A MODALITY-INDEPENDENT MMI SYSTEM ARCHITECTURE

Kouichi Katsurada Yoshihiko Otani Yusaku Nakamura
Satoshi Kobayashi Hirobumi Yamada Tsuneo Nitta

Toyohashi University of Technology, Japan
katurada@tutkie.tut.ac.jp

ABSTRACT

This paper discusses the design of a modality-independent MMI system architecture. In the architecture, the MMI system is divided into three modules: the document-server module which holds dialog scenarios and contents, the dialog manager which controls dialog flow, and the front-end module which manages the user's inputs and the system's outputs. This division enables us to reuse the document-server module and the dialog manager when introducing new terminals with different types of modalities because they are independent of modalities. Moreover, we propose an MMI description language XISL. Since it has the flexibility to describe the user's inputs and the system's outputs, it can be used for describing interactions on various terminals without introducing a new description language and its processor. We show a prototype system of an online shopping application implemented on our architecture, and compare the difference between XISL and other languages.

1. INTRODUCTION

Discussions on multimodal interaction (MMI) have just started. The WWW Consortium organized a multimodal working group [1], and started standardization of the MMI description language. Some other organizations have been specifying their own descriptions to deal with multi-modalities [2] [3] [4]. The purpose of these approaches is to provide seamless web services on various types of terminals such as mobile phones, PDAs, information kiosks and other terminals in addition to ordinary PCs.

MMI on various terminals would enable advanced web services to be provided, but it raises a new problem, the so-called *n*m* problem. If we use *n* applications on *m* terminals with different types of modalities, system developers must write *n*m* programs, which is time consuming. To prevent this problem, the system should be designed so that developers can easily reuse the common modules among terminals. Our proposal is to provide an MMI system architecture in which some modules are easy to reuse, which we achieve by separating the modality-independent modules from the modality-dependent modules.

In the architecture, the MMI system is divided into three modules: the document-server module which holds dialog scenarios and contents, the dialog manager which controls dialog flow, and the front-end module which manages the user's inputs and the system's outputs. Since the document-server module and the dialog manager are independent of modalities, we can reuse them on various terminals. Moreover, we propose a language XISL to describe MMI scenarios. Since XISL has the flexibility to describe the user's inputs and the system's outputs, we can use it for describing interactions on various types of terminals without introducing a new description language and its processor.

This paper is organized as follows: Section 2 outlines the MMI system architecture. Section 3 describes the MMI description language XISL and its input/output specifications. Section 4 then outlines an online shopping system implemented on our architecture. After comparing the difference between XISL and other languages in section 5, we conclude our discussion in section 6.

2. MMI SYSTEM ARCHITECTURE

2.1. Design of MMI system architecture

Our goal is to provide an MMI system architecture that satisfies the following conditions.

- We can easily reuse its modules when introducing a new terminal with different types of modalities.
- We can implement various applications on it.

In order to satisfy these conditions, we designed the MMI system architecture shown in Figure 1.

In the architecture, an MMI system is divided into three modules: the document-server module, the dialog manager, and the front-end module. The dialog manager is independent of both application and terminal, while the document server

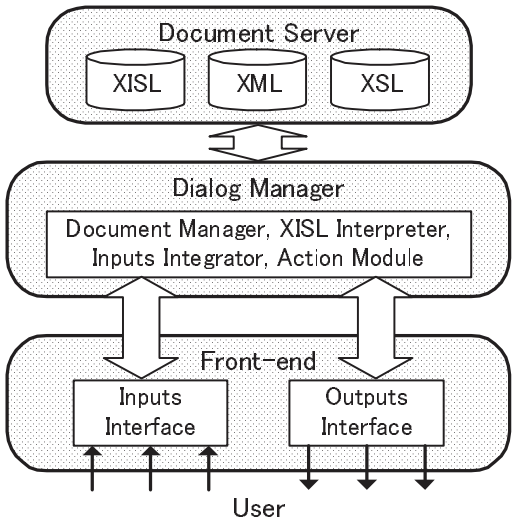

Figure 1: MMI System Architecture.

