IMPLEMENTATION OF A MULTIMODAL DIALOG SYSTEM USING EXTENDED MARKUP LANGUAGES

Kuansan Wang

Speech Technology Group, Microsoft Research
One Microsoft Way
Redmond, Washington 98052, USA

ABSTRACT

In this paper, we describe an implementation of a plan-based multimodal dialog system using the extensible markup language (XML). The dialog manager receives semantic objects representing the user's utterance at the end of each user's turn. We define a semantic markup language (SML), based on XML, to describe these semantic objects. Following the principles of XML Schema, we define the schema of SML in another XML called semantic definition language (SDL). In addition to supporting many discourse and dialog features, SDL is also designed to represent the domain knowledge via the application schema and the hierarchy of the semantic objects. We show that, with a thoughtful design in SDL, SML can be expressive enough that the behavior of a dialog planner can be fully specified in the extensible stylesheet language (XSL), a standardized language with a logical programming model that is most popular for implementing intelligent systems.

1. INTRODUCTION

Despite the revolutionary progress in computing power, human computer interfaces remain esoteric. The interface design often aims at exposing the capabilities of applications, and the modern graphic user interface (GUI) metaphor can usually provide mechanisms to make the presentation obvious and more intuitive. However, more often than not, the users still have to undergo a learning phase and eventually adapt themselves to the interface. Recently, the notion of "natural interfaces" has received a lot of attention. Instead of making users adapt to the computers, natural interfaces strive to reverse the usage model by making the computers infer users' intentions and react intelligently. The benefits of natural interfaces are obvious, yet the technical challenges abound, among them, an intelligent multimodal dialog capability that integrates spoken language seamlessly with other modalities. This is a research topic of the Dr. Who project [9] which this work is part of.

It is widely established that plan-based approaches [1-3] provide a suitable framework for realizing a multimodal dialog interface. A plan-based system is equipped with sufficient intelligence to actively seek out missing information and resolve conflicting factoids. This is usually achieved through logical inference. In [1], we proposed an implementation of such a system in which the system components communicates with one another through events surrounding the semantic objects. Semantic objects are essentially an abstraction of speech acts and domain knowledge, and are designed to encapsulate the language models and dialog actions that govern their instantiation and behaviors. Events are generated and exchanged between the dialog manager and the domain expert when the semantic objects are created and evaluated. For the implementation in [1], we treated the dialog events as *synchronous* GUI events, a decision that appeared reasonable and natural when all the system components reside in a single GUI environment. The design highlighted our view that a seamlessly integrated GUI and speech interface should embrace the same human computer interaction model and, therefore, at the most fundamental level adopt similar design strategies. From a system point of view, however, handling events synchronously is not always the best implementation when computing resources are scattered across a network. This is especially true for the Internet where the quality of service often varies significantly and unexpectedly. It is thus highly desirable that dialog events can be generated, transmitted, and processed in an asynchronous manner. Equally important, however, is that these engineering considerations should maintain and enhance, rather than alter or compromise, the functionality underlying a powerful paradigm.

In this paper, we describe our efforts in extending the previous work to a distributed computing environment. In particular, we have found XSL-transformations (XSLT) [4] to be a suitable language for specifying the behavior of a plan-based dialog system. XSLT, a recent World Wide Web Consortium (W3C) standard, is a specialized XML intended for describing the rules of how one structured XML document can be transformed into another, say, in the hypertext markup language (HTML) or a text-to-speech (TTS) markup language for visual or aural rendering. Its core construct is a collection of predicate-action pairs: each predicate specifies a textual pattern in the source document, and the corresponding action will produce a text segment in the output whenever the pattern specified by the predicate is seen in the source document. The output segment is specified through a programmable, context-sensitive template. XSLT defines a rich set of logical controls for composing the templates. The basic programming paradigm bears close resemblance to a logical programming language, such as Prolog, which is designed to facilitate the incorporation of intelligent behaviors into a computer system. As a result, we have found XSLT possesses sufficient expressive power for implementing crucial dialog components ranging from defining dialog plans, realizing dialog strategies, and generating natural language, to manipulating prosodic markup for speech synthesis and creating dynamic HTML pages for multimodal applications.

