User-Preferred Interface Design with Abstract Interaction Description Language

Takuto Yanagida, Hidetoshi Nonaka, and Masahito Kurihara

***Abstract*— For user-preferred interfaces, we propose a service architecture, the Interface-Client/Logic-Server (ICLS). It separates specific interfaces from services. An ICLS-based service consists of some interface clients and a logic server. The target of this work is the services with intermediary computers in various scenes of our daily activities. Nowadays, against various users' characteristics, the services offer different UIs individually, and they are mostly single GUI. In ICLS, users can switch interfaces of services to suit their preferences and to customize flexibly. Interface clients and logic servers work together independently with common descriptions of interaction. Those descriptions are written in the Abstract Interaction Description Language (AIDL), we propose. AIDL is an application of semantic web technologies, and describes interactions as graphs. These graphs represent a specification of interfaces and the current state of interactions constantly. We propose a framework of ICLS-based service design. Developers can use it like GUI toolkits. With this framework, we developed three clients and two servers.**

I. INTRODUCTION

We propose a service architecture, the *Interface-Client/Logic-Server* (ICLS). It enables users to switch interfaces of services to suit their preferences and to customize flexibly (Fig. 1). A user can use one service by any interface as well as any service by one interface. Furthermore, ICLS permits developing services regardless of details of their interfaces. Thus, ICLS separates interface developers from service developers. We call this environment of ICLS-based services *user-preferred interfaces*.

ICLS separates specific interfaces from services, and constructs a service of some *interface clients* and a *logic server*. Users prepare the client, and service developers set up the server. The clients handle characteristic processes of specific interfaces in all services. On the other hand, the servers handle common processes of all interfaces and each service's own logic processes. The clients and the server work together independently with some *interaction graphs*. These interaction graphs mean specifications of interactions between users and services written in the *Abstract Interaction Description Language* (AIDL), we propose. We also introduce the *Presentation and Indication Model* (PIM). It is a model of interaction with AIDL. In this way, interface clients and logic servers cooperate to compose services.

In this work, we developed a framework of ICLS and several examples.

T. Yanagida, H. Nonaka, and M. Kurihara are with Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan {takty, nonaka, kurihara}@main.ist.hokudai.ac.jp

II. BACKGROUND

The target of this work is the *services* with intermediary computers in various scenes of our daily activities. The services include, for example, web applications such as mail-order of books or CDs, and information appliances such as VCRs operated from remote places. Moreover, they include application softwares on standalone PCs, and immediate electric products with embedded computers. The occasions to access interfaces of the services increase with the pervasion of such services on our daily life.

Under the present circumstances, in spite of individual differences of *users' characteristics*, most of the services offer only fixed and predefined UIs, and they are mostly GUIs with respective design concepts. The users' characteristics involve their environments, physical descriptions, and preferences for interfaces. Services do not consider them in most cases. From an aspect of accessibility, it is desirable to be able to use the services comfortably regardless of their physical disabilities. However, existing approaches are just developing substitutes of traditional mouses or keyboards. Moreover, universal design techniques have not coped with variation of individual users. The inconvenience has been forced on users today.

For service developers, it is hard to make interfaces fitting in various users on each service, because of a combinatorial explosion. To prepare all interfaces for m users on n services, developers have to develop $m \cdot n$ interfaces. Even today, complicated interfaces increase the ratio of interface processes in application codes, and require huge amount of work for their implementations. Using our service architecture, the number of interfaces is reduced to $m+n$; consequently, time and cost for development are prominently slimmed down.

