Natural Language Enabled Web Applications

Kuansan Wang
Speech Technology Group, Microsoft Research, Microsoft Corporation
One Microsoft Way, Redmond, WA 98052 USA
http://research.microsoft.com/stg

Abstract

This article describes the usage of XML in a multimodal dialog system based on the Web architecture. The motivation behind the work is based on the observation that Web based interactions can be viewed as dialog conducted in an iconic language. Since the Web architecture accommodates multiple input/output methods by abstracting device details, it seems a straightforward way to weave natural language (NL) into the Web is to convert NL semantics into objects consistent with the Web architecture. We demonstrate how one can employ the extensibility of XML to implement federated understanding in which multiple sources unbeknownst to one another may collaborate to resolve and fulfill user's commands. We also demonstrate how the separation of content and presentation principle of XML can be used to implement distributed dialog applications that are less susceptible to the diversity of Web accessing devices.

1 Introduction

The rapid adoption of the Web-based client server architecture has put the distributed computing into the central stage. As the Web applications become more sophisticated and the computers morph into many form factors that do not have a sizeable display and easy-to-use input devices, conventional user interface no longer seems to serve the needs. NL coupled with speech inputs has emerged as an ideal candidate that promises a consistent and natural user experience to interact with the computers. There is thus a strong demand for technologies that can seamlessly bring natural language technologies to the Web-based computing architecture. Many of them are based on XML, the extensible markup language, recommended by the World Wide Web Consortium (W3C).

From a high level point of view, XML is simply a collection of protocols for representing structured data in a text format that makes it straightforward to interchange XML documents on different computer systems. However, the strength of XML resides not only on its system independence, but also on the standardized extensibility. The idea of extensibility is to allow new markups, created as an embodiment of new pieces of technology, to be introduced on demand. To insure interoperability, however, any individual must follow the XML convention in extending the existing markups. As long as the standard is followed, the new added elements do not necessarily have to go through a standard body in order for them to be publicly useful. The extensibility indeed is probably the most important feature of XML, especially for use in rapidly advancing areas like natural language technologies.

Another key feature in XML's design goal is to facilitate the separation of content from presentation. XML embraces a declarative syntax that is most suitable for annotating structured data. The data then can be transformed into appropriate formats either procedurally through XML document object model (DOM) (W3C, 2001), or declaratively in XSL or XSLT (W3C, 2001). Separation of content and presentation is especially critical in the Web environment because diverse presentation formats are needed to accommodate various kinds of access devices, ranging from conventional telephones with limited display and input capabilities to personal computers with sophisticated peripherals.

There have been efforts trying to use XML for natural language applications, most notably the VoiceXML specification pioneered by the VoiceXML Forum (2000). VoiceXML aims at providing a simple telephone-based dialog framework based on a finite state machine. VoiceXML models dialog using a form-filling metaphor, namely, a dialog goal is modeled as a form with dialog sub-goals as fields on the form. VoiceXML also defines a form interpretation algorithm that traverses the fields in the order they are defined on the form. The default flow can be modified using procedural flow control tags such as calling into a subroutine and conditional or unconditional jump statements. There are also a set of markups for telephony call controls such as hanging up and call transfer.

Although VoiceXML incorporates XML as part of its name, VoiceXML is really not an XML application other than using XML syntax for the language. Most notably, VoiceXML adopts a procedural programming paradigm in describing the dialog flow. Since much of the XML's extensibility lies within the declarative nature of the language, it is a challenging task to extend VoiceXML at will as envisioned by the XML design. Also, as it is designed for telephony applications that do not use textual or graphical display in mind, VoiceXML can afford not to consider the issues of separating content from presentation. In fact, the 'content' of a dialog, i.e. the dialog goals and sub-goals, is intimately intertwined with its presentation. To facilitate more advanced interaction, fields often must be created *ad hoc* during design time just for dialog flow manipulation, even though these fields have little to do with the task knowledge or semantics at all. Consequently, one may regard VoiceXML as not following the spirits of XML in terms of the extensibility and flexible presentation.

