IEICE TRANS. INF. & SYST., VOL.E97–D, NO.5 MAY 2014

PAPER Special Section on Knowledge-Based Software Engineering

Method for Consistent GUI Arrangements by Analyzing Existing Windows and Its Evaluation

Junko SHIROGANE[†a)], ***Member***, **Seitaro SHIRAI**[††], **Hajime IWATA**[†††], ***Nonmembers***, ***and*** **Yoshiaki FUKAZAWA**[††], ***Member***

SUMMARY To realize usability in software, GUI (Graphical User Interface) layouts must be consistent because consistency allows end users to operate software based on previous experiences. Often consistency can be achieved by user interface guidelines, which realize consistency in a software package as well as between various software packages within a platform. Because end users have different experiences and perceptions, GUIs based on guidelines are not always usable for end users. Thus, it is necessary to realize consistency without guidelines. Herein we propose a method to realize consistent GUIs where existing software packages are surveyed and common patterns for window layouts, which we call layout rules, are specified. Our method uses these layout rules to arrange the windows of GUIs. Concretely, source programs of developed GUIs are analyzed to identify the layout rules, and then these rules are used to extract parameters to generate source programs of undeveloped GUIs. To evaluate our method, we applied it to existing GUIs in software packages to extract the layout rules from several windows and to generate other windows. The evaluation confirms that our method easily realizes layout consistency.

key words: *GUI, usability, consistency, layout*

1. Introduction

Consistency is a crucial factor for usability of software GUIs (Graphical User Interfaces) [1], [2]. Generally, consistency means that the layouts of all windows in a software package, including widget size, position, arrangement, etc., are unified, the same terminology is used for the same meaning text, operations of functions are standardized, etc. Because consistency allows software to be easily operated through practice, end users can employ their experiences to manipulate various functions of GUIs in a software package, even if they are unfamiliar with a specific GUI. Typically user interface guidelines (hereafter, guidelines) are developed to realize consistency for a specific platform, such as Windows User Experience Interaction Guidelines [3] and Mac OS X Human Interface Guidelines [4].

These guidelines include numerous detailed elements, such as widget arrangements, label names in widgets, etc. Developing GUIs via guidelines, which define detailed GUI layouts and terminology, realizes consistency not only in a software package but also between different software packages. However, following guidelines does not always develop suitable GUIs for end users. For end users, GUIs are often developed by layout strategies other than guidelines. Conversely, guidelines have numerous detailed elements, and developers must create numerous GUI windows. Thus, that they cannot be economically and efficiently applied to all guideline elements to all windows.

User interface patterns have been developed for user interface designs [5], [6]. Patterns define solutions to certain usability problems and usable layouts for various situations of users' actions and data representations. Because patterns are individually developed, combining them may not realize consistent GUIs and determining the appropriate patterns for each situation is difficult.

Thus, our research focuses on creating consistent window layouts. We have proposed a method to develop consistent GUIs. Our method assumes that undeveloped windows are generated based on the layout of developed windows. An "undeveloped window" means that although developers have considered the widget list in a window, the source programs have yet to be developed, whereas a "developed window" means a window layout and its source programs are complete. Actual windows in the target and other software programs as well as sample windows for typical strategies of widget arrangements are acceptable as developed windows.

Our previous paper analyzed source programs and extracted layout rules [7]. The layout rules, which indicate the arrangement method for windows in the GUIs (e.g., widget combinations, size, position, etc.), were classified into position and combination rules, and undeveloped windows were generated by applying the layout rules. Our previous paper focused on position rules, but did not provide details for the combination rules or evaluate our method.

Our current research has refined the position rules and defined the combination rules. Specifically, the number of widget-applying position rules has increased, the position rules are extracted and applied in more detail, and combination rules are classified into four sub-rules. Herein we described our overall method and evaluate its applicability.

Our method targets multi-window style GUIs in which widgets can be arranged flexibly in a window (e.g., widgets can be arranged by specifying the size and coordinates). A typical target is GUIs for desktop applications on personal computers where function is often realized via window switching.

Manuscript received June 26, 2013.
Manuscript revised October 29, 2013.

[†]The author is with Tokyo Woman's Christian University, Tokyo, 167–8585 Japan.
[††]The authors are with Waseda University, Tokyo, 169–8555 Japan.
[†††]The author is with Kanagawa Institute of Technology, Atsugi-shi, 243–0292 Japan.
a) E-mail: junko@lab.twcu.ac.jp

DOI: 10.1587/transinf.E97.D.1084

Copyright © 2014 The Institute of Electronics, Information and Communication Engineers

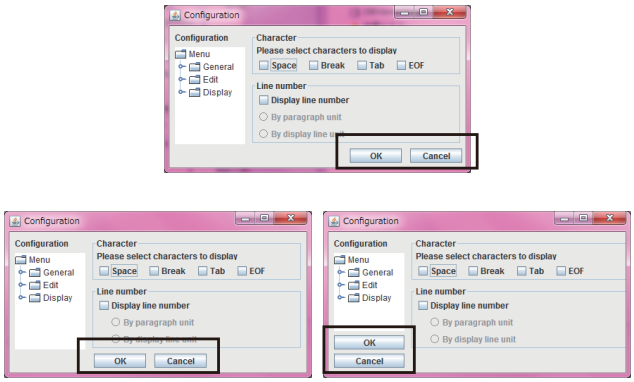

Fig. 1 Example of windows with inconsist layouts.

For high usability, simple and natural interactions, low error rates, etc. must be considered. Consistency is one of the most crucial and basic factors of usability [1]. Because our research aims to support the development of "consistent GUIs", "usability" is synonymous with consistency.

Although consistency includes various factors, in our method consistency means that window layouts, including widget size, position, and arrangement, are unified, while "consistent GUIs" means that the windows in a software program satisfy consistency. Other definitions of consistency, such as operations of functions, color usages, and sounds, are beyond scope of our method.

Figure 1 shows an example of three windows, which have the same content but different layouts. Thus, consistency is not realized in terms of layouts. All of these window layout strategies are used in an actual software package. The most crucial problem is that the locations of the "OK" and "Cancel" buttons differ in each window. These buttons are used to complete the task of a window. When the button locations differ in each window, end users may have to search for the buttons because the actual and assumed positions conflict. This is why window layout is an important factor for usability.

Although our method is currently implemented to analyze and generate GUI source programs written in Java programming language, it can be applied to other programming languages. The developed GUIs can be either actual windows in the software or sample windows consisting of typical widget arrangements. Then the extracted layout rules are used to generate the source programs of the undeveloped GUIs.

This paper is organized as follows. Section 2 describes related works, while Sect. 3 highlights the features of our method. Section 4 shows the layout rules defined in our method. Section 5 provides a detailed description of our method. Section 6 evaluates our method, and Sect. 7 concludes our paper.