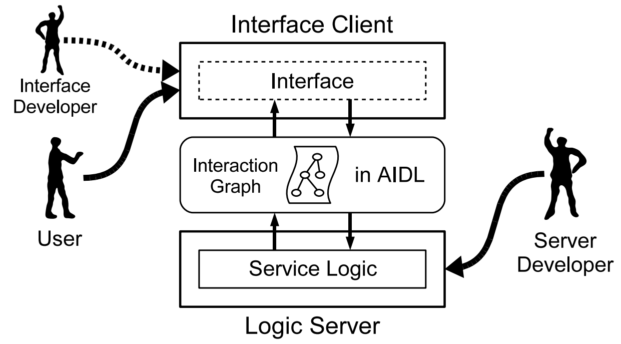


Fig. 1. Interface Client/Logic Server (ICLS) architecture. Services are consisted of users' own clients and the servers provided by server developers. Both of them work together via interaction graphs.

III. Related Work

There are many works to deal with the problem about the services. We can classify them roughly into two categories: to support developers to implement interfaces on multi platforms, and to enable users to operate the services by typical interfaces. As the former's example, UIML [1] is an XML based language for developers to describe interfaces independently from specific platforms. It is effective when users' characteristics can be estimated in a design phase of the services. As the latter's example, the work of an application remote control system by GUI-character UI conversion [2] converts UIs of Windows applications to e-mail or web based interfaces. It has an advantage that existing applications can be used, though it limits its interfaces to mail or web based. The purposes of both are different from ours, which is to enable users to exchange interfaces.

We have to mention about the Ubiquitous Interactor (UBI) [3], [4] and the Personal Universal Controller (PUC) [5], [6]. UBI handles interactions as *interaction acts* and provides a way to develop the services independently from devices. Unlike ICLS, it does not completely entrust displays of interfaces to clients. PUC deal with information sent between appliances and clients as typed state variables. It has *Smart Templates* to address domain-specific design patterns. Different from ICLS, it does not focus on interactions.

IV. Interaction Model

We introduce here an interaction model, the Presentation and Indication Model (PIM). In this model, interactions are abstracted with *selection acts* of *choices* (Fig. 2). The selection acts consist of *presentation acts* and *indication acts*. The former stands for acts that service logics transmit the information about choices to the users. The latter stands for acts that the users transmit to the service logics which choices the users selected. PIM confines interactions to what they can be described as selections, and puts matters that involve highly cognitive processes in the service.

In PIM, how choice sets are provided defines user's operation. We categorize choice sets into the three types: *enumerate choice set*, *range choice set*, and *free choice set*. The first one is used to enumerate all choices. For example, when a user selects a power state of a certain electric appliances, the choice set is as follows: PowerStates = $\{\mathrm{On}, \mathrm{Off}\}$. The second one is used to describe maximums and minimums such as numeric. The last one is used when choices cannot be enumerated like input of users' names.

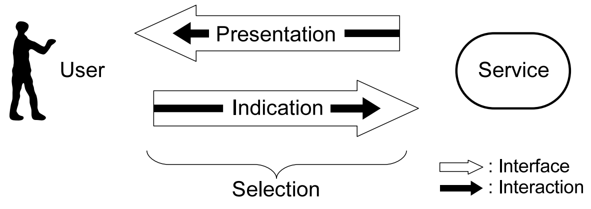


Fig. 2. Presentation and Indication Model (PIM). Interactions, abstracted as selection acts, consist of presentations and indications.

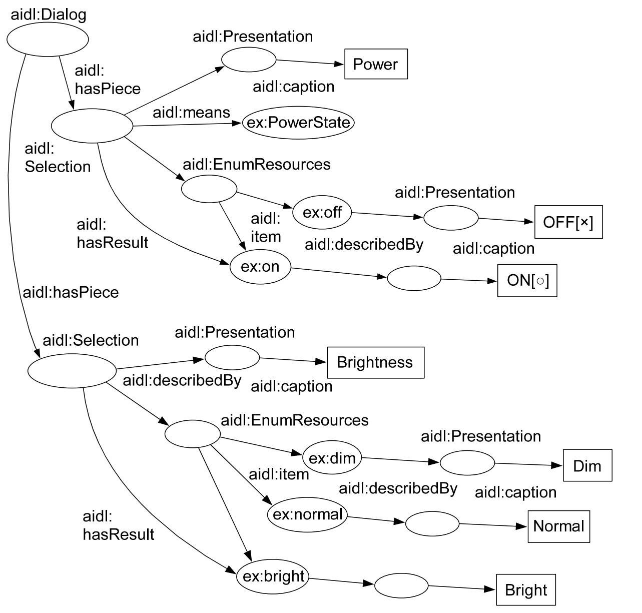


Fig. 3. Example of an interaction graph in AIDL (an interaction with a desk lamp). Ovals, rectangles, and arrows indicate RDF resources, literals, and properties respectively