In this paper, we describe a multimodal dialog system that incorporates the strength of XML and other Web standards to implement the plan-based framework (Sadek 1997, Allen 1995, Cohen 1990) known to be more suitable for dialog purposes. We elaborate how the dynamic schema principle underscoring the extensibility of XML is used for distributed NL understanding, and demonstrate how the separation of content and presentation principle can be used to facilitate distributed dialog system that can accommodate a multitude of diverse accessing devices that pose drastic differences in UI considerations.

2 Dynamic Schema Principle

In order for XML to be freely extensible by anyone, individual XML documents must be able to self-describe the structure of the document. A specification, called XML schema, describes what XML elements and attributes are expected in the document. To avoid name conflicts, one usually designates a namespace to the extended elements and attributes meaningful to a specific domain. Multiple namespaces coexisting in a single XML document are a common scenario that serves to highlight the extensibility of XML.

The self-describing nature of XML inspires a powerful idea known as the dynamic schema design principle for Web services. It addresses the problem of versioning and updating XML schemas published by Web services. Consider, for example, the services of a credit bureau that provides credit reports to financial institutes. A conventional way to structure the Web service is for the credit bureau to publish the XML schema for its customers, as illustrated in Fig. 1(a). A problem for this setup is that, whenever the credit bureau changes or adds new features to its services, all its customers are impacted by the changes in the XML schema. The design becomes problematic for the Web because many services have little control on their content consumers. In lieu of a statically posted schema,

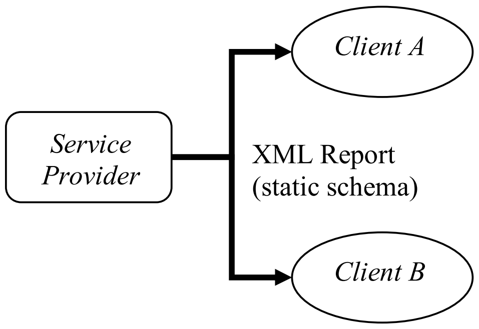

Figure 1(a) Web service using static schema: All clients receive documents in the same schema.

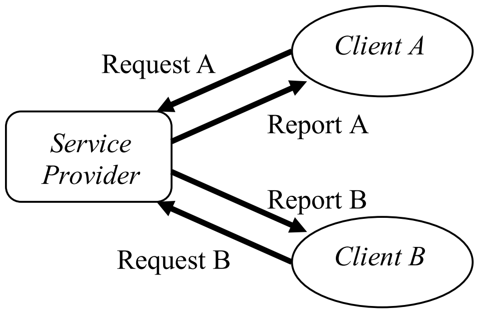

Figure 1(b) Web service using dynamic schema: each client receives a document formatted to the request.

the credit bureau can publish a protocol, called meta-schema, for its clients to specify the XML schema of the desired outcome on a request by request basis, as illustrated in Fig. 1(b). Since the clients always get the exact format they are asking for, the bureau can accommodate a very diverse clientele with various needs, and can freely add functionalities into meta-schema without concerns of adversely affecting its clients unknowingly. On the other hand, the clients have the liberty to experiment with new features at their own pace and choose the data formats that best serve their needs, rather than being dictated to one by their service provider.

Two components in our system employ the dynamic schema principle: speech recognition and parsing that analyzes surface semantics at the utterance level, and the discourse manager that extracts semantics at the discourse level. In both cases, we use XML as a schema definition language for meta-schema specification.

2.1 Surface Semantic Analysis

In our system, speech recognition is packaged as a Web service component that processes speech waveform. There are two types of recognition services: a dictation service transcribes speech into the word string, while a spoken language understanding (SLU) service further parses the utterance into a semantic tree. The SLU service can also receive text as input for the purpose of semantic parsing.

