Multimodal Dialog Description Language for Rapid System Development

Masahiro Araki **Kenji Tachibana**
Kyoto Institute of Technology
Graduate School of Science and Technology, Department of Information Science
Matsugasaki Sakyo-ku Kyoto 606-8585 Japan
araki@dj.kit.ac.jp

Abstract

In this paper, we explain a rapid development method of multimodal dialogue sys-tem using MIML (Multimodal Interaction Markup Language), which defines dialogue patterns between human and various types of interactive agents. The feature of this language is three-layered description of agent-based interactive systems which separates task level description, interaction description and device dependent realization. MIML has advantages in high-level interaction description, modality extensibility and compatibility with standardized technologies.

1 Introduction

In recent years, various types of interactive agents, such as personal robots, life-like agents (Kawamoto et al. 2004), and animated agents are developed for many purposes. Such interactive agents have an ability of speech communication with human by using automatic speech recognizer and speech synthesizer as a main modality of communication. The purpose of these interactive agents is to realize a user-friendly interface for information seeking, remote operation task, entertainment, etc.

Each agent system is controlled by different description language. For example, Microsoft agent is controlled by JavaScript / VBScript embedded in HTML files, Galatea (Kawamoto et al.. 2004) is controlled by extended VoiceXML (in Linux version) and XISL (Katsurada et al. 2003) (in Windows version). In addition to this difference, these languages do not have the ability of higher level task definition because the main elements of these languages are the control of modality functions for each agent. These make rapid development of multimodal system difficult.

In order to deal with these problems, we propose a multimodal interaction description language, MIML (Multimodal Interaction Markup Language), which defines dialogue patterns between human and various types of interactive agents by abstracting their functions. The feature of this language is three-layered description of agent-based interactive systems.

The high-level description is a task definition that can easily construct typical agent-based interactive task control information. The middle-level description is an interaction description that defines agent's behavior and user's input at the granularity of dialogue segment. The low-level description is a platform dependent description that can override the pre-defined function in the interaction description.

The connection between task-level and interaction-level is realized by generation of interaction description templates from the task level description. The connection between interaction-level and platform-level is realized by a binding mechanism of XML.

The rest of this paper consists as follows. Section 2 describes the specification of the proposed language. Section 3 explains a process of rapid multimodal dialogue system development. Section 4 gives a comparison with existing multimodal languages. Section 5 states conclusions and future works.

Proceedings of the 7th SIGdial Workshop on Discourse and Dialogue, pages 109–116,
Sydney, July 2006.

2 Specification of MIML

2.1 Task level markup language

2.1.1 Task classification

In spoken dialogue system development, we proposed task classification based on the direction of information flow (Araki et al. 1999). We consider that the same analysis can be applied to agent based interactive systems (see Table 1).

Table 1: Task classification of agent-based interactive systems

Class	Direction of Info. flow	Typical task
Information assistant	user ← agent	Interactive presentation
User agent	user → agent	control of home network equipments
Question and Answer	user ↔ agent	daily life information query

In the information assistant class, the agent has information to be presented to the user. Typically, the information contents are Web pages, an instruction of consumer product usage, an educational content, etc. Sometimes the contents are too long to deliver all the information to the user. Therefore, it needs user model that can manage user's preference and past interaction records in order to select or filter out the contents.

In the user agent class, the user has information to be delivered to the agent in order to achieve a user's goal. Typically, the information is a command to control networked home equipments, travel schedule to reserve a train ticket, etc. The agent mediates between user and target application in order to make user's input appropriate and easy at the client side process (e.g. checking a mandatory filed to be filled, automatic filling with personal data (name, address, e-mail, etc.)).

