Management of Static/Dynamic Properties in a Multimodal Interaction System

Kouichi Katsurada[1], *Yuji Okuma*[2], *Makoto Yano*[1], *Yurie Iribe*[1], *Tsuneo Nitta*[1]

[1] Toyohashi University of Technology, Toyohashi, Japan
[2] OBIC Co., LTD.
{katurada, nitta}@tutkie.tut.ac.jp, {okuma, yano, iribe}@vox.tutkie.tut.ac.jp

Abstract

This paper provides a mechanism to deal with static/dynamic properties in a web-based Multi-Modal Interaction (MMI) system. The static/dynamic properties in this paper include user profile, user's facial expression, surrounding environment, and so on. By using these properties, the MMI system can make interaction more natural based on context or situation. To consolidate these properties into a module, we have designed and developed a static/dynamic property manager on our MMI system. We have also prototyped a user navigation system that introduced user profile, user's facial expression and GPS information as static/dynamic properties.
Index Terms: multimodal interaction, static/dynamic properties

1. Introduction

Various frameworks are proposed for constructing web-based Multi-Modal Interaction (MMI) systems [1][2][3][4]. Their main purposes are to incorporate speech interface into web-based applications. For this purpose, they prepare XML-based languages to describe speech grammars, prompts, dialog flows, and so on. Although these languages provide enough functions for embedding speech interface into traditional web applications, they do not explicitly show how the passive properties should be handled on MMI systems. Passive properties, such as user's facial expression, eye gaze, etc., are pointed out to improve interpretation accuracy of user's active operations and to enhance system robustness [5]. In this paper we provide a framework to handle passive properties on a web-based MMI system.

There have been proposed a lot of MMI systems that handles passive properties [6], and most of them treat passive properties the same as user's active operations. However, it causes complexity of dialog management because passive properties play a different role from active operations. In an MMI system, user's active operations explicitly express the user's intentions, while passive properties only give implicit intentions of a user. Especially in a web-based system, XML-based language is usually designed for describing user's active operations such as form filling, link clicking, and so on. Therefore if the passive properties are mixed with active operations, the description may be complex, which impairs its readability. For this reason, we provide a separated module to handle passive properties on a web-based MMI system. We call this module the static/dynamic property manager. It does not only manage passive properties but also handles various type of information such as user's profile, surrounding environment, system configuration, user's location, and so on. In this paper we show its module structure and interfaces prepared for accessing the properties.

In the following sections, firstly, we outline our MMI system and its internal language XISL2.0. Then, we present the static/dynamic property manager. After evaluation of the manager, we conclude our work and show future works.

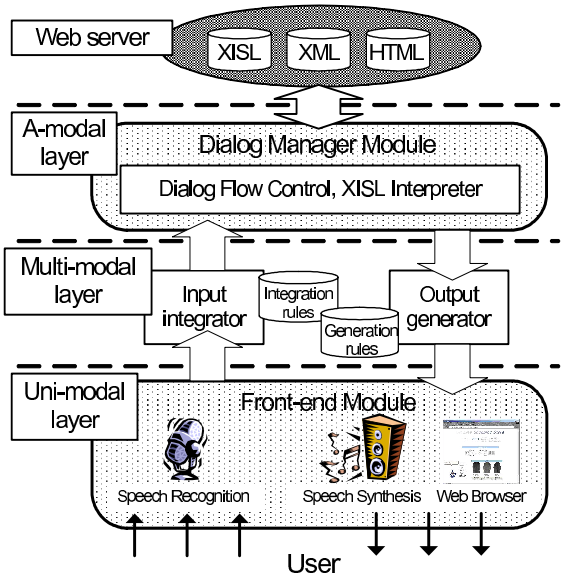


Figure 1: *Framework of MMI system.*

2. Multimodal Interaction System

2.1. System architecture

An MMI system is the system that integrates multiple inputs from various modalities to understand user's intention. Since technological innovation may enable the use of novel modalities in near future, it is desirable that the MMI system has extensibility of modalities. For this reason, we have proposed a framework of MMI system that has enough extensibility of modalities [4][7]. Figure 1 shows the framework.