The semantic tree is represented in XML in a schema dynamically specified in Microsoft SAPI text grammar format (STFG), which is an XML application (Microsoft, 1999). To facilitate tight integration between speech recognition and surface semantic parsing, SLU service extends probabilistic context free grammar (PCFG) used in speech recognition to semantic parsing. For SLU, STFG has two major missions: specifying PCFG rules and output construction rules. The former governs how the input text stream can be parsed, and the letter how the resultant semantic XML can be constructed per a given schema.

2.1.1 PCFG rule specification

Using the XML syntax, each PCFG rule is declared in STGF using a <rule> element with a "name" attribute specifying the LHS. The rule element is composed of mandatory and optional phrase elements, <p> (or <phrase>) and <o> (or <option>) tags, respectively, representing the RHS of the rule. Alternative RHS are contained in a list element, <list> (or <l>). Non-terminals are referred to using a <ruleref> element. For example, the rule for the properties of a meeting can be specified as

```
<rule name="MeetingProperties"/>
 <l>
  <ruleref name="Date"/>
  <ruleref name="Duration"/>
  <ruleref name="Time"/>
  <ruleref name="Person"/>
  <ruleref name="Subject"/>
.. ..
 </l>
 <o>
  <ruleref
    name="MeetingProperties"/>
 </o>
</rule>
```

With recursion, the above STGF rule can resolve meeting properties, such as date, time, duration, attendees of a meeting that may appear in any order in the utterance. The probability of each production can be specified by attaching a "weight" attribute to the corresponding RHS. STFG also provides two additional pre-terminal declarations to simplify the implementation of robust parsing: a <wildcard/> element (or "…") matches any filler or garbage word, and a <dictation/> element (or "*") that matches any word from the lexicon implied by a pre-specified statistical N-gram. For example, the meeting subject, drawing text from an N-gram, can be specified as

```
<rule name="Subject">
 <l>
   <p> regarding </p>
   <p> ?with ?the subject </p>
 </l>
 <dictation max="inf"/>
</rule>
```

with "max" attribute indicating the maximum number of words to be drawn from the N-gram. Note that, although the example is geared towards semantic parsing, the formalism is rich enough to include syntactical rules for text processing. Currently, STGF is evolving more along the line of a context free grammar, and has no immediate plan to natively support unification grammar.

2.1.2 Simple Semantic Tagging

Quite often, the semantics of a phrase manifests itself as simple text normalization. STGF utilizes two XML attributes, “propname” and “valstr”, for simple semantic tagging. When a production is activated, STGF produces an XML element node whose name and value are taken from the propname and valstr, respectively. For example, a production

```
  <l propname="DayOfWeek">
   <p valstr="Sun"> Sunday </p>
   <p valstr="Mon"> Monday </p>
   <p valstr="Mon"> first day </p>
   .. .. ..
   <p valstr="Sat"> Saturday </p>
  </l>
```

generates an XML element

```
 <DayOfWeek
  text="first day">Mon</DayOfWeek>
```

when the user says “first day”. The mechanism is designed to normalize the value of the XML node, and hence is most useful to represent factoid phrases or pre-terminal lexical entries across multiple languages. STGF automatically retains the exact wordings from the input in the “text” attribute as shown above.

2.1.3 Generic Semantic XML generation

More advanced semantic representation needs capability beyond simple semantic tagging. For example, semantic tagging always produces outcome as the value of an XML node. What if the schema requires it to be an attribute? To be useful for any XML schema, one clearly needs a more general transformation of the PCFG parse into the semantic tree. This is particularly true as a good semantic representation should be able to normalize as much as possible the linguistic variations PCFG production rules are designed to accommodate for user utterance.

