User Interface Architecture with Abstract Interaction Description

Takuto YANAGIDA

Laboratory of Intelligent Information Systems
Research Group of Mathematical Information Science
Division of Computer Science
Graduate School of Information Science and Technology
Hokkaido University
Sapporo, Japan

Doctoral Dissertation
submitted to Graduate School of Information Science and Technology,
Hokkaido University

Contents

List of Figures

List of Tables

Chapter 1

General Introduction

The topic handled in this thesis relates to the field of human computer interaction, and user interfaces (UIs), which are sometimes called human interfaces. There are many studies about this topic; they are roughly separated into three types:

- developments of actual UIs such as new devices and new applications,
- evaluations or developments of evaluation methods for existing or newly introduced UIs, and
- offers of new kinds of frameworks, architectures, and concepts for UIs,

The study mentioned in this thesis mainly belongs to the third type, and its part belongs to the first one. In this chapter, I introduce the terms and their definitions used in the field of HCI, explain the objective of this thesis, and mention the structure of the following chapters of this thesis.

1.1 User Interfaces

1.1.1 What are user interfaces?

The term, *user interface* is widely used; what is its definition? User interfaces are the point of contact between human and systems [1], and they themselves behave as systems because some entities are contained there, which are interdependent with each other. Services, especially intermediated by computers in this thesis, are usually seen as systems which perform some functions based on the interaction between the systems and their users. For the users, because the all the user can see is UIs when they utilize the services, the UIs determine the all of look and feel of services. In this thesis, a UI is considered as a system consists of three layers: device, style, and service semantics (Figure 1.1).

Device corresponds to both hardware such as mouse, keyboard, and display of PCs; and driver software for controlling them.

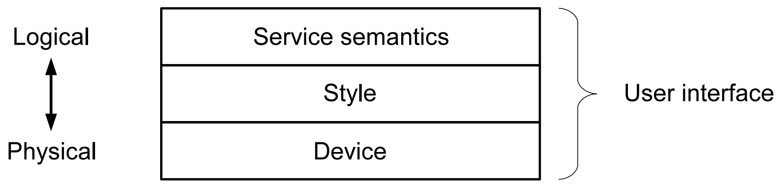


Figure 1.1: Three layers of a user interface as seen as a system. A user interface consists of the three layers of the concepts: device, style, and service semantics from physical one to logical one.

Style corresponds to CLI (command line interface), CUI (character-based user interface or character user interface), GUI (graphical user interface), voice input/output UI, etc.

Service semantics correspond to rules how to relate user operations to service functions of each service.

In comparison with the each layer, the device layer is the most physical, the style layer is more logical than it, and the service semantics layer is the most logical.

Along with the term UI, the term, *interaction* should be defined for the purpose of explanation of the difference between the two terms. Interaction is the communications between user and computers with verbal or nonverbal means [2]. Simple communications and complex ones which are combinations of them are expressed as interactions. On the other hand, UIs are defined as the medium of interactions. That is, there exist UIs as implementations corresponding to each interaction, and UIs are media for representing interactions. Therefore, descriptions of interaction represent the communication of users and services more essentially than the description of UI. This definition is important at the point that it shows there is a one-to-many relationship of interaction and UI (Figure 1.2). That reflects it is enable to construct various UIs from one interaction, and different interactions can correspond to the same UI. Hence, it is difficult to specify interaction from only the description of UIs. In this thesis, I try to generate UIs from descriptions of interactions, and thus, various UIs can be constructed.

1.1.2 Reason of various UIs

There is a simple question about UIs; is there any almighty UI for every person, every service, and every situation? There exist various UIs instead of an almighty one; this might be an answer of the question although I did not study this topic. Generally, various UIs are developed for the following three types of targets:

- specific purposes or services,
- specific contexts or situations, or

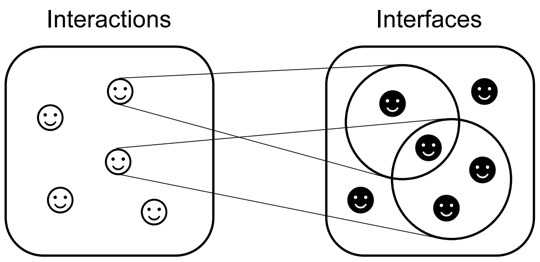


Figure 1.2: Relationship between interaction and UI. The both are in the one-to-many relations, and thus, one interaction can correspond to multiple UIs. In addition, different interactions can correspond to the same UI.

- specific users (who have specific characters).

As mentioned above, just on PCs, GUI and CLI are used according to the purpose done in the UI, and they are the example of UIs *for specific purposes or services*. To widen the range of services, vending machines of rail way tickets, drinks, etc. are also the examples of the UIs for specific services. They usually have control panels with buttons for selecting merchandises and levers for getting them, and those UIs are specific for the services offered by the machines. On the other hand, the users' contexts effect to the decision of used UIs. The frequently used word, *mobile* is used for expressing the context that users are out of offices and put on some devices. For the mobile context, mobile phones, mobile PCs, and PDAs are used, and they are the examples of the UIs *for specific contexts or situations*. The context is not limited to the mobile, for example, at least, the darkness in a theater or when a blackout is a kind of the context, and for this context, appropriate UIs might exist. Furthermore, there exist some UIs *for specific users*. UIs for people with disabilities are the example of those UIs, and in other words, many UIs are developed for people without disabilities. The UIs based on the concept of the universal design are developed *for all users*. Therefore, to finding the mighty UI for all services and all contexts are waste of labor.

There is a relation between various UIs for different targets mentioned above and *context-aware services*. Recently, the concept of context-aware services has been attracting attention as an approach to improving the usability of computer-mediated services [3, 4]. Especially in ubiquitous computing environments [5], context-aware services are expected to become more effective and beneficial. They enable users to utilize dynamically-composed and suitable service provided according to the users' contexts in ubiquitous computing environments. It is a role of context-aware services to determine automatically suitable ones from various UIs, which are categorized into the three types of targets. Therefore, there is a challenge of how to determine suitable UIs, which is a part of the research of context-aware services. In this thesis, it is not handled though related to the theme of the thesis.

1.2 General Objectives

1.2.1 Objectives

The objective of this study is offering *user-preferred UIs*; for that, I implement a new UI architecture, and develop a method of generating UIs based on the architecture. Then, I attempt to improve the traditional environment of UIs of services, and to make the environment of user-preferred UIs. In this thesis, the term user-preferred UIs is used for the following two meanings:

- an architecture that enables users to utilize services with their preferred UIs;
- UIs themselves that correspond to users' preferences for UIs.

The first meaning is addressed as the first topic of this thesis how to realize the situation where users can utilize their preference UIs for accessing any services. In addition, the second meaning is addressed as the second topic; I also develop an example of the user-preferred UIs. For strictly realizing the user-preferred UIs as an environment, I need to develop various UIs as studies, but, in this thesis, I confine them to one implementation and make others future studies.

This work consists of two elements and each one corresponds to Chapter 2 and 3 respectively. The first is to offer a new UI architecture, which realizes both flexible exchange of UIs of services, and flexible customizing. The second is to show an actual example of UIs based on the UI architecture, which offers a user preferred GUI. In the following paragraphs, I explain both how the work will contribute for users and service developers, and the image in the future of the work.

1.2.2 Image and contribution

For users, the work brings new service environment. Users themselves can be to change and to use UIs of the various services that use computers as the point of contact. Therefore, by that users choose and obtain UIs which matched with the users' taste for UIs, they can be to use the services conveniently. In addition, the operability improves at the point to get possible to perform higher customization after having changed it. Because the services pervade the everyday life of the user deeply, by this study that can apply to the whole services, the convenience can be improved in various situations in the life of the users. In addition, it is a characteristic of the proposal architecture not to limit to GUIs. Therefore, in consideration of various users' characteristics such as use environments and the taste for the interface, users can use UIs with devices and styles which match with then. Furthermore, users can use all services comfortably even if they had some kind of physical obstacles by obtaining a UI specific for it. At this time, in the meaning of not assuming the possession of the disabilities to be a special situation, it is important that becomes possible use by matching such a special UI to the situation even if the user does not have the disabilities.

For service developers, this work not only reduces the costs of service developments, but also improves the usability of the services. So far, developers had to make the program that did specific UI processing at each service, coding on each platform of service was needed. However, because it only has to describe abstract processing alone that doesn't depend on a specific interface by using the framework that this work offers, the decrease of the development cost can be attempted. Moreover, it decreases for service developers to consider for the improvement of usability, since it enables the user to change of the UIs.

For UI developers, this work increases the diversity of UIs that can be offered. The UIs are separated from services by the proposal architecture, and they come to cooperate with abstract descriptions. As a result, correspondence and the depending processing to individual services become unnecessary. Therefore, the possibility of achievement of more reformative UIs arises in the UI developments. Moreover, that the UIs can exist independently from services brings that it is possible to sales and to circulate only the UIs. Therefore, it comes to be able to circulate to the market even if it is a special UI, and it comes to be selected by the users.

In this thesis, in future, I assume the situation that the user in the scenario can freely choose the equipment for the system and he or she can buy it in shops. This is similar to the present relation to the mobile phones and the service that can be used by them. To receive the service of the same voice call etc., the user of the mobile phone can freely choose the portable terminal that each manufacturer except to the career offers based on convenience. The application of the proposal architecture as this infrastructure enables the service supplier and the UI supplier to exist independently. And, the environment that the user is freely combined each service and the UI and can used will be achieved.

This thesis consists of four chapters. In Chapter 1, as general introduction, I mentioned the terms used in this thesis and general objectives. Next, in Chapter 2, I explain a new UI architecture for the objectives, and in Chapter 3, I explain an implement of GUI of the architecture for user-preferred UI. Finally, I mention the general conclusion in the last chapter.

Chapter 2

Interface Client/Logic Server

In this chapter, a UI architecture named the interface client/logic server (ICLS) is proposed, which supports migratory user interfaces (UIs) and adaptive UIs for devices and services. As mentioned in the previous chapter, for realizing user-preferred UIs, it is needed to develop both an architecture that enables users to utilize services through preferred UIs, and UIs that correspond to users' preferences for UIs. The proposed architecture is architecture for the user-preferred UIs, and targets dialog-based interactive services like web applications, where some input facilities are used on some dialogs or pages updated as state transitions. In this chapter, the user-preferred UIs are separated to the following two users' demands. The first demand is to be able to use services through different devices and modalities and in accordance with certain contexts. Another demand is to be able to change devices and take their tasks from one device to another This is called migratory UIs. The ICLS is designed for adaptive and migratory UIs In the following sections, the detail of the ICLS is presented, and some implementations of the architecture are shown. After that, some issues about the ICLS are discusses.

2.1 Introduction

2.1.1 Background

Constant improvements in technology have spawned a large variety of platforms (such as mobile phones, PDAs, portable music players, and PCs) used for interactive services, and that has created users' new demands on user interfaces (UIs) of the services. The first demand is to be able to use a service through different devices with different modalities and in accordance with certain contexts. For example, depending on whether users are in their homes or cars, devices they want to use for checking their schedules will be different. I consider that UIs offered by devices should be device- and service-specific ones for the richness of usability, and I call them *adaptive user interfaces*. Another demand is to be able to change devices and take tasks from one device to another, which are also with different modalities. This is called *migratory user interfaces* [6]. When the users change their devices, they should not be obliged to repeat the same processes which have already been completed with the previous device.

Conventional ways of associating devices and services do not meet the users' demands because of costs and inadequate separation between services and devices. Simply developing multiple versions for each platform respectively is expensive for developers in terms of time and money, and it needs to spend labors for maintaining the versions consistency. Hence, in spite of the diversity of devices and platforms, users cannot utilize them effectively when accessing the services. Recently, web browsers have been implemented in many mobile devices, and the advent of web applications for these devices seems to have solved the problem. They need, however, specific versions in order to offer device-specific UIs for each platform, because web architectures including HTML still assume their target platform. Thus, they are often accessed selectively with different URIs. The conventional ways are discussed in detail in Section 2.1.3.2.

2.1.2 Objective

In this chapter, the following scenario is considered for explaining the concept of the proposal: *A man is staying in the Tapioca Hotel in a resort, and he is now in his room feeling a little bit hungry because it is six p.m. the time for dinner. He searches a menu of room service, but, instead of it, he finds there is a sign, which indicates that guests can access room service there. Thus, he uses his own gadget like mobile phone to access the room service. Through this service on the gadget, he can see the menu for dinner and he understands he can order dishes there, but he cannot see the detail of the dishes because of the spec. of his gadget. He looks around the room, and finds the same sign on a sophisticated TV there. He pushes a button on the gadgets toward the TV; therefore, the TV is turned on and shows the pictures of the detail of the dish he wants to order. By watching the TV, he can order his dinner operating his gadget.* To realize this scenario, the service can be accessed through different devices (the gadget and the TV) and can be moved between such devices continuing the use of the service.

In this thesis, a UI architecture named the *interface client/logic server* (ICLS) [7,8] is presented, which supports both migratory UIs and its extension; and offers adaptive UIs (Figure 2.1). The ICLS is one of model-based UI architectures [9,10], and leverages *logical descriptions* of UIs along with the other architectures. Its target is dialog-based interactive services like web applications, where some input facilities (such as buttons, check boxes, and text fields) are used on some dialog boxes or web pages updated as state transitions. In the ICLS, *interface clients* possessed by users are connected to *logic servers*, and the clients dynamically generate device-specific (client-specific) UIs from logical descriptions. The ICLS offers migratory UIs between the clients, which is movements and continuations of tasks between them (Figure 2.2). In addition, it offers *simultaneous UIs* as an extension of migratory UIs, where users can utilize multiple devices simultaneously. In the context of the scenario, the gadget and the TV are interface clients, and the service used for ordering dinner is a logic server. It is an example of simultaneous UIs that the man can order by watching the TV through operating the gadget.

For this UI architecture, a language for logical descriptions as an XML application, the *abstract interaction description language* (AIDL) is developed, which is designed for migratory and adaptive (device- and service-specific) UIs. The logical descriptions written in the AIDL (AIDL documents) are highly abstracted from specific devices and modalities. Therefore, devices (the interface clients) and services (the logic servers) can be developed separately and independently. For implementing migratory UIs, the AIDL is designed to retain the current state of generated UIs in the AIDL documents as well. For implementing adaptive UIs, especially for service-specific UIs, the AIDL is designed to enable developers to declare *UI meanings* in a machine-readable way. The meanings are the way to convey concrete roles of UI elements on each service to the clients from service developers. UIs are dynamically generated in client-side using the meanings, and thus, adaptive, or device- and service-specific UIs can be offered.

2.1.3 Related work

In this section, related work is categorized into two, one is work presented as studies, and another is existing technologies already used.

2.1.3.1 Studies

As a general solution for adaptive UIs, a broad range of research has been proposed, and almost all of it employs a model-based UI design [9, 10], which commonly utilizes logical descriptions. There are some studies that address static (on development times) and dynamic (in run-times) generations of UIs, and others that handle migratory UIs. To generate UIs from logical descriptions requires resolving a mapping between abstracted expressions and concrete UI elements. However, it is difficult for the techniques to represent service-specific UIs because the mappings are simple projections between typed variables and UI elements, especially GUI widgets, there. The proposal approach differs from existing work in terms of the representation of UI meanings in the AIDL.

A web migratory interface system was proposed in [6, 11]. It targets at arbitrary web applications and performs reverse engineering of web pages of the applications in order to obtain their logical information. This approach has the benefit of being able to handle existing web applications, but it does not suit for adaptive UIs. For example, which is the function of an HTML element <`button`> in a web application, a *navigator* or a *selection*? It is often used as one of the both functions, and which one it is used as is not understandable by an HTML document itself. Therefore, it is difficult to obtain information enough to represent an HTML element by other UI elements corresponding to its function.

The ubiquitous interactor (UBI) [12–14] addresses service-specific domains

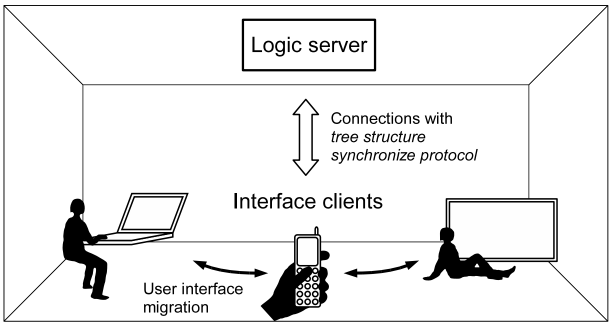


Figure 2.1: Interface client/logic server (ICLS) architecture. In the ICLS, interface clients generate and offer UIs to their users, and logic servers execute service contents. Servers and clients are connected with the tree structure synchronization protocol. Among the clients, the generated user interfaces are migrated from one to another.

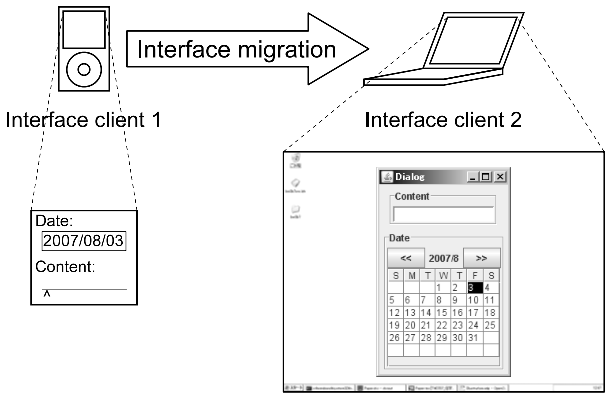


Figure 2.2: Concept of UI migration on the ICLS architecture. Services provided by the logic servers are moved between the interface clients, keeping the tasks on the service while the session of the service is continued.

with *customization forms* in logical descriptions for developing services independent of any devices. It is similar to the ICLS architecture in the point that it can represent various UIs for services using interaction acts as the elements of UI function. However, the UBI handles service-specific UIs in a different way that it uses specialized customization forms for each device for controlling presentations of UIs on specific devices. Since the customization forms have no portability among different devices, developers need to customize their descriptions for each device. Moreover, it does not address the UI migration.

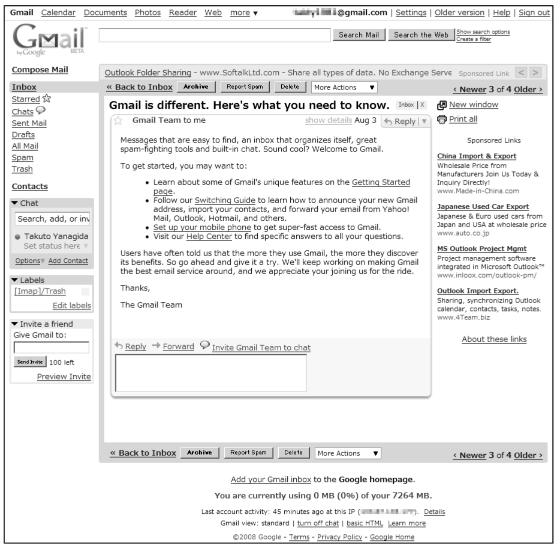
The personal universal controller (PUC) [15–17] was proposed for remote controlling various appliances with only PDAs. The PUC uses a mechanism *smart templates* for generating conventional presentations on some service domains. I think this mechanism is beneficial for generating service-specific UIs and similar to the meaning approach of the AIDL, but the difficulty of defining the smart templates is not mentioned. In addition, there is no consideration of migration there because the system targets at single sort of devices, PDAs.

User interface markup language (UIML) [18] is an XML-based language for developers to describe UIs independently of specific platforms. During the development period, developers need to define relationships between elements in UIML and other UI elements in an existing language such as Java or HTML. It is similar to the AIDL in terms of XML-based UI languages, but it does not support dynamic UI generation and UI migration in run-time.

As for other related work, I can also mention the migratory applications [19], the XWeb [20], the document-based framework [21], and the ICrafter [22]. Migratory applications offer a programming model for migration with distributed scripting language, Obliq, but this system is strongly dependent on the scripting language. Other studies have some similarities regarding the use of XML and adoption of a model-based approach, but it does not address UI migration.

2.1.3.2 Existing technologies

Recently, web applications have become widely accepted, and are expected as a new type of applications in comparison with legacy applications which need to be installed locally on PCs (Figure 2.3). In the web environment, many technologies such as Flash, JavaScript, XML, and CSS have been introduced, and they are used for enriching the web applications. Especially, the success of Ajax (the acronym for asynchronous JavaScript and XML) come to be recognized, accompanied by some practical examples like Gmail, a web mail service [23]. These web applications are sometimes mistakenly believed as the solution for the users' demands mentioned above, because they are accessible by web browsers on many devices. However, in fact, they need to be customized for each device, because the web technologies are not developed with the intention to be such an application platform. For example, Gmail is offered by four versions such as Ajax version, normal HTML version, mobile phone version, and a version for both iPod touch (a PDA-like portable music player) and iPhone (a smart phone).



Outlook Folder Sharing - www.SoftalkLtd.com - Share all types of data. No Exchange Serve Sponsored Link

Gmail is different. Here's what you need to know.

Gmail Team to me

Messages that are easy to find, an inbox that organizes itself, great spam-fighting tools and built-in chat. Sound cool? Welcome to Gmail.

To get started, you may want to:

- Learn about some of Gmail's unique features on the Getting Started page.
- Follow our Switching Guide to learn how to announce your new Gmail address, import your contacts, and forward your email from Yahoo! Mail, Outlook, Hotmail, and others.
- Set up your mobile phone to get super-fast access to Gmail.
- Visit our Help Center to find specific answers to all your questions.

Users have often told us that the more they use Gmail, the more they discover its benefits. So go ahead and give it a try. We'll keep working on making Gmail the best email service around, and we appreciate your joining us for the ride.

Thanks,

The Gmail Team

Sponsored Links

China Import & Export
Wholesale Price from Manufacturers Join Us Today & Inquiry Directly!
www.Made-in-China.com

Japanese Used Car Export
Japanese & Euro used cars from Japan and USA at wholesale price
www.auto.co.jp

MS Outlook Project Mgmt
Project management software integrated in Microsoft Outlook™
www.inloox.com/outlook-pm/

Outlook Import Export.
Sharing, synchronizing Outlook calendar, contacts, tasks, notes.
www.4Team.biz

About these links

Add your Gmail inbox to the **Google homepage**.

You are currently using 0 MB (0%) of your 7264 MB.

Last account activity: 45 minutes ago at this IP ([redacted]). Details
Gmail view: standard | turn off chat | basic HTML Learn more

©2008 Google - Terms - Privacy Policy - Google Home

Figure 2.3: Example of web applications. It is a screenshot of Gmail, the web mail service provided Google. This service is developed with Ajax, and used by web browsers on PCs, PDAs, mobile phones, etc.

