# Architecture for Migratory Adaptive User Interfaces

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

## Abstract

*We propose a new solution named interface client/logic server (ICLS), targeting dialog-based interactive services, supporting user interface (UI) migration, and offering adaptive UIs for devices and services. Constant improvements of technology have brought a large variety of platforms, and that has made users' new demands about the services. The first is that the users would like to use services through different devices and modalities depending on their use contexts. The second is that the users would sometimes like to change devices and take their tasks from one to another, which is called UI migration. Our architecture ICLS is designed based on client/server model. In ICLS, we use XML documents written in abstract interaction description language (AIDL) as logical descriptions of UIs, and introduce one of the semantic web technologies adding the function of expressing meanings of interactions.*

## 1. Introduction

Constant improvements of technology have brought a large variety of platforms (such as mobile phones, PDAs, and music players including desktop PCs) used for interactive services, and that has made users' new demands about the services. The first is that the users would like to use services through different devices and modalities depending on their use contexts. Depending on whether users are in their homes or cars, the devices which they would like to use for checking their schedules must be different. The second is that the users would sometimes like to change devices and take their tasks from one device to another, which is called user interface (UI) migration [3]. When the users change their devices, they wouldn't like to retrace the same process of clicking the same buttons and entering the same contents.

Conventional ways of associating devices and services does not meet the users' demands because of costs and inadequate separation between services and devices. Simply

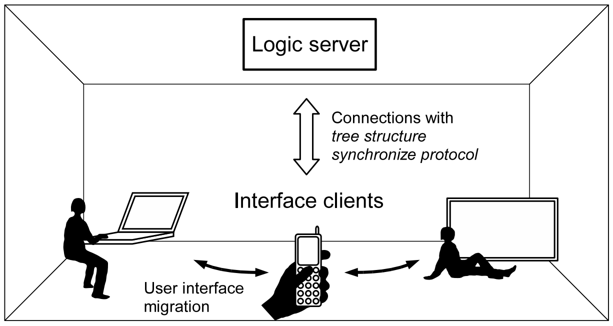


**Figure 1. Interface client/logic server (ICLS) architecture.**

developing multiple versions for each platform respectively is expensive for the developers in terms of time, money, and maintaining the versions consistent. Hence, in spite of the diversity of the devices and platforms, users cannot utilize them effectively for accessing the services. Recently web browsers were implemented in many mobile devices, and the advent of web applications on them seemed to be a solution. They need, however, specific versions for each platform, and they are often accessed selectively with different URI, because HTML and other web architectures still assume their target platform.

In this paper, we propose a new solution named *interface client/logic server* (ICLS) (Figure 1), targeting dialog-based interactive services, supporting UI migration, and offering adaptive UIs for devices and services [17, 18]. Our architecture aims at services like web applications, in which some input facilities (such as buttons, check boxes, and text fields) are used on some dialogs or pages updated as state transitions. It is a kind of model-based UI architecture proposed in a large amount of related work, and this architecture uses some logical descriptions of user tasks, interfaces, or interactions that the architecture supports. These logical descriptions are highly abstracted from specific devices and platforms, and realize separated and independent developments of devices and services. ICLS allows service developers to declare concrete semantics of interactions on services in logical descriptions with a machine-readable way, and realizes the richness of generated UIs on devices. For example, by specifying the meanings such as *date* or *power state* in a description, appropriate widgets like a calendar or a power switch can be generated if they are supported by the device. ICLS has a mechanism for attaching devices to existing session for UI migration, and when the users do not disconnect the previous device after attaching new device, they can utilize the two devices simultaneously. It is useful when we are in multi-device environments like homes or offices and the devices have different characteristics.

In this paper, we first discuss our architecture, next, we show some implementations, discuss related work, and lastly we conclude our work.

## 2. The interface client/logic server

Our architecture ICLS is designed based on client/server model (Figure 2). The term *interface clients* stands for various devices and platforms in which client applications are implemented, and the term *logic servers* stands for various services in which server applications are implemented. Both applications of interface clients and logic servers are assumed to conform to the specifications of ICLS, which is mentioned later. Once devices or services are developed in accordance with the specification of ICLS, no revisions are required when new service or device is introduced.

