widgets, and they do not handle the selection of suitable widgets.

In this paper, we formulate the FWL problem as a *fuzzy constraint satisfaction problem* (FCSP) [12], which is an extension of *constraint satisfaction problem* (CSP), in the field of artificial intelligence. Our system automatically selects appropriate widgets according to logical descriptions and lays them out in certain rectangles. In layout processes, deciding the sizes and the positions of widgets is also performed in existing layout managers, but before that, our system decides which kind of widget and positioning (alignment) is appropriate for the layout. We formulate the desirability of the selection straightforward as *fuzzy constraints*; therefore, we can utilize existing technique of FCSP for the FWL. We divide the layout process into three phases, and using an existing FCSP algorithm, we realize the layout in a short time enough not to keep users of the generated GUI waiting. We claim that our approach can handle adequately complicated real applications, without depending on specific domains, and complete the layout in real time.

In the following sections, we first formulate the FWL problem, next, we show an implementations, discuss some consideration for it, and lastly we conclude our work.

II. Flexible widget layout problem

For the FWL, a system needs to execute both selecting appropriate widgets from sets of widget candidates and putting them in a rectangle of a dialog box. A set of widget candidates corresponds to a certain UI function, and every widget in the set represents the same function. The FWL is executed based on a *UI model* or its descriptions, which contains UI functions and their groupings. A widget selection process has two steps. First, a system resolves the mappings between a UI function and a set of widget candidates, and second, it selects actually used widgets from the mapped sets in terms of the dimensions and usability of the widgets. After that, the system places all selected widgets in a dialog box with no overlapping but gaps among them, and it makes groupings of related widgets in the same rectangles. The complexity of FWL is caused by that widgets have the tradeoff between their usability and the ease of layout involving their dimensions. For example, we can use a list box or a drop down list box for expressing the function of selecting one item from an item list. From the standpoint of usability, since users can view the many items at once, the former is better, but it needs larger area and may not be placed in a small dialog box (or a small screen).

All widgets used here are rectangular and have presentations corresponding to their types of widgets. We defined the desirability $\alpha \in [0,1]$ also corresponding to the types of widgets. In this paper, since they are commonly adopted by many existing toolkits, we use the following subset of widgets W: a check box, radio buttons, a drop down list box, check boxes, and a list box. We distinguish between a check box and check boxes because they are used for different functions. The desirability of each widget type is defined as α_{cb}, α_{rbs}, α_{ddlb}, α_{cbs}, and α_{lb} respectively. Especially, the desirability of a list box is defined according to the rate of its visible items; therefore, it has the range $\alpha_{lb\_min} \leq \alpha_{lb} \leq \alpha_{lb\_max}$. The desirability is ordered in terms of the usability of the widgets. We defined the order of the desirability as $\alpha_{ddlb} < \alpha_{lb\_min} < \alpha_{lb\_max} < \alpha_{rbs} = \alpha_{cbs} < \alpha_{cb}$ referring [13], [14]. The widgets have the reverse order in terms of their dimensions.

As a UI model generally expressed in logical descriptions, in this paper, we adopt *selection act model*, where UI functions are represented as *selection acts* with some parameters, and they are grouped to make a tree graph (Fig. 1). A selection act is represented as a 4-tuple: $s_i = \langle L_i, e_i, t_i, o_i \rangle$, where L_i stands for the list of choices, and $|L_i|$ stands for the number of choices. $e_i \in \{\texttt{single}, \texttt{multiple}\}$ is the selection size, $t_i \in [1, 10]$ is the importance, and $o_i \in \{\texttt{true}, \texttt{false}\}$ denotes the flag meaning whether its choices are opposite when they have two choices. Group elements make groupings of relevant selection acts and group elements as child elements, and they make both parental relationship between itself and its grouped elements and sibling relationship among them. All selection acts are grouped and make a tree graph of UI functions, whose root is a group, and this tree will correspond to a dialog box to be generated. In addition, selection acts and the groups can have a caption string for their explanations.

