# Interaction Description with Service-Specific Meanings

Takuto YANAGIDA and Hidetoshi NONAKA
Hokkaido University, Japan

# 1. INTRODUCTION

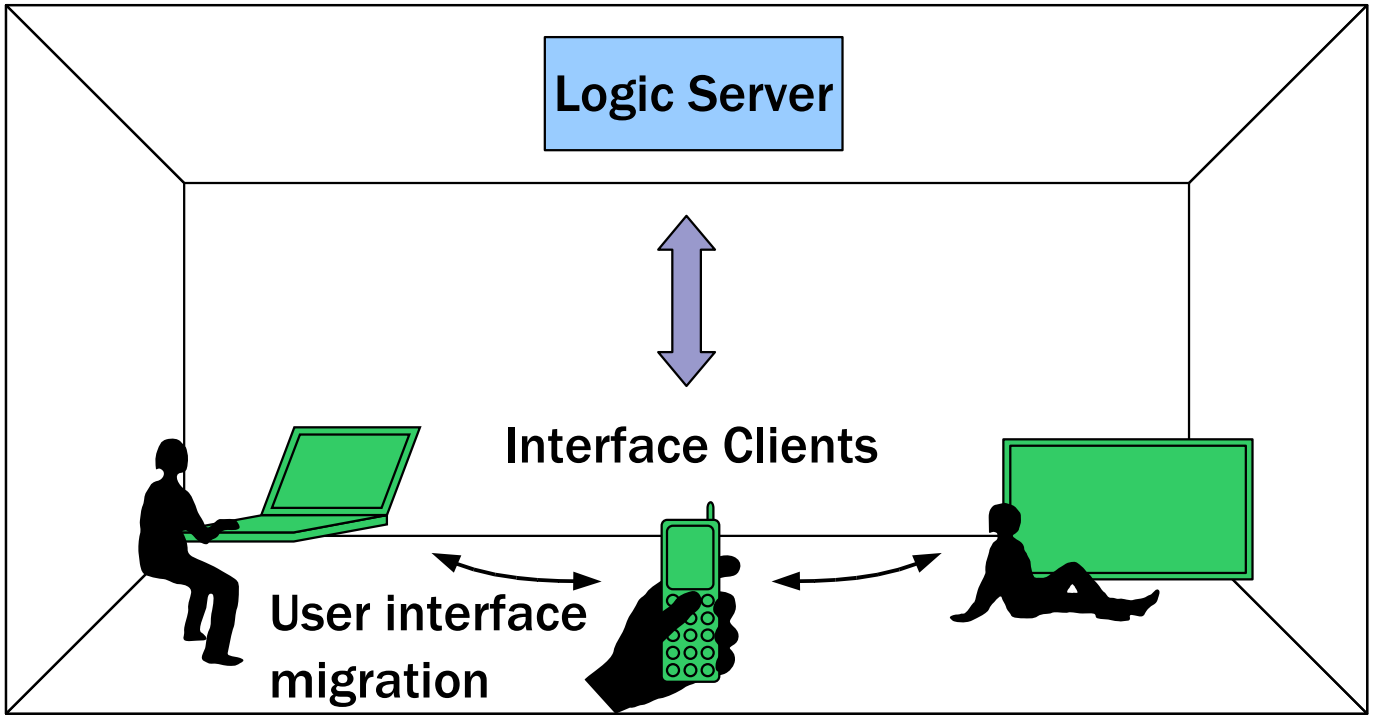
# Background

- Computer-based services (appliances and applications operated with user interface (UI) devices) have pervaded our everyday lives.
- The diversity of the users and the devices has increased the users' demand to be able to use services through different devices and modalities in accordance with certain contexts.

# Architecture

- Our approach, interface client/logic server (ICLS) is a model-based UI architecture, which supports UI separation, generation, and migration comprehensively [15].

[15] 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.




- Users of ICLS-based services possess interface clients, and connect them to logic servers when they use the services provided by the servers.

# Description of interaction

- We developed an XML-based language, abstract interaction description language (AIDL) for describing interactions and their meanings.
- An interface client
  - generates UIs from AIDL documents from a server,
  - shows the UI to its user,
  - applies the users' operation to the documents, and
  - sends messages about it to the server.

# Design principles

- We adopted the following three principles in developing ICLS:
  1. We focus on the aspect of input of interactions rather than output, and place emphasis on realizing the diversity of the interface clients.
  2. We did not add any scripting function to AIDL in order to retain its specifications simplicity.
  3. Since it is difficult to separate the whole UI process from service codes, we have separated only device- or modality-specific processes as the interface clients.