Our MMI system is designed based on a three-layered model. The uni-modal layer is the lowest layer that handles each modality separately. It is implemented as a front-end module in our MMI system. The module actuates a speech recognition engine, a speech synthesis engine, a web browser, and so on. Since this module is implemented separately from the other modules, developers can easily add/modify modalities without changing whole of the system. The multi-modal layer executes modality fusion and media fission. We implemented them as an input integrator module and an output generator module. These modules use integration/generation rules for integrating inputs or generating outputs. The a-modal layer is the highest layer which receives/delivers modality independent information. It is implemented as a dialog manager. The dialog manager

```
<?xml version="1.0" encoding="Shift-JIS"?>
<!DOCTYPE xisl SYSTEM "xisl.dtd">
<xisl version="2.0">
  <body>
    <form id="SHOP"> ……..…….. (1)
      <fe> <!-- Display an HTML content --> </fe> ……..…….. (2)
      <fe> <!-- Set a speech recognition grammar --> </fe> … (3)
      <initial> ……..…….. (4)
        <prompt>
          <fe><!--A prompt to ask item and quantity --></fe>
        </prompt>
      </initial>
      <field name="ITEM"> ……..…….. (5)
        <prompt>
          <fe> <!--A prompt to ask an item --> </fe>
        </prompt>
        <filled> ……..…….. (6)
          <backend action="set_item.cgi" namelist="ITEM"/> (7)
            :
        </filled>
      </field>
      <field name="QTY">
        <prompt>
        <fe> <!--A prompt to ask quantity of the item --> </fe>
        </prompt>
        <filled>
          <backend action="set_quantity.cgi" namelist="QTY"/>
            :
        </filled>
      </field>
      <filled namelist="ITEM QTY">
        <fe> <!--You ordered QTY ITEMs. --> </fe>
          :
      </filled>
    </form>
  </body>
</xisl>
```

Figure 2: *An example of MMI description with XISL2.0.*

controls dialog flows along MMI scenarios described in XISL2.0 [7]. The dialog manager first downloads an XISL2.0 document from a web server, interprets it, and then executes dialog along the scenario.

2.2. XISL2.0

XISL2.0 is an XML-based language that has a similar syntax to VoiceXML [8]. It has various notations to describe MMI such as user's multimodal inputs, system's multimedia outputs, dialog transitions, conditional branches, arithmetic operations, and so on. Figure 2 shows a fragment of an XISL2.0 document for an online shopping application.

XISL2.0 is executed along almost the same algorithm as the one used in VoiceXML, the so-called FIA (Form Interpretation Algorithm). When a <form> element ((1) in Figure 2) is visited, two <fe> elements ((2) and (3) in Figure 2) are executed first. <fe> elements are XISL2.0 original elements that is used for setting some parameters to modalities/media in the MMI system. Then, an <initial> element ((4) in Figure 2) is selected and a prompt is output. The <field> element ((5) in Figure 2) is an input slot that accepts a user's input. In this example two <field> elements are prepared for inputs concerning an item and its quantity. If a <field> is filled, the <filled> element ((6) in Figure 2) in the <field> is executed. The <backend> element ((7) in Figure 2) starts a CGI program on the web server and receives its return value in a variable specified in the "namelist" property. If there are unfilled elements, the FIA algorithm automatically selects a <field> element, outputs a <prompt> in the <field> element, and waits for an input for the <field>. For further details of XISL2.0 tags, the reader can refer to the article [7].