STGF uses the general tree transformation specification defined in W3C XSLT (2001). XSLT adopts a template based transformation mechanism. When an XSLT template matches the input pattern, its content is produced in the output. If the template content contains further XSLT directives, they are executed recursively. The mechanism can be applied to surface semantic analysis in a straightforward manner by treating PCFG parsing as a pattern matching process. Accordingly, each PCFG rule can have an XSLT template that specifies the appropriate output when the rule is activated. Using XSLT is appealing because it is already a well known and widely adopted standard with strong commercial backing and supports. STGF introduces an <output> element to host the XML generation template for each rule. For example, consider the following grammar rule for specifying a meeting:

```
<rule name="Meeting">
 <o> <ruleref
      name="StartingPhrases"/>
 </o>
 <p> <ruleref
      name="MeetingProperties"/>
 </p>
 <o> <ruleref name="EndingPhrases"/>
 </o>
 <output>
   <calendar:meeting>
    <DateTime>
      <xsl:apply-templates
       name="//Date"/>
      <xsl:apply-templates
       name="//Time"/>
      <xsl:apply-templates
       name="//Duration"/>
    </DateTime>
    <xsl:apply-templates
      name="//Person"/>
      .. ..
   </calendar:meeting>
 </output>
</rule>
```

In this example, when the utterance invokes the “Meeting” rule, the template contains in the <output> element will be generated and sent to the output stream. In the case, an XML element with namespace “calendar” and name “meeting” is first produced. The first child of this element is called “DateTime”, whose contents are generated in the specified order from the templates “Date”, “Time”, and “Duration,” all of which, in turn, are defined by some constituents the “Meeting” rule. The notation “//” means the template can come from any child node descended from the current node. In this example, they are from constituents of the “MeetingProperties.” Per XML generation standard defined in XSLT, the template specified by the <output> element of the “Date” rule will take the place of <xsl:apply-template select=”//Date”/> in the final outcome, and the same for the “Time” and others. As shown previously, the production rules of the meeting property grammar aim to give users maximal flexibility in describing meeting properties. Through the use of the <output> element, the semantic XML does not have to bear the complexity induced by the flexibility. The above example demonstrates that, regardless how a user might say it, the semantic XML will have the schema where the meeting properties always occur in a particular order. For instance, the semantic XML will always be

```
<calendar:meeting text="…">
 <DateTime text="…">
  <Date text="…">tomorrow</Date>
  <Time text="…">2:00</Time>
  <Duration text="…">3600</Duration>
 </DateTime>
 <Person>Kuansan Wang</Person>
</calendar:meeting>
```

for utterances "*Schedule a meeting for one hour with Kuansan Wang tomorrow at two o'clock*", or "*Invite Kuansan Wang to a one hour meeting at two o'clock tomorrow*", etc. The normalization is achieved here through the order in which the XLST apply-template directives are declared. This is the same idea widely adopted in general XML document transformation.

As suggested by the name, XSLT is intended to be extensible. Early versions of the working drafts described a method in which the XSLT processing can be extended using procedural programs. Although not included in the current W3C recommendation, many vendors have included the scripting supports in the commercial products. For example, Microsoft's XSLT, as part of Internet Explorer or freely downloadable from the Web, allows an XLST template to use an "eval" element to invoke a program enclosed by a "script" element. The mechanism is used here, for example, to carry out numerical computations and inverse text normalization for numbers, dates, times, and currencies, as in "2:00" for "two o'clock" and "3600" (seconds) for "one hour" above.

We consider the XML usage in STGF abides by the dynamic schema principle because the output schema for the recognition/parsing service is entirely up to the client, not the service provider. Suppose the calendar application from another vendor requires a schema that requires the start time and end time of the meeting instead of the start time and duration as demonstrated above. The semantic XML can be composed and generated by equipping the output template in the "Meeting" rule with the appropriate script that computes the end time from the start time and duration.

More advanced processing not defined by the standard XSLT can be implemented in scripts because the DOM is fully accessible inside the scripts. However, since not all platforms support the same set of scripting functions, and not all script interpreters behave identically on the same statements, heavy usage of scripting is usually not advisable.