Furthermore, PIM handles meanings of selection acts to improve concreteness of descriptions. A meaning can be assigned in a selection act optionally, and it helps interface clients to provide an appropriate affordance to users. For example of the power state, with the meaning of PowerState, the clients can show a user-friendly button with an icon. The meaning introduces a concrete expression of the power icon.

V. Graph Representation of Interaction

Interaction graphs are the representation of interaction based on PIM (Fig. 3). Selection acts, meanings of them, types of choice sets, choices, etc. are described as elements of these graphs. In ICLS, interface clients construct interfaces from the graphs. In other word, we substitute the description of interaction for these of interfaces. Unlike existing methods for interface description, the graphs have no information depends on specific devices and styles. Furthermore, interaction graphs also mean the current state of interactions, which is the result of users' operations.

Abstract Interaction Description Language (AIDL), we developed, is a vocabulary set of the Resource Description Framework (RDF), which is one standard of semantic web technologies [7], [8]. AIDL adopts the grammar and the syntax of RDF. We use its specification handling all resources by URI, in order to describe everything in the real world as choices. In addition, we use the categorization of resources by RDF classes to describe meanings of selection acts. Because the services need specific meanings of selection acts respectively, the increase of ICLS-based services brings the increase of the meanings. In AIDL, many communities and individuals can define meanings as RDF classes on demand; therefore, we do not have to define all of them in advance.

A. Description of Interaction

AIDL expresses indication acts, presentation acts, and selection acts as resources of corresponding RDF classes. These resources are grouped and have inclusive relation to make a tree structure in all. `aidl:Presentation` resources stand for presentation acts, and express explanations (such as captions) of other resources for users (Fig. 4-a). `aidl:Indication` resources stand for indication acts, and express that users have to make a sign in any way (Fig. 4-b). The users' responses to them are transmitted as a form of notification to the servers separately. An `aidl:Selection` resource, as a selection act, has a resource standing for a choice set to express choices (Fig. 4-c). It connects to a choice by `aidl:hasResult` property to express the result of users' selections. Meanings are assigned to both `aidl:Indication` and `aidl:Selection` resources by `aidl:means` property. An `aidl:Group` resource collects up related resources to make a tree in totality. An `aidl:Dialog` resource that extends `aidl:Group` becomes the root element in parent-child relation of a tree structure (Fig. 5).

B. Selection and Response of User

An `aidl:Selection` resource shows a selection act with a choice set and a meaning. The choice sets are defined by definition methods, default types (*resource*, *numeric*, *string*), and choices. Their resources connected via `aidl:selectedFrom` property express the choices. Interface clients use the default types when they cannot get a meaning on a selection, or do not support the meaning. The selection resources have RDF class resources through `aidl:means` property to express meanings of themselves.

Updating graphs represents the results of users' operations (Fig. 6). In enumeration choice sets, the clients connect one of existing choice resources to the `aidl:Selection` resource with `aidl:hasResult` property. In range choice sets or free choice sets, the clients connect a new literal corresponding with the result. On the other hand, because indication acts have no choice, the results of them are not written as the updating directly. However, we handle them with virtual updating to integrate the expressions.