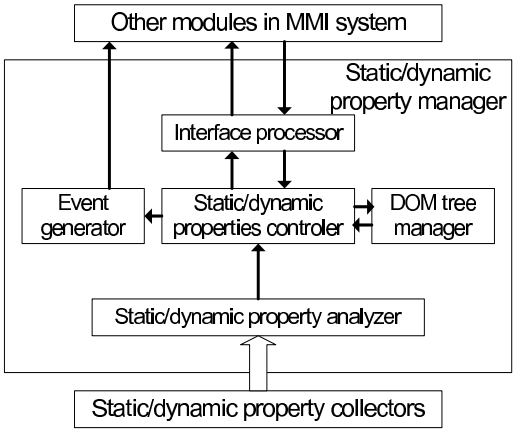


Figure 3: *Static/dynamic property manager.*

3. Static/Dynamic Property Manager

3.1. Static/dynamic properties

Static/dynamic properties in this paper include static information (user profile, system configuration, etc.) and dynamic information (user's facial expression, eye gaze, user's location, surrounding environment, etc.). These properties can be used in various modules of the MMI system. For example, user profile might be used to select appropriate service for a user, user's facial expression will provide important information when interpreting his/her intention, and surrounding environment triggers the change of input/output modalities in a front-end module. Although there are various types of properties, interfaces to access these properties should be unified to avoid any confusion. For this reason, we have incorporated the static/dynamic property manager into our MMI system, and specified some APIs to access to the properties.

3.2. Design of static/dynamic property manager

We extracted the following functions as requirements to be satisfied by the static/dynamic property manager.

- A mechanism to add/modify the properties easily
- Push type interface to notify change of properties
- Pull type interface to access to the properties

To satisfy these conditions, we designed the static/dynamic property manager based on Delivery Context Interfaces (DCI) [9] that is discussed in W3C. DCI is a framework to handle static/dynamic properties in a web-based system. In the DCI framework, the properties are managed as a form of DOM tree structure. It provides some APIs to access to the properties, and uses XML events to notify the change of properties. We employed almost the same mechanism as this framework to handle static/ dynamic properties.

Figure 3 shows the module structure of the static/dynamic property manager. It is composed of five sub-modules. When a property collector sends a property value to the static/ dynamic manager, a static/dynamic property analyzer receives the value first, and sends it to the static/dynamic property controller. Then the controller sends it to a DOM tree manager. Next, the DOM tree manager compares the value with the original one. If they are different, the DOM tree manager sends the change to an event generator through the static/dynamic property controller. The event generator sends

```
<?xml version="1.0" encoding="Shift-JIS"?>
<!DOCTYPE xisl SYSTEM "xisl.dtd">
<xisl version="2.0">
  <body>
    <catch event="SE.chg_user_face_exp"
           return="user_face_emo"> ..... (1)
      <if cond="session.SE.user$state eq 'dissatisfied' "> ..... (2)
        <then>
          <fe><!-- Do you have any problem? --></fe>
          <goto next="#search_landmark">
        </then>
      </if>
    </catch>
      :
  </body>
</xisl>
```

Figure 4: *Description to handle static/dynamic properties.*

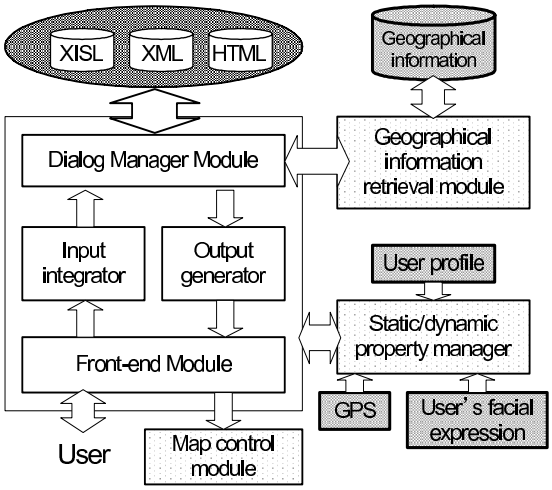


Figure 5: *Module structure of the user navigation system.*

an event to the other modules in the MMI system. On the other hand, when the MMI system asks for some property values, an interface processor accepts the request and refers to the DOM tree via the static/dynamic property controller. Then the interface processor returns the values.

The static/dynamic properties are not only used by the modules in the MMI system, but also referred from XISL documents. XISL prepares two types of interfaces for static/dynamic properties. First one is a push type interface. In XISL, <catch> element is used for this purpose. (1) in Figure 4 shows how XISL receives the information from the static/dynamic property manager. The event name "SE…" expresses the event from the manager. Second interface is a pull type interface. It is realized by using session variables. As described at (2) in Figure 4, the variable name "session.SE…" represents the variable that indicates a static/dynamic property. The path of the DOM tree is described following the prefix string "session.SE.".