2. Related Works

Arranging the layouts of numerous windows with complex GUI structures is a heavy burden on developers. Because GUI layouts directly reflect end user's perceptions of software usability, many works have proposed methods to develop and apply guidelines [8], [9]. However, except for our method, few have aimed to realize consistent GUIs.

For example, Bendsen proposed a method to automatically generate GUI layouts [10] for business applications with numerous windows. Class structures are represented as models, which are transformed into abstract layers. Then GUIs with the appropriate layouts for various platforms are generated. Although their method adopts a simpler strategy, our method focuses on GUIs with more complex layouts.

Lutteroth et al. proposed a method to arrange layouts based on the Auckland Layout Model (ALM) [11], [12]. Codes to extract GUI data are added to the source programs to generate abstract GUIs. Although this method can flexibly generate GUIs by platform, it modifies the source programs and does not always produce consistent GUIs. In contrast, our method focuses on consistency without modifying the developed GUIs.

Raneburger et al.'s method groups widgets by containers, which are structured in the form of a tree within a window (e.g., panel widgets) [13]. A container can include subcontainers and multiple widgets, which are arranged from right to left and from top to bottom. Designers can add extra rules to arrange the widgets. When child containers are placed into a parent container, the parent container is divided into several regions. Then the appropriate regions for the child containers are determined. Widget sizes are calculated automatically based on the size of the root container. Although this method easily arranges widgets, it does not always realize consistent GUIs.

Feuerstack et al. proposed a method to arrange layouts based on existing models that can realize user interface designs, such as task trees and dialog models [14]. In this method, designers interpret these models to describe layout models, which consist of ordered statements with six properties. Designers specify the layout characterization by interpreting existing models to generate statements. Both intended and unintended user interface designs can be realized. However, because designers interpret various existing models and specify the user interface design from numerous model elements, this method can be burdensome.

3. Features of Our Method

Easily realized consistent GUIs

Because guidelines contain numerous detailed elements, applying all elements to numerous windows in software is challenging, and combining user interface patterns does not assure consistent GUIs because these patterns are individually developed. Additionally, software packages contain many windows, and each window has different content, which makes arranging window layouts a challenge.

To resolve the guideline issue, our method allows developers to easily obtain consistent GUIs because it automatically extracts layout rules, arranges consistent GUI lay-

outs, and generates source programs by surveying the guidelines to determine analysis targets of developed GUIs. It should be noted that we limited the scope of elements in our survey because many elements are beyond consistency. Then the process to extract layout rules includes the analysis targets. Thus, only developers create a few developed windows applying guidelines, and our method extracts guideline elements as layout rules and generates undeveloped windows applying the guidelines.

For user interface patterns, we assume that developed GUIs are consistent, even if individual patterns are combined. Thus, our method realizes consistency in the generated GUIs by analyzing some developed windows and extracting layout rules.

Also, it is difficult to arrange GUI layouts, because each window has different contents. However, arrangement strategies are often determined in every widget usages. These arrangement strategies for every widget usages are defined as layout rules, thus, making GUIs consistent can be realized by automatically applying layout rules.

Support to develop guidelines

Our method automatically extracts layout rules, both intended and unintended by the developers, using developed GUIs. These extracted rules are the basis for the guidelines, which are applied to various software packages to realize consistent GUI layouts in the package and between different software packages within a platform. Finally, our method generates the source programs for the undeveloped windows. Typically detailed programming techniques are necessary to apply the items in the layout rules, but because our method automatically arranges GUI layouts, such detailed techniques are not required.

4. Layout Rules

Layout rules, which indicate the arrangement of windows in GUIs, consist of a "position rule" and "combination rules". Combination rules consist of four sub-rules for a total of five layout rules.

Although layout rules except these five rules are used in actual software, such layout rules are used in few windows of software of a certain vendor or a certain software package. We intend that our current method is proposed not for special use to software of a certain vendor or a certain software package but for general use, and our proposed method derives layout rules for general use in various software packages. Here we describe the concept of our extracted layout rules, including order, coordinates of widgets, and arrangements of widgets.

4.1 Position Rules

Position rules are applied to widgets with specific label names (i.e., "OK", "Cancel", "Help", "Next", "Close", "Back", "Complete", and "Previous"). In many cases, buttons with these label names are commonly arranged at specific positions in every window. For example, the "OK" and "Cancel" buttons are often at the bottom of a window. These buttons usually trigger (or cancel) a process. Thus, the label name, order, and position of these buttons within a window are fixed in the software package or platform. Position rules unify the locations of the above eight label names in every window, and concrete positions of these buttons are determined by analyzing developed GUIs.

Table 1 Number and appearance rate of widgets.

Label name	Number	Appearance rate
Cancel	178	85%
OK	124	59%
Help	37	18%
Next	21	10%
Close	17	8%
Back	10	5%
Complete	9	4%
Previous	7	3%

To determine the target label names of the buttons for this position rule, we surveyed 209 windows in 28 common software packages in Windows and Mac OS, such as web browsers (e.g., Internet Explorer[†] and Mozilla Firefox[††]) and image processing tools (Adobe Photoshop[†††] and iPhoto[††††]). Table 1 shows the results where "Label name" indicates the label names of buttons at the bottom of a window. "Number" denotes the number of windows in which the buttons are placed, and "Appearance rate" indicates the rate of windows where the buttons are placed.

According to Table 1, positions of buttons with certain label names are determined based on label names. Thus, these button positions are defined as position rules.

4.2 Combination Rules

Some widgets are grouped together to indicate a purpose within a window. We surveyed the groups in 270 windows of 12 common software packages, such as web browsers (e.g., Internet ExplorerInternet Explorer[†] and Mozilla Firefox[††]) and word processors (Microsoft Word[†††††] and OpenOffice.org Writer[††††††]). Four frequently used sub-rules are defined as combination rules (i.e., "widgets grouped by a border", "widgets with instructions", "grouped widgets with instructions", and "parent and child widgets"). Table 2 shows the survey results where "Windows" indicate the number of windows that each sub-rule is used. Because some sub-rules are used more than once in a window, "Number" indicates the total number of appear-

[†]Internet Explorer: http://windows.microsoft.com/ja-JP/internet-explorer/download-ie

[††]Firefox: http://www.mozilla.org/firefox/

[†††]Adobe Photoshop: http://www.adobe.com/products/photoshop.html

[††††]iPhoto: http://www.apple.com/ilife/iphoto/

[†††††]Microsoft Office: http://office.microsoft.com/

[††††††]OpenOffice.org: http://www.openoffice.org/

Table 2 Number and rate of sub-rules.