# 2. DESCRIPTION LANGUAGE

# AIDL

- Abstract interaction description language (AIDL) is used for describing interactions as logical description, instead of physical descriptions containing UI elements literally.
- It supports the explicit separation of clients and servers as it is based on a web standard XML and has no device- or modality-specific contents.

```
<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 2

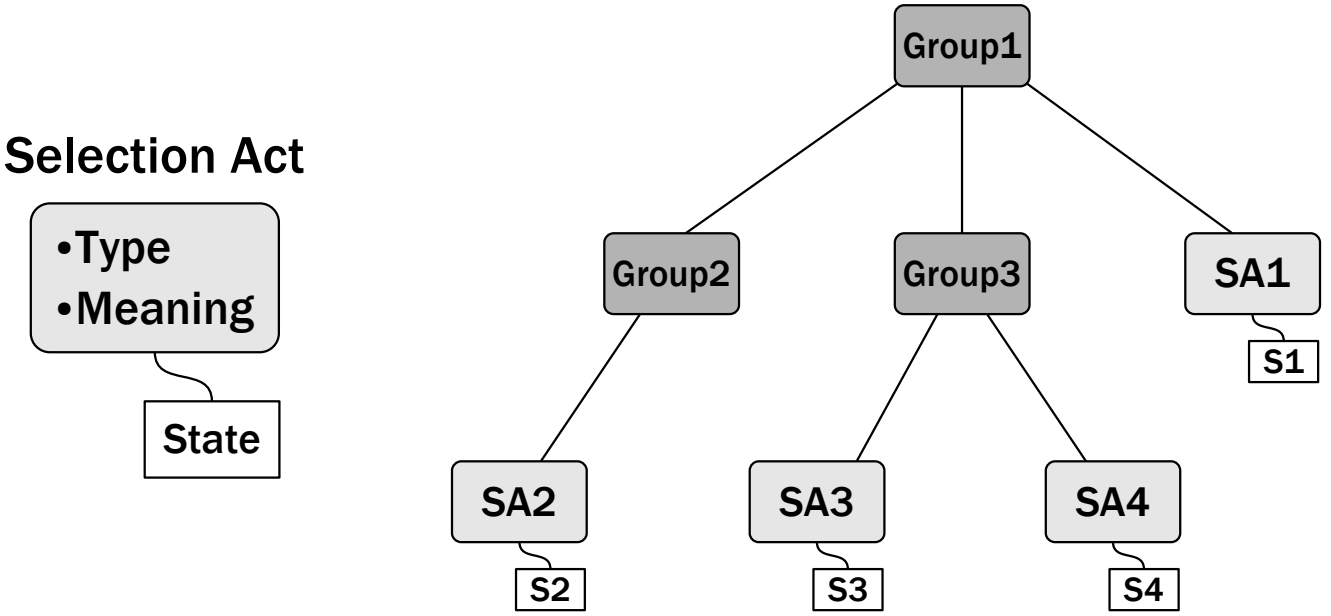
# Use of semantic web

- We developed AIDL for supporting meanings of service specific interactions on ICLS.
- Meanings enable ICLS to combine service specific functions and device- and modality-specific UI elements.
- For expressing meanings, we adopted one of the semantic web technologies, resource description framework (RDF) [7].

[7] E. Miller, R. Swick, and D. Brickley. Resource description framework (RDF), 2004.

# 2.1. Interaction Model

- UI structures and their current states described in AIDL are abstracted as **selection acts**, which represent essential function of UI elements.
- A selection act consists of three elements:
  - a **type** (a set of choices),
  - a **meaning** (a purpose of a selection in a service), and
  - a **state** (a current state of selection).




- Selection acts are grouped with other selection acts and groups, and make an interaction tree.
  - In AIDL documents, selection acts and groups are expressed as XML nodes, and
  - the nodes are added or removed by clients and servers.

# Type

- **A type is one of basic types (`Resources`, `Numerals`, and `Strings`) representing a set of choices.**
- **For expressing subtypes, one of constraints:**
  - ***enumeration*, which enumerates all used choices,**
  - ***range*, which specifies maximum and minimum values**

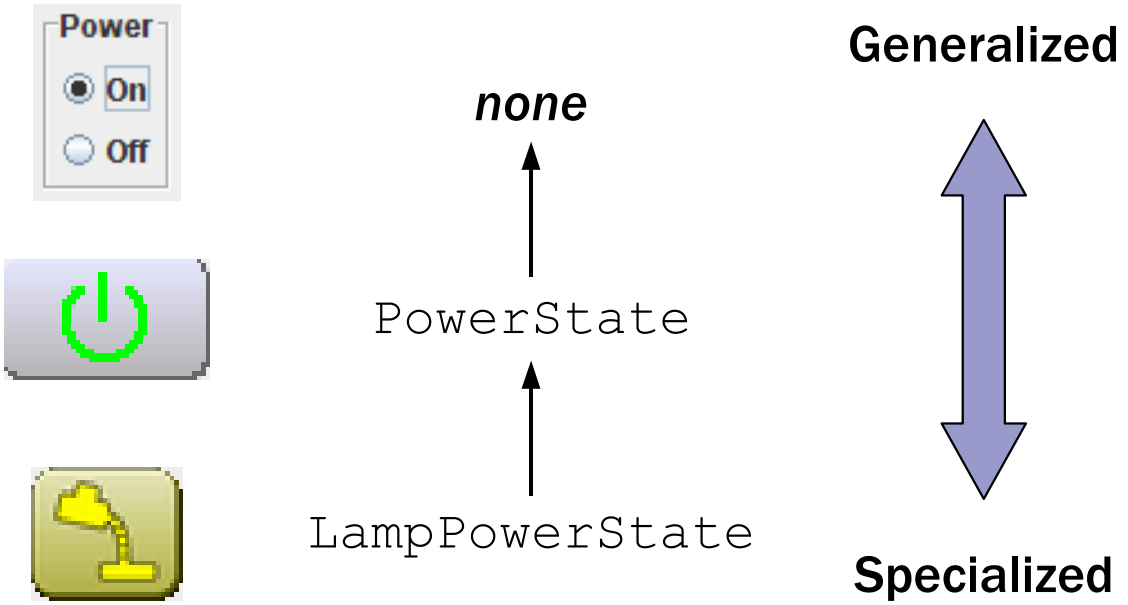
  **can be added to a type.**

# State

- A state is one of choices contained in a type, and it represents which choice is selected.
- A selection act has its state as a result of users' operation or a current state of a logic server.
- The tree represents its own current state with the states of its selection acts.