### 2.1. Clients and servers

Figure 3 shows an example of the flow of a session between a client and a server for a *reminder service*, which manages users' scheduled tasks. The numbers with parentheses in the figure correspond to the numbers of the following scenario: A session of a service starts by a request from a client to a server (1). After the session starts, the client receives a logical description from the server (2).The client constructs DOM tree, and generate a UI based on the tree (3). Logical descriptions are written in the language, *abstract interaction description language* (AIDL), which

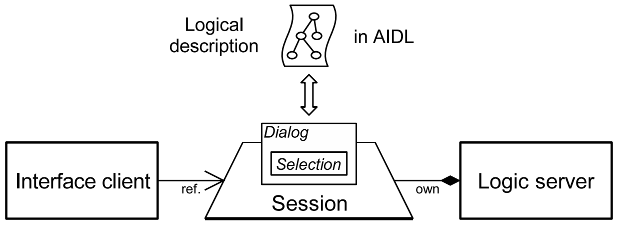


Figure 2. Concept of ICLS.

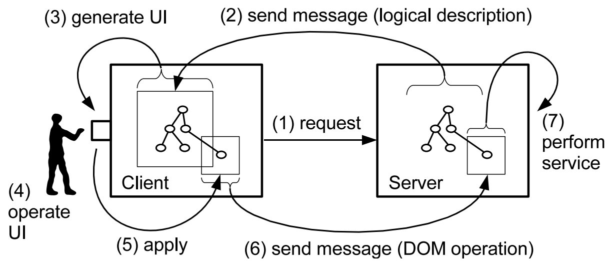


**Figure 3. Flow of communication process between a interface client and a logic server.**

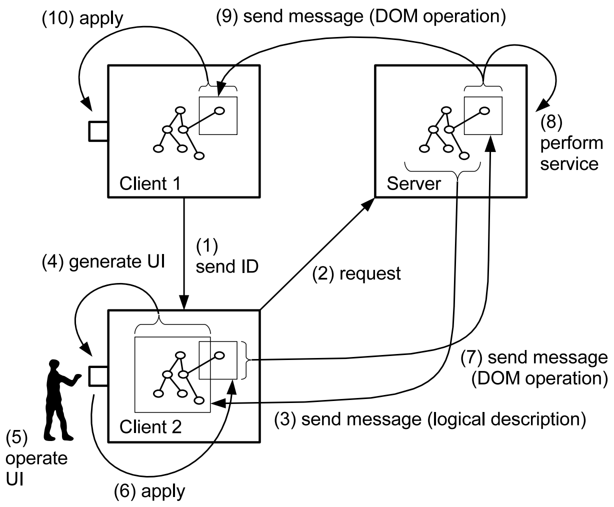


**Figure 4. Flow of migration process from the client 1 to the client 2.**

is an XML application we have developed. After the UI was generated on the client, the user can check his scheduled task list with the device like a portable music player which offers a small screen and a wheel control (4), on the way to his office on a train. At this time, the user notices that he has to add a new task to the list, and he operates the small wheel control on the player to do it. The client changes the DOM tree corresponding to the user's UI operations (5). The client sends this DOM changes as messages to the server (6). The server performs the service according to the message from the client (7).

During a user is accessing a service through one client with a session, the user can also try to access the same service through another client with the existing session. Here, the DOM trees of these clients are synchronized and updated simultaneously, and 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. Using this ID, the user can access the same session. Figure 4 shows an example of the flow of a migration process between two clients: client 1, and client 2. A new client (client 2) obtains a session ID from the existing client (client 1) (1). A simultaneous session of a service starts by a request with an ID from a client to a server (2). After that, the client receives a logical description used in the session from the server (3). The new client executes the same UI generation process and sends a user's operation to the server (4, 5, 6, 7, 8). After performing the service, the server broadcasts the received message to another client (9). The client 1 receives the message, and applies it to the DOM tree and UI of the client (10). In the example of a reminder service, the user can set the limit time for a task item easily with the wheel control of the player, but he might feel it's a pain to fill the content field of the item with the same control. Accessing the server with the same session through a PC, the user can continue to type in the content of the item, without restart of the service or reentering the time data.