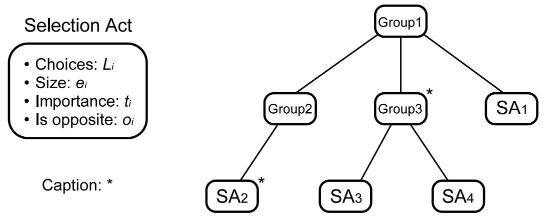

Fig. 1. Selection act UI model.

TABLE I
Widget candidates

Selection size e_i	Item size $\|L_i\|$	Is opposite o_i	Candidates W_i
`single`	$\|L_i\| = 2$	`true`	Check box, Radio buttons, Drop down list box
		`false`	Radio buttons, Drop down list box
	$\|L_i\| < 6$	-	Radio buttons, Drop down list box
	$\|L_i\| \geq 6$	-	List box, Drop down list box
`multiple`	-	-	Check boxes, List box

Selection acts are categorized into some groups and mapped to the corresponding sets of widget candidates $W_i \subset W$, and selection act s_i is expressed with widget $w_i \in W_i$ selected from its set of widget candidates (Table I). We created the widget candidates table also referring [13], [14], but it is possible to apply users' preferences there. Based on the table, for example, widget candidates W_i corresponding to s_i, where $e_i = \texttt{single}$ and $|L_i| = 5$, are determined as a set of a drop down list box and radio buttons (see Table I). In the layout process, it is decided which one of the candidates are used. Each instance of widget w_i has uniquely a minimum size (width: w_{w_i}, height: h_{w_i}), which is defined by parameters ($|L_i|$, the maximum width of item strings of L_i, and o_i) of the corresponding selection act s_i.

A group in UI models and captions of selection acts are represented as *array containers* and *labeled containers* respectively. An array container has three positioning candidates: vertical alignment, horizontal alignment, and tab pages (Fig. 2), and it expresses a grouping of its child elements (widgets and containers). We express the caption of a selection act as a labeled container wrapping the element, because it also has the positioning candidates. The container contains only one child element and has two candidates: the left side label and the upper side label (Fig. 3). On the other hand, the caption of a group is expressed as the label fixed on the upper side of an array container; hence it has no candidates. The containers are mapped to a set of positioning candidates P_j, and each group and caption of selection acts is expressed with positioning $p_j \in P_j$. Each positioning candidate p_j has uniquely a minimum size (width: w_{p_j}, height: h_{p_j}). The minimum sizes of the array containers are defined by the

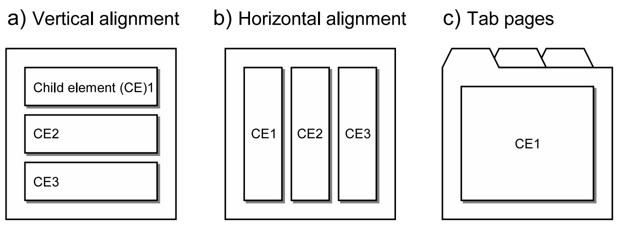

Fig. 2. Three positioning candidates of array container.

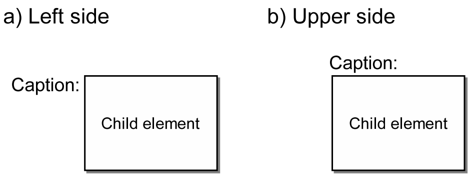

Fig. 3. Two positioning candidates of labeled container.

minimum sizes of its child elements. The minimum sizes of the labeled containers are calculated based on the length of caption and the child element size. We also define desirability α_{ac_v}, α_{ac_h}, and α_{ac_t} for the positioning candidates (vertical alignment, horizontal alignment, and tab pages) of the array containers; α_{lc_l}, and α_{lc_u} for the candidates (left side label and upper side label) of the labeled containers. Especially, the root group of a UI model is expressed as a *dialog*, which has one child element, and it is given a fixed client area size.

Based on the minimum sizes of widgets and containers, a system gives a decision whether it is possible to execute the layout which is defined by the selections from widget candidates and positioning candidates, where the possibility means that the child elements of a container can be placed in its rectangle. We mention the condition expression for that possibility in the next section.