It is also worth mentioning that XSLT is designed primarily for programming in a distributed computing environment, in which minimizing the number of components with temporal or lateral dependencies on one another is highly desirable. It is henceforth not surprising that the specification of an XSLT processor does

not include any persistent memory. The implication is that every document transformation starts anew, and the states of an XSLT program are reset once the source document is fully processed. The key to employing XSLT in a dialog system, therefore, is to embed enough contextual information into the source document so that the discourse semantics can be properly inferred and subsequently appropriate actions can be executed. A possible implementation is described in the following sections.

2. SYSTEM ARCHITECTURE

Our system architecture closely follows the analytical framework described in [5]. Essentially, the major system components are cascaded as schematized in Fig. 1. The raw signals generated by the user are first processed by a semantic parser, one for each modality, into a representation called the *surface semantics*. The surface semantics from all the input media are merged by the discourse manager, a component that has access to the dialog context and domain knowledge. The result is a reply message the system would like to communicate back to the user. The reply, called the *discourse semantics*, is the outcome of the system's best attempt to infer the user's intention. Finally, the response manager synthesizes the proper responses based on the discourse semantics and the capabilities of the user interface. In contrast to our previous work [1], the semantic evaluation and response generation processes, which were carried out by the dialog manager, are now accomplished in two components that are well insulated from each other.

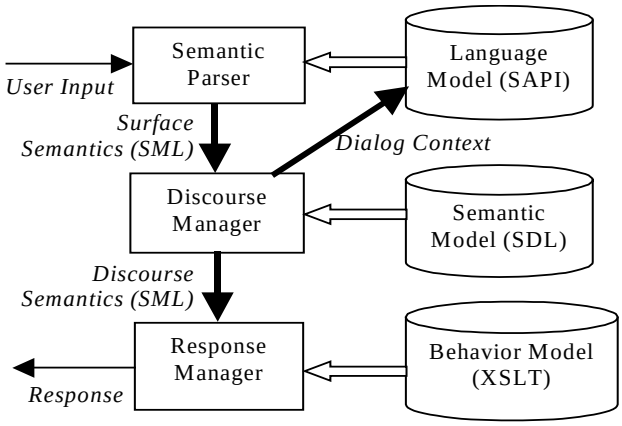

Fig. 1. System Block Diagram

The surface semantics consists of the semantic objects that represent meaning-bearing linguistic units ranging from words and phrases to dialog acts. The semantic parser produces surface semantics by searching for the semantic objects in the user's signals. The search is guided by a language model that dictates how lower linguistic units can be fused into forming a higher level one. For speech and text input devices, the language model is a unified model that integrates context-free grammars with a statistical N-gram [6]. For pointing devices, the language model is a simple lookup table mimicking natural language outputs for the mechanical signals. The language model can be synthesized dynamically by the semantic model based on the dialog context. It is often crafted in such a manner that ordinal anaphora (e.g., "the third one") and the deictic references are resolved at the surface level whenever possible (e.g., "I want two pizzas: one with mushrooms, the other with onions and green peppers."). In our implementation, the language model is authored in the XML format defined in [7].