It is possible for clients to attaching existing sessions, because servers and clients on session maintain the same structured DOM tree. The DOM trees constructed with logical descriptions contain not only UI structures but also current states of generated UIs. Receiving a DOM tree which is used by another client means receiving the whole information about UIs in a session.

### 2.2. Protocol

Interface clients and logic servers communicate with a protocol, *tree structure synchronize protocol* (TSSP), where changes of DOM trees are serialized as messages. Clients and a server connected in a session have the same DOM trees extracted from AIDL documents respectively. The protocol provides the way to send whole or partial trees as documents written in AIDL in order to synchronize two or more DOM trees in each client and server. This synchronization realizes the virtual sharing of one DOM tree, which is changed by the clients and the server. Note that we do not specify the low protocol under TSSP, and many existing protocols which can send text messages are available.

Messages exchanged between a client and a server are categorized into two types: serialized change operations with one of three commands and an XML path, and just sending a whole AIDL documents. Changes on a DOM tree corresponding to a user's UI operations are sent to a server as messages which consist of some tuples, whose elements are one of three commands (`INSERT`, `ERACE`, or `REPLACE`), an XML path (such as "`/group[0]/selection[1]`"), and a piece of AIDL document. The server received the messages applies them to its DOM tree, and broadcast them to the other clients if it has multiple connections. A server expresses the state transitions of UIs by sending whole of new descriptions in AIDL to a client.

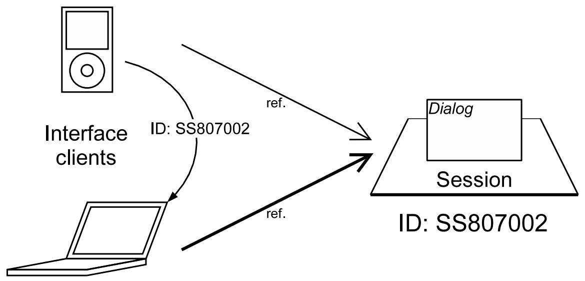


**Figure 5. Sharing of session using with session ID.**

Attaching an existing session for simultaneous or migratory interfaces is performed by passing session IDs given a certain way when a client is connected to a server (Figure 5). When a client is connected with an ID, the server sends the DOM tree of session identified with the ID as a document written in AIDL to the client. Clients also have to communicate each other for obtaining this ID from another client for the connections with the session IDs.

When multiple clients exist in the same session, the priority of their messages retains the consistency of their DOM trees. When a server receives messages which have some conflicts among clients or between the server and the clients, the state in the server takes priority of them in the clients, and the message arriving earliest takes priority of other messages.

## 3. Logical description language

In ICLS, we use XML documents written in AIDL as logical descriptions of UIs, and the documents describe interaction structures, presentations, and task models in services (Figure 6). AIDL is an application of XML, and supports the clear separation of interface clients and logic servers on ICLS, because it is based on web standards XML and has no device- or modality-specific contents. We have defined AIDL with a RELAX NG [6] schema (and we are preparing to release it to the public).