III. Formulation

A. Fuzzy constraint satisfaction

A FCSP is an extension of a traditional CSP. It consists of a finite set of variables $X = \{x_g\}_{g=1}^{q}$, a finite set of domains of values $D = \{d_g\}_{g=1}^{q}$ associated with the each variable, and a finite set of constraints $C = \{c_h\}_{h=1}^{r}$. c_h denotes a fuzzy relation μR_h on a subset $S_h (S_h \subset X)$ of X. S_h is called the *scope* of R_h. If the size of S_h is 1 or 2, then the relation is called a unary or binary relation respectively. Fuzzy relations R_h have their membership functions defined by

$$\mu R_h : \prod_{x_g \in S_h} D_g \to [0, 1].$$

In other words, the membership value is defined by an assignment v_{S_h} to the variables in scope S_h of constraint c_h. It is called the *satisfaction degree*. Since a FCSP requires the satisfaction of the fuzzy conjunction of all the fuzzy constraints, the satisfaction degree of the whole FCSP is defined as the minimum satisfaction degree as follows:

$$C_{\min}(v) = \min_{1 \le h \le r} (\mu R_h(v_{S_h})).$$

The structure of CSPs and FCSPs can be represented by a constraint graph, where nodes and edges of the graph are corresponding to variables and constraints. If a constraint is binary, the two nodes in its scope are connected by an edge. If a constraint is unary, the one node in its scope is connected by an edge as a self-loop. We do not handle constraints which are ternary or of higher order in this paper.

We introduce the framework of FCSP, formulate FWL problems as FCSPs, and represent the desirability of widgets with satisfaction degrees of fuzzy constraints. We use unary fuzzy constraints for expressing the desirability α of widgets. Fuzzy constraints enable us to represent naturally the gradual rules of widget desirability. We represent the parental relationship among widgets with binary crisp constraints, which are particular cases of fuzzy constraints. Using two types of constraints, we formulate FWL problems, which can be applied some existing algorithms for solving FCSP.

B. Flexible widget layout with FCSP

Variables $X = X_W \cup X_P \cup X_D$ express the widget candidates, the positioning candidates, and the dialogs respectively. The values of variables $x_{W_i} \in X_W$, $x_{P_j} \in X_P$, and $x_D \in X_D$ correspond to which candidates are selected. In FWL, parental relationships are expressed as binary constraints between the variables, and a tree structural constraint graph is constructed.

The values of domains are tuples according to each variable type and are used for that constraints calculate their satisfaction degrees. The domain of widget candidate variable x_{W_i} is a set of the tuples consisting of widget w_i and its minimum size $(\mathrm{w}_{w_i}, \mathrm{h}_{w_i})$ as follows:

$$D_{x_{W_i}} = \{\langle w_i, \mathrm{w}_{w_i}, \mathrm{h}_{w_i} \rangle \mid w_i \in W_i\}.$$

The domain of positioning candidate variable x_{P_j} is a set of the tuples consisting of positioning p_j, its minimum size $(\mathrm{w}_{w_i}, \mathrm{h}_{w_i})$, and permissible maximum sizes (width: $\mathrm{W}_{p_{j,k}}$, height: $\mathrm{H}_{p_{j,k}}$) for the child elements as follows:

$$D_{x_{P_j}} = \{\langle p_j, \mathrm{w}_{p_j}, \mathrm{h}_{p_j}, M_{p_j} \rangle \mid p_j \in P_j\},$$

$$M_{p_j} = \langle \mathrm{W}_{p_{j,1}}, \mathrm{H}_{p_{j,1}}, \mathrm{W}_{p_{j,2}}, \mathrm{H}_{p_{j,2}}, \ldots, \mathrm{W}_{p_{j,K_{P_j}}}, \mathrm{H}_{p_{j,K_{P_j}}} \rangle,$$