2.2 Federated Understanding

The extensibility of XML, together with dynamic schema design principle, is a suitable vehicle for implementing federated understanding, in which multiple knowledge sources are joined together to collaborate on a user request. The knowledge sources are usually Web services themselves and may be unrelated and otherwise unbeknownst to each other. For the context of this article, we call the service that combines all the knowledge sources a context manager. A context manager is a personal service that can reside on end user's Web access device or be a Web service itself. The concept is illustrated in Figure 2.

Since the idea of federated understanding is to dynamically combine relevant Web services to fulfill a user task, NL plays a critical role in the user interface design because NL provides an unsurpassed expressive power in describing a task. Consider a usage scenario where the user says "Send driving directions to the dinner guests tomorrow night." Upon understanding the user's sentence, the context manager can

- Invoke the user's calendar service, obtaining information on the dinner event for the following night, including the dinner location and invited guests.
- Cross check the guest list with the user's contact list, finding out the proper way of sending a document to each guest.
- Invoke a geo-information service for

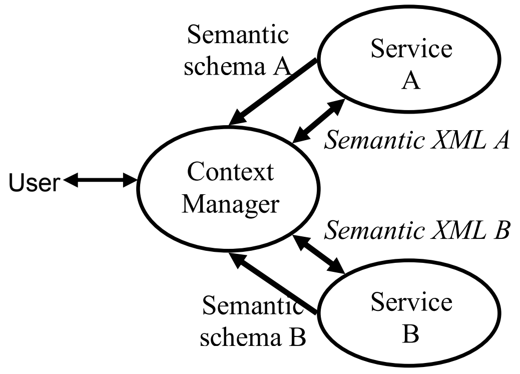

Figure 2: Federated Understanding. User's utterance is parsed into partial semantic XML and sent to related Web services based on their semantic schemas. Service A and B do not have to be designed to work with each other.

computing driving directions to the dinner location.

- Send the driving direction document to each invited guest using the messaging service the user has subscribed to.

Clearly, a user can fulfill all the above steps in a GUI environment, but probably not before wading through several pages, windows or menus with numerous mouse clicks, a laborious, mechanical, and potentially error prone process. However, we surmise that the value of SLU is not to replace GUI but to facilitate federated understanding that provides a very convenient and productive means for the users to accomplish their tasks.

2.2.1 Semantic schema definition

We follow the dynamic schema principle in the implementation of federated understanding. The context manager publishes a schema definition protocol in XML, called semantic definition language (SDL), for the Web services to register the schemas of their services (Wang, 2000a). The collective SDL documents form the basis of the data model underlying the semantic XML (SML) for the user utterance. Each domain defines its own namespace, and hence SML segments from various domains can be lumped together as appropriate without conflicts. As an example, Sec. 2.1.3 shows an instance of SML for the calendar Web service in which the top node corresponding to a new meeting service is included in the “calendar” namespace.

In our system, semantics is represented by basic meaning-bearing units called the *semantic objects*. The role of a semantic schema is to define how semantic objects are related to each other, and how compound semantic objects can be composed of from simple semantic objects. This role is the same as an XML schema that describes the structure of an XML document. As a result, it seems straightforward to represent semantics in XML in which each semantic object is simply a node in the SML so that most of the conventions and standard practices for XML schema can be directly applied to semantic schema as well.

There are considerations, however, needed to be addressed in applying XML for NL purposes, one of which is the “fluid” nature of NL in contrast to the precise machine-to-machine communications for which XML is designed for. For instance, while the ISO-8601 format is adequate to represent date/time for machines, it is more desirable for a NL semantic representation to allow expressions such as “a week before the paper is due” or “two hours before the project review meeting.” The observation that an entity may be referred to via many semantic objects leads us to extend the primitive data types in XML to a more elaborated semantic types for the semantic representation.