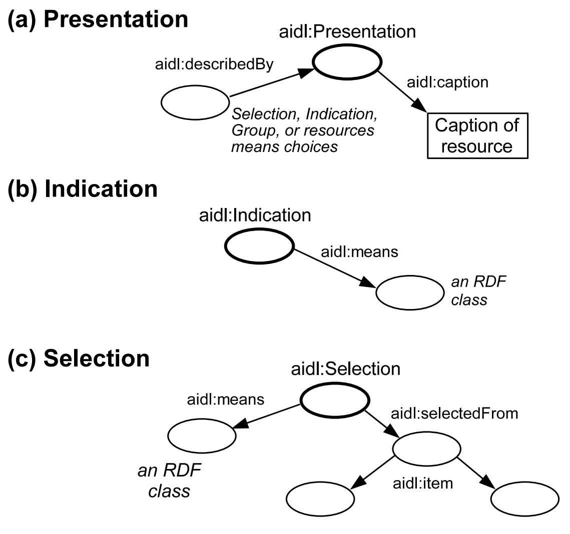


Fig. 4. Graphs representing presentation act, indication act, and selection act. They are written as RDF graphs.

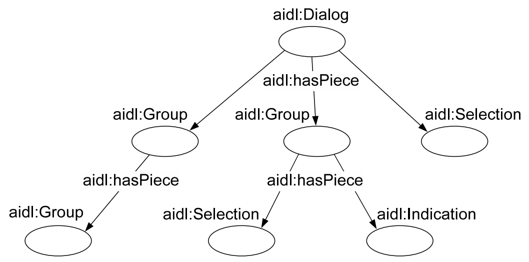


Fig. 5. Tree structure of interaction graph. The resource of aidl:Dialog means the root and the resources of aidl:Group mean the nodes, which are connected via aidl:hasPiece property.

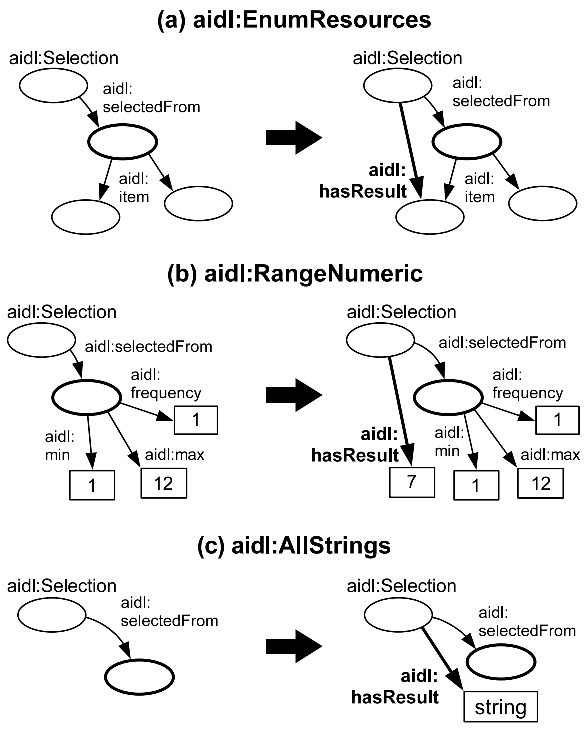


Fig. 6. Graph of selection act and user's response. The bold ovals express choices sets. The lefts and the rights are the states before and after user's operation.

C. Limitation of Presentation Acts

Interaction graphs have to be independent from all devices and styles, and cannot contain device/style-specific media information like graphics, animations, and sounds. The graphs use strings basically for their presentations. However, for rich expression power without the dependence, AIDL enables to link to some media with other RDF vocabularies. There are many RDF vocabularies proposed. For example, it is possible to assign display-sensitive information like graphics with one of FOAF [9] properties. The `foaf:depiction` property connects the `aidl:Presentation` to a resource expresses media. Interface clients use those resources for presentation