where $p_{j,k}$ is the kth child element of container P_j, and K_{P_j} is the number of the child elements of P_j. Containers have different sizes of children; therefore, the sizes of tuples of their domains are also different. The domains of dialog variable x_D have a value, which is a tuple containing a given size of client area of the dialog (W_d, H_d) as the permissible maximum size for its child element as follows:

$$D_{x_D} = \{\langle \mathrm{W}_d, \mathrm{H}_d \rangle\}.$$

Each variable except for a dialog variable is connected by a unary constraint for expressing the desirability, and arbitrary two variables are connected by a binary constraint for expressing a parental relationship. Unary constraints $c_{des\,W_i}$, $c_{des\,P_j} \in C_{des\,WP}$ denote the desirability of the value of its scope x_{W_i}, x_{P_j} as their satisfaction degrees. If the value of x_{W_i} is $v_1 = \langle w_i, \mathrm{w}_{w_i}, \mathrm{h}_{w_i} \rangle$, and the value of x_{P_j} is

$v_2 = \langle p_j, \mathrm{w}_{p_j}, \mathrm{h}_{p_j}, M_{p_j} \rangle$, the satisfaction degree of $c_{des\,W_i}$, $c_{des\,P_j}$ is calculated as follows:

$$c_{des\,W_i}(v_1) = des(w_i), \;\; c_{des\,P_j}(v_2) = des(p_j),$$

where des is the projection from the widget candidates and the positioning candidates to their desirability α. A binary constraint $c_{con\,P_{j,k}} \in C_{con\,P}$ denotes whether container P_j can conclude its children $P_{j,k}$ in its rectangle area on the scope x_{P_j} and $x_{P_{j,k}}$. If the value of x_{P_j} is $v_1 = \langle p_j, \mathrm{w}_{p_j}, \mathrm{h}_{p_j}, \langle \ldots, \mathrm{W}_{p_{j,k}}, \mathrm{H}_{p_{j,k}}, \ldots \rangle \rangle$ and the value of its kth child element $x_{P_{j,k}}$ is $v_2 = \langle p_{j,k}, \mathrm{w}_{p_{j,k}}, \mathrm{h}_{p_{j,k}} \rangle$, the satisfaction degree of $c_{con\,P_{j,k}}$ is calculated as follows:

$$c_{con\,P_{j,k}}(v_1, v_2) = \begin{cases} 1 & \text{if } \mathrm{w}_{p_{j,k}} \le \mathrm{W}_{p_{j,k}} \text{ and } \mathrm{h}_{p_{j,k}} \le \mathrm{H}_{p_{j,k}} \\ 0 & \text{otherwise} \end{cases}$$

IV. Implementation

We implemented an experimental system for FWL, which consists of three phases: (1) creating a FCSP from a UI model, (2) solving of the problem with an algorithm, and (3) performing actual layout based on the result of the algorithm (Fig. 4). In the current implementation, we need to program the mapping from the elements of a UI model to the corresponding sets of the widget candidates and the positioning candidates as input sources for the system. The desirability α_* of both the widget candidates and the positioning candidates are given empirically assuming typical applications.

A. Creating problem phase

In the first phase, a constraint graph is generated from a given UI model, and this graph is a tree structure having one-to-one correspondence to the model. Groups and selection acts in a UI model are seen as each corresponding typed variable (nodes) in the graph, and the parental relationships are seen as composition constraints there. All variables except for a dialog variable have unary constraints for expressing the desirability of the current assignments of the variables.

The minimum sizes of widgets, which are the values of the domains of the variables, are determined by the parameters of the based selection acts, and the minimum sizes of containers are determined by the minimum sizes of the child elements of the containers. The minimum widths of widgets are the sum of the width of one of its items, which has the longest string length, and the widths of the operational parts such as a vertical scroll bar, a radio button, and a check box. The minimum heights are defined by the widget type, their item size, and the item height *item_h* (Table II). The minimum size of an array container (w_{ac_j}, h_{ac_j}) is calculated based on the minimum sizes of its child elements ($\mathrm{w}_{ac_{j,k}}$, $\mathrm{h}_{ac_{j,k}}$) as follows (where gaps among child widgets and tabs space are omitted):

$$\mathrm{w}_{ac_j} = \begin{cases} \max(\mathrm{w}_{ac_{j,k}}) & \text{vertically alignment} \\ \sum \mathrm{w}_{ac_{j,k}} & \text{horizontal alignment} \\ \max(\mathrm{w}_{ac_{j,k}}) & \text{tab pages} \end{cases}$$