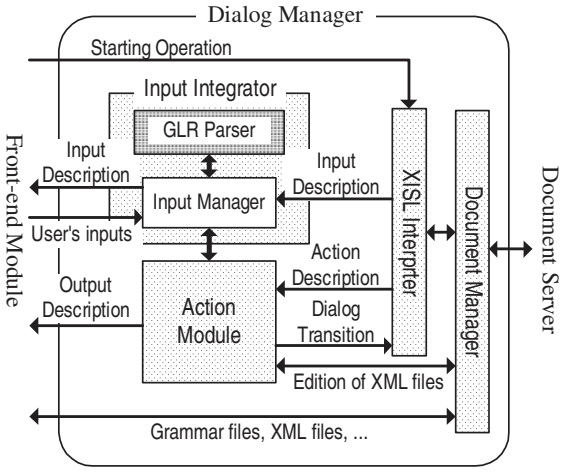

Figure 2: Dialog Manager

module and the front-end module depend on applications and terminals, respectively. In our architecture, applications are written in the MMI description language XISL. In the remainder of this section, we firstly outline XISL and then explain each module.

2.2. MMI description language XISL

XISL is a language for describing MMI scenarios between a user and a system. In principle, a scenario is composed of a sequence of exchanges that contains a set of user's multimodal inputs and the system's actions corresponding to the inputs. Actions include outputs to a user, simple arithmetic operations, conditional branches, and so on. The details of XISL are described in section 3.2.

2.3. Dialog manager

The dialog manager downloads an XISL document, interprets it, and controls the flow of dialog. As shown in Figure 2, it is composed of four sub-modules: XISL interpreter, input integrator, action module, and document manager.

Accepting the starting operation from the front-end module, the XISL interpreter requests the document manager to send an XISL document. After accepting the XISL document, the XISL interpreter checks its syntax, divides it into descriptions of the user's multimodal inputs and the system's actions, and then sends them to the input integrator and the action module, respectively.

The input integrator extracts available inputs from the input descriptions, and sends them to the front-end module. Moreover, it makes a GLR table by regarding the input descriptions as grammar rules. The user's inputs from the front-end module are integrated by a GLR parser.

After the input integration, the action module executes the actions corresponding to the inputs. The actions, which are independent of terminal, are executed inside the action module by communicating with other modules, while the terminal-dependent output descriptions are sent to the front-end module.

The document manager downloads documents from the document server, holds them as temporal files, and sends them to the other modules.

```
<?xml  version="1.0"  encoding="Shift-JIS"?>
<!DOCTYPE  xisl  SYSTEM  "xisl.dtd">
<xisl  version="1.0"> ---------- (a)
  <head> ..... </head> ---------- (b)
  <body> ---------- (c)
    <dialog  id="order"> ---------- (d)
      <begin> ---------- (e)
        <output  type="window"  event="open">
          <![CDATA[
            <param  name="window-name"> top-page </param>
            <param  name="uri"> ./OLS.xml </param>
          ]]>
        </output>
        <output  type="speech"  event="play">
          <![CDATA[  <param  name="speech_text">
            May I help you?
            </param>    ]]>
        </output>
      </begin>
      <exchange> ---------- (f)
        <operation  comb="par"> ---------- (g)
          <input  type="pointing"  event="l_click" ---------- (h)
                  match="/object/goods[item:=@id]/fig"/>
          <input  type="speech"  event="recognize"
                  target="count.grxml"
                  match="/grammar/rule[@id=count]"
                  return="number">
            <param name="mode"> data </param>
          </input>
        </operation>
        <action  comb="par"> ---------- (i)
          <get_value  target="Goods.xml"  var="item_name"
                      match="/item[@id=$item]/name"> ---- (j)
          <output  type="speech"  event="play"> ---------- (k)
            <![CDATA[  <param  name="speech_text">
              You ordered <value expr="number">
              <value expr="item_name">.---------- (l)
              </param>    ]]>
          </output>
          <switch  var="number"> ---------- (m)
            <case  value="one">
              <assign  var="Qty"  expr="1"/> ---------- (n)
            </case>
              :
          </switch>
            :
          <goto  next="payment"> ---------- (o)
        </action>
      </exchange>
        :
    </dialog>
  </body>
</xisl>
```

Figure 3: Example of XISL.