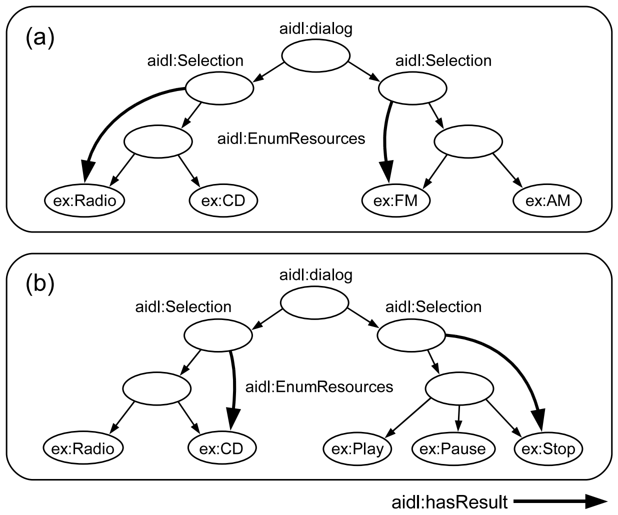


Fig. 7. Diagram of state transition. The graph is replaced depending on which of ex:Radio or ex:CD a user selects, and it expresses the corresponding operation.

acts if they support these vocabularies, or take no account. Hence, server developers cannot force clients to use these media assigned in the graphs.

D. Description of State

An interaction graph represents the current state of a user and a logic server consistently. A graph has one dialog, and an interaction of service consists of one or more dialogs. Thus, in complex services that need multiple dialogs, switching some graphs of dialogs means state transitions (Fig. 7). The servers, instead of AIDL self, handle such state transitions by switching graphs externally.

VI. ICLS Architecture

The Interface-Client/Logic-Server (ICLS) architecture, we propose, consists of interface clients that users prepare and logic servers that service developers set up, and both of them work together. The clients handle characteristic processes of specific interfaces in all services, while the servers handle processes common to all interfaces and logic processes of each service. As sharing one interaction graph, they work together independently. Actually, they have a graph respectively, modify it, and synchronize each structure through exchanging *change logs*. This synchronization process on transmission of logs is executed according to the *graph structure synchronize protocol*. The independence of the clients and the servers brings a commutative property of interfaces of the services.

In clients, the *interface components* interpret graphs and generate real interfaces (Fig. 8). A client coordinates components taken from itself or networks, to consist of an interface. Components exist corresponding to each selection act and indication act in a graph. There are various kinds of components handling choice sets and meanings of selection acts. Hence, clients can compose variety of interfaces using respectively available components.

A. Division of Interface Process

Interface processes of the services are divided into characteristic processes on specific interface and common processes on all interfaces. The first is unique elements to each interface that not be appeared in interactions. In GUI, for example, there are many unique elements of GUI. They process of displaying widgets like windows, buttons, edit boxes, and check boxes, and they accept users' operations for themselves. In contrast, the second, processes common to all interfaces are the elements involving interactions. Event processing and state transitions of interactions are common despite their diverse appearances.

ICLS does not separate all interface processes from the services, but divides them. Interface clients handle only the characteristic processes of specific interfaces, while logic servers handle only the common processes of all interfaces, for example, state transitions of interactions and feedbacks according to users' operations depending on logics. We did not separate interface completely because the processes are sure to contain service-characteristic parts. ICLS has to avoid that clients depend on specific services through delegating that service characteristic processes to servers.

B. Cooperation Process

In the beginning to use a service based on ICLS, users have to connect interface clients with logic servers, and then, clients and servers communicate with each other by the graph structure synchronize protocol. With a typical scenario, we explain the flow of cooperation from interface composition of a client to operations of a user, and server's responses reflected the operations (Fig. 9).

1) When a user connects a client to a server, the client and the server share an interaction graph prepared by the developer of the server.
2) The client gets information from itself or networks about some selection acts in the shared graph.
3) The client gets corresponding components based on the information from itself or networks.
4) The components initialize themselves using the given sub graph, and they generate and display interfaces, which the user operates.
5) The user starts operations on the interfaces displayed, and the components accept that operations.
6) The components update the shared graph to reflect the results of the user's operations, and the server offers the correspondent service.