In the Question and Answer class, the user has an intention to acquire some information from the agent that can access to the Web or a database. First, the user makes a query in natural language, and then the agent makes a response according to the result of the information retrieval. If too much information is retrieved, the agent makes a narrowing down subdialogue. If there is no information that matches user's query, the agent makes a request to reformulate an initial query. If the amount of retrieved information is appropriate to deliver to the user by using current modality, the agent reports the results to the user. The appropriate amount of information differs in the main interaction modality of the target device, such as small display, normal graphic display or speech. Therefore, it needs the information of media capability of the target device.

2.1.2 Overview of task markup language

As a result of above investigation, we specify the task level interaction description language shown in Figure 1.

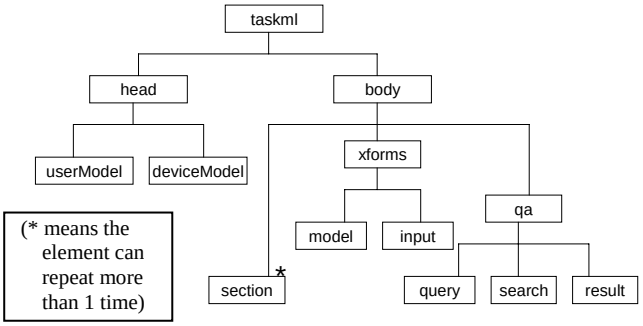

Figure. 1 Structure of the Task Markup Language.

The features of this language are (1) the ability to model each participant of dialogue (i.e. user and agent) and (2) to provide an execution framework of each class of task.

The task markup language <taskml> consists of two parts corresponding to above mentioned features: <head> part and <body> part. The <head> part specifies models of the user (by <userModel> element) and the agent (by <deviceModel> element). The content of each model is described in section 2.1.3. The <body> part specifies a class of interaction task. The content of each task is declaratively specified under the <section>, <xforms> and <qa> elements, which are explained in section 2.1.4.

2.1.3 Head part of task markup language

In the <head> element of the task markup language, the developer can specify user model in <userModel> element and agent model in <deviceModel> element.

In the <userModel> element, the developer declares variables which represent user's information, such as expertise to domain, expertise to dialogue system, interest level to the contents, etc.

In the <deviceModel> element, the developer can specify the type of interactive agent and main modality of interaction. This information is

used for generating template from this task description to interaction descriptions.

2.1.4 Body part of task markup language

According to the class of the task, the <body> element consists of a sequence of <section> elements, a <xforms> element or a <qa> element.

The <section> element represents a piece of information in the task of the information assistant class. The attributes of this element are id, start time and end time of the presentation material and declared user model variable which indicates whether this section meets the user's needs or knowledge level. The child elements of the <section> element specify multimodal presentation. These elements are the same set of the child elements of <output> element in the interaction level description explained in the next subsection. Also, there is a <interaction> element as a child element of the <section> element which specifies agent interaction pattern description as an external pointer. It is used for additional comment generated by the agent to the presented contents. For the sake of this separation of contents and additional comments, the developer can easily add agent's behavior in accordance with the user model. The interaction flow of this class is shown in Figure 2.

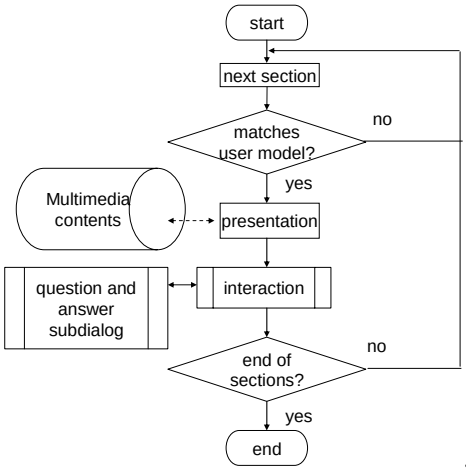

Figure. 2 Interaction flow of Information Assist class

The <xforms> element represents a group of information in the task of the user agent class. It specifies a data model, constraint of the value and submission action following the notation of XForms 1.0.