Note that the dialog manager does not interpret individual inputs and outputs, they are interpreted in the front-end module. This enables the dialog manager to be independent of modalities, and makes it easy to reuse the dialog manager on various terminals.

2.4. Front-end module

The front-end module interprets input/output descriptions that are sent from the dialog manager. When interpreting them, some additional documents such as grammar files and XML

type	*event*	*target*	*match*	*return*
“speech”	“recognize”	Specify a speech grammar	Specify a path to an available grammar rule	Variables to hold speech contents
“keyboard”	“press”	Specify a displayed XML file	Specify a path to the root node of the XML file	Variables to hold pressed key
:	:	:	:	:

Table 1: A fragment of <input> specification

type	*event*	<param>	
		name	value
“agent”	“create”	“agent_name”	name of the agent
		“agent_file”	agent file to display
	“speak”	“agent_name”	name of the agent
		“speech_text”	spoken text
“video”	“open_file”	“id”	video ID
		“uri”	URI of the video file
:	:	:	:

Table 2: A fragment of <output> specification

contents are needed. They are also loaded from the document manager sub-module in the dialog manager. Upon accepting the user’s input, the front-end module compares it with an input description. If the input coincides with an input description, it is transferred to the dialog manager. When the front-end module receives an output description, the module executes it immediately.

2.5. Document server module

The document server module holds XISL, XML and other documents. The module is a general web server that sends those documents to the dialog manager according to requests, and executes server-side scripts.

3. OUTLINE OF XISL AND FRONT-END

3.1. Outline of XISL

XISL is an XML oriented language to describe MMI. Figure 3 shows an example of an XISL document. The root of the XISL document is an <xisl> element ((a) in Fig. 3) that is divided into two elements: a <head> element ((b) in Fig. 3) and a <body> element ((c) in Fig. 3). The <head> element is a header of the document, and the <body> element is a container of <dialog> elements. Each <dialog> element contains at most one <begin> element ((e) in Fig. 3), one or more <exchange> elements ((f) in Fig. 3), and at most one <end> element, which are used to describe initial processes, a unit of interaction, and terminal processes, respectively.

An <exchange> element is composed of at most one <prompt> element, an <operation> element ((g) in Fig. 3), and an <action> element ((i) in Fig. 3). If an <exchange> element contains a <prompt> element, it is a system initiative interaction. On the other hand, if an <exchange> element contains no <prompt> element, it is a user initiative interaction. A <prompt> element, an <operation> element, and an <action> element represent a prompt to the user, a user’s operation and a system action, respectively.

In a <prompt> element and an <operation> element, the author can describe <output> elements and <input> elements ((h) in Fig. 3), respectively. In an <action> element, the author can describe some elements such as <output> elements ((k) in Fig. 3), <get_value> elements ((j) in Fig. 3), <switch> elements ((m) in Fig. 3), <assign> elements ((n) in Fig. 3), <value> elements ((l) in Fig. 3), <goto> elements ((o) in Fig. 3), and so on.

To increase the flexibility of modalities, strict attributes and contents of <input> and <output> elements are not specified by XISL itself. This enables us to expand or modify modalities easily. The descriptive details of them are intended to be specified by the platform developers.

An interaction flow is controlled by a “comb” attribute of the <dialog> tag whose value is *par* (parallel), *seq* (sequential) or *alt* (alternative). When the “comb” is *par*, all <exchange> elements are executed in parallel. If the “comb” is *alt*, one of the <exchange> elements is performed alternatively, and if the “comb” is *seq*, all <exchange> elements are performed in sequence. We can also control interaction flows by means of <par…>, <seq…> and <alt…> tags. The elements bound by these tags are equivalent in functions to the corresponding attributes of the <dialog> tag. These tags and attributes can also be used in an <operation> tag and an <action> tag (*alt* and <alt…> do not appear in an <action> tag). The details of the XISL specifications are on our web site [5].