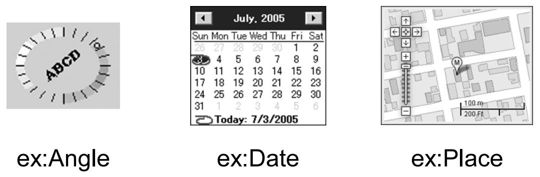


Fig. 8. Examples of interface component

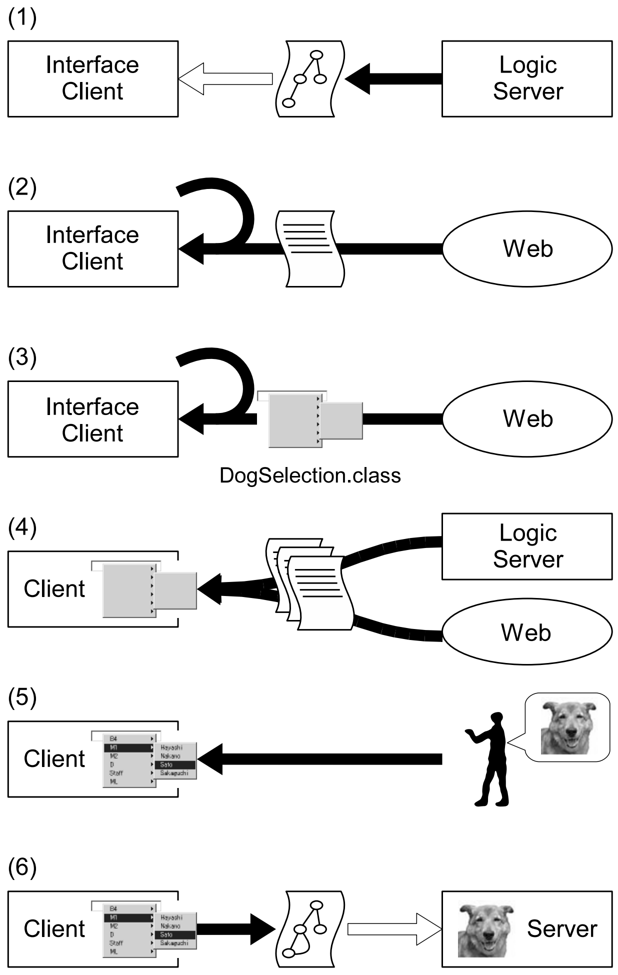


Fig. 9. Cooperation of client and server

VII. GRAPH STRUCTURE SYNCHRONIZE PROTOCOL

We developed a communication protocol, the graph structure synchronize protocol. It is a protocol to maintain the structures of two RDF graphs in an interface client and a logic server by exchanging change logs of the graphs. The client and the server possess a graph and a change log respectively, and reserve their modification for the graphs as adding or removing of RDF statements. After that, they transmit change logs each other at any timings, and apply the logs to update the graphs. This iteration of reservation and application of the logs makes the graphs equal at all times. The flow of transmissions is as follows (Fig. 10):

1) A client connects with a server and receives a graph that the server has.
2) The client and the server change the own graphs in parallel and reserve their logs.
3) The client sends its log to the server, and the server applies it to the own graph. If the server executed a state transition by exchanging the graphs, return to 1).
4) The server sends its log to the client, and the client applies it to the own graph. Return to 2).