In the task of user agent class, the role of interactive agent is to collect information from the user in order to achieve a specific task, such as hotel reservation. XForms is designed to separate the data structure of information and the appearance at the user's client, such as using text field input, radio button, pull-down menu, etc. because such interface appearances are different in devices even in GUI-based systems. If the developer wants to use multimodal input for the user's client, such separation of the data structure and the appearance, i.e. how to show the necessary information and how to get user's input, is very important.

In MIML, such device dependent 'appearance' information is defined in interaction level. Therefore, in this user agent class, the task description is only to define data structure because interaction flows of this task can be limited to the typical patterns. For example, in hotel reservation, as a result of AP (application) access, if there is no available room at the requested date, the user's reservation request is rejected. If the system recommends an alternative choice to the user, the interaction branches to subdialogue of recommendation, after the first user's request is processed (see Figure 3). The interaction pattern of each subdialogue is described in the interaction level markup language.

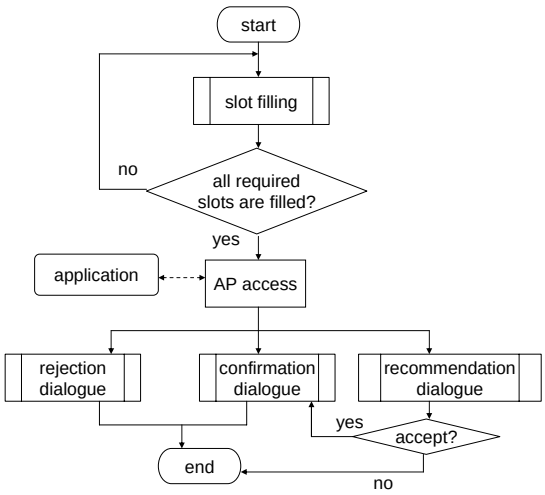

Figure. 3 Interaction flow of User Agent class

The <qa> element consists of three children: <query>, <search> and <result>.

The content of <query> element is the same as the <xforms> element explained above. However, generated interaction patterns are different in user agent class and question and answer class. In user agent class, all the values (except for optional slots indicated explicitly) are expected to be filled. On the contrary, in question and answer class, a subset of slots defined by form description can make a query. Therefore, the first ex-

change of the question and answer class task is system's prompt and user's query input.

The <search> element represents application command using the variable defined in the <query> element. Such application command can be a database access command or SPARQL (Simple Protocol And RDF Query Language)[1] in case of Semantic Web search.

The <result> element specifies which information to be delivered to the user from the query result. The behavior of back-end application of this class is not as simple as user agent class. If too many results are searched, the system transits to narrowing down subdialogue. If no result is searched, the system transits to subdialogue that relaxes initial user's query. If appropriate number (it depends on presentation media) of results are searched, the presentation subdialogue begins. The flow of interaction is shown in Figure 4.

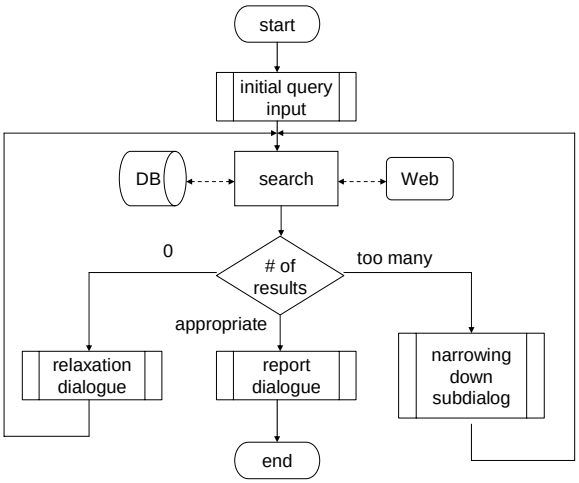

Figure. 4 Interaction flow of Question and Answer class

2.2 Interaction level markup language

2.2.1 Overview of interaction markup language

Previously, we proposed a multimodal interaction markup language (Araki et al. 2004) as an extension of VoiceXML[2]. In this paper, we modify the previous proposal for specializing human-agent interaction and for realizing interaction pattern defined in the task level markup language.