We introduce one of semantic web technologies, resource description framework (RDF) [10] for adding a function of expressing meanings of interactions on certain services in logical descriptions. The easiest way for addressing multi-devices and multi-platforms is limiting, abstracting, and aggregating functionalities of devices, as is pointed out in some previous researches, which I will mention in Section 6. We also adopts the same approach; however, with the expression of interaction meanings in services, our system has an advantage of offering the possibility for combining service specific functionalities and device specific ones. In ICLS, RDF classes are exploited for the meaning

```
<aidl:pane>
   <aidl:dialog>
      <aidl:selection aidl:meaning="http://.../LampPowerState">
         <aidl:description aidl:caption="Power" />
         <aidl:state>http://www.example.com/On</aidl:state>
         <aidl:resources>
            <aidl:choice aidl:uri="http://www.example.com/Off">
               <aidl:description aidl:caption="Off" />
            </aidl:choice>
            <aidl:choice aidl:uri="http://www.example.com/On">
               <aidl:description aidl:caption="On" />
            </aidl:choice>
         </aidl:resources>
      </aidl:selection>
   </aidl:dialog>
   <aidl:knowledge>
      <rdf:RDF>
         <rdf:Description rdf:about="http://.../LampPowerState">
            <rdfs:subClassOf rdf:resource="http://.../PowerState"/>
         </rdf:Description>
      </rdf:RDF>
   </aidl:knowledge>
</aidl:pane>
```

**Figure 6. Simple example of description in AIDL.**

expressions, and the hierarchies of these classes are utilized for the inference of meanings. RDF is a standardized web technology, independent of specific platforms and venders, and designed for addressing *opened information* (meaning information not expected in advance). ICLS inherits such characteristics of RDF.

In AIDL, arbitrary UI structures and their current state (in other words, the history of users' operations) are described as *selection acts*, which represent essential function of UI elements in common among various devices, platforms, and modalities. A selection act is a tuple which consists of three elements: a *type* (a set of choices), a *meaning* (a purpose in a service), and a *state* (a current state). For instance, the operation of the power state of a desk lamp is represented as a selection act composed of the set of two choices (`ON` and `OFF`), current state `ON`, and its meaning `LampPowerState`. Selection acts are grouped with other selection acts and groups, and compose an *interaction tree*. A description of AIDL can be seen as a tree where selection acts, choices, current states, and groups are expressed as XML nodes. 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 for logic servers, all nodes are mutable.

Interface clients can infer meanings based on the hierarchy of RDF classes applied to the general-specific relationship of the meanings, and they can address more meanings than ones actually implemented. Class hierarchy can be constructed with *merged* RDF graphs which clients obtain from servers and the web. RDF has the namespace

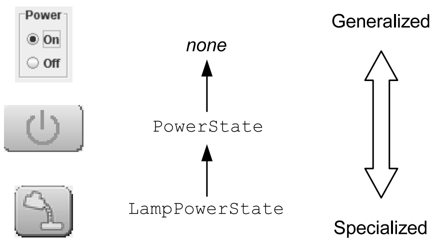


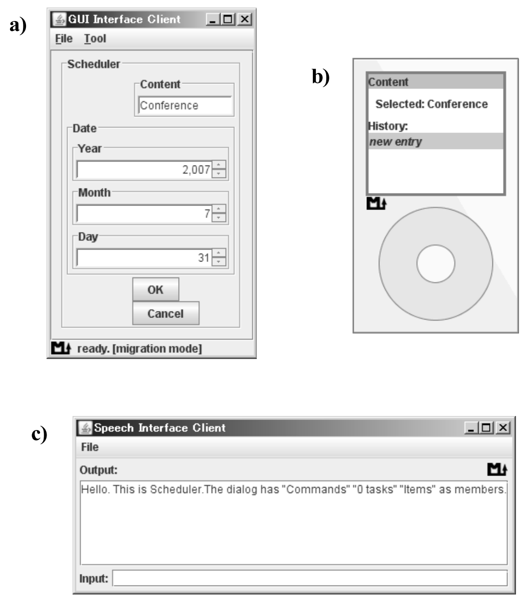
**Figure 7. Example of meaning hierarchy and corresponding UIs.**

mechanism on the web, and those graphs can be merged ensuring their consistency. Meanings in AIDL documents are exploited to specify the purposes of selection acts and to relate the selection acts to client-specific UI elements. In other words, interface clients pick out appropriate UI elements using the meanings. RDF describes knowledge about UIs in a machine-readable way.

## 4. Implementations

We implemented ICLS as a framework, which is a class library written in Java language (JDK 1.6) with a semantic web library Jena [8]. Although various protocols for sending text messages are available as low protocols under the ICLS protocol, we adopted TCP/IP as the default. We used Java and Jena, but it does not mean any dependency since the technologies we adopted are standardized well. In addition there exist some cell phones which can run Java programs in these days, and thus it can be possible to port our library to other platforms.