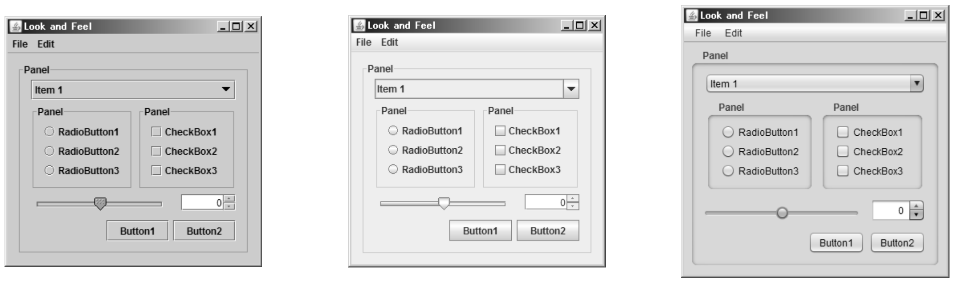


Figure 2.4: Examples of look and feel theme of the Swing toolkit. They are screenshots of the same dialog boxes made of Swing with different look and feel themes. The left is with Metal (Steel), the center is with Metal (Ocean), and the right is with Nimbus.

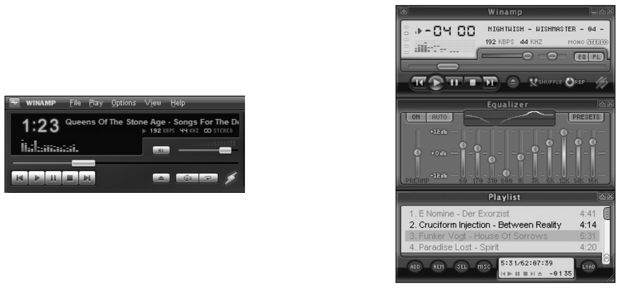

WINAMP
File Play Options View Help
1:23
Queens Of The Stone Age - Songs For The De
Winamp
NIGHTWISH - WISHMASTER - 04 -
Equalizer
ON AUTO
PRESETS
Playlist
1. E Nomine - Der Exorzist 4:41
2. Cruciform Injection - Between Reality 4:14
3. Funker Vogt - House Of Sorrows 5:31
4. Paradise Lost - Spirit 4:20

Figure 2.5: Example of skin function of a media player application. They are screenshots of Winamp, a popular media player application. The left is its default skin (`http://www.winamp.com/player/overview`), and the right is one of its skins (`http://www.winamp.com/skins/details/149846`). The skin function enables to change UIs but it limited in each application.

There are other related technologies mistakenly thought as to be the solution. The look and feel of Swing toolkit in Java language or the theme of window managers used in X window system is widely supported function for customize UIs (Figure 2.4). Needless to say, it is limited to GUI, and it targets literally the look and feel of GUI. As another similar technology, the skin function of applications especially media players can be mentioned here (Figure 2.5). The skins are used for customizing UIs more dynamically than the look and feel, but it limited to specific applications, and it is not shared among applications. In this way, these customizable UIs do not mean the adaptive UIs.

This chapter consists of six sections. In the first section, as introduction of this chapter about the UIs, the background, objective and related work were mentioned. In the following Section 2, the outline of the logical description language is mentioned, and how to describe interactions is explained. In Section 3, the outline of the architecture ICLS is explained, followed by the protocol between the clients and servers. In Section 4, implementations of framework of the ICLS and some clients and servers are shown. In the following Section 5, some consideration about the design of the architecture, the interpretation of meaning, and the feasibility of the ICLS are discussed. Lastly, in Section 6, conclusion and some issues remained as future work are mentioned.

2.2 Logical Description Language

2.2.1 Outline

In the ICLS, XML documents written in the AIDL are used as logical descriptions for describing UI functions for each services (Appendix A.2). The AIDL has been defined with a schema written in RELAX NG [24] (for the detail, see Appendix A.1). For describing UI functions in a general way, I have developed an interaction (UI functions) model between users and services, the *selection act model*. The AIDL is designed to specify UI information of services enough to realize migratory and adaptive (device- and service-specific) UIs.

In this section, one of the semantic web technologies, the resource description framework (RDF) [25] is introduced for adding the function of expressing meanings of UI functions of specific services into logical descriptions. The easiest way to address multi-devices and multi-platforms is by limiting, abstracting, and aggregating the functionalities of devices and platforms. This approach is adopted in many existing studies in common, and its details were provided in Section 2.1.3. Here, the same approach is also adopted; however, with the expression of UI meanings in services, the proposal system has an advantage of offering the possibility for combining service-specific and device-specific functionalities. In the AIDL, RDF classes are exploited for expressing UI meanings, and hierarchies of these classes are utilized for the inference of the meanings. RDF is a standardized web technology, independent of specific platforms and venders, and it is designed for addressing *opened information*, which is information not expected in advance. The AIDL inherits these RDF characteristics.

2.2.2 Interaction model

In the selection act model, UI functions are represented as *selection acts* with some parameters, and they are grouped to make a tree graph. The model consists of some selection acts (or elements), group elements, and their description elements. A selection element represents an essential function of UI elements that are in common among devices and modalities. Group elements make groupings of relevant selection elements and group elements as child elements (Figure 2.6). In addition, selection elements and the group elements can have description elements for their explanations and a flag expressing whether the elements are enabled or not.

A set of UI elements in the model is expressed as $U = U_\mathrm{S} \cup U_\mathrm{G} \cup U_\mathrm{D}$, where U_S is a set of selection elements, U_G is a set of group elements, and U_D is a set of description elements. Selection element $u_i \in U_\mathrm{S}$ is represented as a 5-tuple:

$$u_i = \langle L_i, e_i, t_i, o_i, m_i \rangle, \tag{2.1}$$

and its current state. In the tuple, L_i stands for the typed list of choices, and $|L_i|$ stands for the number of choices. $e_i \in \{\mathtt{single}, \mathtt{multiple}\}$ is the selection size, $t_i \in [1, 10]$ is the importance, $o_i \in \{\mathtt{true}, \mathtt{false}\}$ denotes the flag meaning whether its choices are opposite when they have two choices, and m_i denotes meaning (a purpose in a service). For instance, a selection element representing the operation of the power state of a desk lamp is as follows: $u_1 = \langle L_1 = \{\mathtt{ON}, \mathtt{OFF}\}, e_1 = \mathtt{single}$, $t_1 = 1$, $o_1 = \mathtt{true}$, $m_1 = \mathtt{LampPowerState}\rangle$, and current state is $\mathtt{ON}$. Note that the selection elements are not mere typed variables, because the elements contain those parameters which are specific for generated UIs based on them.

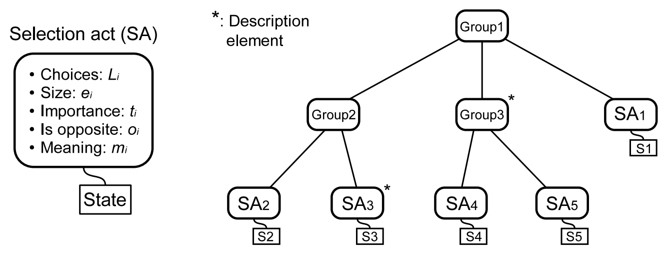


Figure 2.6: Selection act model expressed in an AIDL document. A group element makes both parental relationship between itself and its child elements; and sibling relationship among them. All selection elements are grouped as a tree structure recursively. The root of the tree corresponds to a group.

2.2.3 Semantic web

As mentioned above, it is efficient to handle the meanings of interactions for describing interactions, but it is difficult to enumerate beforehand all of the meanings of selection elements in the real world. Therefore, the semantic web technologies are introduced into the AIDL. The semantic web is a framework for constructing machine-analyzable web by improving simple hyperlink structure in HTML and adding semantics for the hyperlinks [26, 27]. In this section, the outline of the semantic web and one of its components, the resource description framework (RDF) are explained, and how to introduce them into the work is mentioned.

For handling semantics with extending traditional web, it is necessary to add the function that the values indicating relationships can be appended to the concept of hyperlinks, and these values should be machine-readable. Hence, the semantic web is designd based on its design principle, which is not standardized officially, but it has been organized as follows [27, 28]:

Everything Identifiable is on the Semantic Web It must be able to handle not only resources on networks but also physical resources such as peoples, places, and event comprehensively with assigning URI (uniform resource identifier) to each entity.

Partial Information It must enable everyone to mention everything in any wise in a similar way of traditional web, which handles partial and opened information of the real world, which can be known partially.

Web of Trust It must offer for each application the mechanism for evaluating how veridical the information on the web is, but not to declare all of it is truth.

Evolution It must be able to enable both to effectively combine information offered by distributed and independent communities, and to add new information with keeping old information untouched.

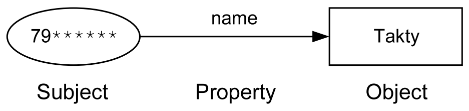


Figure 2.7: Example of a RDF statement. A statement consists of a subject (the oval), a property (the arrow), and a object (the rectangle). A subject can have multiple properties, and if an object is a resource, the object can have another property as a subject.

Minimalist Design It must take a course of keeping simplicity and to simplify complex things as possible, and avoiding too much standardizing.

RDF is a data model for representing information about resource on the web, and it allows adding values representing relationships, to links, which are the components for constructing network structures of documents like HTML. It represents information as statements which consist of a subject, a property, and a object (Figure 2.7), and the set of the statements is called a RDF graph. It handles subjects, properties, and objects as resources, which are represented uniquely by URI. Strings and values except resources are handled as literals. By using this naming mechanism, RDF graphs made by different people can be merged without discrepancy. In addition, RDF schema is standardized for defining RDF vocabularies, which are both the values representing the relationships, and classes meaning the type of resources. The class is the method for grouping resources that have relevant characters, and moreover, the relationship of subclass can connect classes. Resources belong to the class are called instances of the class.

2.2.4 Description

2.2.4.1 Design

The AIDL is newly developed from scratch for supporting the concept of the ICLS. It is designed based on the following three principles:

Comprehensive description independent of specific UIs The AIDL has to handle various UIs in a unified way, and thus, descriptions in it cannot be dependent on specific UIs. It describes interaction information common to every UI, and abolishes other inherent information of specific UIs and media information, for which UIs have deferent capability.

High extendability for the real world The AIDL has to deal with choices and meanings of selection elements in the real world. These cannot be defined beforehand because they differ by services. By adopting RDF, it become enable to define partial information in the real world according to need, and thus, interactions of various services can be described.

Machine-readability enough for automatic UI generation The AIDL has to be well-portable as a language, and has to contain information enough for automatic UI generation. Because RDF is a standard independent of specific platforms, it can be usable on various devices, and its logicality can be exploited for automatic processes.

The selection acts, choices, current states, and groups can be seen as XML nodes in an AIDL document. Each selection node has its own selection state node as a child node, and thus, the current state of the tree graph is represented with all of the state nodes. For interface clients, the nodes expressing the structure of interactions are immutable (meaning they are not modified), and other nodes for expressing the current states are mutable (meaning they are modifiable), while all nodes are mutable for logic servers. Thus, the tree represents a current state of a service interaction with the states of the selection acts.

2.2.4.2 Selection elements

As shown below, a selection element is expressed by `<aidl:selection>` and `</aidl:selection>` tags, which hold child elements expressing a choice list, a current state, and a description element:

```
<aidl:selection aidl:meaning="...">
  <aidl:description ... />
  <aidl:state> ... </aidl:state>
  <aidl:resources> ... </aidl:resources>
</aidl:selection>
```

In the following three subsections, I explain the detail of the constituents of the selection acts, typed lists of choices, current states, and meanings in turn.

Typed choice lists A type of choice lists is one of basic types (resources, numeric, and strings), and they are expressed as following tags: `<aidl:resources>`, `<aidl:numeric>`, and `<aidl:strings>`. Here, the resources mean a set of things specified with unique URI on the concept of semantic web, and the numeric and strings mean a set of literals (values and strings) of RDF. For expressing arbitrary subsets of these basic types, *subtypes* are introduced. A subtype is constructed with one of the basic types and one of constraints: *enumeration* and *range* (Table 2.1). The enumeration is used for enumerating all used choices, and the range is used for specifying maximum and minimum values. In the above example of a desk lamp, the selection element contains a `<aidl:resources>` added an enumeration constraint which specifies its two choices {`On`, `Off`} as follows:

```
<aidl:resources aidl:opposite="true">
  <aidl:choice aidl:uri="http://www.example.com/On">
    <aidl:description aidl:caption="On"/>
  </aidl:choice>
  <aidl:choice aidl:uri="http://www.example.com/Off">
```

Table 2.1: Basic types and permissible constraints

Basic types	Permissible constraints
Resources	Enumeration
Numeric	Enumeration, Range
Strings	Enumeration

```
    <aidl:description aidl:caption="Off"/>
  </aidl:choice>
</aidl:resources>
```

In this example, the property `aidl:opposite` expresses parameter o_i of the corresponding selection element. For another example, a volume control of audio instrument is described as a selection element having a <`aidl:numeric`> added the range constraint such as $\{0 \leq n \leq 10\}$ as follows:

```
<aidl:numerics aidl:frequency="1" aidl:min="0" aidl:max="10"/>
```

Furthermore, for the input of contents without any limitation like to-do items of a PIM service, a <`aidl:strings`> without any constraint is used since it is difficult to enumerate choices and to specify its range.

Current states A current state represents what choice is selected, and it is one of choices contained in a choice list and expressed by <`aidl:state`> and </`aidl:state`> tags. It changes according to user's selections, and thus, the current states of all selection elements express the current state of interaction described in an AIDL document. When the type of a selection element has an enumeration or range constraint, its state satisfies this constraint. Otherwise, when the type has no constraint, the state can be an arbitrary element in the type. In the desk lamp example, the selection element has the choice `On` as its state, which means literally that the state of the lamp is on, and expressed as follows:

```
<aidl:state>http://www.example.com/On</aidl:state>
```

This current state mechanism is designed for the UI migration, and it expresses literary the state of UIs. The migration is explained in Section 3.

UI meanings A UI meaning is represented as the URI of an RDF class, and is exploited to specify the purpose of selection elements and to relate them to service-specific UI elements such as custom widgets on GUI, specific interaction methods, and physical facilities on devices. It is expressed in an AIDL document by `aidl:meaning` property of <`aidl:selection`> tag as follows:

```
<aidl:selection aidl:meaning="http://www.example.com/
    LampPowerState">
  ...
</aidl:selection>
```

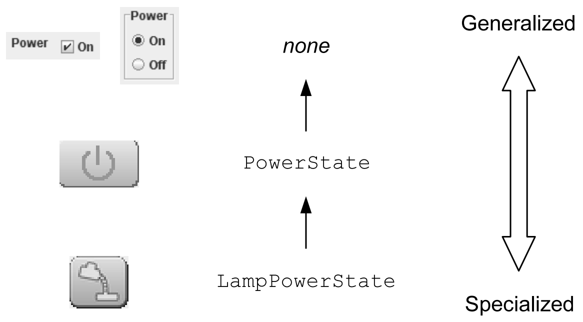


Figure 2.8: Example of meaning hierarchy and corresponding service-specific UI elements. In the example, `LampPowerState` is a sub class of `PowerState`, and `PowerState` is a sub class of *none* meaning nothing is specified. It represents that the selection element with `PowerState` is substituted for the element with `LampPowerState`.

UI meanings can compose their general-specific relationship as a hierarchical structure of RDF classes corresponding to the meanings (Figure 2.8). AIDL documents can include an RDF document which represents the meaning hierarchy. The embedded RDF document can be seen surrounded by <`aidl:knowledge`> and </`aidl:knowledge`> tags as follows (see also Appendix A.2):

```
<aidl:knowledge>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
      ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <rdf:Description rdf:about="http://www.example.com/
        LampPowerState">
      <rdfs:subClassOf rdf:resource="http://www.example.com/
          PowerState"/>
    </rdf:Description>
  </rdf:RDF>
</aidl:knowledge>
```

UI meanings are beneficial in order to increase the concreteness of the descriptions of selection acts and to adapt generated UIs to services. Note that the meaning hierarchy is independent of the relationship of the types and their subtypes. In the desk lamp example, the meaning `LampPowerState` added in the selection element represents that its purpose is the operation of the power state of lamps.

2.2.4.3 Other elements

A group element is expressed by <`aidl:group`> and </`aidl:group`> tags, which has its child elements as the child elements of the tags, and can be specified the ordering of its child elements (Table 2.2). For example, a group having two selection elements and one group element which are ordered is expressed as follows:

```
<aidl:group aidl:ordering="ordered">
```

```
  <aidl:description ... />
  <aidl:selection> ... </aidl:selection>
  <aidl:selection> ... </aidl:selection>
  <aidl:group> ... </aidl:group>
</aidl:group>
```

The property `aidl:ordering` specifies how child elements are ordered, and it is used by the interface clients for aligning the elements. When a description element is added as a child of the group element, the child represents the description of the group. If a group element is the root of a UI model, it is expressed by `<aidl:dialog>` and `</aidl:dialog>` tags, instead.

A description element is expressed by a `<aidl:description/>` tag, which has some properties for holding its description contents. For example, a description having a caption and an abbreviation of the caption is expressed as follows:

```
<aidl:description
  aidl:caption="Power␣switch"
  aidl:abbr="PS"
/>
```

The tag can have the following four properties: `aidl:caption`, `aidl:abbr`, `aidl:message`, and `aidl:resource`, which are all information about the parent element of the tag. The first property expresses literary the caption of an element owning the tag, and the tag must have this property. Property `aidl:abbr` expresses the abbreviation of the caption, and property `aidl:message` expresses an additional explanation of the parent element. Property `aidl:resource` is the URL of an outer resource used for describing the element owing the tag.

2.3 Interface Client/Logic Server

2.3.1 Clients and servers

The ICLS architecture is designed based on the client/server model, and consists of interface clients and logic servers. The term interface clients stand for various devices and platforms in which ICLS client applications are implemented. By the applications, UIs are dynamically generated in the clients to be adapted to clients'

Table 2.2: Orderings of the child elements of a group

Orderings	Explanations
`none`	Not ordered.
`ordered`	Arbitrarily ordered.
`subject_predicate`	Ordered in a subject-predicate relation.
`time_line`	Chronologically ordered.
`writing_direction`	Ordered in a writing direction.

capabilities and service contents of connecting servers. On the other hand, the term logic server stands for various services in which ICLS server applications are implemented. Both applications for the clients and servers are assumed to conform to the specification of the ICLS, the details of which are provided later. Once devices or services are developed in accordance with the specification, no revisions are required when a new service or device is introduced. Because UI processes are depend on service contents provided by the processes, it is generally difficult to separate the whole of UI processing from service codes, although many architectures and frameworks attempt it. Therefore, I separate only device- or modality-specific UI processes and incorporate them into the interface clients.

In the ICLS, AIDL documents are not only used for descriptions of UI functions, but also they are used for communication between clients and servers. After a client generates a UI based on an (original) AIDL document received from a server, the client and server keep the DOM (document object model) tree constructed from the document. They update the DOM tree, and communicate messages, where the changes of the tree are serialized as deference of the AIDL document. By this synchronization process, clients and servers virtually share the same DOM tree constructed from an original AIDL document. DOM trees from AIDL documents also contain the current states of generated UIs. Therefore, receiving a DOM tree which is being used by another client means receiving the whole information about all of the UIs in a session. That enables clients to attach existing sessions for UI migration by simply communicating an AIDL document.

Figure 2.9 shows the flow of a service session between an interface client and a logic server, and Figure 2.10 shows the flow of a migration process between two interface clients. During the time a user is accessing a service through one client in a session, the user can also access the same service through another client in the existing session. At that time, the DOM tree of the first client is transferred to the second client, and the tree of these clients is synchronized and updated simultaneously. Then, a migration is performed when the first client is disconnected. When a client establishes a connection in a session, the server issues a unique session ID to the client. This ID is used when the user access the same session again.

By using the scenario (Section 2.1.2), a use case of the UI migration is shown as follows: In the scenario of a room service, *a man uses his own gadget like mobile phone to access the room service.* Here, this phone-like gadget is an interface client, and the room service is provided as a logic server installed in the room. *Through this service on the gadget, he can see the menu for dinner and he understands he can order dishes there, but he cannot see the detail of the dishes because of the spec. of his gadget.* He uses the client for accessing the room service, and the client is not enough for his purpose. *He looks around the room, and finds the same sign on a sophisticated TV there.* This TV is also an interface client in this scenario. *He pushes a button on the gadgets toward the TV; therefore, the TV is turned on and shows the pictures of the detail of the dish he wants to order.* This operation is an example of a procedure for starting simultaneous (or migratory) UI between the two clients. *By watching the TV, he can order his dinner operating his*

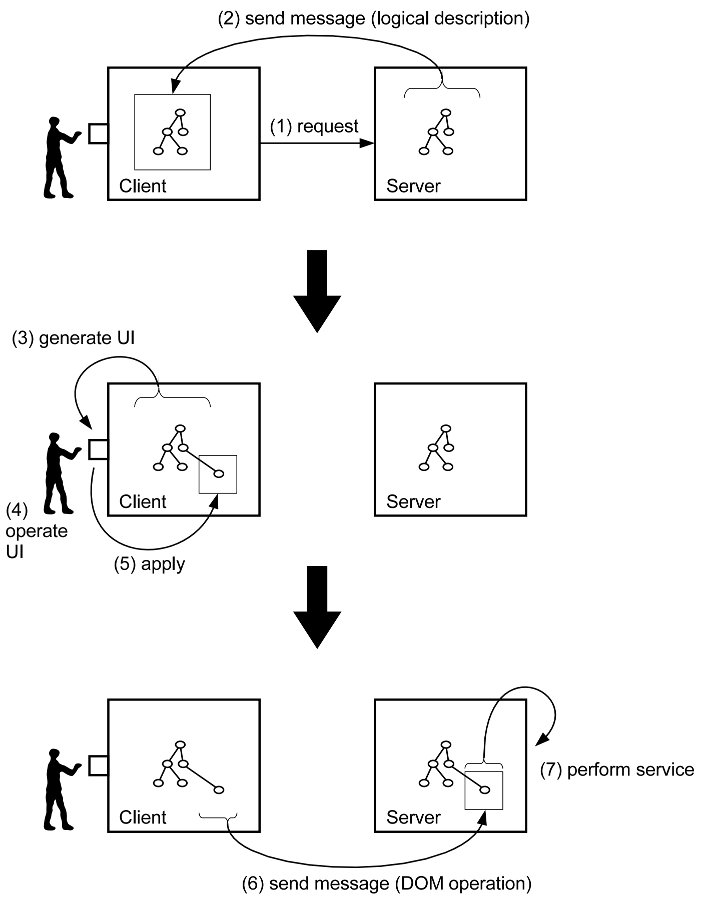


Figure 2.9: Process flow of communication between an interface client and a logic server. (1) A service session starts with a request from a client to a server. (2) After the session starts, the client receives an AIDL document from the server. (3) The client constructs a DOM tree from the document, and generates a UI based on the tree. (4) After the UI is generated, the user can operate the UI. (5) The client changes the DOM tree corresponding to the user's UI operation. (6) The client sends these DOM changes as messages to the server. (7) The server performs the service according to the messages from the client.