Sub rules	Windows	Number
Widgets with instructions	218	1942
Grouped widgets with instructions	91	342
Parent and child widgets	67	287
Widgets grouped by border	71	573

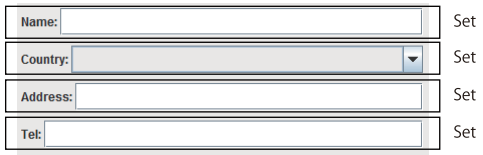

Fig. 2 Examples of widgets with instructions.

Fig. 3 Example of grouped widgets with instructions.

ances of each sub-rule. When a window includes plural tabs that switch the window content, it is counted as one window.

4.2.1 Widgets with Instructions

The "widget with instructions" sub-rule is applied to two widgets where one is a type of widget, usually a text field or a combo box, and the other is a label widget for the former widget's instruction when a label widget and a type of widget are present and both of their variable names include common keywords. In many cases, this sub-rule arranges the label widget to the left or upper side of the type of widget. However, actual arrangement strategies are determined by analyzing developed GUIs. Figure 2 shows examples using four sets of widgets; the second set is a combination of label and combo box widgets, while the other three are combinations of label and text field widgets.

4.2.2 Grouped Widgets with Instructions

This sub-rule is similar to the "widgets with instructions", except that multiple widgets use a single label widget. The "grouped widgets with instruction" sub-rule is applied when multiple widgets use a single label widget for instruction and all their variable names include common keywords. In many cases, this sub-rule arranges the label widget at the left top or upper side of other widgets. However, actual arrangement strategies are determined by analyzing developed GUIs. Figure 3 shows an example using a label and three checkbox widgets where the label widget "E-mail" is the instruction for the checkbox widgets.

4.2.3 Parent and Child Widgets

Parent widgets are selection widgets (i.e., radio button and checkbox). Once the parent is selected, some other widgets are enabled. The other widgets are child widgets. The

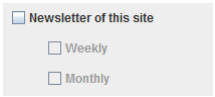

Fig. 4 Examples of parent and child widgets.

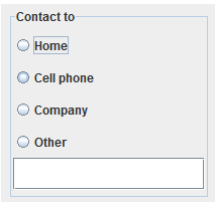

Fig. 5 Example of widgets grouped by border.

"parent and child" sub-rule is applied when there are some radio button or check box (parent) widgets and other (child) widgets in a group that developers specify. Figure 4 shows an example using a set of widgets, which are all checkbox widgets, and selecting the parent widget, "Newsletter of this site", enables the child widgets, "Weekly" and "Monthly".

4.2.4 Widgets Grouped by Border

The "widgets grouped by border" sub-rule places a group of widgets within a border and adds instructions to the border. Any widget type in the group is available. When developed windows include this sub-rule, this rule is applied to a group that developers specify. Figure 5 shows an example where five widgets are placed within a border, and a "Contact to" label is added as the instruction.

5. Process of Our Method

Our method analyzes developed GUI source programs, extracts the layout rules, and generates the undeveloped GUIs automatically. Figure 6 shows the system architecture, while Fig. 7 details the flow of the "grouping widgets" and "extracting layout rules" in Fig. 6.

5.1 Extracting GUI Data

GUI data, including widget size, positions, and combinations, are required to arrange the layout. By analyzing the developed GUI programs, our method extracts this information in two phases (i.e., static and dynamic analyses).

Static analysis

The developed GUI programs are statically analyzed using a parser generated by JavaCC (Java Compiler Compiler) [15]. In this phase, a widget ID is assigned to each widget and window. Then the parser extracts the variable name and type for each widget.

Table 3 Example of widget values.

	Variable	Type	Width	Height	Position (x, y)	Label	Parent
Window	n/a	JFrame	300	200	n/a	n/a	n/a
"OK"	okBut	JButton	51	26	(128, 146)	OK	WindowA:butPanel
"Cancel"	cancelBut	JButton	73	26	(199, 146)	Cancel	WindowA:butPanel

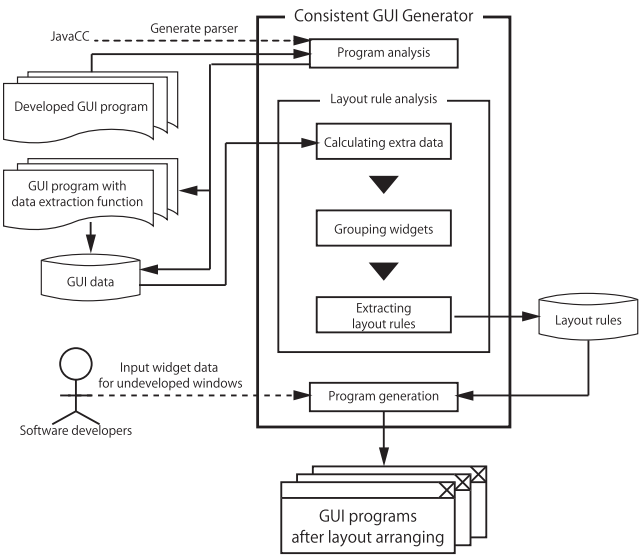

Fig. 6 System architecture.

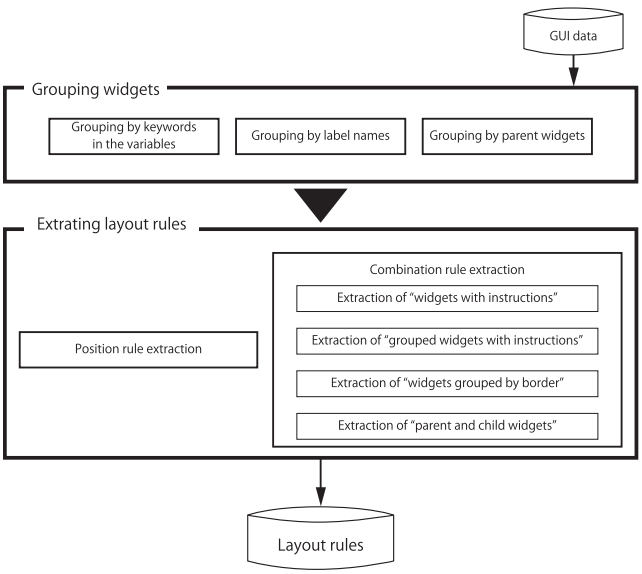

Fig. 7 Flow of analyzing layout rules.

Dynamic analysis

The developed GUI programs are executed, and GUI data are extracted. Because functions to extract GUI data are generated and added to the GUI programs by AspectJ [16] using the GUI data extracted in the static analysis, functions can be added without modifying the developed GUI programs. These functions extract following additional GUI data:

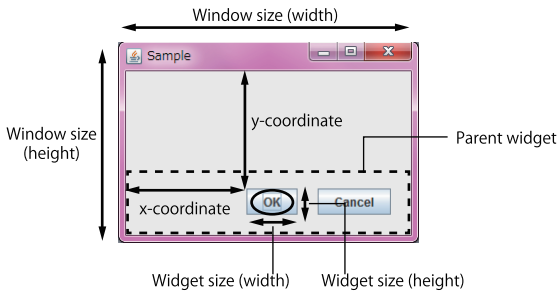

Fig. 8 Example of dynamic analysis.