TABLE II
Minimum widget heights

Widget	Minimum height (without edges)
Check box	*item_h*
Drop down list box	
List box	$\min(\lvert L \rvert, 4)$ *item_h*
Radio buttons	$\lvert L \rvert$ *item_h*
Check boxes	

$$\mathrm{h}_{ac_j} = \begin{cases} \sum \mathrm{h}_{ac_{j,k}} & \text{vertically alignment} \\ \max(\mathrm{h}_{ac_{j,k}}) & \text{horizontal alignment} \\ \max(\mathrm{h}_{ac_{j,k}}) & \text{tab pages} \end{cases}$$

The minimum size of a labeled container (w_{lc_j}, h_{lc_j}) is calculated based on the size of its label (lw_{lc_j}, lh_{lc_j}) and the minimum size of its one child element ($\mathrm{w}_{lc_{j,1}}$, $\mathrm{h}_{lc_{j,1}}$) as follows (where gaps are omitted):

$$\mathrm{w}_{lc_j} = \begin{cases} \mathrm{w}_{lc_{j,1}} + \mathrm{lw}_{lc_j} & \text{left side} \\ \max(\mathrm{w}_{lc_{j,1}}, \mathrm{lw}_{lc_j}) & \text{upper side} \end{cases}$$

$$\mathrm{h}_{lc_j} = \begin{cases} \max(\mathrm{h}_{lc_{j,1}}, \mathrm{lh}_{lc_j}) & \text{left side} \\ \mathrm{h}_{lc_{j,1}} + \mathrm{lh}_{lc_j} & \text{upper side} \end{cases}$$

The permissible maximum sizes of child elements, which are entries of the value of the domains of container variables, are determined from the dialog to the descendant containers in turn, based on the given dialog size. The permissible maximum size of a dialog is the client area size of the dialog, which is given fixed size. The permissible maximum size for child elements ($\mathrm{W}_{ac_{j,k}}$, $\mathrm{H}_{ac_{j,k}}$) of array container ac_j is calculated based on the maximum size of the container (W_{ac_j}, H_{ac_j}) given by its parent container and all of the minimum sizes of its child elements ($\mathrm{w}_{ac_{j,k}}$, $\mathrm{h}_{ac_{j,k}}$) as follows:

$$\mathrm{W}_{ac_{j,k}} = \begin{cases} \mathrm{W}_{ac_j} & \text{vertically alignment} \\ \mathrm{W}_{ac_j} - \sum_{l \ne k} \mathrm{w}_{ac_{j,l}} & \text{horizontal alignment} \\ \mathrm{W}_{ac_j} & \text{tab pages} \end{cases}$$

$$\mathrm{H}_{ac_{j,k}} = \begin{cases} \mathrm{H}_{ac_j} - \sum_{l \ne k} \mathrm{h}_{ac_{j,l}} & \text{vertically alignment} \\ \mathrm{H}_{ac_j} & \text{horizontal alignment} \\ \mathrm{H}_{ac_j} & \text{tab pages} \end{cases}$$

The permissible maximum size for the child element ($\mathrm{W}_{lc_{j,1}}$, $\mathrm{H}_{lc_{j,1}}$) of labeled container lc_j is calculated based on the maximum size of the container (W_{lc_j}, H_{lc_j}) given by its parent container, and its label size (lw_{lc_j}, lh_{lc_j}) as follows:

$$\mathrm{W}_{lc_{j,1}} = \begin{cases} \mathrm{W}_{lc_j} - \mathrm{lw}_{lc_j} & \text{left side} \\ \mathrm{W}_{lc_j} & \text{upper side} \end{cases}$$

$$\mathrm{H}_{lc_{j,1}} = \begin{cases} \mathrm{H}_{lc_j} & \text{left side} \\ \mathrm{H}_{lc_j} - \mathrm{lh}_{lc_j} & \text{upper side} \end{cases}$$