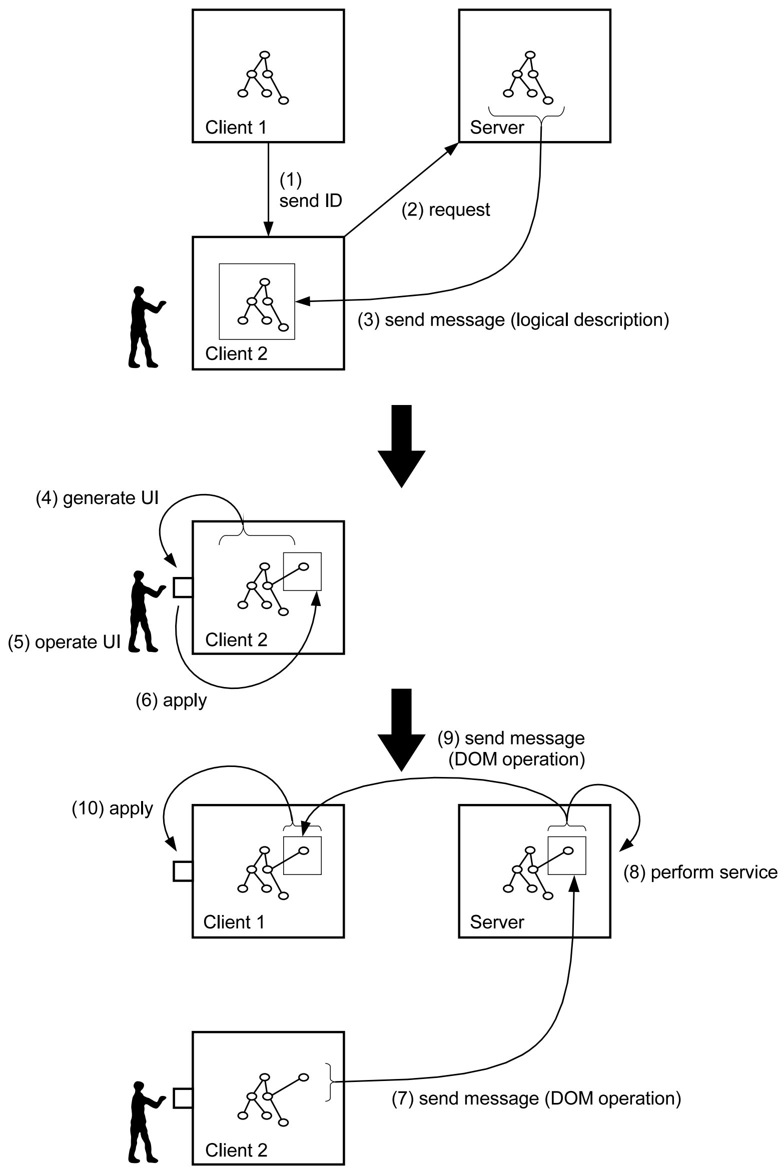


Figure 2.10: Process flow of a migration from the client 1 to the client 2. (1) A new client (client 2) obtains a session ID from the existing client (client 1). (2) A simultaneous service session starts by a request with an ID from a client to a server. (3) After that, the client receives an AIDL document used in the session from the server. (4–8) The new client executes the same UI generation process as Figure 2.9 and sends the user's operation to the server. (9) After performing the service, the server broadcasts the message to another client. (10) Client 1 receives the message, and applies it to the client's DOM tree and UI.

gadget. He uses the two clients simultaneously here.

2.3.2 Protocol

Interface clients and logic servers communicate in accordance with the protocol, *tree structure synchronization protocol* (TSSP), which establishes how to express changes in DOM trees as messages for asynchronous communication. In the protocol, changes in DOM trees are serialized as XML-based messages containing whole or partial AIDL documents in order to synchronize two or more DOM trees. This synchronization brings about the virtual sharing of one DOM tree, which is changed by clients and the server. To avoid conflicts of messaging when a server receives from different clients multiple messages, among which there are some discrepancies, the following priority is given:

1. the state of the server takes priority over its clients, and

2. the message that arrives first takes priority over the other messages.

Note that the lower protocols under the TSSP are not specified, and many existing protocols which can send text messages are available.

Messages exchanged with the TSSP are categorized into the following two types: serialized change operations, and just sending whole AIDL documents. A serialized change operation consists of 3-tuples, whose elements are

- one of the three commands (`insert`, `erase`, or `replace`),

- an XML path (such as "`/group[0]/selection[1]`"), and

- a piece of AIDL document

(Figure 2.11). A user's operation changes the current state of a selection element of a DOM tree in a client, and the client sends the change to a server as a message. The server receives the message, applies it to the server's DOM tree, and broadcasts