The goal of the discourse processing is to understand the user's intention and take proper actions. In our system, this amounts to converting the surface semantics into domain entities and fulfilling the actions implied by the dialog acts. As a result, the semantic objects in the discourse semantics are dominated by the meaning-bearing units related to the application domain. This is in contrast to the surface semantics where the semantic objects mostly represent the linguistic units in the user's utterance. A domain entity can be derived only for non-ambiguous semantic objects. As a result, one way to tell how far a dialog state is from achieving the dialog goal is to count how many semantic objects representing the user's intentions are yet to be converted. To reduce dialog turns, we design the discourse manager to process the semantic objects for maximum conversions. As a side effect, the discourse manager can exhibit some intelligence by automatically taking care of "the obvious" such as inferring missing data from the dialog history. The process can be further assisted by a *semantic model* that defines the extent of automatic inference, within which the semantic integrity is safeguarded by the domain knowledge and common sense. In our system, both the surface and discourse semantics are represented in the semantic markup language, or SML, which we extended from XML. From a system point of view, the semantic model defines the rules to convert the surface SML into discourse SML. The semantic model, therefore, must define how to construct a legitimate SML document, i.e. the schema for SML. Instead of defining the schema statically, we follow the principles of XML Schema and design a semantic definition language (SDL), a specialized XML to specify the semantic model. Aside from defining the legitimate constructs of an SML document, SDL is primarily designed for specifying the relationships among the semantic objects and the semantic inference rules for the discourse manager (e.g., see [5]).

As in a typical plan-based system, the end of the semantic inference process implies a proper course of dialog actions, which, in this work, is described in the discourse semantics. To generate a proper response, the limitations and the strengths of the physical device must be taken into account and the generation strategy must be tailored to the user interface. For maximal expressive power, the response strategy is also defined in terms of logical inference, the rules for which are called the *behavior model*. The predicates of the behavior model refer to the domain entities and semantic objects in the discourse semantics, and the actions consist of instructions to operate the physical device. From a programmatic viewpoint, the behavior model consists of the rules to transform the structured document of the discourse semantics into another structured document of device operating instructions. As a result, the representation of a behavior model falls squarely into the intended use of XSLT.

3. SEMANTIC MARKUP LANGUAGE

The purpose of surface semantics is to transcribe and annotate the user's utterance in a meaningful way. From the discussion in the previous section, surface SML basically represents a semantic

parse of the user's utterance. The terminal and non-terminal nodes of the parse tree are defined in SDL by tags <verbatim> and <class>, respectively. They refer to the semantic objects and both have a "name" and a "type" attribute. The type attribute corresponds to the type of the entity the semantic object would be converted to, and plays a key role in semantic inheritance and polymorphism [1]. When a semantic object is unique in its type, SDL automatically assumes its type as the object's name. SDL also defines a <cfg> tag for the language model that governs the instantiation of a semantic object, and a <expert> tag for the system resource that is responsible for physically converting a semantic object into a domain entity. Finally, the tag <slot> in SDL defines the descendants of a non-terminal node.

As an example, consider the semantic model for the Microsoft employee directory application. The simple application answers queries on employees' data such as office location, phone number, hiring date, etc. The main speech act, the query, is modeled by the following semantic object:

```
<class type="DirectoryQuery" ...>
   <slot type="Person"/>
   <slot type="DirectoryItem"/>
   <expert clsid="..."/>
   <cfg ref="Directory.cfg"/>
</class>
<include ref="PeopleGrammar.sdl"/>
```

The semantic object is instantiated in accordance with the language model "Directory.cfg" and, once instantiated, is handled by a system object identified by its class id (clsid) that retrieves the data from the database. One can elect to embed the XML version of the query language (e.g., XQL) as the content of the <expert> tag. In the same manner the <cfg> tag can enclose the XML of the language model. The declaration of semantic objects can be nested and reused, as shown in the <include> tag in the above. The employee's directory items that one can ask are the semantic terminal modeled as

```
<verbatim type="DirectoryItem" ...>
  <prod name="office"/>
  <prod name="phone"/>
  <prod name="hiring date"/>
  ...
</verbatim>
```

The <prod> tags inside a terminal semantic object indicate the terminal is of an enumeration type, of which the possible values are defined by "name" attribute. This highlights the fact that the directory items are semantic terminals, but not linguistic terminals since they may have many equivalent expressions.