3.3. Sample application development

We have prototyped a user navigation system constructed on our MMI system. The system navigates various institutions or locations according to user preference. Figure 5 illustrates the system structure and Figure 6 shows its screen shot. As shown in Figure 5, it contains a map control module and a geographical information retrieval module in addition to our MMI system. It handles three types of static/dynamic property collectors: a GPS information collector, a user's facial expression collector, and a user profile manager.

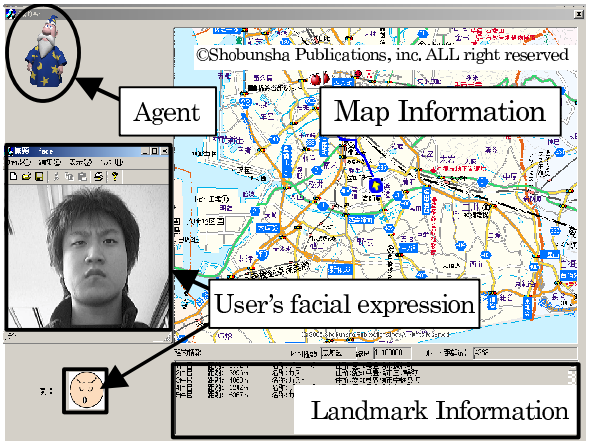


Figure 6: *A screen shot of the user navigation system.*

A user is driving a car. He likes beef bowl.
User: I'm very hungry.
(His favorite food is registered as user profile.)
(The system judges that he is on a car from GPS Information.)
System: There is a beef bowl restaurant about 1.5km from here. But it is in the opposite direction.
User: (Dissatisfied expression.)
System: Do you want to search in the forward direction?
User: That's more like it.
System: The nearest restaurant is a hamburger shop. It is about 3.2km from here.
User: O.K. Please navigate.

Figure 7: *Dialog between a user and the navigation system.*

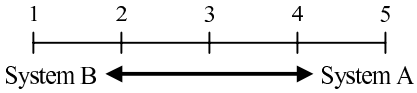
The GPS information collector can receive user's location, movement speed and direction. User's location is used to decide scope of geographical information retrieval. Movement speed and direction are used to judge whether the user is walking or using transportation. If the user seems to drive, the system expands the scope of retrieval. User's facial expression is used to evaluate the navigation. If the user looks dissatisfied when the system presents some restaurants, the system changes the scope of retrieval and gives other restaurants to the user. Figure 7 shows an example of dialog between a user and the navigation system.

3.4. Evaluation of static/dynamic property manager through application development

Some benefits and issues of the static/dynamic property manager become apparent through developing the speech navigation system. In this section, we evaluate the manager with regard to its interfaces, variety of data, and functions. Firstly, push and pull type interfaces work well for constructing the navigation system. We could appropriately refer to the user's facial expression and some other static/dynamic properties by using <catch> elements and session variables. As for variety of data, our manager only handles latest data given by the information collectors. Although it does not cause any inconvenience while developing our navigation system, the other types of data such as time series data may be useful for more natural user adaptation. We will introduce these data into the future version of the static/dynamic properties manager. Lastly, as for functions, our manager only gives static/dynamic properties to the modules in our MMI system. However, if it provides the properties to the other modules such as information collectors, these modules can use them for

Table 1: Easiness of system development

Items	rate
Which system is easy to develop?	4.5
Which system do you want to use when you have to introduce a new static/dynamic property collector?	4.5
Which system is easy to comprehend data flows?	4.25
Which system could you finish developing in a short time?	4.5
Which system has the simple development process?	4.5



modifying their internal parameters. We would like to redesign the manager to enable such data transfer.

3.5. Experimental evaluation

We have made experimental evaluation of validity of the static/dynamic property manager. In the evaluation, four subjects were asked to develop two MMI systems using facial expression as a static/dynamic property. One system uses the static/dynamic property manager (system A) and the other one does not use it (system B). In system B, they have to develop some functions of the static/dynamic property manager such as push/pull type interfaces, an event generator, and so on. Subjects are asked to give five-level rating about easiness of system development of two systems. Table 1 shows the result. It shows that system A gets higher scores in all the items. This result indicates that the static/dynamic property manager plays an important role on making it easy to construct MMI systems with static/dynamic properties.