# Meaning

- A meaning is represented as a URI of an RDF class, and help clients to generate UIs which offer appropriate affordances for users.
- Meanings are useful in order to increase the concreteness of selection acts, or interactions in services.




- Meanings compose their general-specific relationship as a hierarchical structure of RDF classes corresponding to the meanings.

# Spread of meanings

- The more ICLS-based services are developed, the more meanings are needed; therefore, it is unrealistic to define them all beforehand.
- In ICLS, arbitrary developers can define meanings as RDF classes with relations with existing meanings.
- We are considering that the consensus of the interpretation of meanings will be built according to the spread of AIDL.

# 2.2. Interpretation of Meanings

- Interface clients can infer meanings from the hierarchy of RDF classes, which offers the general-specific relationship of meanings.
- Clients can download and merge external RDF documents containing class trees, because RDF has the namespace mechanism for the use on the web.

1. The client searches itself for implementations corresponding to the given meaning.
2.
   - When the client finds some implementations, it uses one of them and ends this interpretation process.
   - 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 some class hierarchy tree.

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 uses one of them and ends this process.
   - When the client cannot find any implemented meaning, it renders the most general UI without meanings.

# Implementation of meaning

- Meanings are just labels for the concepts shared by developers, and client developers need to determine which meanings they implement.
- With the inference mechanism, clients can handle meanings that have not been actually implemented.

# Example

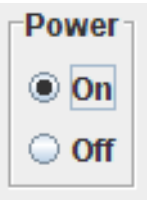
- **In the example (fig. 2), there exists a selection act representing a power state of a desk lamp (`LampPowerState`).**
  - **The document contains a RDF document describing *`LampPowerState` is sub class of `PowerState`*.**