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<icls:tssp xmlns:icls="http://.../iclsns/">
  <icls:replace icls:path="/aidl:dialog/aidl:selection[1]/aidl:state">
    <aidl:state xmlns:aidl="http://.../aidlns/">http://.../On</aidl:state>
  </icls:replace>
</icls:tssp>
```

Figure 2.11: Example of messages used in the tree structure synchronization protocol (TSSP). It expresses a DOM tree change when a user operates the power state of the desk lamp control service written in Appendix A.2. It replaces the state of the selection act for the power state in the server-side DOM tree.

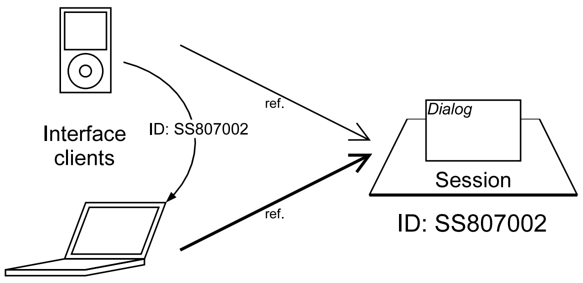


Figure 2.12: Sharing a session using the session ID. With the session ID, multiple clients can refer the same session. Clients need to communicate with each other to obtain this ID from other clients for the connections with the session IDs.

them to the other clients if it has multiple connections. The server then expresses the state transitions of UIs by sending entire new AIDL descriptions to the client(s).

Attaching an existing session for simultaneous or migratory UIs is performed by passing session IDs, provided in a certain way when a client is connected to a server (Figure 2.12). When a client is connected with an ID, the server sends the DOM tree of the session identified with the ID as a AIDL document to the client. In that case, the DOM tree is shared and synchronized by two clients and a server. The means to transfer the session ID among clients is not defined in the protocol, and this is an issue discussed in Section 2.5.

2.3.3 Interpretation of UI meanings

UI meanings are *inferred* based on the hierarchy of RDF classes, which is applied to the general-specific relationship of the meanings. Therefore, interface clients can address more meanings than ones actually implemented (Figure 2.13). The meanings are just labels for concepts of UI functions shared by developers of both clients and servers. Hence, they require to be interpreted to UI elements actually generated, and client developers need to determine which meanings are adopted and implemented. While it is a common problem in related work including the study of the PUC [17], the proposal system handles this problem with the meaning inference. UI meaning (class) hierarchy can be enlarged by *merging* RDF graphs that the clients obtain from the servers and the web. That is possible because RDF has the namespace mechanism on the web, and those graphs can be merged ensuring their consistency. On client side, the step-by-step process of meaning interpretation for the generation of UIs is as follows:

1. An interface client searches itself for implementations of UI corresponding to the given meaning.
2. - When the client finds some implementations, it employs one of them for generating UIs and ends this interpretation process.

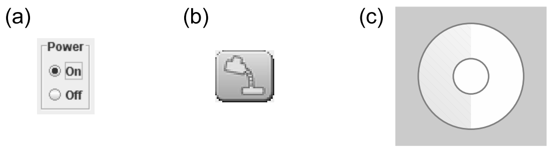


Figure 2.13: Results of UI generation in each client. From the same AIDL document (Appendix A.2), a GUI-based interface client can generate (a) two radio buttons without any meanings and (b) a custom-designed button with the meaning if it has its implementation. A portable client like a music player can generate a UI that corresponds to (c) a hardware switch on the device with the meaning `PowerState` inferred from the given meaning.

- When the client cannot find any implementation, it downloads some RDF documents about the meaning from web or internal database.

3. The client merges some downloaded RDF documents and generates one or more class hierarchy tree(s).

4. The client traverses `rdfs:subClassOf` property from the given meaning to its ancestors until it reaches an implemented meaning (inference).

5. - When the client finds some implemented meanings, it employs one of its implementations and ends this process.
   - When the client cannot find any implemented meaning, it renders the most general UI regardless of the specified meaning.

2.4 Implementations

2.4.1 Framework

The ICLS is implemented as a framework, which is a class library written in Java language (the Java development kit 1.6) with a semantic web library Jena 2.5.2 [29] (Appendix B). It consists of 35 classes and 12 interfaces, and offers event-driven programming model like GUI toolkits. In this framework, although TCP/IP is adopted as the default lower protocol under the TSSP, other communication protocols can be usable. Although the specific language and library, Java and Jena are adopted, this does not imply any dependency since the technologies which we adopted are well standardized. For example, the library for PC can be easily ported to cell phones which support Java applications.

It is easy to develop the interface clients and logic servers with the ICLS library. For developing GUI dialogs with toolkit like Swing, in many cases, some widgets are composited, and event handlers are added into the widgets for customizing widgets' operations. In development with the ICLS library, selection elements, group elements, and description elements are created and composited, and in the same way, event handlers are added into the elements. User operations to the selection elements are automatically applied to the corresponding AIDL documents represented as classes as well. The communications between the interface clients and the logic servers are capsulated into the classes, `InterfaceClientProxy` and `LogicServerProxy`. Therefore, developers can simply develop the clients and servers respectively without implementing the TSSP.

2.4.2 Clients and servers

In order to verify the feasibility of the ICLS specification and the stability of the communication protocol, two interface clients and two logic servers are developed with the ICLS library. The two clients are a graphical user interface (GUI) client (Figure 2.14, 2.15) and a voice output client (Figure 2.16). The two servers are services of remote controller of virtual appliances: a desk lamp and an audio system. Although these clients and servers are simple and small in-scale, they include the essence of more general and typical applications. In addition, the two clients show that the ICLS can handle the different modalities comprehensively. In this implementation, the drag-and-drop mechanism of the desk top environment offered by Java is used for passing the session IDs for UI migration. This is just for the experiment of the implementation of UI migration, and it is future work to develop an actual inter-client communication protocol.

The GUI client generates GUI dialog boxes using Swing toolkit in Java language, according to AIDL documents received from logic servers (Figure 2.14, 2.15). Automatic GUI generation from AIDL documents in client-side requires both (1) deciding which widget types and their positioning are used and (2) completing the layout immediately. I call the widget layout satisfying the two conditions, *flexible widget layout* (FWL) problem, and propose its solution in Chapter 3. This client is an example of device-specific UIs in GUI generation, and it is a benefit which comes from the design of the AIDL. In this client, I implemented meanings `LampPowerState`, `PowerState`, and `PlaybackController` as examples. Its users can customize whether or not the client uses these meanings. In addition, the client has a simple mechanism for inferring the meanings with given RDF class hierarchies written in AIDL documents.

The voice output client offers hierarchical menu UIs with voice synthesis and gets input from buttons (in a key board in this implements) (Figure 2.16). For the output of voice sounds, I adopt the library, FreeTTS 1.2 [30], which is an implementation of Java speech API [31]. It is assumed that this client is an example of an interface client implemented on mobile devices like cell phones, which have a speaker and a key pad, although currently it is implemented on PC. Recently, voice input has become available, but in mobile environments, the combination of voice output and key input might be still reasonable.

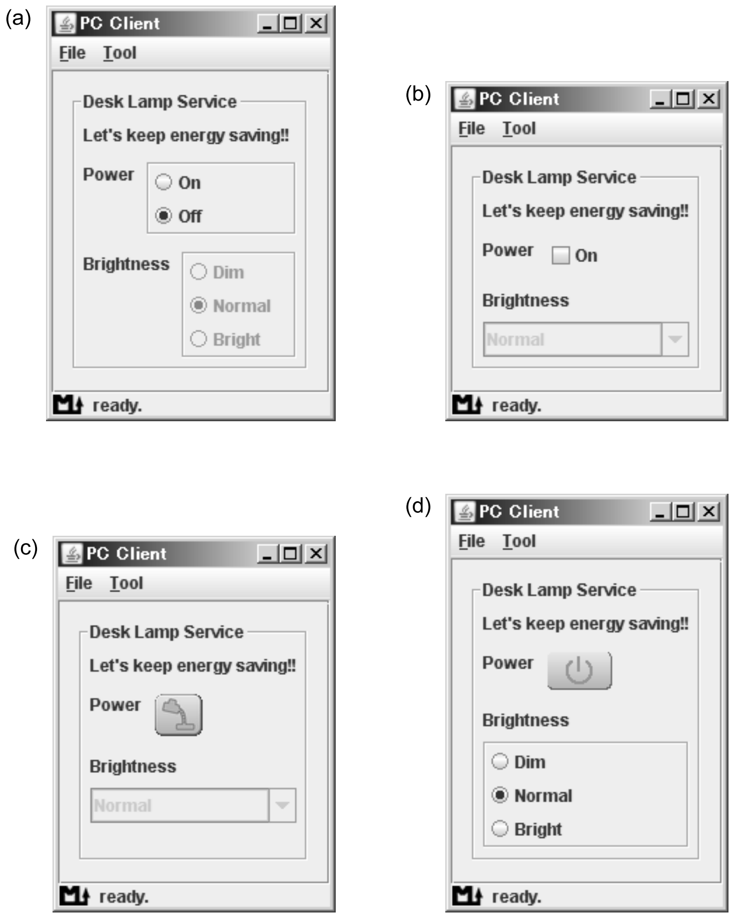


Figure 2.14: Screenshots of the GUI interface client. They are ones when the client connects to the desk lamp control service and its window size is changed. It is seen that the appropriate widgets (radio buttons, a custom button, and a check box) are selected automatically for the same UI function (the power switch) according to the window size. Figure (d) is one when the widget for `LampPowerSwitch` is disabled, and the widget for `PowerSwitch` is used instead.

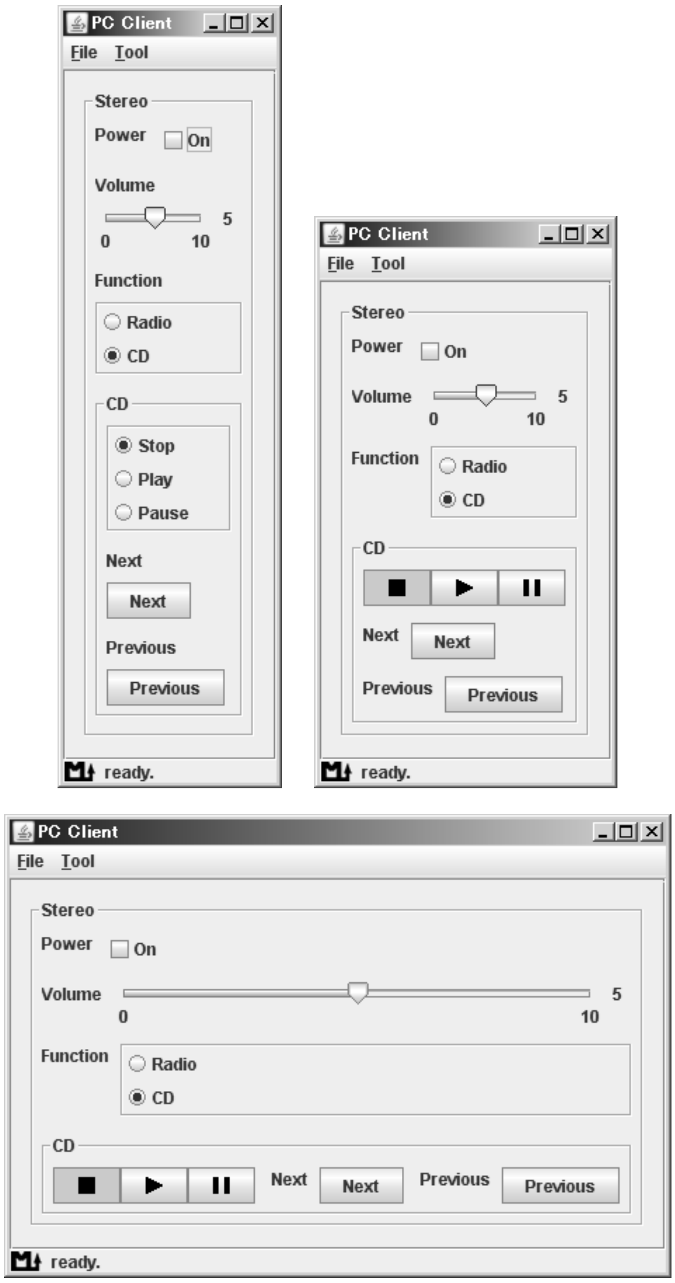
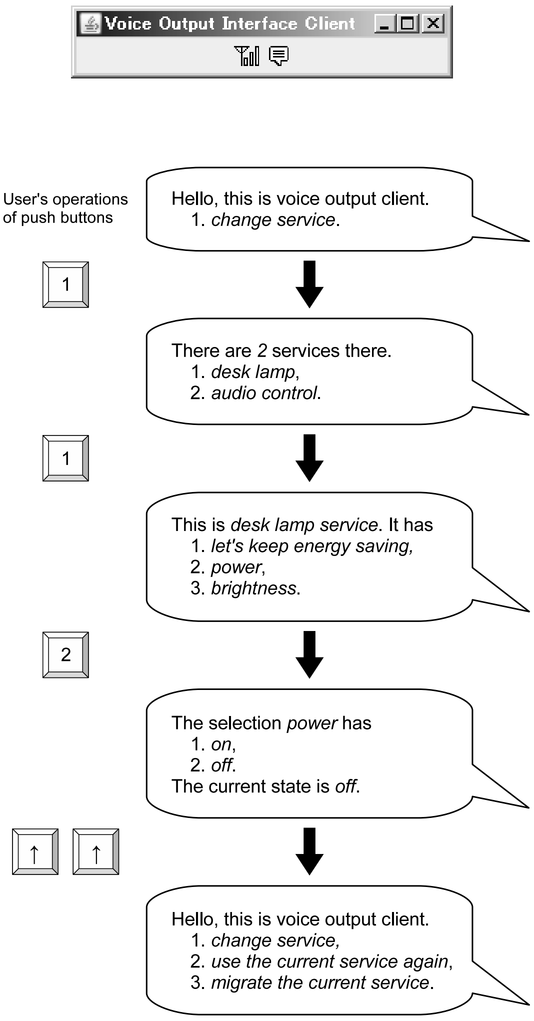


Figure 2.15: Other screenshots of the GUI interface client. They are ones when the client connects to the audio control server and its window size is changed. It is seen that the appropriate widget (a slider) is selected for the volume control. The selection element of the playback has meaning `PlaybackController`, and the client has an implementation; thus, three buttons with appropriate icons are shown.


Voice Output Interface Client
User's operations of push buttons
1
1
2
↑ ↑

Hello, this is voice output client.
1. *change service*.

There are *2* services there.
1. *desk lamp*,
2. *audio control*.

This is *desk lamp service*. It has
1. *let's keep energy saving,*
2. *power*,
3. *brightness*.

The selection *power* has
1. *on*,
2. *off*.

The current state is *off*.

Hello, this is voice output client.
1. *change service,*
2. *use the current service again*,
3. *migrate the current service*.

Figure 2.16: Screenshot of the voice output client connecting to the desk lamp control server and the flow of its use. In the flow, the speech balloons represent the voice output of the client, and the buttons in the left side mean its user's key typing.

2.5 Discussion

2.5.1 Design

I am emphasizing the diversity of devices and platforms, and consequently, have designed architecture in which the servers handle all functionalities except for specific UIs, and I have focused on the aspect of input.

As mentioned in Section 3, the ICLS is designed so that interface clients handle only device- and modality-specific UI processes, leaving other UI processes in logic servers. You may think the model-view-controller architecture is similar to the ICLS, and it might separate UI processes from other parts better than the ICLS does. Furthermore, the document-view architecture has been well-known and used as a standard architecture in some application frameworks. Those architectures are used for having application developments easier; however, they are not intended as to handle and separate various UIs from services like the ICLS offers. In other words, the ICLS is an architecture which replaces the view of the model-view-controller or document-view in existing architectures.

To address many kinds of devices and platforms in a standardized architecture, I must consider both input and output of contents. In this paper, I focused mainly on the input aspect because this is a first step. The easiest method for output customized for each device and platform is to reserve beforehand multiple media such as multi-size pictures, multi-sampling-rate sounds, and multi-quality videos. However, it is just platform-specific developments, which is avoided by the ICLS architecture. As another way, it is considered to covert media according to the capabilities of platforms, which is called media transcoding, and forms a research field. It is future work to utilize achievements in the field for the ICLS.

I intend to engage clients in utilizing their resources for the generation of various UIs, and do not add any scripting functionality to the current architecture for design simplicity. In the early step of this work, mobile devices were not powerful like now, and some Java virtual machines had begun to be implemented; hence, I implemented the ICLS architecture in Java. However, in these days, many platforms support a scripting language JavaScript in addition to Java, and this scripting language is widely used in rich web applications in the context of Ajax. In the current implement of the ICLS, all user operation done in clients is immediately transferred to servers, and the response speed is sensitive to communication bands. Therefore, it is effective to reduce communication traffic by using JavaScript, which enables handling some processes in device locally.

2.5.2 Interpretation of meanings

By using the AIDL, developers can define arbitrary meanings as RDF classes relating to existing meanings also defined as the classes. The more ICLS-based services are developed, the more meanings are needed because each service has specific meanings for its domain. Therefore, it is unrealistic to define them all beforehand.

RDF has flexible scalability for handling information on the open world, and it has the function of specifying resources with URI and the mechanism of class hierarchy. Using RDF, the meanings of selection elements can also have the same scalability. I am considering that the consensus of the interpretation of meanings will be built according to the spread of the AIDL.

The infinite meanings can be defined arbitrarily as RDF classes by service (logic server) developers, but they pose a problem of how client developers decide which meanings need to be implemented. It is a common problem in related studies including [17], because real world problems are addressed in this thesis. In the ICLS and the AIDL, however, the meanings represented by RDF classes allow a client to infer them by traversing class hierarchy trees until it finds a meaning that can be interpreted. In addition, since RDF documents are considered to be handled by arbitrary communities with consistency, the architecture can deal with meanings flexibly by collecting and merging RDF documents written by current and future developers of services and devices.

2.5.3 Feasibility

Since the ICLS requires keeping client-server connections alive, I performed a preliminary experiment for checking the response time of generated UIs on the ICLS. I used two PCs: PC 1 (Pentium 4, 3.2 GHz CPU, 1 GB memory) for the GUI clients and PC 2 (Trion 64 Mobile, 2 GHz CPU, 1 GB memory) for the desk lamp control server. The two PCs are connected with 100 Mbps LAN. After a user operates the power state of the server and the client sends a message, the server sends another message (indication of changing the availableness of another selection act) back to the client. In that case, the size of the message from the client to the server (Figure 2.11) is 350 bytes, and the size of the message from the server to the client is 219 bytes. Changing the number of clients connected to the server simultaneously from 1 to 10 on the PC 1, I measured the time of the bidirectional communication. As the result of the experiment, the response time was 140 msec. at worst (Figure 2.17), and it is enough performance for real use of UIs.

2.6 Conclusion and Future Work

2.6.1 Conclusion

In this chapter, the users' new demands for services were raised, and adaptive and migratory UIs were handled as the solution for the demands. After that, the interface client/logic server was proposed as UI architecture, and the abstract interaction description language was presented as a method for supporting adaptiveness and migration of UIs. Lastly, the implementations of the ICLS are also presented.

The interface client/logic server (ICLS) architecture was presented as a new solution, which not only supports migratory UIs but also offers adaptive UIs. To develop the ICLS, the abstract interaction description language (AIDL) is proposed,

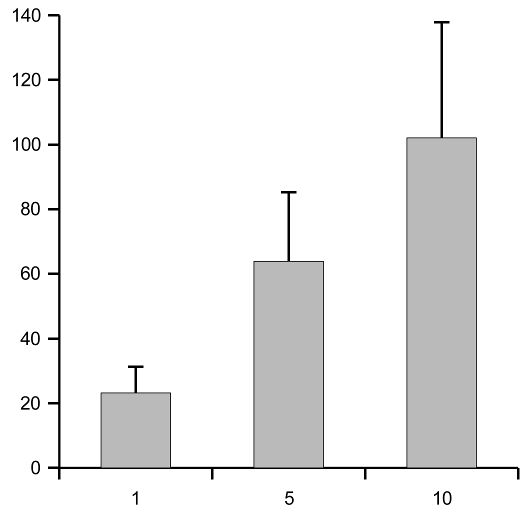


Figure 2.17: Averages of response time according to the number of clients. The x axis is the number of clients connected to one server simultaneously, and the y axis is a time scale (msec.) of the responses. The error indicators show the standard deviation of each data.

which describes UI functions for device- and service-specific UIs. Moreover, an ICLS framework was developed, and by using this framework, some implementations of interface client and logic server were developed in order to verify the feasibility of the architecture. The concept of simultaneous UIs shown in this chapter is an extension of migratory UIs, and it is a new technique on the research field of UIs. The AIDL can express the meanings of UI functions with the one of the semantic web technologies, RDF. That enables clients to perform meaning inference for service-specific UI generations. This is a new application of semantic web technologies, for the human computer interaction field.

The first point of this chapter is that the ICLS supports the migratory and simultaneous UIs as its architecture design, but as an ad-hoc way for existing architectures. In general, when UIs migrate from one device to another, their states should be gathered and transported along with their structures. In the ICLS, the states of UIs are always kept in the synchronized AIDL documents; thus, the state gathering step become not to be needed. The protocol for synchronizing the states is the mechanism for supporting the architecture in simple way.

The second point of this chapter is that the AIDL can express the meanings of interactions for describing service-specific interactions with their purposes. For generating various UIs for devices from one specification, the specification should be abstracted, and this causes abstracted UIs, which mean ones that do not utilize device-specific and service-specific features. The representation of meanings introduced here is one approach for this problem, and this feature enables clients to perform meaning inference for adaptive UI generations. In this approach, one of semantic web technologies, RDF is utilized as an application of this technology.

2.6.2 Future work

For advancing this work, it is necessary to advance the architecture, to evaluate its availability, and to consider exploiting the outcomes of other work.

As future work for advancing the architecture, there are following two topics: generation schemes of UIs and a mechanism for choosing appropriate UI components with given meanings. In the next chapter, a new method for generation GUIs from logical description is proposed, but for other sort of UIs, any well-established schemes are not shown. This will be an important task of this work for future. On the other hand, there is another issue for considering algorithms for matching the meanings in AIDL documents to implemented meanings in clients. This will be a key for exploiting the meaning mechanism newly introduced in this thesis. In addition, development and evaluation of authoring tools for AIDL documents in ICLS services are needed for further research, as well.

There are three points to be evaluated: the feasibility of the ICLS, the description capability of the AIDL, and the usability of services based on this architecture. The first and second ones are evaluated with implementing some practical services in imitation of existing applications provided on multiple platforms. The third point to be evaluated is examined with the practical services and their user as examinees and questioners. In addition, I have to evaluate the response speed of UIs on networks in detail, although I performed its preliminary experiment, because our architecture requires keeping client-server connections alive.

It is necessary to consider exploiting achievements of other studies and existing technologies. For example, outcomes in the area of transcoding, which handles conversion of the presentation of output information, can be usable for handling media data on multiple devices. In the way that is mentioned in section 5, the current design of ICLS does not utilize any scripting technologies like JavaScript of web applications. However, depending on the outcome of the evaluation of the feasibility, it might need to reconsider adopting such scripting languages.

Chapter 3

Flexible Widget Layout

In this chapter, a new solution for the flexible widget layout (FWL) problem is proposed, which formulates this problem as a fuzzy constraint satisfaction problem (FCSP). In the previous chapter, the UI architecture ICLS is proposed, where the interface clients generate UIs based on AIDL documents, or logical description of UIs. However, the task for considering how to generate actual UIs, especially GUIs from AIDL documents remains to be handled. FWL is the automatic layout of GUI widgets which requires both (1) deciding which widget types and their positioning are used and (2) completing the layout immediately especially when the system dose this at run time. In the following sections, the FWL problem is formulated as FCSP and a method for solving the problem is proposed. The formulation and method are implemented as a GUI interface client.

3.1 Introduction

3.1.1 Background

The automation of widget layout is one of the most important challenges [32] in the context of dynamic generation of graphical user interfaces (GUIs). In this thesis, by *widget layout* we mean the process of determining the positions and the sizes of widgets on a dialog box; it is also used for mentioning the result of the process (Figure 3.1). The widgets, such as list boxes, radio buttons, and grouping panels, offer specific functions, and they constitute GUIs hierarchically. The layout has a significant impact on the usability of applications and services using GUIs, and it determines the ease of tasks which can be accomplished with them.

In the field of model-based user interface (UI) design [9, 10], widget layout is more complicated because a layout system needs *to select widgets* before actually lay them out. Many systems proposed in researches in the field automatically generate GUIs based on *logical descriptions* through layout processes. The abstract interaction description language (AIDL) mentioned in the previous chapter is a sort of logical description languages, too. The logical descriptions specify

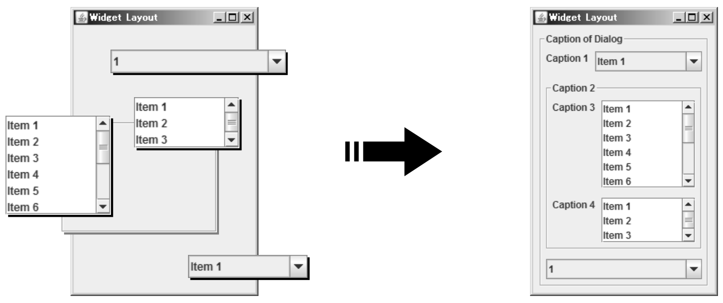


Figure 3.1: Concept of (normal) widget layout. Some assigned widgets are placed in a dialog box while their sizes, relationships, and hierarchy are being considered. This process is usually performed by layout managers offered by GUI toolkits, and the managers are specified by developers.