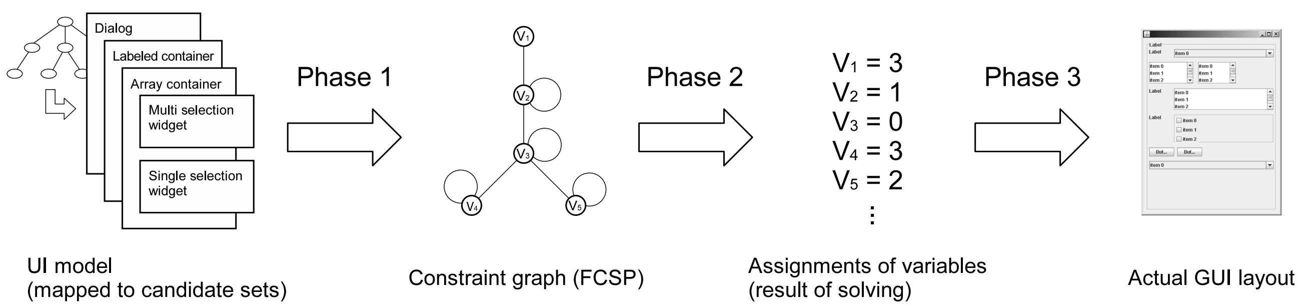

Fig. 4. Phases of performing flexible widget layout.

B. Solving problem phase

In the second phase, the system iterates solving the FCSP generated in the previous phase using the forward checking algorithm in order to search better solution, or the assignments of the variables. The system prunes the domains according to a *worst satisfaction degree*.

The precise description of the process of solving the FCSP is as follows. (Step 1) Before the iteration, the system prepares satisfaction degree set A by collecting possible satisfaction degrees from all unary constraints. Since the all unary constraints express desirability α of each widget candidates and positioning candidates, A is a discrete set. (Step 2) The system chooses the maximum one a from A as the worst satisfaction degree and removes it. It prunes worse values of the domains than a based on the unary constraints. At this step, the minimum heights in the domains of widgets in the list box state are reset as follows:

$$\mathrm{h}_{lb_i} = |L_i|\, item\_h \left(1 - \frac{\alpha_{lb\_max} - a}{\alpha_{lb\_max} - \alpha_{lb\_min}}\right)^{\frac{1}{t_i}}.$$

(Step 3) The system solves the FCSP with forward checking algorithm, which is extended for handling fuzzy problems improving the worst constraint satisfaction degree. (Step 4) If the system can find a solution, or an assignment for the variables, it moves to the next phase in order to perform an actual layout; otherwise, it moves back to the step 2. If there is no value in A, the system stops in failure.

For solving the problem rapidly, the system prunes the domains based on the given worst constraint satisfaction degrees before applying the algorithm. The forward checking algorithm searches systematically through the search space of the possible assignments of values to the variables until it finds a solution. It is guaranteed to find a solution if one exists but has the disadvantage that it requires large cost of time; hence, pruning the domains and reducing the scale of the problem is effective. Since possible satisfaction degrees of the constraints are finite discrete sets, the system can prune the domains. We mention the effect of the pruning in the next section.

C. Layout with result phase

In the last phase, based on the assignments of the variables, or the selected candidates, the system decides the positions and the sizes of the selected widgets, and then it places them. In FWL, the variables express the selections of both of widget candidates and positioning candidates; therefore, the solution of a FCSP is not a layout. The system generates widgets adopted in the solution; then it sets their concrete positions (pixels) and sizes (pixels) based on the values of container variables. In our implementation, the resize of the dialog box is re-flected in the relayout of the widgets (Fig. 5).