```
<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>
```

**Fig. 2**

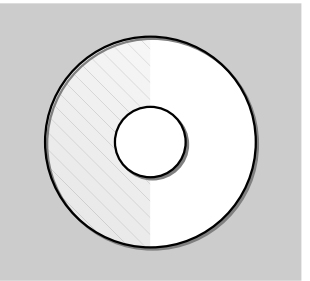
- From this description,

  a) A GUI-based interface client can generate two radio buttons without the meaning.




  b) It can also generate a custom-designed button with the meaning if it has the implementation.

  c) A portable client like a music player can use its hardware switch as the representation of a meaning `PowerState` inferred from the given meaning.



# 3. IMPLEMENTATION

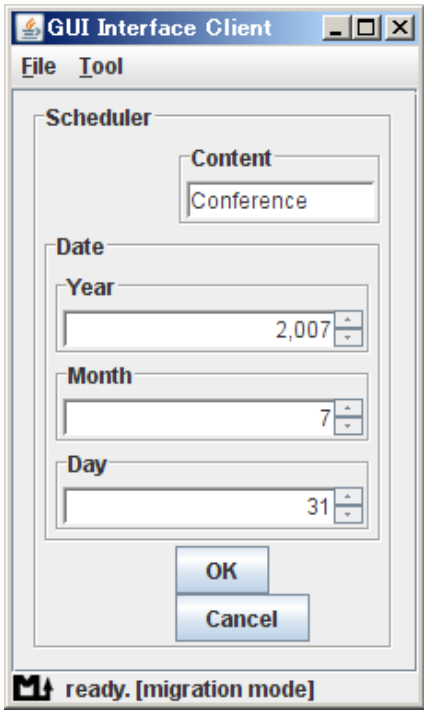
# Framework

- We implemented ICLS as a framework, which is a class library written in Java (JDK 1.6) with a semantic web library Jena [5].
- Although various protocols for sending text are available as a low protocol, we adopted TCP/IP as the default.
- We developed three interface clients and two logic servers (a reminder service and a remote controller of a virtual appliance).

[5] Hewlett-Packard Development Company, L.P. Jena – a semantic web framework for java.

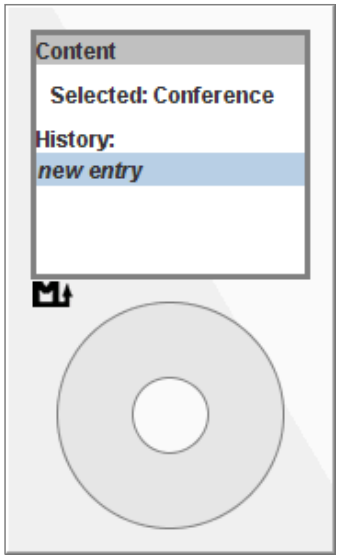
# Clients and servers

a)

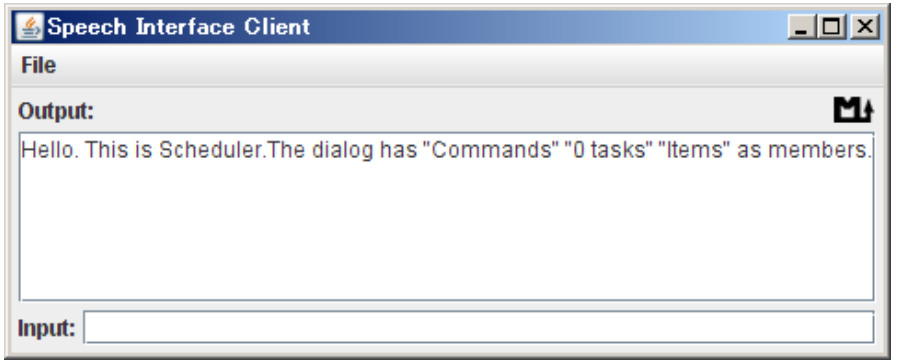



a) The GUI client adopts Swing as a GUI toolkit, and it generates GUI dialog boxes based on AIDL documents.

b)




c)




b) The mobile client has a small screen and a wheel control, and offers hierarchical menu UIs.

c) The client of voice UI simulator expresses voice communication with strings.

# 4. RELATED WORK

# Model-based UI development

- For UI generations, generators are required to resolve a mapping between abstracted expressions and concrete elements of UIs.
- There is a lot of related studies, and this mapping is common among them involved in model-based UI design [2, 4, 13].

[2] S. Berti, F. Correani, G. Mori, F. Patern`o, and C. Santoro. TERESA: A transformation-based environment for designing and developing multi-device interfaces. In CHI 2004, 2004.
[4] J. Eisenstein, J. Vanderdonckt, and A. Puerta. Applying model-based techniques to the development of UIs for mobile computers. In IUI '01, 2001.
[13] J. M. Vanderdonckt and F. Bodart. Encapsulating knowledge for intelligent automatic interaction objects selection. In CHI '93, 1993.

# Ubiquitous interactor

- Ubiquitous interactor (UBI) [10] is a method of developing services without depending on devices through describing interactions.
- UBI offers *customization forms* for controlling presentations, but developers have to customize the forms for each device because the forms have no portability.

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

# Personal universal controller

- In personal universal controller (PUC) [8, 9], users can remote control various appliances with a PDA where the system is running.
- It offers *smart templates* for generating conventional presentations on some service domains, but how to define the templates beforehand is not mentioned.

[8] 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.
[9] J. Nichols, B. A. Myers, and K. Litwack. Improving automatic interface generation with smart templates. In IUI 04, 2004.

# UIML

- User interface markup language (UIML) [1] is an XML based language for the purpose of describing interfaces independently of specific platforms.
- It offers no runtime support which addresses automatic UI generations.

[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.

# 5. CONCLUSION AND FUTURE WORK

# Conclusion

- We have proposed the abstract interaction description language (AIDL) for describing interactions for our UI architecture, interface client/logic server (ICLS).
- It utilizes RDF for expressing meanings of interactions and describing service-specific interactions, and it enables clients to perform meaning inference for flexible UI generations.

# Future work

- We have to establish generation schemes for each modality.
- We are also considering exploiting outcomes in the area of transcodings, which handles conversion of the presentation of output information.