While we use static tags such as <class> and <verbatim> in SDL for the data model, in SML the instance of a semantic object assumes the object name as the tag name. For example, the surface SML for an utterance "*What is the phone number for Kuansan?*" is

```
<DirectoryQuery ...>
  <PersonByName type="Person" parse="kuansan">
    kuansan
  </PersonByName>
  <DirectoryItem type="DirectoryItem" parse="phone number">
    phone
  </DirectoryItem>
</DirectoryQuery>
```

The discourse manager would find that all three semantic objects, the person, the directory item, and the directory query itself are resolvable and produce the discourse SML

```
<DirectoryQuery ...>
  <Person id="kuansanw" parse="kuansan">
    <First>Kuansan</First>
    <Last>Wang</Last>
    ...
  </Person>
  <DirectoryItem parse="phone number">
    <phone>+1(425)703-8377</phone>
  </DirectoryItem>
</DirectoryQuery>
```

Note that the parse string from the user's original utterance is kept throughout the process so that the response manager can choose to phrase the response in the user's own wording. Rules that render the discourse SML into text can be programmed in XSLT as:

```
<xsl:template match="DirectoryQuery[@not(status)]">
    For <xsl:apply-templates select="Person"/>, the
    <xsl:apply-templates select="DirectoryItem"/>.
</xsl:template>
<xsl:template match="Person">
    <xsl:value-of select="First"/>
    <xsl:value-of select="Last"/>
</xsl:template>
<xsl:template match="DirectoryItem">
    <xsl:apply-templates/>
</xsl:template>
<xsl:template match="phone">
    phone number is <xsl:value-of/>
</xsl:template>
```

This leads to a response "For Kuansan Wang, the phone number is +1(425)703-8377." Advanced functions, such as prosodic manipulations, can be included in a straightforward manner. The above XSLT stylesheet can be slightly modified for rendering as a HTML table:

```
<xsl:template match="DirectoryQuery[@not(status)]">
    <TABLE border="1">
        <THEAD><TR>
            <TH>Properties</TH>
            <TH><xsl:apply-templates select="Person"/> </TH>
        </TR></THEAD>
        <TBODY><xsl:apply-templates select="DirectoryItem"/>
        </TBODY>
    </TABLE>
</xsl:template>
<xsl:template match="phone">
    <TR> <TD>phone</TD> <TD> <xsl:value-of /> </TD> </TR>
</xsl:template>
```

The discourse manager can insert an <error> tag with a status code ("scode") into the semantic objects when exceptions occur during semantic evaluation. For example, if the query is for a person named "*Derek*" that has 27 matches in the database, the discourse SML will look like

```
<DirectoryQuery status="TBD" focus="Person" …>
  <PersonByName type="Person" parse="Derek" status="TBD" …>
     <error scode="1" count="27"/>
     <Person id="derekba">
        <First>Derek</First>
        <Last>Baines</Last>
           …
      </Person>
      <Person id="dbevan">
         <First>Derek</First>
         <Last>Bevan</Last>
           …
      </Person>
           …
  </PersonByName>
  …
</DirectoryQuery>
```

Semantic objects that cannot be converted, such as DirectoryQuery and PersonByName in the above example, are flagged with a status "TBD". Discourse SML also marks the dialog focus, as in the DirectoryQuery, to indicate the places where the semantic evaluation is discontinued. These two cues assist the behavior model in choosing appropriate responses. The HTML stylesheet can simply present all the 27 possibilities on the display with hyperlinks and resolve the ambiguities in a single dialog turn, whereas the text-based stylesheet will need a more elaborated strategy resulting in several dialog turns. When the number of possibilities is small, a text-based stylesheet can generate a response that enumerates the alternatives, as in *"Do you mean (choice 1) or (choice 2)?"* However, when there are too many choices, the behavior model should guide the response manager to browse through discourse SML and generate a question that can efficiently resolve the ambiguities. The differences in dialog strategies can be fully accounted for and programmed into customized XSLT stylesheets, and changes to the behavior model have minimal impacts on the rest of the system.