common *UI functions* independently of specific devices and platforms, instead of specifying widgets to be used. Although that is useful for realizing the diversity of UIs like the UI architecture ICLS, the systems must select widgets which provide the UI functions required by the description before they place widgets. These widgets are often determined as some sets of widget candidates because there are multiple widgets offering the same UI function.

3.1.2 Objective

The task tackled in this chapter is automatic GUI generation from logical descriptions, which requires considering the following two issues:

- How to decide which widgets are used from widget candidates according to given logical descriptions?
- How to complete the layout in a certain time especially when a system generate it at run time?

I call the layout handling those two issues, *the flexible widget layout* (FWL). In addition, I call the problem of determining a layout which fulfills conditions associated with a dialog box and widgets, *the FWL problem* (Figure 3.2). The FWL problem is a combinatorial optimization, and the systems for the FWL search the combination of widgets selecting from their candidates. As solving this problem, a system can select small widgets with less usability for small screens, or large widgets with enough usability for large screens. This feature can expand the possibilities of layout, but, the systems need to finish the layout in real time when the generation processes are performed at run time of service use.

In this chapter, the FWL problem is formulated as a *fuzzy constraint satisfaction problem* (FCSP) [33]. Constraint conditions of the FWL problem includes, except for physical conditions, subjective ones involved with usability, sensitivity,

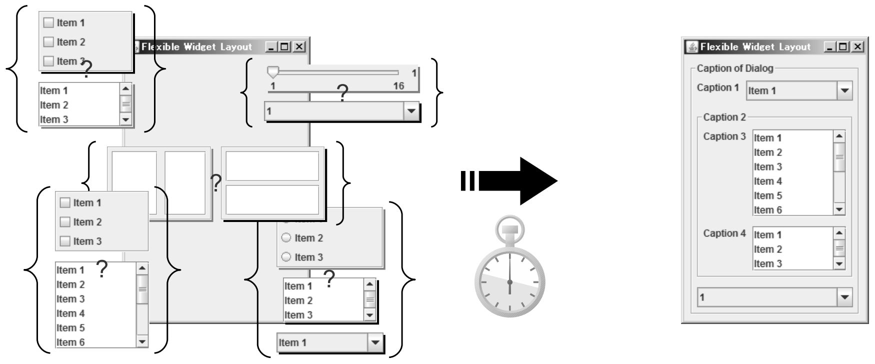


Figure 3.2: Concept of flexible widget layout. Before placing widgets, they need to be selected from some candidates of widgets. Here, the candidates include some container widgets. In addition, the time for the layout is limited when it is done at run time.

etc. such as *desirable* widgets and *conspicuous* layout. Thus, it is difficult to obtain a precise solution which satisfies all constraints. For tackling this difficulty, I introduce *fuzziness* into those constraint conditions, and aim at obtaining a practical solution at run time. FCSP is an extension of *constraint satisfaction problem* (CSP) for handling the fuzziness of constraints of real world problems. As using FCSP, the desirability of selection is expressed as *fuzzy constraints* straightforward. In this way, existing technique of FCSP can be utilized without extending the original framework of FCSP, and layouts as desirable as possible can be obtained.

In addition to the formulation, I propose a method for selecting appropriate widgets automatically and putting them in a dialog box, and show an implemented layout system of the method. The layout process is divided into three phases, and adopts a few optimizing techniques for realizing the layout in a practical time enough not to keep GUI users waiting. Along the process, the layout system generates a GUI dialog box based on a logical description given as an input. I claim that the approach can handle adequately complicated real applications, without depending on specific domains, and complete the layout in real time.

3.1.3 Related work

The related work of the FWL can be separated into four categories. As a part of model-based UI architecture, the FWL should be compared to the related work mentioned in Section 2.1.3 and other studies especially addressing the problem of GUI layouts for multi-platforms. On the other hand, as a sort of layout problem, the FWL relates to the field of facility layout problem (FLP), and the field of LSI layout should be evaluated as the related work of the FWL. In addition, except for current studies, layout managers of GUI toolkits can be mentioned as existing and widely used layout systems. In the following paragraphs, similarity and difference of some studies in each category are explained respectively.

Researches in the field of model-based UI design mentioned in Section 2.1.3 handle how to realize the utilization of various devices and platforms rather than how to generate GUI layouts, but, there are some studies addressing the same problem as in this thesis. The ubiquitous interactor [12] shows the results of GUI generations from logical descriptions, where it seems that their system performs layouts. However, the authors do not mention how any concrete layout methods are used and its automation is done in the system. The personal universal controller [15] is proposed for remote controlling various appliances with PDAs. Although a rule-based layout algorithm is explained, it is dependent on specific application domain and does not handle the widget selection corresponding to the same functions. In XWeb [20], the need of both the widget selection and layout is mentioned, but the proposed system does not handle their combination. As named as the design of graceful degradation of UIs, a method of building usable UIs for multi-platform systems is proposed [34, 35]. It addresses GUI generations for platforms having different screen resolutions more generally than as addressed in this thesis; therefore, it is very suggestive for the future improvements of the FWL although it does not propose actual method for the automation of GUI layouts. In addition, an intelligent editor for GUIs based on the concept of plasticity of UIs [36]. It proposes a model and a visualization technique for managing the plasticity, and it implemented them in the PlastiXML tool, a plug-in of existing tool the GrafiXML. Although it also does not handle the automation of GUI layout like that handled in this thesis, it models the plasticity of UIs as Moore machines, a variant of finite state machines, while the FCSP is used here.

The FLP is for managements of facility positioning and recourse movements in manufacturing plants, hospitals, etc., and researched for two and more decades [37, 38]. It is a combinatorial optimize problem for minimizing investments, transfer costs, etc., and it is involved in adjacency of placed items and flow of resources between the items. As an application for human computer interaction, a system was proposed where the FLP is used for the UI components layout [39]. It is to locate the menu/icon items on the screen/keyboard/mouse in order to achieve the greatest efficiency in exchanging the inputs and outputs between the user and the system. The authors of [39] claim that a one-to-one relationship exists between the manufacturing facilities layout problem in a plant and the textual and graphical user interface components layout problem. They showed an example addressing the closeness relationship of UI components (such as Del, Cut, Paste, Right, and Spell check in MS-WORD). As saying in the context of FWL in this thesis, what the FLP handles is generating a UI model, which consists of parental and sibling relationships of UI elements. It is different that the FWL considers physical alignments of UI components based on UI model, and I am exploring FLP to utilize it as future work. In addition, in papers of FLP, generally speaking, CSP, especially FCSP are not adopted for its methodology.

Many studies for the LSI or VLSI layout problem are proposed, but they do not consider the speed of layouts because they do not need it. In the early period, the objective of these layout problems is to wire completely and to shrink their dimensions, and it has advanced to the multi-criterion optimization [40]. The automation of LSI layout is a combinatorial optimization problem, and various algorithms are proposed for it; thus, they can be applied to other study fields. However, in the FWL, the practical speed of solving layout problems is required, and it is the characteristic which is not handled in the LSI layout.

Existing *layout managers* may seem to be solutions for the problem. In these days, many GUIs are developed with toolkits such as Swing [41], the windows forms [42], the visual component library [43], GTK+ [44], and Qt [45]. They offer layout managers or related mechanisms called as other names. They perform at run time and decide the positions and sizes of widgets (Figure 3.1), but they do not handle the selection of suitable widgets. In the present circumstances, the widget selection is done by GUI developers considering layout.

This chapter consists of six sections. In the first section, as introduction of this chapter about the FWL, its background, objective and related work were mentioned. In the following Section 2, the outline and structure of the FWL problem is mentioned, and how the FWL is to be handled as a problem is explained. In Section 3, the outline of FCSP is explained, followed by the formulation of the FWL with FCSP and the reason that FCSP is introduced to the FWL are explained. In Section 4, a method for solving the problem and an implementation of a layout system are shown with the explanation of multiple phases of the layout process and how it is optimized. In the following Section 5, some consideration about the speed of the system and the validity of the formulation of the FWL are discussed. Lastly, in Section 6, conclusion and future work are mentioned.

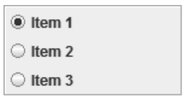
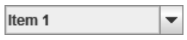
3.2 Layout Problem

3.2.1 Outline

The flexible widget layout problem is a solution search problem, whose search space is the combinations of widgets, and therefore, solving a FWL problem means finding better combinations of widgets. The widget combination must be layout-possible and should be as desirable as possible. Each widget is selected from *a widget candidate set*, which correspond to a certain UI function, includes some widgets, which can represent the same UI function or composition of UI functions. Each widget has different *minimum size* and *desirability*. The sizes of widgets are used for hard constraint conditions which decide possibility of the layout, and their desirability is used for soft constraint conditions which decide its suitability. A widget selection process from the candidate sets has following two steps:

1. A system resolves the mappings between a UI function and a set of widget candidates.

Table 3.1: Tradeoff between usability and layout-ease of widgets

	Radio buttons	Drop down list box
	 ◉ Item 1 ○ Item 2 ○ Item 3	 Item 1
Function	Same	Same
Usability	Better	Worse
Layout-ease	Worse	Better

2. It selects actually used widgets from the mapped sets in terms of the dimensions and usability of the widgets.

After that selection process, the system places all selected widgets in a dialog box with no overlapping but gaps among them, and it makes some groupings of related widgets in the same rectangles.

The complexity of the FWL is caused by the tradeoff between widget usability and the ease of layout involving their dimensions (Table 3.1), and especially, there is a tendency that lager widgets have more difficulty for layout [46–48]. For example, you can use a list box or a drop down list box for expressing the UI function of selecting one item from an item list. From the standpoint of usability, since users can view many items at once, the former is better, but it needs a larger area and may not be placed in a small dialog box (or a small screen). In the FWL, this tradeoff must be considered when performing optimum layout.

As a UI model generally expressed in logical descriptions, in this chapter, the selection act model is adopted. In the model, UI functions are represented as selection acts (or elements), and which are grouped to make a tree graph (Figure 2.6, repeated below as Figure 3.3). A set of UI elements in the model is expressed as $U = U_\mathrm{S} \cup U_\mathrm{G} \cup U_\mathrm{D}$, where U_S is a set of selection elements, U_G is a set of group elements, and U_D is a set of description elements. Selection element $u_i \in U_\mathrm{S}$ is represented here as a modified 6-tuple:

$$u_i = \langle L_i, e_i, t_i, o_i, m_i, r_i \rangle, \tag{3.1}$$

where L_i is the list of choices, and $|L_i|$ is 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. r_i is a newly added element for convenience of explanation, and it is a flag which is true if the type of choice list is `Numeric` and constrained by `Range`, and otherwise, it is false. Group elements make groupings of relevant selection elements and other group elements (Figure 2.6). All selection elements 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, the selections and groups can have a description element for their explanations.

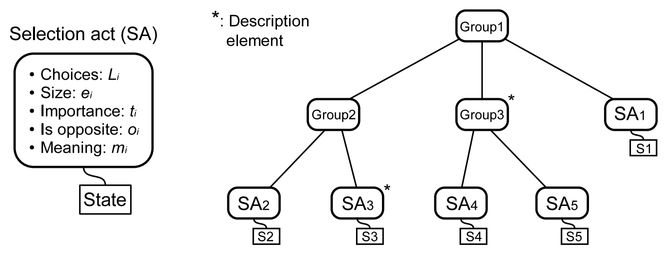


Figure 3.3: Selection act model expressed in an AIDL document. The group elements make both parental relationship between itself and its child elements; and sibling relationship among them. All selection elements are grouped and make a tree graph of UI functions, whose root is a group.

3.2.2 Used widgets

The UI elements of the model are represented as widgets. All widgets used here $W = W_N \cup W_C$ are rectangular and have their own presentations; also they are divided into two categories: normal widgets W_N for representing selection elements and container widgets W_C for group elements and labeling. In this thesis, as the normal widgets, since they are adopted by many toolkits (such as [41, 42, 44, 45]), a subset of widgets is used (Figure 3.4) and their characteristics are as follows:

Check box (CB) A single check box is used for selecting one item between on and off, which items are inter-exclusive (or boolean type). Its two items must be totally opposite, and it is then easily understandable for users.

Radio buttons (RBS) Radio buttons are buttons used for changing multiple states, of which only one is selected in one time. They are grouped and aligned in a frame indicating their grouping.

Drop-down list box (DLB) A drop-down list box is used for selecting one item from pop-up list, and useful when there is less space on a dialog box. However, it can show only one item before its pop-up button is clicked.

Check boxes (CBS) Multiple check boxes are used for selecting multiple items. All items must be shown in a screen, and thus, a user can figure out them at once, although a large space is used.

List box (LB) A list box is used for both single and multiple selections. It can have one or two scroll bars, and handle large number of items. It is better than a drop-down list box because it can show a part of items at once, but it consumes larger space.

Button (B) A button is used for indicating a command, and it only has no states. It is sometimes used as a toggle button but that is not considered here.

Spinner (SP) A spinner or a spin box is used for selecting one from serial items on a space-limited dialog box. It has two buttons of up and down, and they are small for clicking, and thus, it should not be used the items are large.

Slider (SL) A slider is used for selecting a value from serial ranged values, and shows current selection beside it. It is understandable because it itself indicates what a user need to input is a value.

Note that a single check box and multiple check boxes are distinguished because they are used for different functions. In addition to the widgets for selection, for representing description elements, *caption label* (CL) and *abbreviation label* (AL) are also used as the normal widgets. They are the same in respect of that they are label widgets usually offered in toolkits, but different their string.

There are some container widgets used in this thesis. As container widgets, *vertical array* (VA), *horizontal array* (HA), and *tab pages* (TP) are used for representing group elements (Figure 3.5). They contain multiple child widgets, and align them vertically or horizontally, or lap over with tab pages respectively. For representing the *positioning* of description elements, container widgets *left labeling* (LL) and *top labeling* (TL) are used (Figure 3.6). They contain one child widget and one description widget, and decide their alignments. As an exception, description widgets of vertical array, horizontal array, and tab pages are always expressed with top labeling widget and fixed their top. Especially, a dialog box is handled as a container widget, which contains only one child widget.

3.2.3 Structure

Widget candidate sets of FWL are determined according to logical descriptions of UI model, which contains UI functions and their groupings.

The desirability $\alpha \in [0, 1]$ is also defined corresponding to the types of normal widgets. The desirability of each normal widget is defined as α_{CB} for a check box, α_{RBS} for radio buttons, α_{DLB} for a drop-down list box, α_{CBS} for check boxes, α_{LB} for a list box, α_{B} for a button, α_{SP} for a spinner, and α_{SL} for slider. Especially, the desirability of a list box is defined as mutable according to the rate of its visible items; therefore, it has the range $\alpha_{LBmin} \leq \alpha_{LB} \leq \alpha_{LBmax}$. The desirability is ordered in terms of the usability of the widgets. In this thesis, referring to [46, 47], in terms of the usability of the widgets, the order of the desirability is defined as

$$\alpha_{DLB} < \alpha_{LBmin} < \alpha_{LBmax} < \alpha_{SP} < \alpha_{RBS} < \alpha_{CB} < \alpha_{CBS} = \alpha_{B} = \alpha_{SL}. \tag{3.2}$$

The widgets have the reverse order in terms of their dimensions. Moreover, the desirability of caption label and abbreviation label are defined as α_{CL} and α_{AL} respectively. Their order is defined as follows:

$$\alpha_{AL} < \alpha_{CL}. \tag{3.3}$$

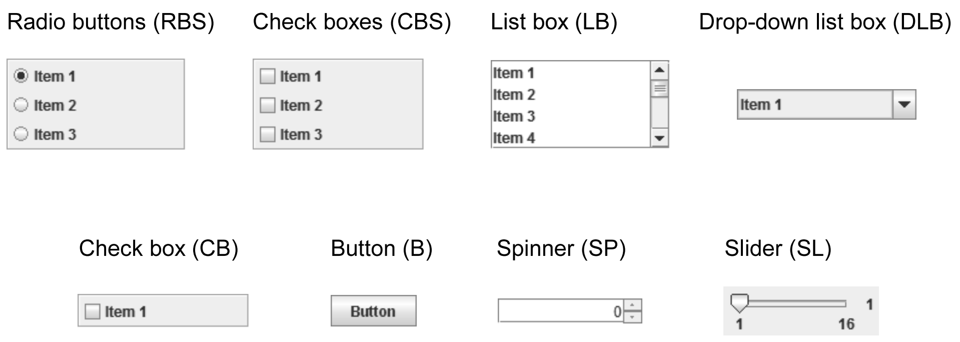


Figure 3.4: Widgets used in this thesis. These are screenshots of Swing widgets, and they are usually adopted in other toolkits, although they have a little different appearance.

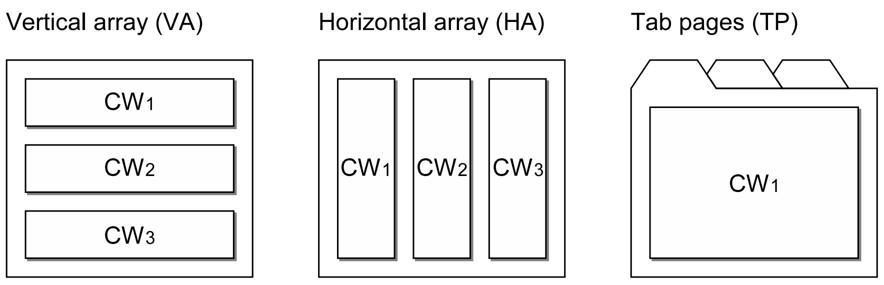


Figure 3.5: Three group widgets for positioning child widgets of a group element (CW stands for child widget). Tab pages widget has no advantage over other two widgets because a user cannot understand what widgets are shown before he or she clicks tabs.

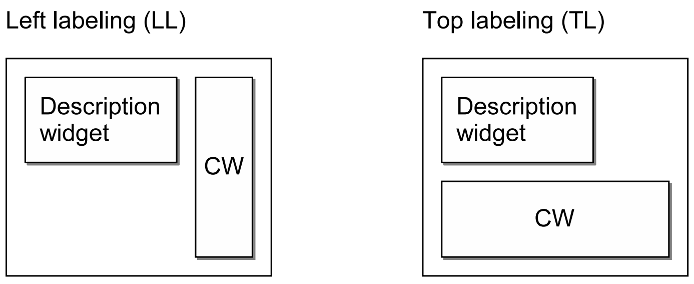


Figure 3.6: Two labeling widgets for positioning a child widget and its description (CW stands for child widget).

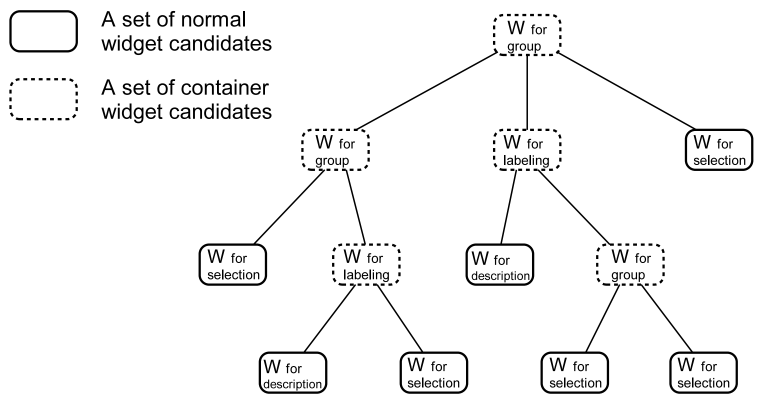


Figure 3.7: Tree structure of widget candidate sets. UI elements of the model and sets of widget candidates are related, and the sets construct a tree structure as well. Note that the candidate sets of labeling containers do not correspond directly to UI elements.

Table 3.2: Selection widget candidates

Candidate $w \in W_i$	Conditions			
	Selection size e_i	Item size $\lvert L_i \rvert$	Is opposite o_i	Is ranged r_i
Check box	`single`	$\lvert L_i \rvert = 2$	`true`	-
Radio buttons	`single`	$0 < \lvert L_i \rvert < 6$	-	-
List box	-	$6 \leq \lvert L_i \rvert$	-	-
Drop-down list box	`single`	$0 < \lvert L_i \rvert$	-	-
Spinner	`single`	$0 < \lvert L_i \rvert < 10$	-	`true`
Slider	`single`	$10 \leq \lvert L_i \rvert$	-	`true`
Check boxes	`multiple`	$0 < \lvert L_i \rvert$	-	-
Button	-	$\lvert L_i \rvert = 0$	-	-

Similarly, each container widgets also has desirability. The desirability α_{VA}, α_{HA}, and α_{TP} are defined for vertical array, horizontal array, and tab pages. α_{LL} and α_{TL} are defined for left labeling and top labeling. Their order is defined as follows:

$$\alpha_{TP} < \alpha_{VA} = \alpha_{HA}, \tag{3.4}$$

$$\alpha_{TL} < \alpha_{LL}. \tag{3.5}$$

Selection elements and description elements are mapped to the corresponding sets of normal widget candidates $W_i \subset W_{\mathrm{N}}$ (Figure 3.7). Selection element u_i is expressed with widget $w \in W_i$ chosen from its corresponding widget candidate set (Table 3.2). The widget candidates table is created by also referring to [46, 47], but it is possible to apply users' preferences there. Based on the table, for example, widget candidates W_i corresponding to u_i, where $e_i = \texttt{single}$ and $|L_i| = 5$, are determined as the set containing a drop-down list box and radio buttons (see Table 3.2). A description element is expressed with the set of widget candidates containing a caption label and an abbreviation label if it is available. In the layout process, it is decided which one of the candidates are used. Each widget candidate $w \in W_i$ has uniquely a minimum size $ms_w = \langle ms.width_w, ms.height_w \rangle$, which is defined by corresponding UI element. If w is a representation of selection element u_i, its minimum size is derived by the parameters of u_i ($|L_i|$, the maximum width of item strings of L_i, and o_i). Its detail is mentioned in Section 3.3.2.

Group elements and positioning of description elements are mapped to a set of container widget candidates $W_i \subset W_{\mathrm{C}}$, and they are expressed with widget $w \in W_i$ (Figure 3.7). A group element is represented as a set containing a vertical array, a horizontal array, and (a container of) tab pages. A positioning of a description is represented as a set containing a left labeling and a top labeling. A description element of a group is fixed on the top of one of the widgets for group element; hence it has no candidates. Each container widget candidate $w \in W_i$ also has a unique minimum size $ms_w = \langle ms.width_w, ms.height_w \rangle$. The minimum sizes of container widgets are defined by the minimum sizes of candidates of its child widgets. Its detail is also mentioned in Section 3.3.2.

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

3.3 Formulation

3.3.1 Fuzzy constraint satisfaction

Constraint satisfaction problem (CSP) is one of fundamental technologies in the field of the artificial intelligence. It is a generic term of search problem finding value combinations which satisfies given constraints, and it is NP-complete problem. It is defined by the following components:

- a finite set of variables $X = \{x_i\}_{i=1}^m$,
- a finite set of domains of values $D = \{D_i\}_{i=1}^m$ for the each variable, and
- a finite set of constraints $C = \{c_k\}_{k=1}^r$.

Constraint c_k denotes a relation R_k on a subset $S_k (S_k \subset X)$ of X. S_k is called the *scope* of R_k. In other words, when $S_k = \{x_{k_1}, \ldots, x_{k_{|S_k|}}\}$,