4. Comparison with Other Approaches

4.1. Management of static/dynamic properties on the other MMI systems

A lot of previous MMI systems use static/dynamic properties. SmartKom Public uses user's facial expression and prosody for obtaining user's intension, and SmartKom Mobile uses GPS information for user navigation [10], whereas MATCH system uses user profile for navigation [11]. Some other systems uses eye gaze, facial image recognition, and so on [6]. Most of these systems employ either open agent architecture or blackboard type architecture for managing user's active operations and static/dynamic properties [12]. The open agent architecture uses a facilitator agent for the management, while the blackboard architecture prepares a blackboard for sharing information. Our framework is essentially different from these approaches because we explicitly distinguish user's active operations and the static/dynamic properties. However, the structure of static/dynamic property manager is almost same as the open agent architecture.

4.2. Relation to some web technologies

W3C specifies some standards related to static/dynamic property management. CC/PP specifies a data format for submitting properties to a web server [13]. Although it is designed for sending user profile, preference, device configuration, and so on, it does not specify timing of data transfer and APIs for referring to the properties.

DCI is a framework to handle static/dynamic properties in web-based systems as described in section 3.2 [9]. It defines how to manage the properties and gives some APIs for data access. However, it does not show how the system structure should be. This paper shows a concrete implementation of the manager and evaluates them through developing an application.

5. Conclusions

We have designed and developed the static/dynamic property manager that handles properties such as user property, user's facial expression and surrounding environment in a web-based MMI system. These properties are very important to understand user's intention and make interactions more natural with them. To show the validity of the module, we have prototyped a user navigation system that includes user profile, user's facial expression and GPS information as the static/dynamic properties. Future work is to extend the manager concerning the issues discussed in section 3.4 (such as variety of data and system stricture), and to develop a dialog system that can adjust the style of output contents to each user according to his/her property such as age, sex, and so on.

6. Acknowledgements

This work was supported by Grant-in-Aid for Young Scientists (B) 17700185 2007, from the ministry of Education, Culture, Sports, Science and Technology.

7. References

[1] http://www.w3.org/2002/mmi/
[2] http://www.saltforum.org/
[3] http://www.w3.org/TR/xhtml+voice/
[4] K. Katsurada, Y. Otani, Y. Nakamura, S. Kobayashi, H. Yamada, and T. Nitta, "A modality-independent MMI system architecture", Proc. ICSLP'02, Denver, United States, pp.2549-2552, (2002).
[5] S. Oviatt, P. Cohen, L. Wu, L. Duncan, B. Suhm, J. Bers, T. Holzman, T. Winograd, J. Landay, J. Larson, and D. Ferro, "Designing the user interface for multimodal speech and pen-based gesture applications: State-of-the-art systems and future research directions", Human-Computer Interaction, Vol.15, No.4, pp.263-322 (2000).
[6] D. Gibbon, I. Mertins and R.K. Moore (ed.): Handbook of Multimodal and Spoken Dialogue Systems, Kluwer Academic Publishers, section 2.2, pp.118-122 (2000).
[7] K. Katsurada, K. Aoki, H. Yamada, and T. Nitta, "Reducing the description amount in authoring MMI applications", Proc. INTERSPEECH'05, Lisbon, Portugal, pp.873-876 (2005).
[8] http://www.w3.org/TR/voicexml20/
[9] http://www.w3.org/TR/DPF/
[10] N. Reithinger, J. Alexandersson, T. Becker, A. Blocher, R. Engel, M. Lockelt, J. Muller, N. Pfleger, P. Poller, M. Streit nad V. Tschernomas, "SmartKom – Adaptive and flexible multimodal access to multiple applications", Proc. of ICMI'03, pp.101-108, (2003).
[11] M. Johnston, S. Bangalore, G. Vasireddy, A. Stent, P. Ehlen, M. Walker, S. Whittaker and P. Maloor, "MATCH: An architecture for multimodal dialogue systems", Proc. of the Annual Meeting of the Association for Computational Linguistics, pp.376-383 (2002).
[12] R. L. Delgado and M. Araki: Spoken, Multilingual and Multimodal Dialogue Systems, John Wiley & Sons, Ltd, section3.2.3, pp.67-70 (2005).
[13] http://www.w3.org/TR/CCPP-struct-vocab/