3.2. Outline of front-end

We implemented a front-end module on an ordinary PC connected to a touch screen display and specified its description of <input> and <output> elements. The system configuration is as follows.

- OS: Windows® 2000
- Input devices: touch screen (or mouse), keyboard, microphone
- Output devices: touch screen, speaker
- Applications: Microsoft® Agent2.0

Figure 4: An Online Shopping System.

Toshiba LaLaVoice™ 2001
IBM® ViaVoice™ 8 Pro
Microsoft® Mediaplayer
RealNetworks™ RealPlayer™
Microsoft® MSXML3.0
Microsoft® InternetExplorer6

Tables 1 and 2 describe fragments of <input> and <output> specifications. The <input> tag includes five attributes: "type" (input modality), "event" (input event), "target" (target XML file), "match" (path to a target XML element), and "return" (return values) in XISL specifications. The <output> tag contains two attributes: "type" (output modality), and "event" (output event) in XISL specifications. <param>s in the <output> element are described in the CDATA section. The values of attributes and <param>s are terminal-dependent specifications.

In addition to the modalities shown in Tables 1 and 2, the front-end module can handle touch operations (click, drag, etc.), window actions (display, move, etc.), and other operations and actions. The details of the front-end specifications are on our web site [6].

4. A PROTOTYPE SYSTEM

We implemented an OnLine Shopping (OLS) application by using XISL. The OLS application is composed of the dialog scenario for shopping (XISL), the data of items and customers (XML), and the style to display on the screen (XSL). Upon starting the system, a user firstly inputs his ID for authentication. The page shown in Figure 4 is then displayed. This page presents a shopping cart ((1) in Fig. 4), an item list ((2) in Fig. 4), and the list of categories of the items ((3) in Fig. 4). The user can modify the shopping cart, modify the item list, and go to the explanation page. After deciding the items to buy, the user determines the number of items, inputs his personal information, and ends shopping by registering his personal information. The help page can be called up whenever the system is running. All of the user's inputs are made by speech, pointing, or both of them.

5. XISL AND OTHER LANGUAGES

Some organizations have specified their own descriptions to deal with multi-modalities. SALT [2] and XHTML+ VoiceXML [3] attempt to add voice interaction to HTML pages. SALT defines additional tags for describing speech recognition and TTS that are attached to an HTML document. The XHTML+VoiceXML approach attempts to use VoiceXML for additional voice interaction. These approaches effectively expand existing technologies such as HTML and VoiceXML, however, they assume only voice interaction as additional modalities. On the other hand, our approach has no restriction on additional modalities.

The SMIL based approach [4] realizes event-based multimodal dialog control by SMIL and ReX (Reactive XML). In this approach, temporal control of the scenario is described in SMIL, and event control is described in ReX. Since this approach is based on event control, it is easy to extend modalities. However, in this approach, the modality-dependent descriptions are not explicitly separated from the modality-independent descriptions. On the other hand, our approach explicitly separates them. Therefore, it is easy to read and reuse the dialog scenario written in XISL.

6. CONCLUSIONS

This paper provides an MMI system architecture, and MMI description language XISL. The dialog manager of the architecture is explicitly separated from the front-end module that depends on a certain modality. Moreover, modality-dependent description of XISL is concentrated in <input> and <output> tags, and they are interpreted only in the front-end module. These features make it easy to reuse the dialog manager on various types of terminals, and easy to read the dialog scenario written in XISL. Therefore the architecture and XISL are useful for constructing general MMI systems.

Future problems are extension of XISL and implementing MMI system on various terminals.

7. REFERENCES

[1] http://www.w3.org/2002/mmi/
[2] http://www.saltforum.org/
[3] http://www.w3.org/TR/xhtml+voice/
[4] Beckham, J.L., Fabbrizio, G.D., and Klarlund, N., "Towards SMIL as a Foundation for Multimodal, Multimedia Applications", Proc. of EUROSPEECH 2001, pp.1363-1366 (2001).
[5] http://www.vox.tutkie.tut.ac.jp/XISL/XISL-E.pdf
[6] http://www.vox.tutkie.tut.ac.jp/XISL/front-end-E.pdf