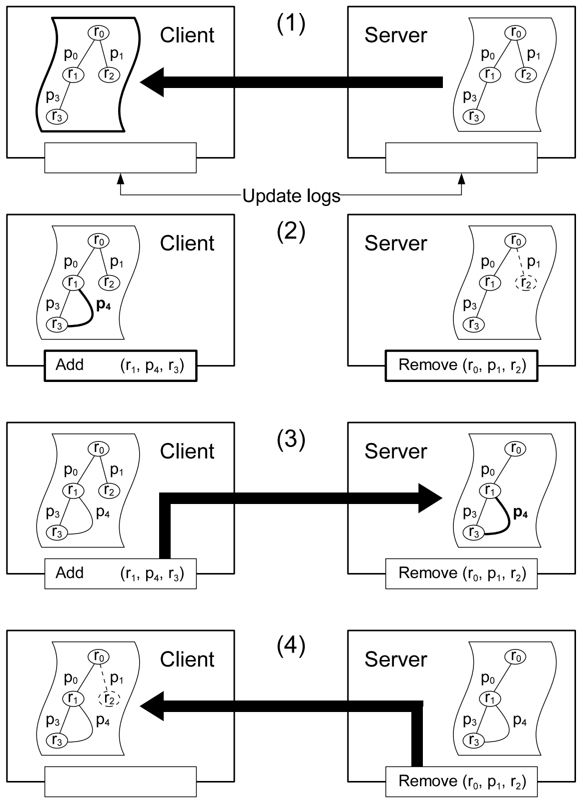


Fig. 10. Synchronization process by Graph Structure Synchronize Protocol. The left side indicates clients, and the right side indicates servers.

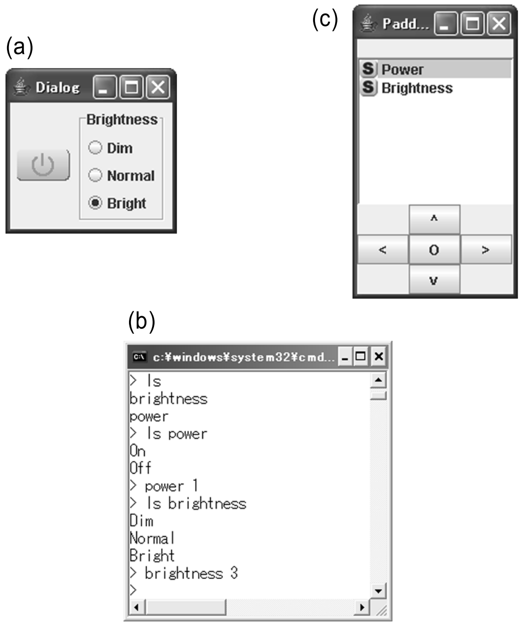
There are two types of the updates, the one is by exchanging change logs and the other is that a server sends a graph responding to sending log of a client. A server sends a whole graph when a client connects at the beginning, or the graph in server is modified drastically. In addition, because they communicate each other with difference information by the logs, the amount of the transmission can be reduced than sending the graph in whole.

VIII. IMPLEMENTATION

We propose a framework of ICLS to make developments of services and interfaces, and developed three interface clients and two logic servers by using it. This framework is class library written in Java language. We designed its structure to emulate existing GUI libraries. Hence, the developers can use it in the same way as those libraries. In the framework, TCP/IP protocol using socket was adopted as the default implementation.

A. Generation of Graph and Event Handling

Using the framework, developers can generate interaction graphs like creating dialog boxes with some GUI toolkits. In this framework, a `Graph` object shows an interaction graph. First, the object is generated, and then an `ElementFactory` object that the `Graph` object has generates some objects such as `Dialog` and `Selection`. After configurations of these generated objects, they are added to `Dialog` and `Group` objects by the `add` methods. As a result, RDF resources that



```
> ls
brightness
power
> ls power
On
Off
> power 1
> ls brightness
Dim
Normal
Bright
> brightness 3
>
```

Fig. 11. Screen shots of Interface Clients

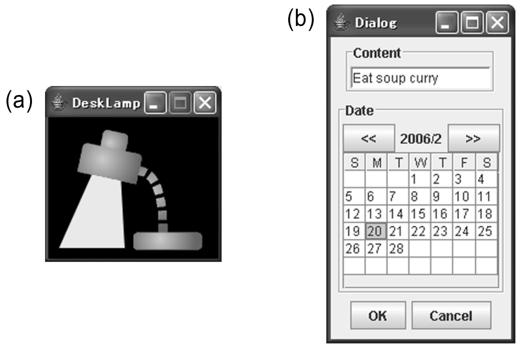


Fig. 12. Screen shots of client that connect desk lamp and scheduler services

`Graph` maintains and corresponding to elements of the graph is generated, and a necessary graph is completed.