- Window width (windowW) and height (windowH)
- x-coordinate (x) and y-coordinate of widget
- Widget width (widgetW) and height (widgetH)
- Label name displayed on widgets
- Parent widget ID (ID on which widgets are place)

Thus, dynamic analysis extracts concrete GUI data. Figure 8 shows an example for a window where two button widgets ("OK" and "Cancel") are on a panel widget. According to the analyses, all the values of the GUI data are extracted. Table 3 shows examples of the GUI data values in Fig. 8.

5.2 Analyzing Layout Rules

Using the GUI data extracted in 5.1, the implemented layout rules are analyzed. The observed pixels/ratios are calculated using the GUI data and the widgets are grouped based on GUI data. Finally, the layout rules are extracted.

5.2.1 Calculating Observed Pixels/Ratios

Although GUI data extracted in 5.1 can be extracted directly from a program, the following must be calculated using the formulae below and the extracted GUI data to derive the layout rules. Figure 9 shows an example calculation.

center x-Coordinate of the center of a widget
Formula: $center = x + widgetW \div 2$

right x-Coordinate of the right edge of a widget
Formula: $right = x + widgetW$

xfr Distance between the right edge of a window and left edge of a widget
Formula: $xfr = windowW - x$

dfr Distance between the right edge of a window and the right edge of a widget
Formula: $dfr = windowW - right$

dfc Distance in the x direction between the center of a window and the center of a widget

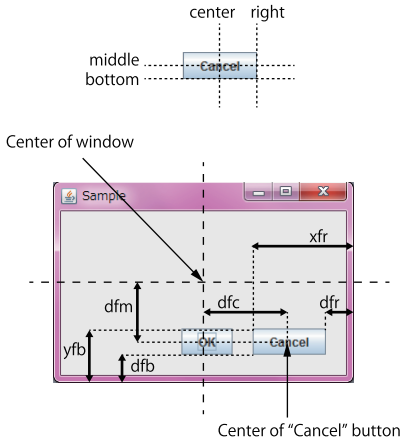

Fig. 9 Example calculation of observed pixels/ratios.

Formula: $dfc = windowW \div 2 - center$

xP Ratio of the x-coordinates of a widget to the window width
Formula: $xP = x \div windowW$

dfrP Ratio of dfr of a widget to the window width
Formula: $dfrP = dfr \div windowW$

dfcP Ratio of dfc of a widget to the window width
Formula: $dfcP = dfc \div windowW$

middle y-Coordinate of the center of a widget
Formula: $middle = y + widgetH \div 2$

bottom y-Coordinate of the bottom edge of a widget
Formula: $bottom = y + widgetH$

yfb Distance between the bottom edge of a window and top edge of a widget
Formula: $yfb = windowH - y$

dfb Distance between the bottom edge of a window and the bottom edge of a widget
Formula: $dfb = windowH - bottom$

dfm Distance in the y direction between the center of a window and the center of a widget
Formula: $dfm = windowH \div 2 - middle$

yP Ratio of the y-coordinates of a widget to the window height
Formula: $yP = y \div windowH$

dfbP Ratio of dfb of a widget to the window height
Formula: $dfrP = dfr \div windowW$

dfmP Ratio of dfm of a widget to the window height
Formula: $dfmP = dfm \div windowH$

5.2.2 Grouping Widgets

Widgets are grouped using three strategies (label names, keywords, and parent widgets) based on the extracted and observed pixels/ratios. Each widget can belong to multiple groups.

Grouping by label names

Widgets can be grouped by their label names (e.g., "OK" and "Cancel") based on 4.1. Table 4 shows widgets grouped by label names where "Label name" indicates the name of the grouped widgets. "Window name" denotes the name of the window in which the grouped widgets are placed, and "Variable name" represents the variable name of the grouped widget.

Table 4 Example of grouping by label names.

Label name	Window name	Variable name
OK	Window A	okButton
	Window C	ok
	Window D	ok
Cancel	Window A	cancelButton
	Window B	cancelBut
	Window D	cancel
	Window E	cancelButton

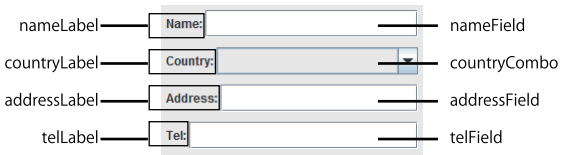

Fig. 10 Example of variables with common keywords.

Table 5 Example of grouping by keywords of variable names.

Keyword	Variable name
name	nameLabel, nameField
country	countryLabel, countryCombo
address	addressLabel, addressField
tel	telLabel, telField

Fig. 11 Example of arranging widgets using panel widgets.

Grouping by keywords in the variables

Often keywords, which represent the meaning, are used as variables. In the naming rules of the Java programming language, the first keyword should be all lower case, but the first letter in subsequent words should be upper case. Using these naming rules, keywords in variables can be extracted.

Additionally, common keywords are often used for related variables. For example, a text field widget and label widget are used for the instruction of a text field widget. Figure 10 shows an example where the label widgets on the left represent the instruction widgets on the right, while Table 5 shows an example grouping.

Grouping by parent widgets

To arrange GUI layouts in Java, related widgets are placed in a panel widget and grouped. Figure 11 shows an example

Table 7 Example of widget position dteremination.

"OK"	Value items based on x-coordinate									Value items based on y-coordinate								
buttons	x	center	right	xfr	dfr	dfc	xP	dfrP	dfcP	y	middle	bottom	yfb	dfb	dfm	yP	dfbP	dfcP
A	214	250	286	196	124	45	0.52	0.3	0.11	234	243	252	66	48	93	0.78	0.16	0.31
B	394	430	466	196	124	135	0.67	0.21	0.23	488	497	506	66	48	220	0.88	0.09	0.4
C	570	613	656	210	124	223	0.73	0.16	0.29	666	676.5	687	69	48	309	0.91	0.07	0.42
D	202	245	288	210	124	39	0.49	0.3	0.09	428	437	446	90	72	178	0.83	0.14	0.34
E	8	102.5	197	727	538	265	0.01	0.73	0.36	318	329	340	97	75	121.5	0.77	0.18	0.29
F	268	311	354	173	87	90.5	0.61	0.2	0.21	216	227	238	65	43	86.5	0.77	0.15	0.31

Table 6 Example of grouping by panel widgets.