The main extension is a definition of modality independent elements for input and output. In VoiceXML, system's audio prompt is defined in <prompt> element as a child of <field> element that defines atomic interaction acquiring the value of the variable. User's speech input pattern is defined by <grammar> element under <field> element. In our MIML, <grammar> element is replaced by the <input> element which specifies active input modalities and their input pattern to be bund to the variable that is indicated as name attribute of the <field> element. Also, <prompt> element is replaced by the <output> element which specifies active output modalities and a source media file or contents to be presented to the user. In <output> element, the developer can specify agent's behavior by using <agent> element. The outline of this interaction level markup language is shown in Figure 5.

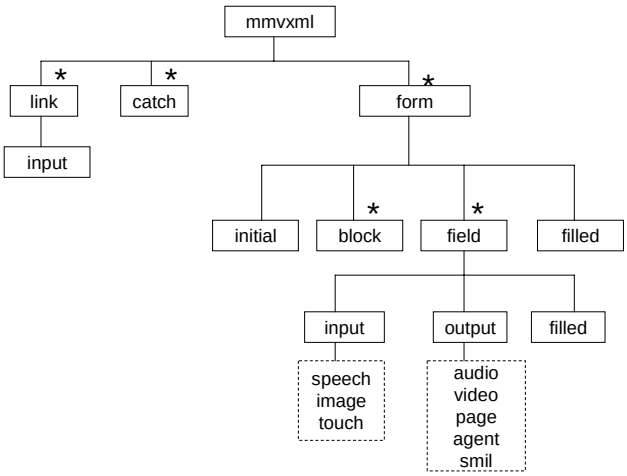

Figure. 5 Structure of Interaction level Markup Language

2.2.2 Input and output control in agent

The <input> element and the <output> element are designed for implementing various types of interactive agent systems.

The <input> element specifies the input processing of each modality. For speech input, grammar attribute of <speech> element specifies user's input pattern by SRGS (Speech Recognition Grammar Specification)[3], or alternatively, type attribute specifies built-in grammar such as Boolean, date, digit, etc. For image input, type attribute of <image> element specifies built-in behavior for camera input, such as nod, faceRecognition, etc. For touch input, the value of the variable is given by referring external definition of the relation between displayed object and its value.

The <output> element specifies the output control of each modality. Each child element of

[1] http://www.w3.org/TR/rdf-sparql-query/
[2] http://www.w3.org/TR/voicexml20/
[3] http://www.w3.org/TR/speech-grammar/

this element is performed in parallel. If the developer wants to make sequential output, it should be written in <smil> element (Synchronized Multimedia Integration Language)[4], For audio output, <audio> element works as the same way as VoiceXML, that is, the content of the element is passed to TTS (Text-to-Speech module) and if the audio file is specified by the src attribute, it is a prior output. In <video>, <page> (e.g. HTML) and <smil> (for rich multimedia presentation) output, each element specifies the contents file by src attribute. In <agent> element, the agent's behavior definition, such as move, emotion, status attribute specifies the parameter for each action.

2.3 Platform level description

The differences of agent and other devices for input/output are absorbed in this level. In interaction level markup language, <agent> element specifies agent's behavior. However, some agent can move in a real world (e.g. personal robot), some agent can move on a computer screen (e.g. Microsoft Agent), and some cannot move but display their face (e.g. life-like agent).

One solution for dealing with such variety of behavior is to define many attributes at <agent> element, for example, move, facial expression, gesture, point, etc. However, the defects of this solution are inflexibility of correspondence to progress of agent technology (if an agent adds new ability to its behavior, the specification of language should be changed) and interference of reusability of interaction description (description for one agent cannot apply to another agent).