The event processing mechanism is similar to GUI. Three kinds of event listener *interfaces*: `IndicationListener`, `SelectionListener`, and `SessionListener` are used for various event processing. As assigning each implementation of the *interfaces* in `Indication`, `Selection`, and `Session`, appropriate methods are called when events are generated.

B. Development of Clients and Servers

We developed the interface clients of a GUI version, a CLI version, and a simulator of a portable terminal version. The GUI client constructs a dialog box from an interaction graph (Fig. 11-a). We created two components corresponding to `ex:PowerState` and `ex:Date` as examples. The CLI client shows a graph structure as a directory structure of file system, and a user inputs commands to operate it (Fig. 11-b). The portable terminal type client has a small screen and 5 buttons, and operated like music player (Fig. 11-c).

The logic servers of desk lamp simulator and scheduler service were also developed. Users can use these servers through the clients. The desk lamp simulator is a remote control service for a simulated desk lamp. We developed it as one example of service operating appliances in home or from remote places (Fig. 11, 12-a). The scheduler service is another example of applications that run on general PCs and PDAs. When users connect clients with this server, they are shown a list of schedules, and can correct or remove them or add new one (Fig. 12-b).

IX. CONCLUSIONS AND FUTURE WORK

In this paper, to improve the current condition about interfaces of service, we have proposed the architecture of interface-client/logic-server (ICLS). It enables users to choose preferred interfaces depending on their situation or individual difference. Moreover, we developed the Abstract Interaction Description Language (AIDL) using RDF to describe interactions, and proposed an interface composition technique based on AIDL. In addition, by the several implementations, we have illustrated that ICLS can become a way of improvement of services. There are the three points of this work as follows:

- The abstraction of interactions focused on selection acts.
- The description technique of interactions that identify specifications of interfaces with states of interactions.
- The application of semantic web technique to interface.

ICLS improves the present circumstances that most of the services offer only fixed and predefined UIs, which are mostly GUIs with respective design concepts.

Future work involves an improvement of ICLS and provision of development tools. It is important that studies of an automatic customizing technique from components and users' characteristics. We will offer some visual development tool that can describe interaction graphs in the sight. Finally, we will achieve the interfaces that understand more meanings of interactions on them.

References

[1] Harmonia, "Tutorial booklet december," 1997.
[2] H. Okada and T. Asahi, "PC remote controller based on user interface transformation," *The Transactions of Human Interface Society*, vol. 4, no. 4, pp. 235–244, 2002.
[3] S. Nylander and M. Bylund, "The ubiquitous interactor–universal access to mobile services," in *HCII 2003*, 2003.
[4] S. Nylander, M. Bylund, and A. Waern, "The ubiquitous interactor–device independent access to mobile services," in *CADUI'2004*, 2004, pp. 274–287.
[5] J. Nichols, B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, and M. Pignol, "Generating remote control interfaces for complex appliances," in *UIST 2002*, 2002, pp. 161–170.
[6] J. Nichols, B. A. Myers, and K. Litwack, "Improving automatic interface generation with smart templates," in *IUI 04*, 2004, pp. 286–288.
[7] E. Miller, R. Swick, and D. Brickley, "Resource Description Framework (RDF)," available at `http://www.w3.org/RDF/`.
[8] E. Miller, R. Swick, D. Brickley, B. McBride, J. Hendler, G. Schreiber, D. Wood, and D. Connolly, "W3C Semantic Web," 2001, available at `http://www.w3.org/2001/sw/`.
[9] FOAF, "The Friend of a Friend (FOAF) project," available at `http://www.foaf-project.org/`.