4. DISCUSSION

In this paper, we describe an implementation of a plan-based dialog system using XML and XSLT. In addition to the logical inference power in XSLT, we believe the proposed system has the following desirable characteristics. First, the implementation separates actions from semantics. By doing so, it is easier to experiment with various dialog strategies with identical semantics. Most importantly, since the stylesheets can be dynamically attached to an SML document, it is possible to adapt dialog strategies whenever appropriate. This is particularly crucial for mixed initiative systems in which both the user and the computer can take the lead in the interaction. When the computer detects confusion or a serious digression, it must be able to guide the user by scaling back to a more system-initiated strategy. On the other hand, when the system interacts with an experienced user, it should lean towards a user-initiated mode that empowers the user and expedites the completion of the user's tasks.

As demonstrated in the above example where both visual and speech only rendering are considered, the encapsulation of dialog strategies in the form of dynamically swappable stylesheets also makes it simpler to accommodate a wide variety of access devices and allow users to switch modes on demand. For example, while presented with a list of choices on a speech only device, the user can walk near a computer with a suitable display and ask to continue the interaction on the screen. In the proposed system, this functionality can be provided by redirecting the semantic markup document and replacing the speech only style sheet with a visual one.

Since XSLT was designed to function in a GUI, implementing multimodal dialog is simplified by reusing the vast supports in the underlying system. This is especially the case as the semantic objects are designed to communicate with various system components in the same event mechanism for the GUI. The use of existing and popular standards also enables the designers to tap into the tools and reusable components already available. For example, our text to speech implementation takes advantage of the fact that the underlying XML/XSL processor has access to the speech interface exposed by the operating system. One can use the same mechanism to include other system functions, such as telephony supports if so desired. Tight integration with the underlying platform also enables advanced implementations of back-channel communication. One such example is the progress bar in the MiPad application [8], for which the user is shown the voice volume and the progress of the understanding process. This display fills the time lag between the user's utterance and any visible actions taken by the device. Without such a back-channel turn, the user can easily mistake a prolonged recognition process as a rejection, thus degrading the user's experience. This back-channel turn also serves as the most appropriate time for the user to cancel the operation, a very important feature from the viewpoint of user interface design. These properties can be effectively introduced by visually rendering the recording volume events from the audio object and the phrase hypothesis events from the recognizer. Since these events are only meaningful when captured synchronously, the response manager must have suitable access to the real-time supports on the physical device. A tight integration with the device is a logical and natural choice.

5. REFERENCES

[1] Wang K., "An event based dialog system," *Proc. ICSLP-98*, Sydney Australia, 1998.

[2] Cohen P.R., Perrault C.R., "Elements of a plan-based theory of speech acts," Cognitive Science, pp. 177-212, vol. 3, no. 3, 1979.

[3] Sadek M.D., "Dialogue acts are rational plans," *Proc. ESCA/ETRW Workshop on the structure of multimodal dialogue*, Maratea Italy, 1991.

[4] W3C recommendation, "XSL-Transformations version 1.0," http://www.w3.org/TR/xslt, November, 1999.

[5] Wang K., "A plan-based dialog system with probabilistic inferences", *Proc. ICSLP-2000*, Beijing China, 2000.

[6] Wang Y.-Y, Mahajan M., Huang X., "A unified context-free grammar (CFG) and N-gram model for spoken language processing," *Proc. ICASSP-2000*, Istanbul Turkey, 2000.

[7] Microsoft Speech Application Program Interface (SAPI) version 5.0, http://www.microsoft.com/speech.

[8] Huang X. *et al.*, "MiPad: a next generation PDA prototype," *Proc. ICSLP-2000*, Beijing China, 2000.

[9] http://www.research.microsoft.com/srg/drwho.asp