Our solution is to use the binding mechanism in XML language between interaction level and platform dependent level. We assume default behavior for each value of the move, emotion and status attributes of the <agent> element. If such default behavior is not enough for some purpose, the developer can override the agent's behavior using binding mechanism and the agent's native control language. As a result, the platform level description is embedded in binding language described in next section.

3 Rapid system development

3.1 Usage of application framework

Each task class has a typical execution steps as investigated in previous section. Therefore a system developer has to specify a data model and specific information for each task execution. Web application framework can drive interactive task using these declarative parameters.

As an application framework, we use Struts[5] which is based on Model-View-Controller (MVC) model. It clearly separates application logic (model part), transition of interaction (controller part) and user interface (view part). Although MVC model is popular in GUI-based Web application, it can be applied in speech-based application because any modality dependent information can be excluded from the view part. Struts provides (1) a controller mechanism and (2) integration mechanism with the back-end application part and the user interface part. In driving Struts, a developer has to (1) define a data class which stores the user's input and responding results, (2) make action mapping rules which defines a transition pattern of the target interactive system, and (3) make the view part which defines human-computer interaction patterns. The process of Struts begins by the request from the user client (typically in HTML, form data is submitted to the Web server via HTTP post method).

The controller catches the request and stores the submitted data to the data class, and then calls the action class specified by the request following the definition of action mapping rules.

The action class communicates with the back-end application, such as database management system or outside Web servers by referring the data class, and returns the status of the processing to the controller. According to the status, the controller refers the action mapping rules and selects the view file which is passed to the user's client. Basically, this view file is written in Java Server Pages, which can be any XML file that includes Java code or useful tag libraries. Using this embedded programming method, the results of the application processing is reflected to the response. The flow of processing in the Struts is shown in Figure 6.

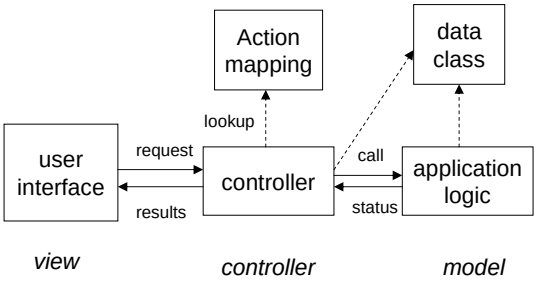

Figure. 6 MVC model.

[4] http://www.w3.org/AudioVideo/

[5] http:// struts.apache.org

The first step of rapid development is to prepare backend application (Typically using Database Management System) and their application logic code. The action mapping file and data class file are created automatically from the task level description described next subsection.

3.2 Task definition

Figure 7 shows an example description of the information assistant task. In this task setting, video contents which are divided into sections are presented to the user one by one. At the end of a section, a robot agent put in a word in order to help user's understanding and to measure the user's preference (e.g. by the recognition of acknowledging, nodding, etc.) . If low user's preference is observed, unimportant parts of the presentation are skipped and comments of the robot are adjusted to beginner's level. The importance of the section is indicated by interestLevel attribute and knowledgeLevel attribute that are introduced in the <userModel> element. If one of the values of these attribute is below the current value of the user model, the relevant section is skipped. The skipping mechanism using user model variables is automatically inserted into an interaction level description.

```
<taskml type="infoAssist">
  <head>
    <userModel>
      <interestLevel/>
      <knowledgeLevel/>
    </userModel>
    <deviceModel
        mainMode="speech" agentType="robot"/>
  </head>
  <body>
    <section id="001"
        s_time="00:00:00" e_time="00:00:50"
        intersetLevel="1"  knowledgeLevel="1">
      <video src="vtr1.avi" />
      <interaction name="interest1.mmi"
            s_time="00:00:30"/>
    </section>
    ...
  </body>
</taskml>
```

Figure. 7 An Example of Task Markup Language.

3.3 Describing Interaction

The connection between task-level and interaction-level is realized by generation of interaction description templates from the task level description. The interaction level description corresponds to the view part of the MVC model on which task level description is based. From this point of view, task level language specification gives higher level parameters over MVC framework which restricts behavior of the model for typical interactive application patterns. Therefore, from this pattern information, the skeletons of the view part of each typical pattern can be generated based on the device model information in task markup language.