In SDL, each semantic object is associated with the type of entity it is referring to. In turns, the composition of a compound semantic object can be based not only on semantic objects but also on semantic types alone. Semantic types have a hierarchy that supports inheritance among types. When a constituent is declared only with a semantic type, any object of a compatible type can be used to instantiate the constituent. In other words, the type system brings *polymorphism* into the semantic representation. Consider, for example, the semantic schema for sending new email:

```
<command type=”email:command”
    name=”email:send”>
  <slot type=”Person”
    maxOccurence=”Infinite”/>
  <slot type=”email:Subject”
    maxOccurence=”1”/>
  <slot type=”email:Body”
    maxOccurence=”1”/>
  <expert server=
    ”http://PIM/email.dll”/>
</command>
```

The semantic object declares a constituent (using the “slot” subelement) of type “Person” for mail recipient. In addition to the standard way of identifying a person by surname and given name, one might want to allow the user to specify mail recipients in a more elaborated way, for example, “send mail to *those receiving the meeting report last Tuesday*.” The semantic object for the mail recipients here are identified through another email message, which can be modeled as

```
<entity type=”Person”
    name=”email:recipient”>
  <slot type=”email:Subject”/>
  <slot type=”DateTime”/>
</entity>
```

The two schemas dictate that the SML for the above sentence should be

```
<email:command text=”send mail…”
   name=”email:send”>
  <Person text=”those receiving…”
      name=”email:recipient”>
```

```
    <email:Subject text=”meeting
      report”/>
    <DateTime text=”last Tuesday”>
      <Month>September</Month>
      <Date>25</Date>
      <Year>2001</Year>
    </DateTime>
  </Person>
</email:command>
```

Polymorphism allows objects from other services to be integrated in a straightforward manner. The calendar service, for example, can expose the attendees of a meeting as a semantic object of type “Person”:

```
<entity type=”Person”
    name=”calendar:attendees”>
  <slot type=”calendar:meeting”/>
</entity>

<entity type=”calendar:meeting”>
  <slot type=”DateTime”
    name=”calendar:start”/>
  <slot type=”DateTime”
    name=”calendar:end”/>
  <slot type=”calendar:subject”/>
            …
</entity>
```

The context manager can merge this schema with the email schema to accommodate the sentence that crosses the boundary between these two domains, e.g., “send email to those in the design review meeting last Friday.” Though from independent domains, the two schemas jointly define a valid SML should be

```
<email:command name=”email:send”>
  <Person name=”calendar:attendees”>
    <calendar:meeting>
       <DateTime text=”last Friday”
         name=”calendar:start”>
        <Month>…</Month>
              …
       </DateTime>
       <calendar:subject text=”SLU
         design”/>
   </calendar:meeting>
  </Person>
</email:command>
```

2.2.2 Distributed Execution

The Web service architecture provides a suitable infrastructure to dissect the task encompassing multiple domains into executable components that can be distributed to their respective owners. We utilize this infrastructure to facilitate domain collaboration. Each domain that is responsible for evaluating a semantic object is required to make itself available as a Web service using Web Service Description Language (WSDL) being standardized in W3C (2001).

The context manager follows the declaration in the semantic schema to invoke the proper Web services to evaluate the segments of SML. In SDL, domain Web services are declared using the “expert” element that specifies the URL of the Web service, as the email example shown above. The SML segment is replaced by the context manager with the result returned by the domain Web service. In the example above, for instance, the context manager invokes the calendar Web service to evaluate the Person SML node, for which the calendar Web service returns an XML segment representing the referred attendees. This XML segment replaces the original Person node and is passed on to the email Web services for execution.

The XML document after semantic evaluation is called a discourse SML, in contrast to the surface SML describing a user’s utterance. A discourse SML typically describes the results of the evaluation, including errors and exceptions. Since the domain Web service always returns the result of the same semantic type, discourse SML has the same schema as the surface SML defined in SDL. In addition to maintaining discourse semantic XML and coordinating the semantic evaluation among relevant domain Web services, the context manager also manages an entity memory and implements a reference resolution algorithm. The details are further described in (Wang, 2000b).