V. Discussion

We need to evaluate our system in detail, but for now, we have confirmed that it can finish performing the layout of the example (Fig. 5) fast enough for GUI generation. We implemented the system with Java 6 on a notebook computer (Pentium M 1.10 GHz CPU, 512 MB main memory, and Windows XP Professional edition). The system can finish the layout of the example within 250 msec. We also executed the system with the same layout without pruning and got the result that it takes more than 50000 msec. That shows the pruning is effective in the system.

In early phases of our research, we tried expressing a FWL problem with the variables corresponding to widgets sizes and positions, but we were not able to obtain enough speed for solving it. That is because these variables expressing positions and sizes have large domains, and the scale of the problem is enlarged. For the large scale problem, systematic search algorithms take long time; also, some stochastic search algorithms, we tested, is not efficient because the problem has large *plateau*, which is a flat part of the search space.

We need to program the input of the system in the current implementation, and we are going to apply another research of UI architecture and UI description language by us [4], [5]. We omitted here, but the UI model of selection acts is originally adopted in our abstract interaction description language (AIDL), which is a UI specification language independent of specific devices and modalities. Hence, we have a plan of developing the new version of the system, which receives AIDL document as description of UI model and generates GUI performing its layout.

VI. Conclusion and future work

In this paper, we have formulized the layout problem accompanied by widget selection, named the flexible widget layout problem, as a fuzzy CSP, and we have offered the solution

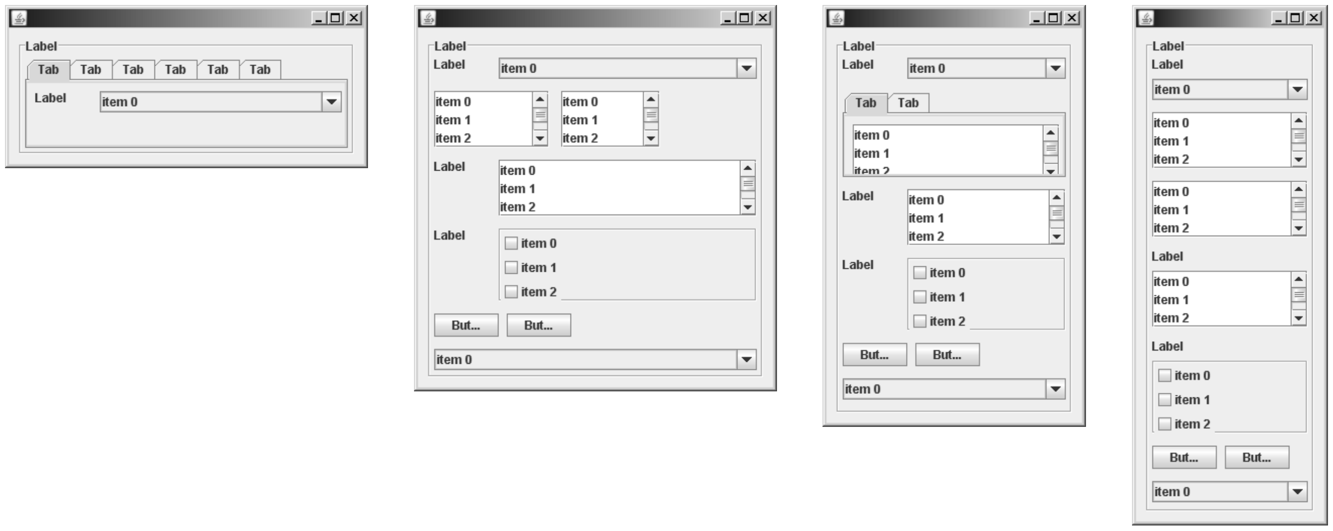

Fig. 5. Results of flexible widget layout from the same UI model with different dialog sizes.

solving it in a practical time for the users. The widgets used in our implementation are a typical subset of ones in existing GUI toolkits; hence, the problem which we have addressed is just a small size one. Besides, the constraints in the problem are used for only parent-child compositions, desirability of widgets, and placements of captions. These limitations, however, are posed just in the current implementation, but not in our approach itself. The selection of widgets before their layout is general, and it is not specific for the model-based GUI generations, because designers also need to select widgets when they execute layout by hand. The points of our work are that we have utilized the achievements in the FCSP domain of artificial intelligence field, where our method is one of their practical applications, and we have offered the method which can solve the layout in a certain time.