Panel widget	Variable name
namePanel	nameLabel, nameField
countryPanel	countryLabel, countryCombo
addressPanel	addressLabel, addressField
telPanel	telLabel, telField
butPanel	okBut, cancelBut

of arranging widgets using panel widgets. The related widgets often have meaningful relationships. For example, the "namePanel" widget includes a label widget and text field widget to input a name. Table 6 shows an example of grouping by this strategy.

5.2.3 Extracting Layout Rules

Position and combination rules are extracted using widget groups.

Extracting position rules

The position rules are applied to buttons with specific label names described in 4.1 to uniformly coordinate the positions of these buttons in every window. Because the position rules are applied widgets with specific label names, widgets must be classified by label names and position in a window.

To extract position rules, groups by label names in 5.2.2 are used. For each group of widgets, the determination strategy of widget position is described below using values of the "OK" button in six windows as an example (Table 7).

Step 1
Widgets from a group are classified by 18 value items (x-coordinate, center, right, xfr, dfr, dfc, xP, dfrP, dfcP, y-coordinate, middle, bottom, yfb, dfb, dfm, yP, dfbP, and dfmP) to create sub-groups. Sub-groups are created by classifying widgets in a group by the same values of these 18 value items. Table 8 shows examples of the sub-groups created using Table 7 where "Sub-group" indicates the numbers of sub-groups and "Widget list" indicates the lists of "OK" buttons included in the sub-groups. "Value item (Value)" indicates whether widgets in the sub-groups have the same value, and their values are shown in parentheses.

Step 2
The sub-group with the most widgets and its value item is specified. For example, according to Table 8, the sub-group with the most widgets is sub-group 3 and the value item is dfr.

Table 8 Example of sub-group classification.

Sub-group	Widget list	Value item (Value)
1	A, B	xfr (196)
2	C, D	xfr (210)
3	A, B, C, D	dfr (124)
4	A, B	yfb (66)
5	A, B, C	dfb (48)
6	E, F	yP (0.77)
7	A, F	dfmP (0.31)

Table 9 Example of sub-sub group classification.

Sub-sub group	Widget list	Value item (Value)
1	A, B	yfb (66)
2	A, B, C	dfb (48)

Step 3
When the specified value item in step 2 is based on x-coordinate (y-coordinate), widgets in the sub-group are classified by the same value items based on the y-coordinate (x-coordinate). By classifying widgets in the sub-group, sub-sub groups are created. For example, widgets of sub-group 3 are classified by value items based on the y-coordinate because the specified sub-group in step 2 is sub-group 3, which has a value item (dfr) based on the x-coordinate. Table 9 shows the sub-sub groups from sub-group 3 in Table 7. The meanings are the same as Table 8.

Step 4
For the sub-sub group with the most widgets, its value item is extracted. For example, in Table 9, sub-sub group 2 has the most widgets, and its value item is dfb.

Step 5
The values of the value items extracted in steps 2 and 4 are defined as position rules. For the "OK" buttons used as an example, the extracted position rule is a dfr value of 124 and a dfb value of 48.

Extracting combination rules

To extract combination rules, groups by variable keywords and by parent widgets described in 5.2.2 are used.

Extracting "widgets with instructions" and "grouped widgets with instructions"

Combination sub-rules for "widgets with instructions" and

Table 10 Example of calculated distances between the label and other widgets.

		Horizontal distances					Vertical distances				
Label	Other	dll	drl	dcc	drr	dlr	dtt	dbt	dmm	dbb	dtb
Name	text field (Name)	39	0	200	361	400	0	27	0	0	27
Country	combo box (Country)	50	0	200	350	400	0	27	0	0	27
Address	text field (Address)	54	0	200	346	400	0	27	0	0	27
Tel	text field (Tel)	23	0	200	377	400	0	27	0	0	27

"grouped widgets with instructions" are the label widgets used as instructions for other widgets. A label widget and another widget are a set in the sub-rule "widgets with instructions", and the label widget and some other widgets are a set in the sub-rule "grouped widgets with instructions". To extract these sub-rules, it is necessary to identify sets of label and other widgets.

These combination sub-rules are extracted in the following manner.

Step 1
From the groups by variable keywords in 5.2.2, groups with two or more widgets and groups in which only one of the widgets is a label widget are extracted. If groups satisfy both of these conditions, then widgets may be arranged by the sub-rule "widgets with instructions" or "grouped widgets with instructions".

Step 2
For the extracted groups, the distances between the label and other widgets within the groups are extracted. Distances consist of the following and can be calculated by formulae below and the extracted GUI data. In these formulae, "label" indicates a label widget, while "other" indicates other widget. Additionally, the spacing is called the "distance value". When there are multiple widgets in a group, these widgets are arranged vertically and the left edges are unified in many cases. Thus, distances are calculated between a label widget and the most upper and left other widget as well as between adjacent other widgets.

dll Horizontal distances between the left edge of the label widget the left edge of the other widget
Formula: $dll = |x(other) - x(label)|$

drl Horizontal distances between the right edge of the label widget and the left edge of the other widget
Formula: $drl = |x(other) - right(label)|$

dcc Horizontal distances between the center of the label widget and the center of the other widget
Formula: $dcc = |center(other) - center(label)|$

drr Horizontal distances between the right edge of the label widget and the right edge of the other widget
Formula: $drr = |right(other) - right(label)|$

dlr Horizontal distances between the left edge of the label widget and the right edge of the other widget
Formula: $drr = |right(other) - x(label)|$

dtt Vertical distances between the top edge of the label widget and the top edge of the other widget
Formula: $dtt = |y(other) - y(label)|$

dbt Vertical distances between the bottom edge of the label widget and the top edge of the other widget
Formula: $dbt = |y(other) - bottom(label)|$

dmm Vertical distances between the center of the label widget and the center of the other widget
Formula: $dcc = |middle(other) - middle(label)|$

dbb Vertical distances between the bottom edge of the label widget and the bottom edge of the other widget
Formula: $dbt = |bottom(other) - bottom(label)|$

dtb Vertical distances between the top edge of the label widget and the bottom edge of the other widget
Formula: $dbt = |bottom(other) - y(label)|$

Table 10 shows an example of the calculated distances between label widgets and other widgets in Fig. 2. "Label" indicates the label names of label widgets, while "Other" indicates widget types of the other widgets, and label names of their label widgets are shown in parentheses.

Step 3
The most common distances between the label and other widgets are defined by the sub-rules "widget with instructions" and "grouped widgets with instructions". Distances between other widgets are defined similarly.

For example, according to Table 10, the most common value is drl for the horizontal distance and dtt for the vertical distance. Thus, the horizontal distance between the label widget and other widget is defined as 0, while the vertical distance is defined as 0.

Extracting "parent and child widgets"

In the "parent and child widgets" sub-rule, parent widgets represent the selection, such as a radio button or checkbox widget, which make child widgets available. In many cases, child widgets are arranged immediately below and indented relative to the parent widget. To extract this rule, it is necessary to identify selection widgets and indented widgets immediately below the selection widgets.