3 Distributed Dialog Management

As typical in a plan-based dialog framework, proper dialog actions are naturally implied in the semantic evaluation results, which in our case, are represented by the discourse SML. In other words, discourse SML can be viewed as the “content” of a dialog for which a “presentation” shall be generated to illicit user actions. It is desirable to be able to dynamically adapt the presentation without having to duplicate all the components of an application. Again, XML is suitable because the separation of content from presentation is a primary design goal for XML.

The principle can be further elaborated in the following two aspects. On the input side, the user interface is responsible for capturing user’s intention in surface SML. As GUI actions can also generated semantic objects, NL inputs may be integrated with GUI inputs through semantic polymorphism to facilitate multimodal dialog and shield the rest of the system from the device details. While UI resides on the browser, the rest of the system, including the context manager, is located at the Web server to carry out federated understanding and distributed execution as described above.

On the generation side, hiding the device details is crucial for adapting the application to different interaction styles. For example, devices with a sizeable display can accommodate many dialog actions in one turn because several questions can be asked and rich contents can be presented to the user simultaneously. Clearly, more dialog turns are needed if the same interaction takes place on a less capable device. In addition to device variation, changes in interaction styles may also be required due to the dialog progress or lack thereof. For instance, it is always advisable to switch from a user initiative to a system initiative dialog when little progress is being made. However, when the system encounters an experienced user, a more user initiative dialog is often desirable.

Based on the above discussion, we adopt a dialog management approach in which the dialog is decomposed into sub-dialogs embodied in Web pages. Under the page-based proposal, a page is responsible for generating and managing dialog actions to fulfill a specific sub-goal. The server logic inspects the discourse SML and determines which subgoal to achieve, or equivalently, which page to be furnished to the user. Once the flow control is transferred to a page, the control stays within the page until a digression occurs or the subgoal is achieved. There may be multiple pages designed to achieve the same objective, each has its own with-in page logic. The within-page logic may implement different interaction styles (e.g. system vs. mixed-initiative), or are optimized for different access devices (e.g. telephone vs. hand-held computer). The delineation of the overall and the with-in page logic constitutes the foundation of the page-based distributed dialog system. Note that, since domains unbeknownst to each other can be teamed up dynamically, the pages are usually well encapsulated within their domain boundary, manifesting themselves as "reusable" dialog components.

4 Summary

In this paper, we describe our implementation of a plan-based multimodal dialog system using the Web architecture and XML. Core Web design principles play a critical role in the system. The dynamic extensibility of XML and Web service architecture give rise to a NL understanding framework where multiple knowledge domains can be pooled together dynamically to fulfill a user's request. XML's separation of content and presentation principle enables dialog interactions to take place on diverse Web access devices, including NL enabled or GUI media. As these technologies mature, we believe NL is ready for Web adoption and usher in an era of natural user interface.

References

World Wide Web Consortium (W3C), 2001. http://www.w3c.org.

VoiceXML Forum, 2000. VoiceXML Specification, http://www.voicexml.org.

Sadek M.D., Bretier P., and Panaget F., 1997. "ARTIMIS: Natural Dialogue Meets Rational Agency," *Proc. IJCAI-97*.

Allen J.F., 1995. *Natural Language Understanding*, 2nd Ed., Benjamin-Cummings, Redwood City CA.

Cohen P.R., Morgan J., and Pollack M.E., 1990. *Intentions in Communications*, MIT Press, Cambridge MA.

Bradshaw J.M. (Ed.), 1996. *Software Agents*, AAAI/MIT Press, Cambridge MA.

Microsoft, 1999. Microsoft Speech Application Interface (SAPI) Software Developers Kit (SDK), http://www.microsoft.com/speech.

Wang K., 2000a. "Implementation of a multimodal dialog system using extended markup language,"in *Proc. ICSLP-2000*, Beijing, China.

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