$$R_k \subseteq D_{k_1} \times \cdots \times D_{k_{|S_k|}}, \tag{3.6}$$

where c_k is called a unary constraint when the size of scope $|S_k| = 1$, or it is called a binary constraint when $|S_k| = 2$. An assignment to the variables in scope S_k of constraint c_k is defined as follows:

$$v[S_k] \in D_{k_1} \times \cdots \times D_{k_{|S_k|}}. \tag{3.7}$$

If $v[S_k] \in R_k$, constraint c_k is satisfied, otherwise, is not satisfied. To finding a solution of a CSP is to finding an assignment of variables $v[X]$ which satisfies its all constraints. For CSP, some general-purpose solvers exist.

The structure of CSP can be represented by a constraint graph, where nodes and edges of the graph are corresponding to variables and constraints (Figure 3.8). If constraint c_k is binary, the two nodes in its scope are connected by an edge. If c_k is unary, the one node in its scope is connected by an edge as a self-loop. If c_k is ternary or of higher order, c_k is represented by a hyperedge, and the graph becomes a hypergraph. However, only unary and binary constraints are used in this thesis.

Fuzzy constraint satisfaction problem (FCSP) is one of the extensions of traditional CSP. CSP is a simple model and its hard (or crisp) constraints are too rigid for formulating real world problems. Therefore, FCSP introduces one of soft constraints, *fuzzy constraint*, which is not necessarily satisfied, and considers the *satisfaction degree* of the constraint. In FCSP, constraint c_k denotes a fuzzy relation μR_k, which has its membership functions defined by

$$\mu R_k : \prod_{x_i \in S_k} D_i \to [0, 1]. \tag{3.8}$$

In other words, the membership value is defined by assignment $v[S_k]$ to the variables in scope S_k of constraint c_k. This value is called the satisfaction degree of a fuzzy constraint. Since a FCSP requires the satisfaction of the fuzzy conjunction of all fuzzy constraints, the overall satisfaction degree of the whole FCSP is defined as the minimum satisfaction degree as follows:

$$C_{\min}(v) = \min_{1 \le k \le r} \Big(\mu R_k(v[S_k])\Big). \tag{3.9}$$

To solve a FCSP means to solve a optimum problem, which is finding the assignment accompanied by the best satisfaction degree of the problem from all combinations of assignments (Figure 3.9).

3.3.2 Flexible widget layout with FCSP

The FCSP framework is introduced to the FWL here, and the desirability α of the widgets is represented with satisfaction degrees of fuzzy constraints. Unary fuzzy constraints are used for expressing the desirability α. Fuzzy constraints enable you to represent naturally the gradual rules of the widget desirability without introducing any other objective functions. The parental relationship among widgets is also represented with binary crisp constraints. Crisp constraints can be handled as particular cases of fuzzy constraints. Using these crisp (or hard) and fuzzy (or soft) constraints accordingly, the FWL problem is formulated, which can be applied some existing algorithms for solving FCSP.

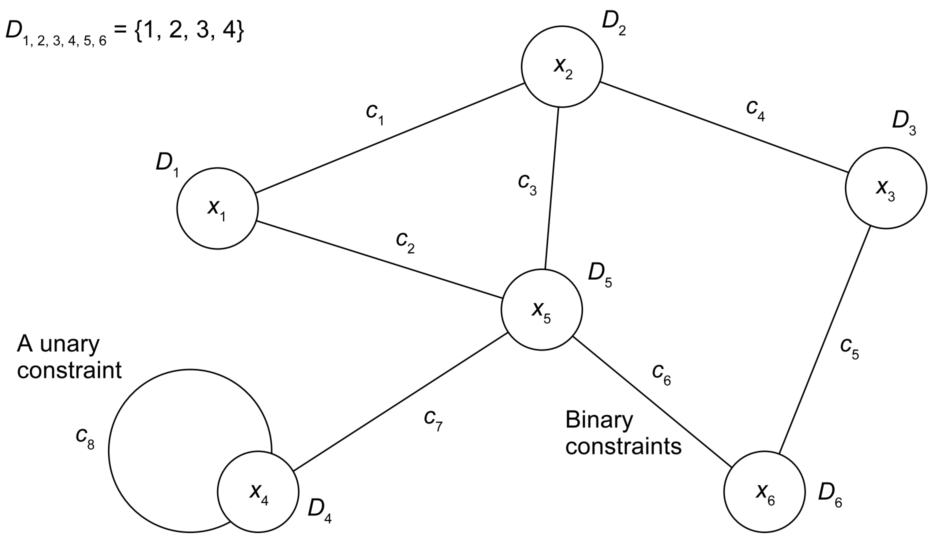


Figure 3.8: Example of constraint graph. In this graph, the six variables (nodes) are connected with the eight constraints (edges), and in each variable, the domain is set. A binary constraint is expressed as the edge connected to two nodes, and a unary constraint is expressed as a self-loop.

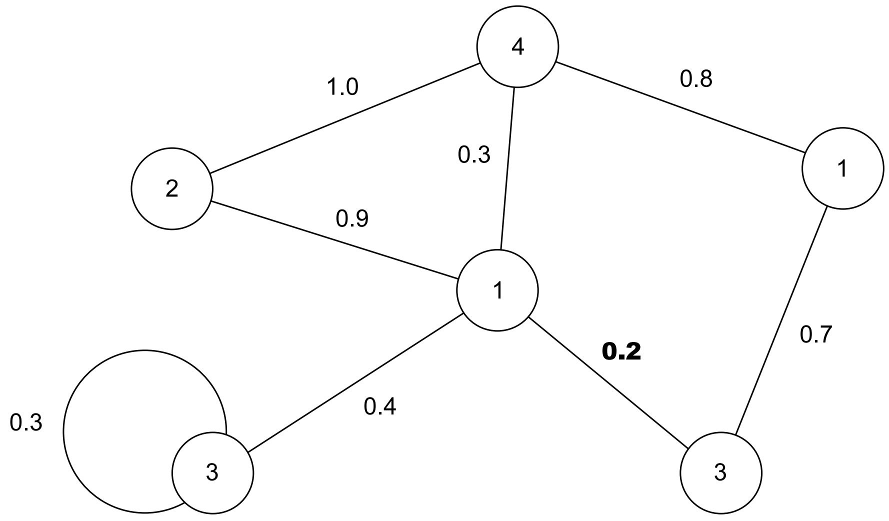


Figure 3.9: Example of assignments of constraint graph of fuzzy constraint satisfaction problem (FCSP) (see also Figure 3.8). The values in the nodes indicate the assignments to the variables, and the values beside the constraints indicate their satisfaction degrees. In these assignments, the lowest value is 0.2 (bold typed), and thus, the overall satisfaction degree of this FCSP is also 0.2.

3.3.2.1 Variables

In the formulation, variable $x_i \in X = X_{\mathrm{N}} \cup X_{\mathrm{C}}$ corresponds to widget candidate set W_i and the value assigned in it expresses a selected candidate from the set. Variables X_{N} and X_{C} express the set of variables for the normal widget candidates and container widget candidates respectively. For example, UI element $u_i \in U$ corresponds to widget candidate set $W_i \subset W$, and next it corresponds to variable $x_i \in X$. In FCSP, parental relationships of the UI model are expressed as binary constraints between the variables (it is mentioned above). Thus, a tree structural constraint graph is constructed from the tree of the model. Note that UI elements, groups, descriptions, and the positioning of the descriptions, are also expressed with the variables. A group or a positioning of the description $u_i \in U_{\mathrm{G}}$ is expressed with $x_i \in X_{\mathrm{C}}$, while a description $u_i \in U_{\mathrm{D}}$ itself is expressed with $x_i \in X_{\mathrm{N}}$.

3.3.2.2 Domains

The values of domains are tuples according to each variable type, and they are used for that constraints calculate their satisfaction degrees. As mentioned below, each tuple of a domain consists of a widget and a minimum size of the widget. In addition to those elements, a tuple of a domain for container widget candidates contains the combination of the values of the domains for its child widgets. The minimum sizes of normal widgets are determined by the parameters of the based selection elements and description elements. The minimum sizes of containers are calculated by the minimum sizes of the child widgets of the containers.

Domains for normal widget candidates variables The domain of normal widget candidates variable $x_i \in X_{\mathrm{N}}$ is a set of the tuples, which consists of normal widget w and its minimum size $ms_w = \langle ms.width_w, ms.height_w \rangle$ as follows:

$$D_i\ (\in D_{\mathrm{N}}) = \{\langle w, ms_w \rangle \mid w \in W_i \subset W_{\mathrm{N}}\}. \tag{3.10}$$

The minimum widths of normal widgets are the sum of the width of the widest one of its items and the widths of the control parts such as a vertical scroll bar, a radio button, and a check box. The minimum heights are defined by the type of widget, their item size, and the item height *item_h* (Table 3.3).

Domains for container widget candidates variables The domain of container widget candidates variable $x_i \in X_{\mathrm{C}}$ is a set of the tuples, which consists of container widget w, a combination of values of child widget candidates M, and its minimum size $ms_{w,M} = \langle ms.width_{w,M}, ms.height_{w,M} \rangle$ as follows:

$$\begin{aligned} D_i\ (\in D_{\mathrm{C}}) = \{\langle w, M, ms_{w,M} \rangle \mid \\ w \in W_i \subset W_{\mathrm{C}},\ M \in D_{\mathrm{child}(i,1)} \times \cdots \times D_{\mathrm{child}(i,\mathrm{cn}(i))}, \\ \mathrm{checksize}(W_i, ms_{w,M})\}, \end{aligned} \tag{3.11}$$

Function 1 checksize(W_i, ms)

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

where 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. The function checksize(W_i, ms) checks whether the combination of its parameters is available or not with the estimated minimum sizes (ems) of the child widgets (Function 1). With this function, the domains for container variables are pruned when the problems are being constructed. In the function, *given_width* and *given_height* are the size of the client area of the dialog box, and function ems is defined as follows:

$$W_i \subset W_{\mathrm{C}},\ \mathrm{ems}'(W_i, ms_{w\in W_{i,j}}) = \min_{w\in W_i}\left(ms''_{w,\{\mathrm{ems}(W_{i,1}),\ldots,ms_w,\ldots,\mathrm{ems}(W_{i,\mathrm{cn}(i)})\}}\right), \tag{3.12}$$

$$W_i \subset W_{\mathrm{C}},\ \mathrm{ems}(W_i) = \min_{w\in W_i}\left(ms'_{w,\{\mathrm{ems}(W_{i,1}),\ldots,\mathrm{ems}(W_{i,\mathrm{cn}(i)})\}}\right), \tag{3.13}$$

$$\begin{aligned} W_i \subset W_{\mathrm{N}},\ \mathrm{ems}(W_i) &= \min_{w\in W_i}(ms_w) \\ &= \left\langle \min_{w\in W_i}(ms.width_w), \min_{w\in W_i}(ms.height_w)\right\rangle. \end{aligned} \tag{3.14}$$

The container widgets have different sizes of child widgets; therefore, 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 based on the minimum

Table 3.3: Minimum widget heights

Widget	Minimum height (without edges)
Check box	*item_h*
Drop-down list box	
Spinner	
List box	min(|L|, 4) *item_h*
Radio buttons	|L| *item_h*
Check boxes	
Slider	*slider_h*
Button	*button_h*

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 among child widgets and tabs space are omitted):

$$ms_{\mathrm{VA}\in W_i} = \left\langle \max_{1\le j\le \mathrm{cn}(i)} \left(ms.width_{w_{i,j}}\right),\ \sum\nolimits_{j=1}^{\mathrm{cn}(i)} ms.height_{w_{i,j}} \right\rangle, \tag{3.15}$$

$$ms_{\mathrm{HA}\in W_i} = \left\langle \sum\nolimits_{j=1}^{\mathrm{cn}(i)} ms.width_{w_{i,j}},\ \max_{1\le j\le \mathrm{cn}(i)} \left(ms.height_{w_{i,j}}\right) \right\rangle, \tag{3.16}$$

$$ms_{\mathrm{TP}\in W_i} = \left\langle \max_{1\le j\le \mathrm{cn}(i)} \left(ms.width_{w_{i,j}}\right),\ \max_{1\le j\le \mathrm{cn}(i)} \left(ms.height_{w_{i,j}}\right) \right\rangle. \tag{3.17}$$

The minimum sizes of a left labeling (LL) and a top labeling (TL) are also calculated based on the minimum size of their child widgets. The sizes are defined by the minimum sizes of their description widget $ms_{w_{i,\mathrm{D}}} = \langle ms.width_{w_{i,\mathrm{D}}}, ms.height_{w_{i,\mathrm{D}}} \rangle$ and their one labeled widget $ms_{w_{i,\mathrm{C}}} = \langle ms.width_{w_{i,\mathrm{C}}}, ms.height_{w_{i,\mathrm{C}}} \rangle$ as follows (where gaps between the two widgets are omitted):

$$\mathrm{ms}_{\mathrm{LL}\in W_i} = \left\langle ms.width_{w_{i,\mathrm{D}}} + ms.width_{w_{i,\mathrm{C}}},\ \max\left(ms.height_{w_{i,\mathrm{D}}}, ms.height_{w_{i,\mathrm{C}}}\right) \right\rangle, \tag{3.18}$$

$$\mathrm{ms}_{\mathrm{TL}\in W_i} = \left\langle \max\left(ms.width_{w_{i,\mathrm{D}}}, ms.width_{w_{i,\mathrm{C}}}\right),\ ms.height_{w_{i,\mathrm{D}}} + ms.height_{w_{i,\mathrm{C}}} \right\rangle. \tag{3.19}$$

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 there.

The method to construct the domains for container widget candidates is the same method as seen in the binarization of arbitrary n-ary constraints with the hidden variable encoding [49, 50]. To simply formulate composite relations between a container widget and its child widgets, n-ary constraints could be used. There, an n-ary constraint is connected to a container variable and its child variables, and its constraint condition is that the total size of its children is less than its container size. In this thesis, since many existing FCSP solvers are designed for binary problems, for keeping the FWL problem available, it is formulated as binary problem by the binarization method. However, the binarization requires calculating all the combination of child values and enlarges the domain sizes. To avoid this enlargement, the domains are constructed in the reverse breadth first order (or from bottom to top) of the tree structure of candidate sets, doing pruning based on the estimation of minimum sizes of child widgets.

3.3.2.3 Constraints

Each variable except for a dialog variable is connected by one of desirability constraints C_D, and two variables corresponding to a container and its child can be connected by one of parental relationship constraints C_P. Constraint $c_k \in C_\mathrm{D}$ is unary and denotes the desirability of the value of its scope x_{k_1} as their satisfaction

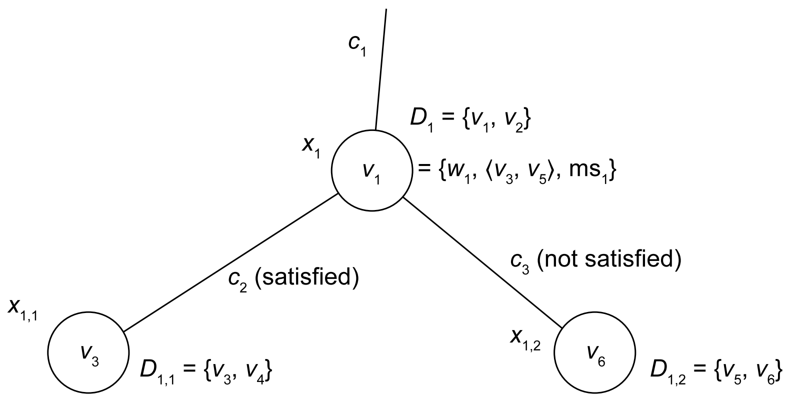


Figure 3.10: Example of the binary constraints, their scopes, and the assignments. In the constraint graph, X_1 is a container widget and others are its child widgets. In these assignments, value v_3 is assigned to the variable of the 1st child $X_{1,1}$, and value v_6 is assigned to the variable of the 2nd child $X_{1,2}$. The value assigned to the variable of the container is $\{w_1, \langle v_3, v_5\rangle, ms_1\}$, and it contains the value assigned to $X_{1,1}$ as the 1st element of the bracket. Thus, binary constraint c_2 is satisfied. On the other hand, v_1 does not contain the value assigned to $X_{1,2}$ as the 2st element of the bracket; hence, binary constraint c_3 is not satisfied.

degrees. If the scope of c_k is $S_k = \{x_{k_1}\}$ and the value of x_{k_1} is $v\ (\in D_{k_1}) = \langle w, ...\rangle$, where $w \in W_{k_1}$, the satisfaction degree of c_k is calculated as follows:

$$c_k(v)\ (\in C_{\mathrm{D}}) = des(w), \tag{3.20}$$

where des is the projection from the widget candidates to their desirability α. Constraint $c_k \in C_{\mathrm{P}}$ is binary and denotes whether the assignments of the variables of its scope correspond with each other. As mentioned above, 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 $v_p\ (\in D_{k_1}) = \langle w, M, ms_w\rangle$, 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:

$$c_k(v_p, v_c)\ (\in C_{\mathrm{P}}) = \begin{cases} 1 & \text{if } v_c = M\left[\text{childindex}(x_{k_1}, x_{k_2})\right] \\ 0 & \text{otherwise} \end{cases}. \tag{3.21}$$

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 $x_1 \in X_{\mathrm{C}}$) (Figure 3.10). These constraints connect tuples are called as compatibility constraints in [50].

A FCSP consists of variables, domains, and constraints; thus, after the formulation mentioned above, the FWL problem is completely represented as a FCSP.

For instance, an example showed in Section 3.4.3 (Figure 3.12) consists of 19 variables and domains, and 31 constraints, and the average size of the domains is 1215. It can be solved with general-purpose FCSP algorithms.

3.4 Method of Layout

3.4.1 Three phases

In this thesis, a layout method is proposed as a layout system for generating GUI dialog boxes. After formulating of the FWL problem, it is still remained to consider how to construct the FCSPs of the FWL and how to solve them. The layout system receives an AIDL document as input and outputs a dialog box with a layout composed of Swing [41] widgets. The layout system consists of the following three phases of transformation (Figure 3.11):

Phase 1 creating a FCSP from a UI model in an AIDL document,

Phase 2 solving the problem with an algorithm for obtaining a solution, and

Phase 3 performing an actual layout based on the solution.

Creating problem phase In the first phase, a FCSP is generated from a given AIDL document through the pruning process. First, group elements, selection elements, and description elements including the positioning of description elements in a document are related to corresponding widget candidate sets. Next, the candidate sets are related to the variables and their domains, which are pruned as mentioned in the previous section. Finally, a constraint graph having one-to-one correspondence to the UI model created, where the UI elements are seen as each corresponding typed variable and their parental relationships are seen as binary constraints. This pruning process is called the static pruning comparing to another pruning (the dynamic pruning) done in the next phase.

Solving problem phase In the second phase, for a better assignment of the variables or the best combination of widgets, the system iterates solving the FCSP generated in the previous phase. The system uses the *forward checking algorithm* [51, 52], which is extended for FCSP. In this method, the algorithm assigns the variables in the reversed breadth first order like when the construction of the domains. While iterating solving, the system sets a threshold of satisfaction degrees, and prunes the values of the domains which satisfy the corresponding unary constraints below the threshold. The threshold is called *the worst satisfaction degree* (WSD), and the pruning is called the dynamic pruning. The precise description of the process of solving the FCSP is as the following 4 steps:

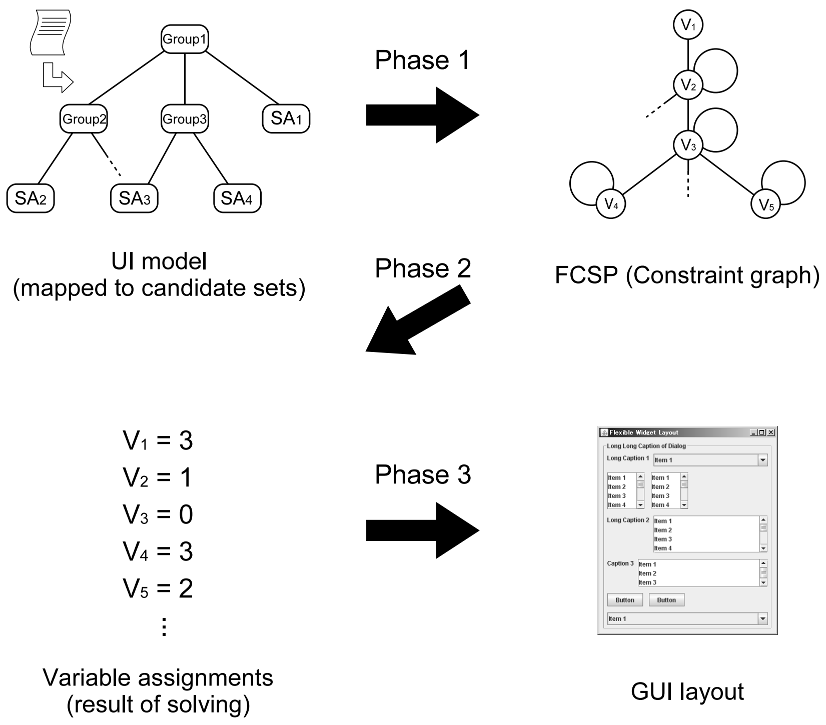


Figure 3.11: Three phases for flexible widget layout. Each phase is transformation of input and output. At the first phase, a UI model is transformed into a FCSP, next, the FCSP is solved and the solution (an assignment of the variables) is obtained, and finally, the actual GUI dialog box is generated from the solution.

Step 1 The system prepares satisfaction degree set A by collecting possible satisfaction degrees from all unary constraints. Since the all unary constraints express the desirability α of each widget candidates, A is a discrete set.

Step 2 The system chooses and removes the maximum one $a \in A$, and prunes values in the domains worse than a based on the unary constraints. At this step, the minimum heights of list boxes in the domains are reset as follows (where $|L_i|$ and t_i are the parameters of selection elements):

$$ms.height_{\mathrm{LB} \in W_i} = \min(|L_i|, 4)\, item\_h \left(1 - \frac{\alpha_{LBmax} - a}{\alpha_{LBmax} - \alpha_{LBmin}}\right)^{\frac{1}{t_i}}. \qquad (3.22)$$

Step 3 The system solves the FCSP with the algorithm.

Step 4 If the system finds an assignment of the variables whose satisfaction degree is equal to or better than a, it moves to the next layout phase; otherwise, it moves back to the step 2. If A is empty, the system stops in failure.

Placing widget 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 a dialog box. In the FWL, the variables express the selections of widget candidates; therefore, the solution of a FCSP is not an actual layout. The system generates widgets adopted in the solution, and then it sets their concrete positions (pixels) and sizes (pixels) based on the selected container widgets (Figure 3.12). Note that this process of deciding positions and sizes itself is the same as the process performed by existing layout managers in toolkits except our system does based of the solution.

3.4.2 Optimization

For the FWL, the speed of solving its problem is important because it is intended for using practically as GUI interface client of ICLS architecture. Therefore, at the creating problem phase, the layout system pruned the domains statically based on the estimated minimum sizes. At the solving phase, in addition, the system iteratively pruned the domains dynamically with the worst constraint satisfaction degrees. As another optimization, I tried to utilize the improved version of the algorithm with the technique of *dynamic variable ordering* [53].

For solving the problem rapidly, the system prunes the domains based on giving WSDs before applying the algorithm. The forward checking algorithm searches systematically through the search space of the possible combinations of the assignments to the variables until it finds a solution. It is guaranteed to find a solution if one exists, but it has the disadvantage that it requires large cost of time. Hence, it is effective to prune the domains and to reduce the problem scale.

The dynamic variable ordering (DVO) technique optimizes the order in which the variables are assigned during searching with systematic algorithms [53]. For the FWL system, it is possible to apply the forward checking algorithm improved with the DVO with the minimum remaining values (MRV) heuristic. The MRV is one of the popular reordering heuristic for the DVO, and it assigns next the variable that has the fewest values (the smallest domain) compatible with the previous assignments. It means assignments are done almost in the reversed breadth first order, because the variables of the *leaves* in the tree structure of FCSP have a few widget candidates, and other variables have more their combinations.

3.4.3 Implementation

The layout system is developed as a part of GUI interface client of ICSL architecture. It is implemented using Java 6 on a PC (AMD Turion 64 CPU 2.0 GHz, 768 MB main memory, and Windows XP Professional edition). For implementing the system, I have developed a class library for CSP and FCSP with Java, and named Stlics. This library represents variables, domains, and constrains as Java objects, and offers some solvers (algorithms) including the forward checking. Using them, arbitrary FCSP can be constructed and solved with the solvers. The desirability α_*

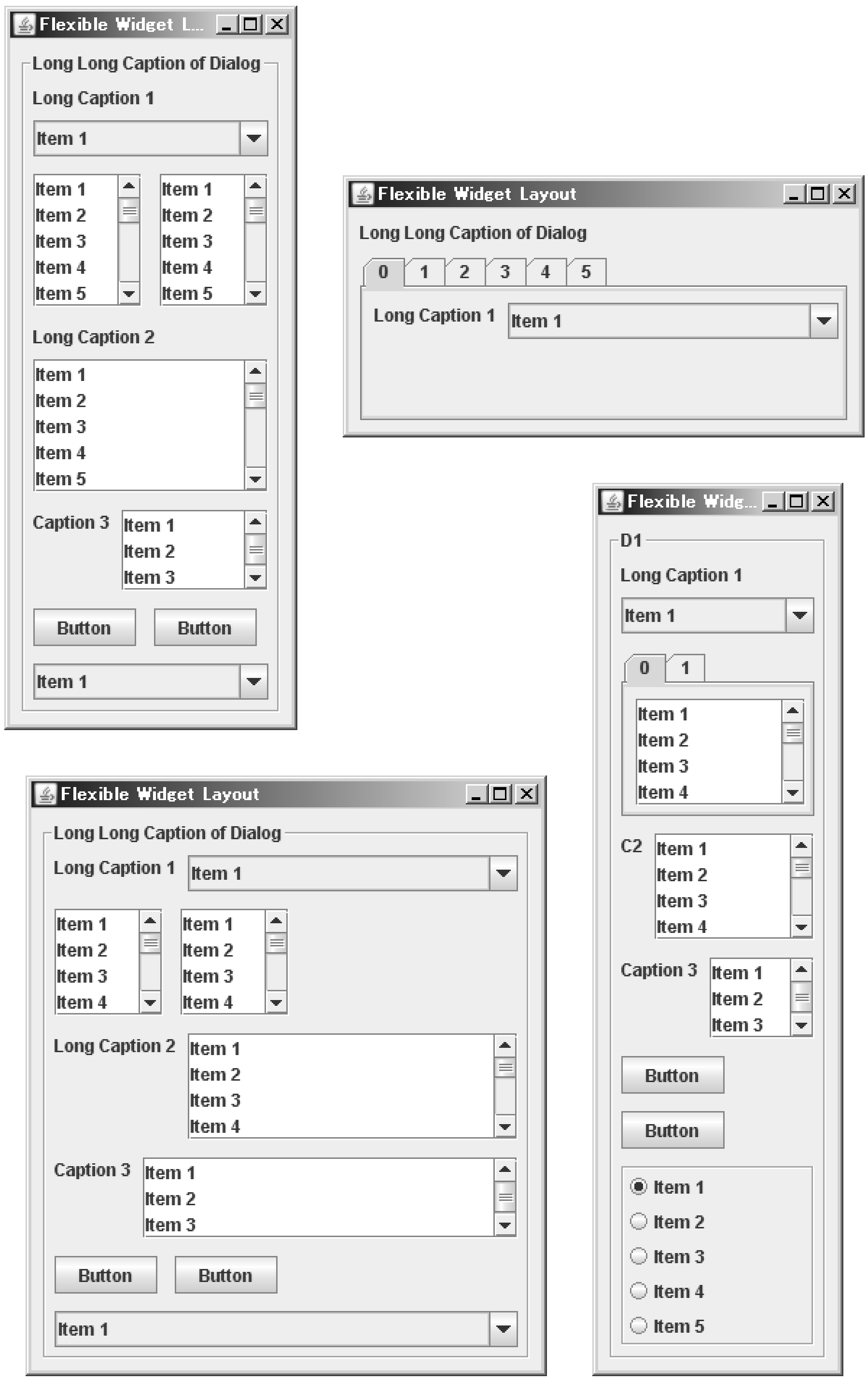


Figure 3.12: Results of flexible widget layout. They are generated from the same AIDL document (UI model) with different dialog sizes.

of the widget candidates are given empirically assuming typical applications (Table 3.4). The implementation reflects the resize of the dialog box in the re-layout of the widgets (Figure 3.12). In the implementation, the specific GUI toolkit is used, but the method proposed here does not depend on it

3.5 Discussion

3.5.1 Speed of layout

The following preliminary experiments are performed for investigating the relationships between the speed of the FWL and some conditions: the complexity of a base UI model, the size of a dialog box, and an algorithm for solving. For experiments, I developed a measurement application, which automatically changes the layout size (instead of dialog size) and measures the time for layout and the desirability (satisfaction degree) of the layout.

By using the measurement application, the following two preliminary experiments are performed for checking the relationship between the complexity and the speed when applying the forward checking algorithm without the DVO. First, I examined the relationship between the complexity of UI model and the average of the layout time and the desirability. For this experiment, the example model (Figure 3.12), and extended example model with one, two, or three additional UI elements are used (Figure 3.13). The results show the FWL problem is sensitive to the scale of the UI model, and it might be directly affected by the enlargement of the widget combinations. Second, I examined the relationship between the size of dialog box and the average of the layout time and the desirability with the example four UI models (Figure 3.14, 3.15, 3.16, 3.17). The results show that the area of dialog boxes also affects the time. One of its main reasons is because when the dialog size is small, in the solving phase, applying the algorithm is iterated many times with from better WSD to worse one. In addition, when the WSD is low, many widget candidates are not pruned and the search space remains large, and therefore, the standard deviations of each example also increase according to the complexity of the model. The average desirability is also affected by the scale of the UI model, and decreases in accordance with the complexity