This combination sub-rule is extracted using the following steps.

Step 1
In many cases, parent widgets of "parent and child widgets" are radio button or checkbox widgets. Groups that satisfy either of the two following conditions are extracted:

- For groups by parent widgets, widgets are only radio button or checkbox widgets
- For widgets arranged immediately above and below, one widget is not indented and the others are indented

Table 11 Example of calculated distances for the "parent and child widgets" sub-rules.

Labe 1	Labe 2	ind	Label 1	Label 2	ver
Newsletter	Dayly	36	Newsletter	Dayly	0
Newsletter	Weekly	36	Dayly	Weekly	2
Newsletter	Monthly	36	Weekly	Monthly	2

Table 12 Example of distances of groups for "parent and child widgets" sub-rules.

Group	ind	ver
1	36	2
2	36	0
3	36	4
4	30	2

Step 2
The following distances among the widgets are calculated using the extracted GUI data described in 5.1 and 5.2.1. These distances are calculated for each group. The values of all the widget combinations using each formula are calculated, and the smallest values are adopted for each distance.

ind Distances between the left edge of a non-indented widget (widget A) and the left edge of indented widget (widget B)
Formula: $ind = |x(widgetB) - x(widgetA)|$

ver Distances between the bottom edge of a widget (widget C) and the top edge of the widget immediately above widget D
Formula: $dv = |y(widgetD) - bottom(widgetC)|$

Table 11 shows an example of the calculated distances widget groups in Fig. 4. "Label1" and "Label2" indicate the label names of the widgets used to calculate distances ("Newsletter" indicates the label name of "Newsletter of this site" checkbox).

Step 3
For each group, the most common distances of "ver" and "ind" are defined as the distance of each group. For example, according to Table 11, the most common value of "ind" is 36 and "ver" is 2.

Step 4
The most common distances in each group are defined as the rule "parent and child widgets". For example, Table 12 shows examples of the distances for each group of widgets. Because the most common values are 36 for "ind" and 2 for "ver", these values are the defined as the rule.

Extracting "widgets grouped by a border"

The combination sub-rule "widgets grouped by a border" is a set of widgets arranged on a panel widget to which a border with an instruction is added. To extract this sub-rule, the widgets on the panel widgets with borders and instructions must be identified. This combination sub-rule is extracted following steps.

Table 13 Example of calculated distances for "widgets grouped by a border" sub-rules.

Labe 1	Labe 2	dv
Home	Cell phone	9
Cell phone	Company	9
Campany	Other	9
Other	(text field)	6

Table 14 Example of distances of groups for "widgets grouped by a border" sub-rules.

Group	dv
1	9
2	6
3	6
4	6

Step 1
If groups by parent widgets and groups described in 5.2.2 with more than two widgets in a panel are extracted, they can be arranged by the rule "widgets grouped by a border".

Step 2
The following distances among the widgets are calculated using the extracted GUI data described in 5.1 and 5.2.1. These distances are calculated for each group. The values of all widget combinations using each formula are calculated, and the smallest values are adopted for each distance.

dh Distances between the right edge of a widget (widget A) and the left edge of the widget (widget B) when the widgets are arranged horizontally and widget B is arranged at immediately left of widget A.
Formula: $|x(widgetB) - right(widgetA)|$

dv Distance between the bottom edge of a widget (widget C) and the top edge of the widget (widget D) immediately above widget B when the widgets are arranged vertically
Formula: $|y(widgetD) - bottom(widgetC)|$

Table 13 shows an example of the calculated distances widget groups in Fig. 5. Because widgets in this group are arranged vertically, only the "dv" values are described. "Label 1" and "Label 2" indicate the label names of widgets used to calculate distances. (Because the text field widget in Fig. 5 does not have the label name, it is represented by "text field").

Step 3
For each group, the most common distances of "dv" is defined as the distance for each group. For example, according to Table 13, the most common "dv" value of 9 is defined as the value for the group of widgets.

Step 4
The most common distance in each group is defined as the rule "widgets grouped by a border". Table 14 shows examples of distances for each group of widgets. The most common value of 6 for "dv" is defined as this sub-rule.

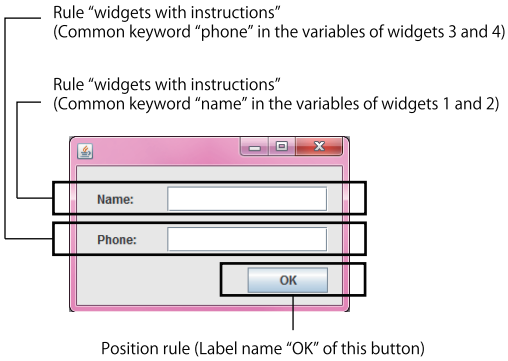

Fig. 12 Example of generated window.

Table 15 Results of extracting layout rules.

Layout rule	Software A	Software B	Software C
Widgets with instructions	1.5	4	1
Position rule	3	4.5	4

Table 16 Results of applying layout rules to windows in software.

	Widgets with inst.	Position rule
Window A-1	Partical	Same
Window A-2	Different	Same
Window A-3	Same	Same
Window B-1	Different	Same
Window B-2	Partial	Same
Window B-3	Same	Same
Window C-1	Different	Different
Window C-2	Partial	Different
Window C-3	Partial	Different

5.3 Generating Programs

After the layout rules are defined, the source programs of the undeveloped GUIs are generated. Developers input items, such as the types of widgets, variable names of widgets, label names for each window, and widget grouping. Then the inputted data is compared to the layout rules. Window layouts are arranged when the layout rules are applicable to the widgets, and the source programs of the window are generated. Figure 12 shows an example arrangement when a developer inputs the following for a window:

Widget 1: Type: label widget, variable name: nameLabel, label name: Name
Widget 2: Type: text field widget, variable name: nameText, label name: n/a
Widget 3: Type: label widget, variable name: phoneLabel, label name: Phone
Widget 4: Type: text field widget, variable name: phoneText, label name: n/a
Widget 5: Type: button widget, variable name: ok, label name: OK
Group 1: Widgets 1 and 2
Group 2: Widgets 3 and 4

6. Evaluation

The evaluation consists of extracting and applying the layout rules. Three software packages are employed [i.e., a word processor (Microsoft Word[†], software A), a photo manager (iPhoto[††], software B), and an ftp client (FileZilla[†††], software C)].

6.1 Extracting Layout Rules

To confirm the efficiency of extracting layout rules, we evaluated the number of developed windows that our method requires to extract layout rules using three software packages. Although there are numerous windows in a software program, if the number of required developed windows is small, most undeveloped windows can be generated by our method.