For example, by the task level description shown in Figure 7, data class is generated from <userModel> element by mapping the field of the class to user model variable, and action mapping rule set is generated using the sequence information of <section> elements. The branch is realized by calling application logic which compares the attribute variables of the <section> and user model data class. Following action mapping rule, the interaction level description is generated for each <section> element. In information assistant class, a <section> element corresponds to two interaction level descriptions: the one is presenting contents which transform <video> element to the <output> elements and the other is interacting with user, such as shown in Figure 8.

The latter file is merely a skeleton. Therefore, the developer has to fill the system's prompt, specify user's input and add corresponding actions.

Figure 8 describes an interaction as follows: at the end of some segment, the agent asks the user whether the contents are interesting or not. The user can reply by speech or by nodding gesture. If the user's response is affirmative, the global variable of interest level in user model is incremented.

```
<mmvxml>
 <form>
  <field name=”question”>
   <input>
     <speech type=”boolean”/>
     <image type=”nod”/>
   </input>
   <output>
     <audio> Is it interesting? </audio>
   </output>
   <filled>
     <if cond=”question==true”>
       <assign name=”intersestLevel”
               expr=” intersestLevel+1”/>
     </if>
     <submit src=”http://localhost:8080/step2/>
   </filled>
  </field>
 </form>
</mmvxml>
```

Figure. 8 An Example of Interaction level Markup Language.

3.4 Adaptation to multiple interaction devices

The connection between interaction-level and platform-level is realized by binding mechanism of XML. XBL (XML Binding Language)[6] was originally defined for smart user interface description, extended for SVG afterwards, and furthermore, for general XML language. The concept of binding in XBL is a tree extension by inheriting the value of attributes to the sub tree (see Figure 9). As a result of this mechanism, the base language, in this the case interaction markup language, can keep its simplicity but does not loose flexibility.

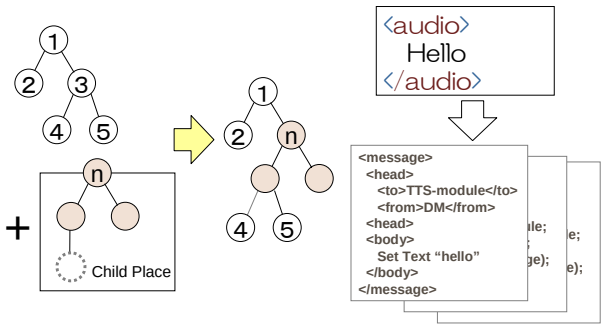

Figure. 9 Concept of XML binding.

By using this mechanism, we implemented various types of weather information system, such as Microsoft agent (Figure 10), Galatea (Figure 11) and a personal robot. The platform change is made only by modifying agentType attribute of <deviceModel> element of taskML.

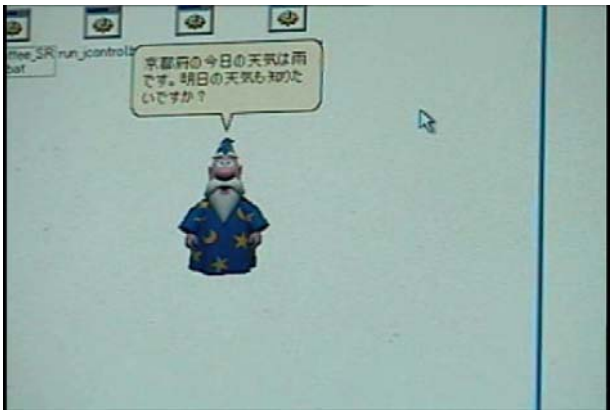

Figure. 10 Interaction with Microsoft agent.