By using the same measurement application, I performed another experiment for checking the efficiency of the DVO with the MRV heuristic in the algorithm for the FWL problem (Figure 3.18). The result shows this extension of the forward checking is less effective for this problem in comparison with the result without the DVO (see Figure 3.13). The reason might come from that the default order to assign the variables is the reversed breadth first order (RBFO) it might be a sort of heuristic like the MRV, in the case of the FWL problem. The result does not show the MRV is not effective, but it shows the RBFO is more suitable than the MRV for the tree structure of the constraint graph unique to the FWL problem. As another experiment for examining the effect of the pruning in the solving phase, I also executed the system with the default sample layout without the pruning. However,

Table 3.4: Samples of α_*

Alpha	Value
α_{CB} (check box)	0.98
α_{DLB} (drop-down list box)	0.70
α_{SP} (spinner)	0.90
α_{LB} (list box)	$0.75 \leq \alpha_{LB} \leq 0.85$
α_{RBS} (radio buttons)	0.95
α_{CBS} (check boxes)	1.00
α_{SL} (slider)	1.00
α_{B} (button)	1.00
α_{AL} (abbreviation label)	0.65
α_{CL} (caption label)	1.00
α_{VA} (vertical array)	1.00
α_{HA} (horizontal array)	1.00
α_{TP} (tab pages)	0.60
α_{TL} (top labeling)	0.90
α_{LL} (left labeling)	0.95

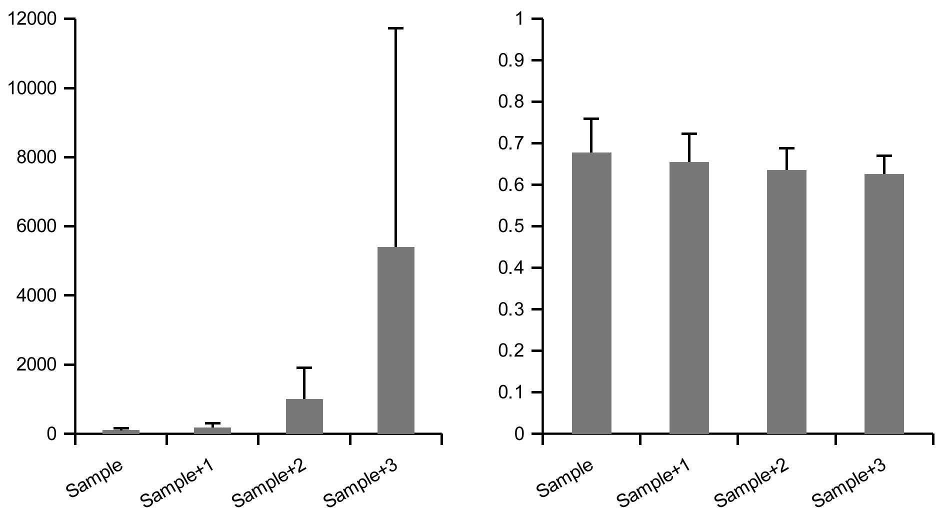


Figure 3.13: The relationship between the complexities of UI model and the average layout times (left), and the relationship between the complexities and average desirability (right). The y axis is a time scale (msec.) (left) or desirability (right) and the x axis is the models. The error bars are the standard deviations of each model.

it did not stop for more than two hours because the domain size got too large to be handled. That shows that pruning is effective in the system.

These experiments mentioned above are not comprehensive, and the scales of the examples used for them are not enough for all applications and purposes of GUI generations. For performing the comprehensive experiment, in addition, it is also need to investigate how large scale problem or UI model is suitable for investigating whether or not the system has effective enough. However, the result shows that the system is enough for the scale of the default example model, and this implementation indicates it is possible to automate the FWL, which is performed by GUI developers by hands before.

3.5.2 Validity of formulation

The formulation method of FWL proposed in this thesis is not trivial. In early phases of the research, the FWL problem was expressed with the variables corresponding to widgets sizes and locations. This could be the easiest approach for formulating the problem, but it could not offer enough speed of solving the problem. That is because these variables expressing sizes and locations have large domains, and the scale of the problem is enlarged. For the large scale problem, systematic search algorithms take long time. Therefore, it was attempted to apply some local search algorithms, which are known that it often can offer approximate solution relatively faster, and the combination of the both of them [54]. However, it also delivered inefficient result because the problem has large *plateau*, which is a flat part of its search space. In the current formulation, the container widgets function effectively for reducing the scale of the problem.

3.5.3 Other considerations

For the usability of the FWL, the result of the FWL problem should be stable, which means that layouts in different dialog sizes should be similar to each other. The satiability of the solution of FCSP is discussed in existing studies for FCSP, for example [55, 56], and these achievements are usable for the FWL problem as well. However, the stability of the FWL is not exactly the same as them because the similarity of widgets and their positioning should be considered. These similarities are also fuzziness and subjective ones like the desirability of widgets discussed in Section 1. It must be considered as future work.

How to decide the desirability α_* is a point to be considered. Although, in this thesis, the values are defined empirically (Table 3.4), it is difficult to define because these values should be totally ordered as constraint satisfaction degrees of FCSP. Since the framework of FCSP is introduced and the desirability can be defined with fuzziness, pairwise comparison method can be utilized as a mean for defining the desirability. I would like to put it as future work.

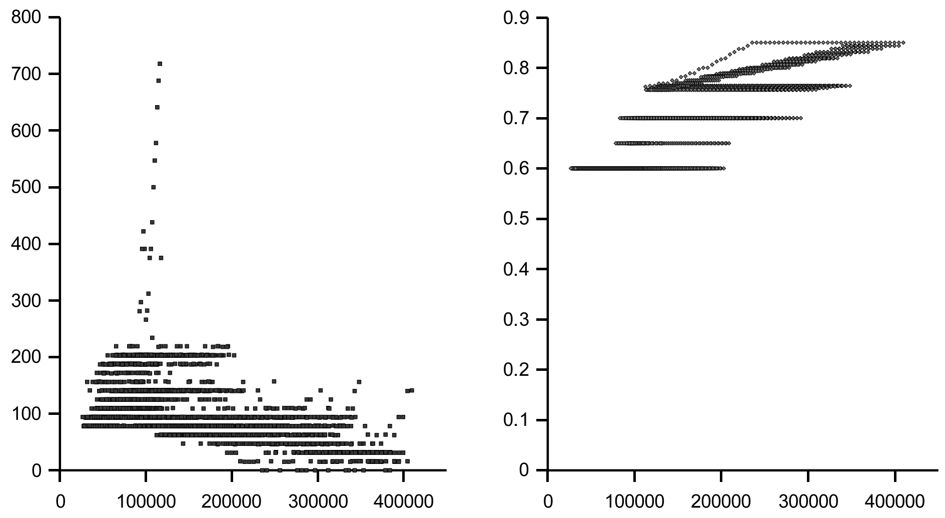


Figure 3.14: Relationship between the dialog box area, time (left) and desirability (right) of the default example UI model. The y axis is a time scale (msec.) (left) or desirability (right) and the x axis is the area of dialog boxes (pixel^2).

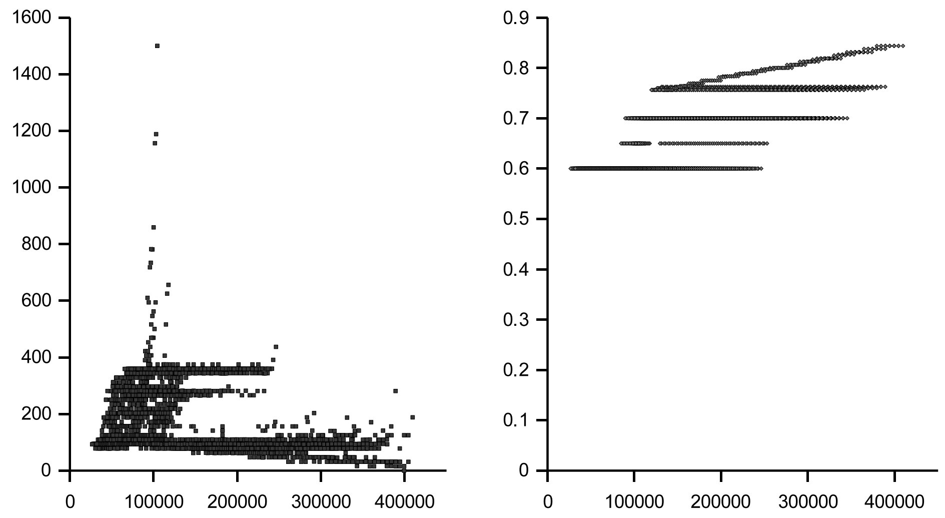


Figure 3.15: Relationship between the dialog box area, time (left) and desirability (right) of the example UI model added one widget. The y axis is a time scale (msec.) (left) or desirability (right) and the x axis is the area of dialog boxes (pixel^2).

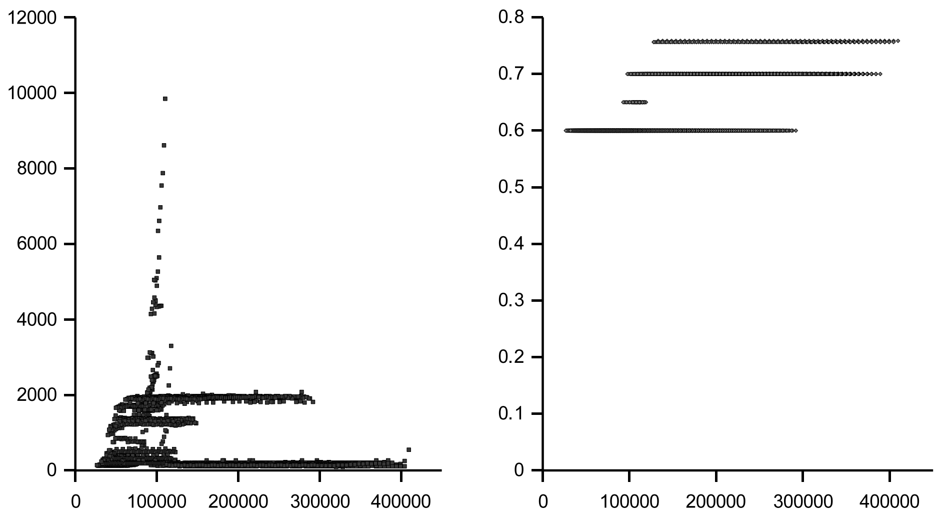


Figure 3.16: Relationship between the dialog box area, time (left) and desirability (right) of the example UI model added two widget. The y axis is a time scale (msec.) (left) or desirability (right) and the x axis is the area of dialog boxes (pixel^2).

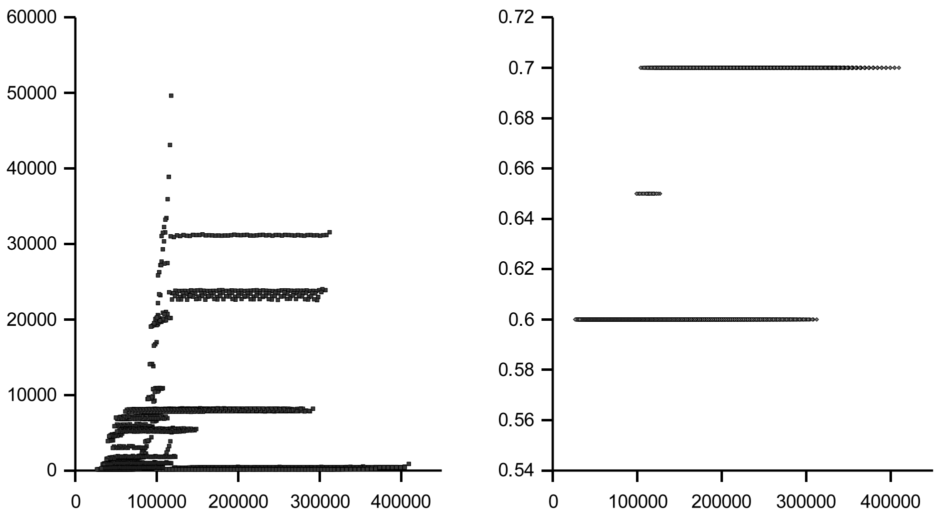
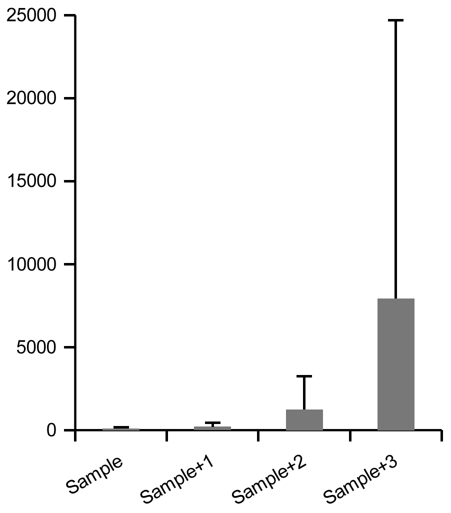


Figure 3.17: Relationship between the dialog box area, time (left) and desirability (right) of the example UI model added three widget. The y axis is a time scale (msec.) (left) or desirability (right) and the x axis is the area of dialog boxes (pixel^2).



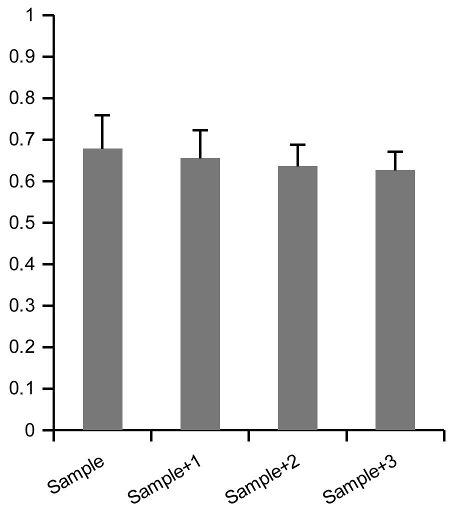


Figure 3.18: Relationship between the complexities of UI model and both of the average layout times (left) and the desirability (right), when using the forward checking with the dynamic variable order. The y axis is a time scale (msec.) (left) or desirability (right) and the x axis is the models. The error bars are the standard deviations of each model.

3.6 Conclusion and Future Work

3.6.1 Conclusion

In this chapter, a new sort of layout problem accompanied by widget selection was showed and named the flexible widget layout (FWL) problem. After that, the problem was formulated as a fuzzy constraint satisfaction problem (FCSP), and a layout system for the FWL was offered, which solves the problem in a practical time, and generates dynamically GUIs.

The FWL is the automatic layout of GUI widgets based on the logical descriptions, which requires considering how to decide which widgets should be used and how to complete the layout in a practical time. In this chapter, the FWL was handled as a combinatorial optimization problem. The widgets used in our implementation are a typical subset of ones in existing GUI toolkits; hence, the problem addressed in this thesis is a small size one. Besides, the constraints in the problem are used for only parent-child compositions and desirability of widgets. These limitations, however, are posed just in the current implementation, but not in our approach itself. 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 perform layouts by hand.

The first point of this chapter is that, for the challenge of UI, the achievements in the FCSP domain of artificial intelligence field are utilized, where the proposal method is one of its practical applications. The constraints of the FWL problem include subjective ones involved with usability, sensitivity, etc; therefore, fuzziness of constraints was introduced. The formulation of the problem was done in the existing FCSP framework without not to be added any extension, and thus, existing techniques can be usable. It means future achievements in the field of FCSP will be also utilized for the FWL problem.

The second point of this chapter is that the layout method for the FWL is offered as the layout system which can solve the layout in feasible time. The time-sensitiveness of the problem is the remarkable character of this work, which is not emphasized in the related studies, and some optimization techniques were tried and evaluated. Some experiments have showed the size-sensitiveness of the problem, and the uniqueness of the structure of its constraint graph when being solved as a FCSP. To mention that the speed of layout is enough for the purpose requires more experiments for researching the relation between the scale of problem, meaning the complexity of layout, and the layout speed.

3.6.2 Future work

For advancing this work, it is necessary to add some layout rules based on GUI guidelines, to evaluate the relation between problem scales and solving times, and to consider other FCSP algorithms.

In this thesis, parental composition and widget desirability are used as layout rules, but other rules can be added to the FWL. For example, constraints between sibling widgets can be added for keeping the same types and states among them for aesthetic, which does not exist in the current implementation. If the system can available the semantic information between UI elements, it might be used for alignment of corresponding widgets and reordering sibling widgets. As mentioned as another consideration, how to determine the desirability of widgets is one of the topics related to layout rules. In addition, as mentioned in Section 3.1.3, facility layout problem will be usable for making the grouping and tree-structurizing of UI elements before relating them to widget candidates.

Generally speaking, it is difficult to evaluate the effectiveness and usability of UI systems, which include the layout system offered in this thesis. There are consideration extracting GUI dialog boxes from existing applications and reconstructing them with the method introduced in this thesis. It might be able to evaluate the availability of the layout system. In the perspective of usability, whether the result of layout is useful or not might be evaluated using some examinees and questionnaires. There might be some evaluation techniques for GUI dialog box; therefore they might be researched and used for it.

Algorithms used for the FWL in this thesis also need to be reconsidered more for improving the layout system. In the current implementation, the forward checking algorithm is used, which is systematic search algorithm, but the layout method is not limited to this algorithm. As formulated as FCSP, the layout problem allows using suboptimal results. Therefore, it can be considered to adopt some local search (stochastic) algorithms such as the breakout [57], the local changes [58], and the fuzzy GENET (FGENET) [59, 60] as alternatives other than the forward checking. In addition, one of the hybrid algorithms of systematic and local search, SRS [54] should be examined. When trying these algorithms, the tree structure of the constraint graph of the problem might be considered. Moreover, for stabilizing the result after re-layout accompanied with the size changing of the dialog box, the stability of solutions of FCSP also needs to be considered. Because the stability of the FWL requires considering some problem specific issues such as the similarity among widgets, it is somewhat different from the simple one of FCSP.

Chapter 4

General Conclusion

4.1 Conclusion

This thesis is separated into the two parts: the part of the UI architecture, interface client/logic server (ICLS), and the part of the GUI generation on the architecture, flexible widget layout (FWL). As mentioned in Chapter 1, the objectives through the whole of the thesis are the two: the first one is offering the architecture that enables users to utilize computer-mediated services through their preferred UIs; another is developing the UIs themselves that correspond to users' preferences for UIs. For the first objective, in Chapter 2, I mentioned the two challenges: the adaptive UIs and the migratory UIs should be handled, and offered the UI architecture handling them. For the second objective, in Chapter 3, I mentioned the necessity of dealing with the layout problem occurs when GUIs are generated from logical descriptions, and offered its solution.

The two topics of the studies are related to each other, but they are also independent. I first started tackling the ICLS as a study of UI architecture, and in the study, I noticed that the necessity of the consideration of UI generation method for the ICLS. In this sense, the study of the FWL is the derivation of the first study. However, the FWL is independent of the original study, because, the method proposed in Chapter 3 is not limited to the use as an interface client. I summarize each part of this thesis respectively and the relation of each other below.

The interface client/logic server is a UI architecture designed for flexible exchange of UIs of services and customizing, and consequently for enabling users to utilize their preferred UIs. In Chapter 2, the users' new demands for services were raised, and adaptive and migratory UIs are handled as the solution for the demands. The ICLS architecture was shown as a new solution, which not only supports the migratory UIs but also offers the adaptive UIs. The points of the chapter are both that the ICLS is supports the migratory and simultaneous UIs as its design; and that the AIDL can express the meanings of interactions for describing service-specific interactions. In addition, for showing the feasibility, I explain the two examples of UIs based on the UI architecture, the interface clients. The study of the ICLS itself shows nothing about how to generate actual UIs from the description of the AIDL, and therefore, as the second topic, the next chapter was needed.

The flexible widget layout is a solution for the challenge how to generate user-preferred UIs on the ICLS architecture, and one example of user preferred UIs. In Chapter 3, a new sort of layout problem accompanied by widget selections was presented and named the flexible widget layout (FWL) problem. The FWL is the automatic layout of GUI widgets based on the logical descriptions, which requires considering both how to decide which widgets should be used; and how to complete the layout in a practical time. The points of the chapter are both that, for the challenge of UI, the achievements in the FCSP domain of artificial intelligence field are utilized; and that the layout method for the FWL is offered as the layout system which can solve the layout problem in feasible time. By offering the FWL, the efficiency of the AIDL and describing UIs with abstracted manner was shown, and the feasibility of the ICLS was shown.

4.2 Future Work

This thesis has left plenty of room for the improvements of the proposal method. In addition, as the study of the FWL is a derivation of the study of the ICLS, various studies might be considered as the derivation of the two studies (Figure 4.1).

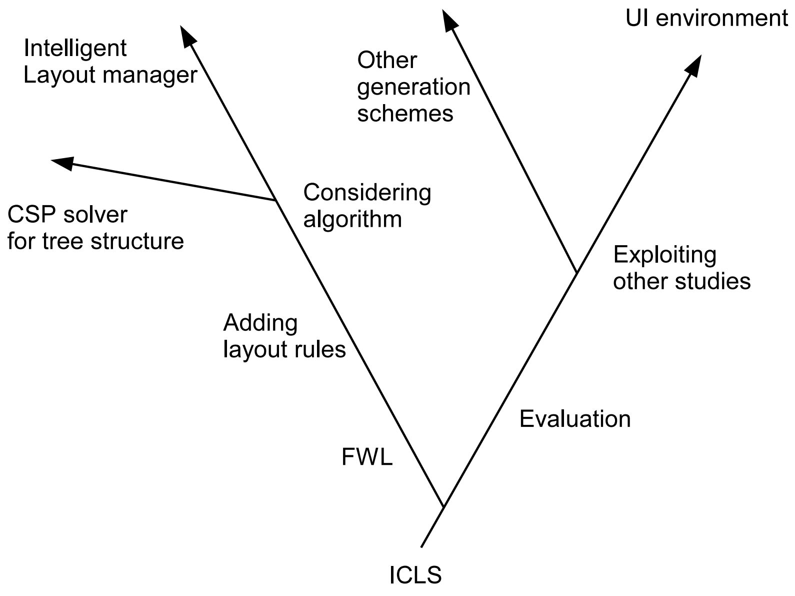


Figure 4.1: Derivations of this study as future work. Because this study concludes some cross-cutting elements, there might exist various sorts of derivation studies.

The ultimate goal of the study of the ICLS is to establish the architecture and its framework as an infrastructure of UIs. For that, it is necessary to evaluate its availability, to advance the architecture, and to consider exploiting outcomes of other work. There are three points to be evaluated: the feasibility of the ICLS, the description capability of the AIDL, and the usability of services based on the architecture. As other studies of derivation from the study of ICLS, I need to consider generation schemes of UIs and the mechanism for choosing appropriate UI components with given meanings. It is also necessary to consider exploiting achievements of other studies and existing technologies.

The study of the FWL problem contains some other studies to be handled as future work. As mentioned in Chapter 3, it is necessary to add other layout rules based on GUI guidelines, to evaluate the relation between problem scales and solving times, and to consider other FCSP algorithms. In this thesis, adopted layout rules and widgets are limited, but for fully implementing the specification of the ICLS, additional widgets are needed. Algorithms used for the FWL in this thesis also need to be reconsidered, because the FCSP of the FWL has specific character; the FCSP is structured as a tree, which is not general in ordinal FCSPs. The study of tree-structured FCSP might be another topic in the field of FCSP. In addition, it also has possibility to be used as an intelligent layout manager in GUI toolkits.

Appendix A

Definition and Examples of the AIDL

A.1 Definition of the AIDL with RELAX NG

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:aidl="http://aiwww.main.ist.hokudai.ac.jp/~takty/aidlns
      /">
  <start>
    <element name="aidl:field">
      <interleave>
        <ref name="dialog"/>
        <optional>
          <element name="knowledge">
            <ref name="rdf"/>
          </element>
        </optional>
      </interleave>
    </element>
  </start>
  <define name="dialog">
    <element name="aidl:dialog">
      <ref name="id"/>
      <attribute name="serviceID"/>
      <attribute name="baseURI"/>
      <interleave>
        <optional>
          <ref name="description"/>
        </optional>
        <optional>
          <ref name="disabled"/>
      </optional>
      <zeroOrMore>
        <ref name="presentation"/>
      </zeroOrMore>
      <zeroOrMore>
        <ref name="selection"/>
      </zeroOrMore>
      <zeroOrMore>
        <ref name="group"/>
      </zeroOrMore>
      <optional>
        <element name="user">
          <attribute name="id"/>
        </element>
      </optional>
    </interleave>
  </element>
</define>
<define name="group">
  <element name="aidl:group">
    <ref name="id"/>
    <interleave>
      <optional>
        <ref name="description"/>
      </optional>
      <optional>
        <ref name="disabled"/>
      </optional>
      <zeroOrMore>
        <ref name="presentation"/>
      </zeroOrMore>
      <zeroOrMore>
        <ref name="selection"/>
      </zeroOrMore>
      <zeroOrMore>
        <ref name="group"/>
      </zeroOrMore>
    </interleave>
  </element>
</define>
<define name="presentation">
  <element name="aidl:presentation">
    <ref name="id"/>
    <interleave>
      <choice>
        <interleave>
```

```
            <attribute name="aidl:caption"/>
            <optional>
              <attribute name="aidl:abbr"/>
            </optional>
            <optional>
              <attribute name="aidl:message"/>
            </optional>
          </interleave>
          <interleave>
            <optional>
              <attribute name="aidl:caption"/>
            </optional>
            <optional>
              <attribute name="aidl:abbr"/>
            </optional>
            <attribute name="aidl:message"/>
          </interleave>
        </choice>
        <optional>
          <element name="knowledge">
            <ref name="rdf"/>
          </element>
        </optional>
      </interleave>
    </element>
  </define>
  <define name="selection">
    <element name="aidl:selection">
      <ref name="id"/>
      <interleave>
        <optional>
          <ref name="description"/>
        </optional>
        <optional>
          <ref name="disabled"/>
        </optional>
        <optional>
          <choice>
            <element name="aidl:resources">
              <choice>
                <oneOrMore>
                  <element name="aidl:item">
                    <attribute name="aidl:uri"/>
                    <optional>
                      <ref name="description"/>
                    </optional>
                  </element>
                </oneOrMore>
                <attribute name="aidl:regex"/>
              </choice>
            </element>
            <element name="aidl:numerics">
              <choice>
                <oneOrMore>
                  <element name="aidl:item">
                    <attribute name="aidl:num"/>
                    <empty/>
                  </element>
                </oneOrMore>
                <group>
                  <attribute name="aidl:min"/>
                  <attribute name="aidl:frequency"/>
                  <attribute name="aidl:max"/>
                </group>
              </choice>
            </element>
            <element name="aidl:strings">
              <choice>
                <oneOrMore>
                  <element name="aidl:item">
                    <attribute name="aidl:str"/>
                    <empty/>
                  </element>
                </oneOrMore>
                <attribute name="aidl:regex"/>
              </choice>
            </element>
          </choice>
        </optional>
        <optional>
          <element name="aidl:state">
            <text/>
          </element>
        </optional>
      </interleave>
    </element>
  </define>
  <define name="disabled">
    <element name="aidl:disabled">
      <empty/>
    </element>
  </define>
```

```
<define name="description">
  <element name="aidl:description">
    <choice>
      <group>
        <attribute name="aidl:caption"/>
        <optional>
          <attribute name="aidl:abbr"/>
        </optional>
        <optional>
          <attribute name="aidl:message"/>
        </optional>
      </group>
      <group>
        <optional>
          <attribute name="aidl:caption"/>
        </optional>
        <optional>
          <attribute name="aidl:abbr"/>
        </optional>
        <attribute name="aidl:message"/>
      </group>
    </choice>
  </element>
</define>
<define name="id">
  <optional>
    <attribute name="aidl:id">
      <text/>
    </attribute>
  </optional>
</define>
<define name="rdf">
  <element>
    <choice>
      <nsName ns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          />
      <nsName ns="http://www.w3.org/2000/01/rdf-schema#"/>
    </choice>
    <zeroOrMore>
      <choice>
        <attribute>
          <choice>
            <nsName ns="http://www.w3.org/1999/02/22-rdf-
                syntax-ns#"/>
            <nsName ns="http://www.w3.org/2000/01/rdf-schema#"
                />
          </choice>
        </attribute>
        <text/>
        <ref name="rdf"/>
      </choice>
    </zeroOrMore>
  </element>
</define>
</grammar>
```