In order to verify the feasibility of the ICLS specification and the stability of the communication protocol, we developed three interface clients and two logic servers (Figure 8). Three clients are a GUI client (a), a simulator of mobile device (b), and a simulator of voice UI (c). Two servers are a reminder service and a remote controller of a virtual appliance. Although these clients and servers are simple and small-scale ones, they include the essence of more general and typical applications. We are also developing a prototype resolving layout problem from logic description in terms of GUI. The GUI client generates GUI dialog boxes using Swing as a GUI toolkit, according to the AIDL documents received from servers (Figure 8-a). We implemented meanings `ex:LampPowerState`, `ex:PowerState`, and `ex:Date` into the client. Users can customize whether the client reflects these meanings or not with a setting dialog box. The mobile client has a small screen and a wheel control, and offers hierarchical menu UIs (Figure 8-b). We implemented a meaning `ex:PowerState`, so that the client can offers its wheel as a metaphor of a power



Hello. This is Scheduler.The dialog has "Commands" "0 tasks" "Items" as members.

**Figure 8. Screenshots of the interface clients: a GUI, a mobile device, and a voice UI.**

switch. The client of voice UI simulator expresses voice communication with text strings (Figure 8-c). Users can utilize connected services through command inputs with keyboard and outputs of text.

## 5. Discussion

In this paper, we are placing emphasis on the diversity of devices and platforms, and thus, we designed the architecture in which the servers handle all functionalities except for about specific UIs, and we focused on the aspect of input. To address many kinds of devices and platforms in a standardized architecture, we must consider both input and output of contents, but we addressed the input aspect mainly as a first step. Since it is difficult to separate the whole UI processes from service codes, we designed interface clients to handle only device- and modality-specific UI processes leaving common UI processes in servers We intend to engage clients in utilizing their resources for the generation of various UIs, and we did not add any scripting functionality to the architecture for retaining the design simplicity.

The infinite meanings can be defined freely as RDF classes by service (logic server) developers, and they pose a problem for how client developers decide which meanings they implement. It is a common problem in existing contributions including [12], because we are addressing the real world problem. In our architecture, 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, our system can deal with meanings flexibly with RDF documents written by current and future developers of service and devices.

In the above implementations, we have adopted tentative rules of generating UIs from logical descriptions, but at least for the GUI client, we have ended up finding a certain solution. For generating GUI, the rule is reduced to the mapping between selection acts with meanings to appropriate widgets, and the layout problem of these widgets.

## 6. Related work

As a general solution, a broad range of research was proposed, and almost all of them employ the approach of model-based UI design [7, 16], which commonly utilize logical descriptions. There are some researches addressing the static generations of UIs on development times, dynamic generations for remote controlling with multiple devices, and ones handling migratory interfaces.

Web migratory interface system was proposed in [2], which targets arbitrary web applications and performs a reverse engineering of existing web pages in order to obtain their logical information. The previous version of the system [3] is based on model-based UI design tool TERESA [4], and handles web applications developed with that specific tool. This is a strong limitation, and thus, the improved solution was presented, in which their system handles existing web applications with a reverse engineering of web pages. This approach has a benefit for handling specific web applications, but it does not suit our purpose. For example, a *input button* of HTML elements means not only a *navigator* but also a *selection* in some web applications, and still more in general UIs.

Ubiquitous interactor (UBI) [13] addresses service specific domains with *customization forms* in logical descriptions for developing services without depending on devices, but it does not consider migration. It is similar to our solution in terms of that it can represent various UIs for services using interaction acts as the elements of interaction, but it is different in that UBI uses specialized customization forms for each device for controlling presentations on specific devices. Since the customization forms have no portability among different devices, developers have to customize their descriptions for each device.

Personal universal controller (PUC) [11, 12] was proposed for remote controlling various appliances with only PDAs, but there is no consideration of migration there. PUC uses smart templates for generating conventional presentations on some service domains, but the difficulty of defining the smart templates is not mentioned.