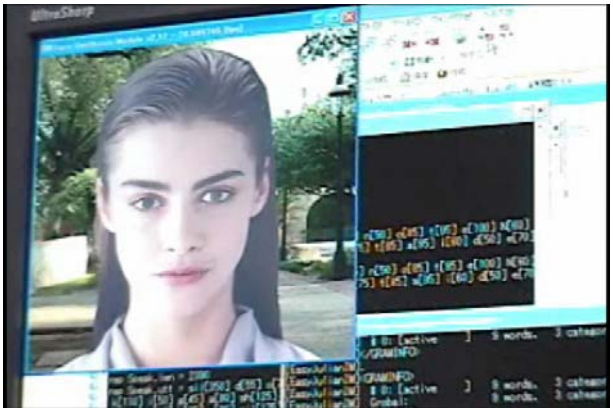

Figure. 11 Interaction with Galatea.

4 Comparison with existing multimodal language

There are several multimodal interaction systems, mainly in research level (López-Cózar and Araki 2005). XHTML+Voice[7] and SALT[8] are most popular multimodal interaction description languages. These two languages concentrate on how to add speech interaction on graphical Web pages by adding spoken dialogue description to (X)HTML codes. These are not suitable for a description of virtual agent interactions.

(Fernando D’Haro et al. 2005) proposes new multimodal languages for several layers. Their proposal is mainly on development environment which supports development steps but for language itself. In contrary to that, our proposal is a

[6] http://www.w3.org/TR/xbl/

[7] http://www-306.ibm.com/software/pervasive/multimodal/x%2Bv/11/spec.htm

[8] http://www.saltforum.org/

simplified language and framework that automate several steps for system development.

5 Conclusion and future works

In this paper, we explained a rapid development method of multimodal dialogue system using MIML. This language can be extended for more complex task settings, such as multi-scenario presentation and multiple-task agents. Although it is difficult to realize multi-scenario presentation by the proposed filtering method, it can be treated by extending filtering concept to discrete variable and enriching the data type of <userModel> variables. For example, if the value of <knowledgeLevel> variable in Figure 7 can take one of “expert”, “moderate” and “novice”, and each scenario in multi-scenario presentation is marked with these values, multi-scenario presentation can be realized by filtering with discrete variables. In case of multiple-task agents, we can implement such agents by adding one additional interaction description which guides to branch various tasks.

Acknowledgments

Authors would like to thank the members of ISTC/MMI markup language working group for their useful discussions.

References

M. Araki, K. Komatani, T. Hirata and S. Doshita. 1999. *A Dialogue Library for Task-oriented Spoken Dialogue Systems*, Proc. IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems, pp.1-7.

M. Araki, K. Ueda, M. Akita, T. Nishimoto and Y. Niimi. 2002. *Proposal of a Multimodal Dialogue Description Language*, In Proc. of PRICAI 02.

L. Fernando D’Haro et al. 2005. An advanced platform to speed up the design of multilingual dialog applications for multiple modalities, Speech Communication, *in Press*.

R. López-Cózar Delgado, M Araki. 2005. *Spoken, Multilingual and Multimodal Dialogue Systems: Development and Assessment*, Wiley.

K. Katsurada, Y. Nakamura, H. Yamada, T. Nitta. 2003. *XISL: A Language for Describing Multimodal Interaction Scenarios*, Proc. of ICMI'03, pp.281-284.

S. Kawamoto, H. Shimodaira, T. Nitta, T. Nishimoto, S. Nakamura, K. Itou, S. Morishima, T. Yotsukura, A. Kai, A. Lee, Y. Yamashita, T. Kobayashi, K. Tokuda, K. Hirose, N. Minematsu, A. Yamada, Y. Den, T. Utsuro and S. Sagayama. 2004. *Galatea: Open-Source Software for Developing Anthropomorphic Spoken Dialog Agents*, In Life-Like Characters. Tools, Affective Functions, and Applications. ed. H. Prendinger and M. Ishizuka, pp.187-212, Springer.