A.2 Examples of the AIDL

Desk lamp control service

The following XML document is a simple example of AIDL documents (descriptions in AIDL) for the service of a desk lamp remote control. It expresses the UI functions of the service; it has two functions: power state (on and off) selection, and brightness (bright, normal, and dim) selection.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<aidl:aidl xmlns:aidl="http://aiwww.main.ist.hokudai.ac.jp/~takty/aidlns/">
  <aidl:session aidl:sessionID="http://aiwww.main.ist.hokudai.ac.jp/~takty/desklamp/"/>
  <aidl:dialog>
    <aidl:description aidl:caption="Desk␣Lamp␣Service"/>
    <aidl:presentation aidl:message="Let's␣keep␣energy␣saving!!"/>
    <aidl:selection aidl:id="ps" aidl:meaning="http://aiwww.main.ist.hokudai.ac.jp/~takty/aidl/LampPowerState">
      <aidl:description aidl:caption="Power"/>
      <aidl:state>http://www.example.com/Off</aidl:state>
      <aidl:resources aidl:opposite="true">
        <aidl:choice aidl:uri="http://www.example.com/On">
          <aidl:description aidl:caption="On"/>
        </aidl:choice>
        <aidl:choice aidl:uri="http://www.example.com/Off">
          <aidl:description aidl:caption="Off"/>
        </aidl:choice>
      </aidl:resources>
    </aidl:selection>
    <aidl:selection aidl:id="bs" aidl:meaning="http://aiwww.main.ist.hokudai.ac.jp/~takty/aidl/Brightness">
      <aidl:disabled/>
      <aidl:description aidl:caption="Brightness"/>
      <aidl:state>http://www.example.com/Normal</aidl:state>
      <aidl:resources>
        <aidl:choice aidl:uri="http://www.example.com/Dim">
          <aidl:description aidl:caption="Dim"/>
        </aidl:choice>
        <aidl:choice aidl:uri="http://www.example.com/Normal">
          <aidl:description aidl:caption="Normal"/>
        </aidl:choice>
        <aidl:choice aidl:uri="http://www.example.com/Bright">
          <aidl:description aidl:caption="Bright"/>
        </aidl:choice>
      </aidl:resources>
    </aidl:selection>
   </aidl:dialog>
   <aidl:knowledge>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
   <rdf:Description rdf:about="http://aiwww.main.ist.hokudai.ac.jp/~takty/aidl/LampPowerState">
   <rdfs:subClassOf rdf:resource="http://aiwww.main.ist.hokudai.ac.jp/~takty/aidl/PowerState"/>
   </rdf:Description>
 </rdf:RDF>
   </aidl:knowledge>
 </aidl:aidl>
```

Audio set control service

The following XML document is another example of AIDL documents (descriptions in AIDL) for the service of an audio system. It expresses the UI functions of the service; it has two functions: power state (on and off) selection, function (CD and radio) selection, volume (0 to 10) selection, frequency (76.0 to 92.0) selection (but it is not seen in the source because the function is switched the CD mode), and playback control (stop, play, pause, next, and previous).

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<aidl:aidl xmlns:aidl="http://aiwww.main.ist.hokudai.ac.jp/~takty/aidlns/" aidl:baseURI="http://www.example.com/avc">
  <aidl:dialog aidl:id="http://www.example.com/avc">
    <aidl:description aidl:caption="Stereo"/>
    <aidl:selection aidl:meaning="http://aiwww.main.ist.hokudai.ac.jp/~takty/aidl/PowerState">
      <aidl:description aidl:caption="Power"/>
      <aidl:state>http://www.example.com/Off</aidl:state>
      <aidl:resources aidl:opposite="true">
        <aidl:choice aidl:uri="http://www.example.com/On">
          <aidl:description aidl:caption="On"/>
        </aidl:choice>
        <aidl:choice aidl:uri="http://www.example.com/Off">
          <aidl:description aidl:caption="Off"/>
        </aidl:choice>
      </aidl:resources>
    </aidl:selection>
    <aidl:selection aidl:id="http://www.example.com/avc#s2">
      <aidl:description aidl:caption="Volume"/>
      <aidl:state>5</aidl:state>
      <aidl:numerics aidl:frequency="1" aidl:id="0" aidl:max="10" aidl:min="0">
```

```
            <aidl:mapping aidl:type="left_to_right"/>
         </aidl:numerics>
      </aidl:selection>
      <aidl:selection aidl:id="http://www.example.com/avc#s3">
         <aidl:description aidl:caption="Function"/>
         <aidl:state>http://www.example.com/avc#cd</aidl:state>
         <aidl:resources aidl:id="1">
            <aidl:choice aidl:uri="http://www.example.com/avc#
                radio">
               <aidl:description aidl:caption="Radio"/>
            </aidl:choice>
            <aidl:choice aidl:uri="http://www.example.com/avc#
                cd">
               <aidl:description aidl:caption="CD"/>
            </aidl:choice>
         </aidl:resources>
      </aidl:selection>
      <aidl:group aidl:id="http://www.example.com/avc#g2">
         <aidl:description aidl:caption="CD"/>
         <aidl:selection aidl:id="http://www.example.com/avc#s5
             " aidl:meaning="http://aiwww.main.ist.hokudai.ac.
             jp/~takty/aidl/Playback">
            <aidl:state>http://www.example.com/Stop</aidl:state
                >
            <aidl:resources aidl:id="3">
               <aidl:choice aidl:uri="http://www.example.com/
                   Stop">
                  <aidl:description aidl:caption="Stop"/>
               </aidl:choice>
               <aidl:choice aidl:uri="http://www.example.com/
                   Play">
                  <aidl:description aidl:caption="Play"/>
               </aidl:choice>
               <aidl:choice aidl:uri="http://www.example.com/
                   Pause">
                  <aidl:description aidl:caption="Pause"/>
               </aidl:choice>
            </aidl:resources>
         </aidl:selection>
         <aidl:selection aidl:id="http://www.example.com/avc#c1
             ">
            <aidl:description aidl:caption="Next"/>
         </aidl:selection>
         <aidl:selection aidl:id="http://www.example.com/avc#c2
             ">
            <aidl:description aidl:caption="Previous"/>
            </aidl:selection>
        </aidl:group>
    </aidl:dialog>
</aidl:aidl>
```

Appendix B

Summary of the ICLS library

The ICLS library consists of the following four packages: `jp.ac.hokudai.ist.main.aiwww.aidl`, `jp.ac.hokudai.ist.main.aiwww.core`, `jp.ac.hokudai.ist.main.aiwww.net`, and `jp.ac.hokudai.ist.main.aiwww.util`.

B.1 Package aidl

The package `ac.hokudai.ist.main.aiwww.aidl` contains interfaces, classes, and enums which represent the elements of AIDL and some facilities.

Interface summary

Class name	Description
AidlDocumentListener	The listener interface for receiving events invoked by an AIDL document.
GroupListener	The listener interface for receiving events invoked by a group element.
KnowledgeFactory	The interface of knowledge base factory.
MeaningHierarchy	The interface for expressing a UI meaning hierarchy.
SelectionListener	The listener interface for receiving events invoked by a selection element.

Class summary

Class name	Description
AidlDocument	The class for representing an AIDL document.

AidlElement	This class is the super class of the all classes of AIDL elements.
Choice	The abstract class for representing a choice.
ChoiceSet	The abstract class for a set of choices.
DescribableInteraction	The class for representing a describable interaction element.
Description	The class for representing a description element.
Dialog	The class for representing a dialog element.
Group	The class for representing a group element.
Interaction	The abstract class for representing a interaction element.
Knowledge	The class for representing a knowledge base of an AIDL document.
Mapping	The class for representing a choice mapping.
NumericSet	The class for representing a numeric choice set.
Presentation	The class for representing a presentation element.
ResourceSet	The class for representing a resource choice set.
Selection	The class for representing a selection element.
SelectionAdapter	The adapter of a selection listener, which offers an empty implementation of SelectionListener.
StringSet	The class for representing a string choice set.

Enum summary

Class name	Description
Choice.Type	The enum for representing the type of a choice.
ChoiceSet.QualifierType	The enum for representing the constraint of a choice set.
Group.Ordering	The enum for representing the order of child elements of a group element.
Mapping.Type	The enum for representing the type of a mapping of choices.

B.2 Package core

The package `ac.hokudai.ist.main.aiwww.core` contains interfaces and classes which represent the ICLS framework, or the entities of the architecture.

Interface summary

Class name	Description

InterfaceClient	The interface for representing an interface client.
LogicServer	The interface for representing a logic server.
LogicServerOutlet	The interface for representing an outlet, which waits for connections from clients and connects to servers.
PaneClientListener	The listener interface for receiving events for clients invoked by a pane.
PaneServerListener	The listener interface for receiving events for servers invoked by a pane.

Class summary

Class name	Description
InterfaceClientProxy	The proxy class of an interface client.
LogicServerProxy	The proxy class of a logic server.
Pane	The class for representing an interaction pane.
PaneClientAdapter	The listener adapter for handling events for clients by pane.
Proxy	The class for handling functions as proxy of both interface clients and logic servers.
ServiceDetector	The class of a service detector.
ServiceDetector .ServiceInformation	The class for storing service information which is found by a service detector.
ServiceProvider	The class for representing a service provider.
SocketLogicServerOutlet	The class of a socket outlet for logic servers.
TemplateNode	The class for representing a node of UI meaning templates.
TemplateNode .MatchResult	The class for representing the result of UI meaning template matching.

B.3 Package net

The package `ac.hokudai.ist.main.aiwww.net` contains interfaces and classes which represent the extended sockets of networking, and are independent of the ICLS.

Interface summary

Class name	Description
MessageListener	The listener interface for receiving message arriving and disconnecting events.

SimpleServerSocketListener	The listener interface of connection to server sockets which are waiting for connections.

Class summary

Class name	Description
MessageSocket	Sockets for handing string messages.
SimpleServerSocket	A simple wrapper of a server socket.
SimpleSocket	A simple wrapper of a socket, which handles multiple lines of strings.

B.4 Package util

The package `ac.hokudai.ist.main.aiwww.util` contains interfaces, classes, and enums which represent the elements of AIDL and some facilities.

Class summary

Class name	Description
AidlUtilities	The utility class for being used when AIDL documents are generated.

Acknowledgements

My deepest appreciation goes to my supervisor, Associate Professor Hidetoshi Nonaka in the laboratory of intelligent information systems (Hokkaido University) for his continuous guidance, valuable advice, and helpful discussions. His comments and suggestions were of inestimable value for my study for six and a half years. I have been impressed in his ideas and the way of thinking, and I would not study on the human computer interaction without his kind advice. Special thanks go to Professor Masaaki Miyakoshi in the laboratory of mathematical science for computing; Professor Mineichi Kudo in the laboratory for pattern recognition and machine learning; and Masahito Kurihara in the formative system engineering laboratory (Hokkaido University). Their comments and advices made enormous contribution to my thesis. Especially, Professor Kurihara was my supervisor when I wrote my master thesis before, and now when I wrote the doctoral thesis, he also gave me kind advice on the artificial intelligence.

I am grateful to thank Assistant Professor Yasuhiro Sudo (Kanagawa Institute of Technology) for his kind advice on the field of constraint satisfaction problems (CSP). The flexible layout problem is based on his research field CSP, and discussion with him has inspired me. Without his help, I can not only study the problem but also I would not write the last half of this thesis. Moreover, it was him who told me how wonderful soup curry was. I also thank Assistant Professor Takeshi Yoshikawa in the laboratory of intelligent information systems (Hokkaido University) for his advice in the same period of time. His opinions were very useful for me especially when I was trying to formulate the layout problem mentioned in Chapter 3. Without discussions with him, I would have not achieved it.

My gratitude is to all the former and present members of the laboratory of intelligent information systems in Hokkaido University. Although I cannot write down the name of every member, I have spent a great time with the colleagues. Active discussions with them, talks on coffee and alcohol, and kind advices from them are the greatest help for me. Furthermore, I would like to thank all the people who gave me useful comments on the research in workshops, symposiums, and conferences for discussions, cooperation, and friendship.

Finally, I would also like to express my gratitude to my parents, brother, and grandparents for their moral support, patience, and warm encouragements. In addition, I cannot forget to be grateful for my dog Pal, who always brings relief for me with his cute face and nice brown fur.

This thesis is partially supported by the program of the cultivating young researchers of the Hokkaido University global COE “the formation of a center for next generation information technology that supports knowledge creation”.

Bibliography

[1] Ken-ichi Okada, Shogo Nishida, Hideaki Kuzuoka, Mie Nakatani, and Hidekazu Shiozawa. *Human computer interaction*. Ohmsha, 2002. (in Japanese).

[2] Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland, and Tom Carey. *Human-Computer Interaction*. ADDISON-WESLEY PUBLISHING COMPANY, 1994.

[3] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Proceedings of the 1st IEEE Workshop on Mobile Computing Systems and Applications (WMCSA1994)*, pages 85–90, Santa Cruz, CA, USA, 1994. IEEE.

[4] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A context-based infrastructure for smart environments. Technical Report GIT-GVU-99-39, Georgia Institute of Technology, 1999.

[5] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.

[6] Renata Bandelloni and Fabio Paternò. Flexible interface migration. In *Proceedings of the 2004 International Conference on Intelligent User Interfaces (IUI 2004)*, pages 148–155, Funchal, Madeira, Portugal, 2004. ACM.

[7] Takuto Yanagida and Hidetoshi Nonaka. Architecture for migratory adaptive user interfaces. In *Proceedings of the IEEE 8th International Conference on Computer and Information Technology (CIT 2008)*, pages 450–455, Sydney, Australia, July 2008. IEEE.

[8] Takuto Yanagida and Hidetoshi Nonaka. Interaction description with service-specific meanings. In *Proceedings of the 5th International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA 2008)*, pages 185–188, Orlando, FL, USA, July 2008. IIIS.

[9] Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta. Applying model-based techniques to the development of UIs for mobile computers. In *Proceedings of the 2001 International Conference on Intelligent User Interfaces (IUI 2001)*, pages 69–76, Santa Fe, NM, USA, 2001. ACM.

[10] Jean M. Vanderdonckt and François Bodart. Encapsulating knowledge for intelligent automatic interaction objects selection. In *Proceedings of CHI '93*, pages 424–429, Amsterdam, The Netherlands, 1993. ACM.

[11] Renata Bandelloni, Giulio Mori, and Fabio Paternò. Dynamic generation of migratory interfaces. In *Proceedings of Mobile HCI 2005*, pages 83–90, Salzburg, Austria, 2005. ACM.

[12] Stina Nylander, Markus Bylund, and Annika Waern. The ubiquitous interactor–device independent access to mobile services. In *Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces (CADUI 2004)*, pages 274–287, Funchal, Portugal, 2004. Kluwer.

[13] Stina Nylander, Markus Bylund, and Annika Waern. Ubiquitous service access through adapted user interfaces on multiple devices. *Personal and Ubiquitous Computing*, 9(3):123–133, 2005.

[14] Stina Nylander. Semi-automatic generation of device adapted user interfaces. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST 2005) (doctoral symposium)*, page 4, Seattle, WA, USA, 2005. ACM.

[15] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld, and Mathilde Pignol. Generating remote control interfaces for complex appliances. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology (UIST '02)*, pages 161–170, Paris, France, 2002. ACM.

[16] Jeffrey Nichols, Brad A. Myers, Kevin Litwack, Michael Higgins, Joseph Hughes, and Thomas K. Harris. Describing appliance user interfaces abstractly with xml. In *Developing User Interfaces with XML: Advances on User Interface Description Languages*, 2004.

[17] Jeffrey Nichols, Brad A. Myers, and Kevin Litwack. Improving automatic interface generation with smart templates. In *Proceedings of the 2004 International Conference on Intelligent User Interfaces (IUI 2004)*, pages 286–288, Funchal, Madeira, Portugal, 2004. ACM.

[18] Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, and Jonathan E. Shuster. UIML: An appliance-independent XML user interface language. In *Proceedings of the 8th International World Wide Web Conference*, pages 1695–1708, Toronto, Canada, 1999. Elsevier Science.

[19] Krishna A. Bharat and Luca Cardelli. Migratory applications. In Jan Vitek and Christian Tschudin, editors, *Proceedings of the 8th Annual ACM Symposium on User Interface Software and Technology (UIST '95)*, volume 1222, pages 131–148. Springer-Verlag: Heidelberg, Germany, 1995.

[20] Dan R. Olsen, Sean Jefferies, S. Travis Nielsen, William Moyes, and Paul Fredrickson. Cross-modal interaction using XWeb. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST 2000)*, pages 191–200, San Diego, CA, USA, 2000. ACM.

[21] Todd D. Hodes and Randy H. Katz. A document-based framework for internet application control. In *Proceedings of the 2nd Conference on USENIX Symposium on Internet Technologies and Systems*, pages 59–70, Boulder, CO, USA, 1999. USENIX Association.

[22] Shankar R. Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. ICrafter: A service framework for ubiquitous computing environments. *Lecture Notes in Computer Science*, 2201:56–74, 2001.

[23] Google. Gmail, 2008. Available at `http://mail.google.com/`.

[24] James Clark, Makoto Murata, and OASIS. RELAX NG, 2001. Available at `http://www.relaxng.org/`.

[25] Eric Miller, Ralph Swick, and Dan Brickley. Resource description framework (RDF), 2004. Available at `http://www.w3.org/RDF/`.

[26] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. The MIT Press, 2004.

[27] Masahide Kanzaki. *An Introduction to RDF/OWL for Semantic Web*. Morikita Shuppan Co., Ltd., 2004. (in Japanese).

[28] Eric Miller. Digital Libraries and the Semantic Web, 2001. Available at `http://www.w3.org/2001/09/06-ecdl/`.

[29] Hewlett-Packard Development Company, L.P. Jena - a semantic web framework for java. Available at `http://jena.sourceforge.net/`.

[30] Sun Microsystems. FreeTTS 1.2, 2005. Available at `http://freetts.sourceforge.net/`.

[31] Sun Microsystems, Inc. Java speech API, 2008. Available at `http://java.sun.com/products/java-media/speech/`.

[32] Simon Lok and Steven Feiner. A survey of automated layout techniques for information presentations. In *Proceedings of the 1st International Symposium on Smart Graphics*, pages 61–68, Hawthorne, NY, USA, 2001. ACM.

[33] Zsófia Ruttkay. Fuzzy constraint satisfaction. In *Proceedings of the 3rd IEEE Conference on Fuzzy Systems*, pages 1263–1268, Orlando, FL, USA, 1994. IEEE.

[34] Murielle Florins and Jean Vanderdonckt. Graceful degradation of user interfaces as a design method for multiplatform systems. In *Proceedings of the 2004 International Conference on Intelligent User Interfaces (IUI 2004)*, pages 140–147, Madeira, Funchal, Portugal, 2004. ACM Press.

[35] Murielle Florins, Francisco Montero Simarro, Jean Vanderdonckt, and Benjamin Michotte. Splitting rules for graceful degradation of user interfaces. In *Proceedings of the 8th International Working Conference on Advanced Visual Interfaces (AVI 2006)*, pages 59–66, Venice, Italy, 2006. ACM Press.

[36] Benoît Collignon, Jean Vanderdonckt, and Gaëlle Calvary. An intelligent editor for multi-presentation user interfaces. In *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC 2008)*, pages 1634–1641, Fortaleza, Ceará, Brazil, 2008. ACM Press.

[37] S. P. Singh and R. R. K. Sharma. A review of different approaches to the facility layout problems. *The International Journal of Advanced Manufacturing Technology*, 30:425–433, 2006.

[38] Russell D. Meller and Kai-Yin Gau. The facility layout problem: Recent and emerging trends and perspectives. *Journal of Manufacturing Systems*, 15:351–366, 1996.

[39] S. K. Peer and Dinesh K. Sharma. Human-computer interaction design with multi-goal facilities layout model. *Computer and Mathematics with Applications*, 56:2164–2174, 2008.

[40] Hitoshi Kitazawa. Overview of the LSI layout CAD algorithms and their applications to image processing. In *Technical Report of the Institute of Electronics, Information and Communication Engineers VLD2006–38*, volume 106, pages 25–30, 2006. (in Japanese).

[41] Sun Microsystems, Inc. JDK 6 swing (java foundation classes), 2005. Available at `http://java.sun.com/javase/6/docs/technotes/guides/swing/index.html`.

[42] Microsoft Corporation. Windows forms. Available at `http://msdn2.microsoft.com/en-us/netframework/aa497342.aspx`.

[43] Inc. Embarcadero Technologies. Codegear home page. Available at `http://www.codegear.com/`.

[44] The GTK+ Team. The GTK+ project. Available at `http://www.gtk.org/`.

[45] Trolltech ASA. Qt. Available at `http://trolltech.com/products/qt/`.

[46] Apple Inc. Apple human interface guidelines, 6 2008. Available at `http://developer.apple.com/documentation/UserExperience/Conceptual/AppleHIGuidelines/OSXHIGuidelines.pdf`.

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

[48] Jenifer Tidwell. *Designing Interfaces*. O'Reilly Media, Inc., 2005.

[49] Fahiem Bacchus and Peter van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of 15th National Conference on Artificial Intelligence (AAAI-98)*, pages 311–318, Madison, WI, USA, 1998. AAAI Press/The MIT Press.

[50] Kostas Stergiou and Toby Walsh. Encodings of non-binary constraint satisfaction problems. In *Proceedings of the 16th national conference on artificial intelligence and the 11th conference on innovative applications of artificial intelligence (AAAI '99/IAAI '99)*, pages 163–168, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[51] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.

[52] Pedro Meseguer and Javier Larrosa. Solving fuzzy constraint satisfaction problems. In *Proceedings of the 6th IEEE International Conference on Fuzzy Systems*, volume 3, pages 1233–1238, Barcelona, Spain, 1997. IEEE.

[53] Fahiem Bacchus and Paul van Run. Dynamic variable ordering in CSPs. In *Proceedings of the 1st International Conference on Principles and Practice of Constraint Programming (CP '95)*, pages 258–275, Cassis, France, 1995. Springer.

[54] Yasuhiro Sudo and Masahito Kurihara. Spread-repair-shrink: A hybrid algorithm for solving fuzzy constraint satisfaction problems. In *Proceedings of the 2006 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2006)*, pages 2127–2133, Vancouver, BC, Canada, 2006. IEEE.

[55] Yasuhiro Sudo, Masahito Kurihara, and Takuto Yanagida. Keeping the stability of solutions in dynamic fuzzy CSPs. In *Proceedings of the 2008 IEEE Conference on Soft Computing in Industrial Applications (SMCia/08)*, pages 382–386, Muroran, Japan, 2008. IEEE.

[56] Yasuhiro Sudo, Masahito Kurihara, and Takuto Yanagida. Keeping the stability of solutions to dynamic fuzzy CSPs. In *Proceedings of the 2008 IEEE International Conference on Systems, Man and Cybernetics (SMC 2008)*, Singapore, 2008. IEEE.

[57] Paul Morris. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pages 40–45, Washington, DC, USA, 1993. AAAI Press/The MIT Press.

[58] Gérard Verfaillie and Thomas Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 307–312, Seattle, WA, USA, 1994. AAAI Press.

[59] Edward P. K. Tsang and Chang J. Wang. A generic neural network approach for constraint satisfaction problems. In J. G. Taylor, editor, *Proceedings of the 2nd British Neural Network Society Meeting (Neural Network Applications)*, pages 12–22, London, UK, 1992. Springer-Verlag.

[60] Jason H. Y. Wong and Ho fung Leung. Extending GENET to solve fuzzy constraint satisfaction problems. In *Proceedings of the 15th national/10th conference on Artificial intelligence/Innovative applications of artificial intelligence (AAAI '98/IAAI '98)*, pages 380–385, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.