User interface markup language (UIML) [1] is an XML based language for developers to describe interfaces independently of specific platforms. In development period, developers have to define relationships between elements in UIML and other elements in existing language such as Java and HTML. Thus, there is no runtime support for automatic generation and migration of UIs.

As other related work, we can also mention Migratory applications [5], XWeb [14], Document-based framework [9], and ICrafter [15]. Migratory applications offer a programming model for migration with distributed scripting language Obliq, but this system is strongly dependent on the scripting language. The other work has some similarities in point of using XML and adopting model-based approach, but they do not address UI migration.

## 7. Conclusion and future work

We presented a new solution, interface client/logic server (ICLS) architecture, which has some limitations but offers simultaneous interfaces, and we showed the new application of outcome on the other research field. Simultaneous interfaces as the extension of migratory interfaces are new technique on UI fields we presented. Furthermore, we utilized one of semantic web technologies, RDF in order to express the meanings of interactions for describing service-specific interactions with the meanings, which is a new application of semantic web for the human computer interaction field.

As our future work, we have to remove some limitations imposed on our solution, to consider exploiting the outcomes of other work, and to evaluation it. The current implementation of GUI interface client uses some layout managers implemented in Swing, and has no intelligent mechanism for laying out widgets. Here, we are exploiting accomplishments in the fields of artificial intelligence, especially constraint satisfaction problems (CSPs), and prototyping some clients. We are also considering exploiting outcomes in the area of transcodings, which handles conversion of the presentation of output information. In other respects, because our architecture requires keeping client-server connections alive, we have to evaluate the response speed of UIs on networks. Development and evaluation of authoring tools for AIDL documents in ICLS services are needed for further research.

## References

[1] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster. UIML: An appliance-independent XML user interface language. In *The eighth international World Wide Web conference*, 1999.

[2] R. Bandelloni, G. Mori, and F. Paternò. Dynamic generation of migratory interfaces. In *Proceedings Mobile HCI 2005*, 2005.

[3] R. Bandelloni and F. Paternò. Flexible interface migration. In *IUI 04*, pages 148–155, 2004.

[4] S. Berti, F. Correani, G. Mori, F. Paternò, and C. Santoro. TERESA: A transformation-based environment for designing and developing multi-device interfaces. In *CHI 2004*, pages 793–794, 2004.

[5] K. A. Bharat and L. Cardelli. Migratory applications. In Jan Vitek and Christian Tschudin, editors, *UIST '95 (Mobile Object Systems: Towards the Programmable Internet)*, volume 1222, pages 131–148. Springer-Verlag: Heidelberg, Germany, 1995.

[6] J. Clark, M. Murata, and OASIS. RELAX NG, 2001. Available at `http://www.relaxng.org/`.

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

[8] Hewlett-Packard Development Company, L.P. Jena. Available at `http://jena.sourceforge.net/`.

[9] T. D. Hodes and R. H. Katz. A document-based framework for internet application control. In *USENIX Symposium on Internet Technologies and Systems*, 1999.

[10] E. Miller, R. Swick, and D. Brickley. Resource description framework (RDF), 2004. Available at `http://www.w3.org/RDF/`.

[11] 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*, pages 161–170, 2002.

[12] J. Nichols, B. A. Myers, and K. Litwack. Improving automatic interface generation with smart templates. In *IUI 04*, pages 286–288, 2004.

[13] S. Nylander, M. Bylund, and A. Waern. The ubiquitous interactor–device independent access to mobile services. In *CADUI'2004*, pages 274–287, 2004.

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

[15] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. ICrafter: A service framework for ubiquitous computing environments. *Lecture Notes in Computer Science*, 2201:56–74, 2001.

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

[17] T. Yanagida and H. Nonaka. Interface migration using abstract interaction description. In *Technical Report of the Institute of Electronics, Information and Communication Engineers SS2007-24*, volume 107, pages 49–52, 2007. (in Japanese).

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