As our future work, we need to add some layout rules based on GUI guidelines, to evaluate the relation between problem scales and solving times, and to consider other algorithms for FCSPs. For example, we can add constraints between sibling widgets for keeping the same types and states among them, which does not exist in the current implementation. We are considering extracting GUI dialog boxes from existing applications and reconstructing them with our method, and we might be able to evaluate its availability. In the current implementation, we use the forward checking algorithm, which is systematic approach, but our method is not limited this algorithm. Formulated as FCSP, the layout problem allows us to use suboptimal results; therefore, we can also consider adopting of stochastic approaches such as fuzzy GENET (FGENET) [15], [16], and SRS [17] as alternatives of algorithms for FCSP other than the forward checking.

References

[1] S. Lok and S. Feiner, "A survey of automated layout techniques for information presentations," in *SmartGraphics '01*, 2001.

[2] J. Eisenstein, J. Vanderdonckt, and A. Puerta, "Applying model-based techniques to the development of UIs for mobile computers," in *IUI '01*, 2001.

[3] J. M. Vanderdonckt and F. Bodart, "Encapsulating knowledge for intelligent automatic interaction objects selection," in *CHI '93*, 1993.

[4] T. Yanagida and H. Nonaka, "Interface migration using abstract interaction description," in *Technical Report of the Institute of Electronics, Information and Communication Engineers SS2007-24*, vol. 107, no. 176, 2007, pp. 49–52, (in Japanese).

[5] T. Yanagida, H. Nonaka, and M. Kurihara, "User-preferred interface design with abstract interaction description language," in *IEEE International Conference on Systems, Man and Cybernetics*, 2006.

[6] S. Nylander, M. Bylund, and A. Waern, "The ubiquitous interactor–device independent access to mobile services," in *CADUI'2004*, 2004, pp. 274–287.

[7] J. Nichols, B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, and M. Pignol, "Generating remote control interfaces for complex appliances," in *UIST 2002*, 2002, pp. 161–170.

[8] D. R. Olsen, S. Jefferies, S. T. Nielsen, W. Moyes, and P. Fredrickson, "Cross-modal interaction using XWeb," in *UIST 2000*, 2000, pp. 191–200.

[9] Sun Microsystems, Inc., "JDK 6 swing (java foundation classes)," available at http://java.sun.com/javase/6/docs/technotes/guides/swing/index.html.

[10] Microsoft Corporation, "Windows forms," available at http://msdn2.microsoft.com/en-us/netframework/aa497342.aspx.

[11] Trolltech ASA, "Qt," available at http://trolltech.com/products/qt/.

[12] Z. Ruttkay, "Fuzzy constraint satisfaction," in *Proceedings 1st IEEE Conference on Evolutionary Computing*, Orlando, 1994, pp. 542–547.

[13] S. L. Fowler, *GUI Design Handbook*. Mcgraw-Hill Companies, Inc., 1997.

[14] Apple Inc., "Apple human interface guidelines," 2008, available at http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/OSXHIGuidelines.pdf.

[15] E. Tsang and C. Wang, *A Generic Neural Network Approach for Constraint Satisfaction Problems*. Springer-Verlag, 1992, pp. 12–22.

[16] J. H. Y. Wong and H. fung Leung, "Extending GENET to solve fuzzy constraint satisfaction problems," in *AAAI '98/IAAI '98: Proceedings of the fifteenth national/10th conference on Artificial intelligence/Innovative applications of artificial intelligence*. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1998, pp. 380–385.

[17] Y. Sudo and M. Kurihara, "Spread-repair-shrink: A hybrid algorithm for solving fuzzy constraint satisfaction problems," in *IEEE International Conference on Fuzzy Systems (WCCI 2006)*, 2006.