In the analysis, the GUIs are implemented in the same way as the original software. Table 15 shows the minimum number of required windows to extract the layout rules from the left side of the menu items and the average value for each rule. For example, the position rules for some target widgets are for buttons such as "OK" and "Cancel", and their values are the average of the minimum number of windows necessary to extract the position rules.

The layout rules for the GUIs of software A, which is arranged strictly based on Windows User Experience Interaction Guidelines [3], show consistency, but the GUIs of software B and C do not. Some buttons with certain label names in software C are arranged in different positions. In addition, except for the rules in Table 15, the layouts are inconsistent in the three software packages. Because the expected results were not obtained, even for software A, more detailed layout rules need to be extracted to improve our method.

On the other hand, less than five windows are required to extract the layout rules for the "position rules" and "widgets with instructions" for all three test cases. Because the typical software package contains many windows, this is a small number. Although the effectiveness of our method is limited, it can easily extract the layout rules from developed GUIs.

6.2 Applying the Layout Rules

To confirm the appropriateness of the extracted layout rules in 6.1, we evaluated whether the layouts of the generated windows are the same as the actual windows.

We selected three windows for each software packages (nine total windows) and applied the extracted layout rules. Then we compared the generated windows to the actual windows (Table 16).

In this table, "Window *X*-*n*" indicates a window in software *X*. "Same" indicates that the arranged layout of the

[†]Microsoft Office: http://office.microsoft.com/
[††]iPhoto: http://www.apple.com/jp/creativity-apps/mac/
[†††]FileZilla: http://filezilla-project.org/

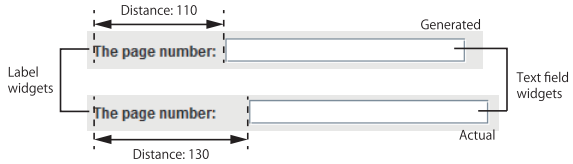

Fig. 13 Example of "Partial".

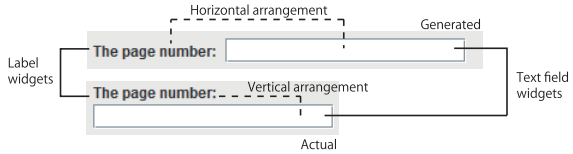

Fig. 14 Example of "Different".

generated window is almost identical to the actual window. "Partial" indicates that the order and arrangement strategy in the generated windows are the same as the actual windows, but the distances among widgets differ. "Different" indicates that the order or arrangement strategy of a set of widgets in the generated window is inconsistent with that of the actual window.

Figure 13 shows an example of "Partial". For a set of widgets in both the generated and actual window, the label widget is on the left and the text field widget is the on right (order), and these widgets are arranged horizontally (arrangement strategy). Thus, the order and the arrangement strategy are the same. However, the distance between the widgets differs, although the dll values should be unified.

Figure 14 shows an example of "Different". The set of widgets is arranged horizontally in the generated windows, but the widgets are arranged vertically in the actual windows.

The arrangements of GUIs in software A are more appropriate than those in software packages B and C because the GUIs in software A are consistently arranged, whereas those in software packages B and C are not. When GUI layouts are inconsistently arranged, extracting the layout rules is difficult, which likely affects the results for software packages B and C. Additionally, for the "widget with instructions" rule, the layouts of the seven windows are all "Partial" or "Different", suggesting that except for label widgets, the arrangement strategies may differ based on the widget type (e.g., text field and combo box widgets). To resolve these problems, grouping widgets in more detail (i.e., by widget type and window structure) may be more effective. However, Table 16 confirms that layout rules can be extracted almost appropriately using our method when the developed GUIs are consistent.

6.3 Comparing with Guidelines

The purpose of our method is to support the development of consistent GUIs. Developing GUIs using guidelines realizes consistent GUIs [1], [2]. To confirm the generated GUIs are consistent, we compared them to those generated using guidelines.

Because software A is arranged strictly based on Windows User Experience Interaction Guidelines [3], the windows of software A are compared with the guidelines. Although the guidelines have numerous elements, we extracted elements in terms of the meanings of consistency in our method and compared them to the generated windows. Examples of the extracted elements are as follows:

- Buttons that complete tasks (e.g., "OK" and "Cancel" buttons) should be arranged at the right bottom of window
- Width of buttons should be unified
- Checkboxes should be arranged vertically
- Widgets should be arranged by left alignment, etc.

Because software B and C are not strictly arranged by existing guidelines, we developed individual guidelines for software B and C. Concretely, guideline elements in terms of the meanings of consistency in our method are extracted from Windows User Experience Guidelines and Mac OS X Human Interface Guidelines [4], and all windows of software B and C are surveyed manually. Then the arrangement strategies of each guideline element are modified based on the surveying results. In cases where multiple arrangement strategies exist for a certain element, the most frequent strategy is adopted. Examples of the guideline element are as follows:

- For a set of label and other widgets, the label widgets should be arranged by right alignment and the others should be arranged by left alignment (Software B)
- Parent and child widgets should arranged horizontally, or when child widgets are arranged vertically, they should be indented from the parents (Software B)
- UI elements that complete tasks (e.g., "OK" and "Cancel" buttons) should be centered at the bottom of the window (Software C)
- Child widgets should be arranged vertically and indented from the parent widgets (Software C)

Figure 15 shows sample windows for this evaluation. The left window is the identical window strictly based on guidelines, while the right window is the window generated using our method. Although these windows are from the generic proxy configuration of FileZilla[†], we developed actual windows using Java Swing Packages for the comparison. The numbers in this figure are examples of guideline elements that should be applied; (2) ("radio buttons should be arranged vertically") is applied, while (1) ("size of boxes should be adjusted so that end users can input text without scrolling") and (3) ("UI elements that complete tasks should be centered at the bottom of the window") are not applied.

In Table 17, "Total" indicates the number of guideline elements that can be applied to each window. Because there are various types of elements, such as specific types of widgets and widget combinations, all elements cannot always

[†]FileZilla: http://filezilla-project.org/

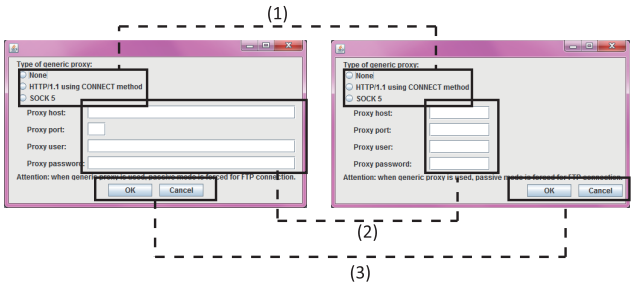

Fig. 15 Example of identical and generated windows.

Table 17 Results of the guideline comparison.

Window	Total	Applied	Not
Window A-1	54	38	16
Window A-2	44	34	10
Window A-3	48	34	14
Window B-1	34	26	8
Window B-2	38	29	9
Window B-3	39	33	6
Window C-1	41	31	10
Window C-2	32	23	9
Window C-3	35	31	4

be applied to each window. "Applied" indicates the number of elements that are applied to each window, while "Not" indicates the number of elements that are not applied.

"Not" elements can be classified as either applicable elements if our method is improved or inapplicable elements regardless of our method. Examples of applicable elements include "related controls should be arranged horizontally" and "the group with the longest content should be arranged at the horizontal center of the window". Although the former elements should be applied to sets of labels, buttons, and other widgets, the combination sub-rule "widgets with instructions" is applied to the label and other widgets, and the buttons are arranged vertically. Thus, our current combination rules are insufficient, but this issue can be resolved by increasing the number of combination sub-rules. For the latter elements, although the widget positions within a window, except for buttons in 4.1, are not considered in our current method, expanding the scope of method and extending the position rules will resolve this type of applicability issue.

Examples of inapplicable elements are "size of boxes should be adjusted so that end users can input text without scrolling" and "buttons that complete tasks ("OK" and "Cancel" buttons) should be arranged at the bottom center of the window". For the former element, it is impossible to recognize the volume of text in a box. Thus, our method cannot appropriately adjust the box size. For the latter, elements defined manually differ from the extracted layout rules (position rules). The buttons are arranged at the right bottom of the generated windows. This problem is due to the fact that the software program has an inconsistent layout strategy, and the layout rules are extracted using some but not all of the windows in the program. Although this problem may be resolved if more windows are used to extract layout rules, it is difficult in practice.

However, according to Table 17, 76% of the total elements are applied to the generated windows. Thus, we confirm that our method can generate consistent GUIs similar to applying guidelines.

7. Conclusion

Herein we proposed a method to automatically generate source programs of undeveloped GUIs by analyzing the source programs of developed GUIs and extracting layout rules. We surveyed existing software packages to identify common layout rules. Then source programs of GUIs were analyzed using various parameters based on the identified layout rules. Next the extracted rules were used to arrange undeveloped GUIs. Although the evaluation revealed some issues with our method, we confirmed that the layout rules could be extracted using only a few windows and that undeveloped GUIs are generated appropriately.

Our future research interests include improving the input widgets of undeveloped windows, increasing extractable and realizable layout rules, and arranging widgets across windows. In addition, we plan to group widgets and analyze layout rules in more detail.

References

[1] J. Nielsen, Usability Engineering, Morgan Kaufmann, 1994.
[2] J. Nielsen, Coordinating User Interfaces for Consistency, Academic Press, 1989.
[3] Windows User Experience Interaction Guidelines, http://msdn.microsoft.com/en-us/library/windows/desktop/aa511258.aspx
[4] Mac OS X Human Interface Guidelines, https://developer.apple.com/library/mac/#documentation/UserExperience/Conceptual/AppleHIGuidelines/Intro/Intro.html#//apple_ref/doc/uid/20000957
[5] J. Tidwell, Designing Interfaces, O'Reilly Media, 2010.
[6] J. Janeiro, S.D. Barbosa, T. Springer, and A. Schill, "Enhancing user interface design patterns with design rationale structures," Proc. 27th ACM International Conference on Design of Communication (SIGDOC '09), pp.9–16, 2009.
[7] S. Shirai, J. Shirogane, H. Iwata, and Y. Fukazawa, "Automatic generation of consistent GUI by analyzing developed windows," Procs. of IADIS International Conference Information Systems 2012, pp.446–448, 2012.
[8] A. Sajedi, M. Mahdavi, A. Pourshirmohammadi, and M.M. Nejad, "Fundamental usability guidelines for user interface design," Proc. of Computational Sciences and Its Applications (ICCSA '08), pp.106–113, 2008.
[9] M.Z.A. Obeidat and S.S. Salim, "Integrating user interface design guidelines with adaptation techniques to solve usability problems," Proc. 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE 2010), pp.V1280–V1284, 2010.
[10] P. Bendsen, "Model-driven business UI based on maps," Proc. of the 2004 ACM SIGMOD international conference on Management of data, pp.887–891, 2004.
[11] C. Lutteroth, "Automated reverse engineering of hard-coded GUI layouts," Proc. Ninth Conference on Australasian User Interface (AUIC '08), vol.76, pp.65–73, 2008.
[12] C. Lutteroth, R. Strandh, and G. Weber, "Domain specific high-level constraints for user interface layout," Constraints, vol.13, no.3, pp.307–342, 2008.
[13] D. Raneburger, R. Popp, and J. Vanderdonckt, "An automated layout

approach for model-driven WIMP-UI generation," Proc. 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '12), pp.91–100, 2012.
[14] S. Feuerstack, M. Blumendorf, V. Schwartze, and S. Albayrak, "Model-based layout generation," Proc. Working Conference on Advanced Visual Interfaces (AVI '08), pp.217–224, 2008.
[15] JavaCC Home, http://javacc.java.net/
[16] The AspectJ Projec, http://www.eclipse.org/aspectj/

Junko Shirogane received B.E., M.E., and D.E. degrees in information and computer science from Waseda University, Tokyo, Japan, in 1997, 1999, and 2002, respectively. She joined the Media Network Center of Waseda University as a Research Assistant in 2000 and the Department of Communication of Tokyo Woman's Christian University as a Lecturer in 2003. Currently, she is an Associate Professor of the School of Arts and Sciences, Tokyo Woman's Christian University. Her research interests include support tools for software development with GUIs. She is a member of IPSJ, IEICE Japan, JSSST, HIS, IEEE, and ACM.

Seitaro Shirai received B.E. and M.E. degrees in information and computer science from Waseda University, Tokyo, Japan, in 2010 and 2012, respectively. His research interests include support tools for software development with usable GUIs.

Hajime Iwata received B.E., M.E., and D.E. degrees in information and computer science from Waseda University, Tokyo, Japan, in 2002, 2004, and 2008, respectively. He joined the Media Network Center of Waseda University as a Research Assistant in 2005 and Department of Network and Communication of Kanagawa Institute of Technology as an Assistant Professor in 2008. His research interests include support tools to learn application operating methods. He is a member of IPSJ and ACM.

Yoshiaki Fukazawa received B.E., M.E., and D.E. degrees in electrical engineering from Waseda University, Tokyo, Japan, in 1976, 1978, and 1986, respectively. He joined the Department of Computer Science of Sagami Institute of Technology as a Lecturer in 1983 and the Department of Electrical Engineering of Waseda University as an Associate Professor in 1987. Currently, he is a Professor of the Department of Information and Computer Science, Waseda University. His research interests include software engineering, program optimization, and computer aided design. He is a member of IPSJ, IEICE Japan, JSSST, ACM, and IEEE.