Compositional Specification and Realization of Mixed-Initiative Web Dialogs

Naren Ramakrishnan and Manuel A. Pérez-Quiñones[†]
Department of Computer Science
Virginia Tech, VA 24061, USA
naren@cs.vt.edu, perez@cs.vt.edu

Atul Shenoy
Microsoft Corporation
Redmond, WA 98052, USA
ashenoy@microsoft.com

Abstract

We present DialogXML – a markup language approach to specifying and realizing mixed-initiative web dialogs on mobile devices. By capturing the functional structure of the dialog independent of the modalities used to realize it, DialogXML facilitates the implementation of web interfaces that integrate hyperlink and speech modes of interaction. It enables the creation of websites that adapt to the needs of users yet permits the designer fine-grained control over what interactions to support. The framework uses an algorithm based on staging transformations – an approach that represents dialogs by programs and uses program transformations to simplify them based on user input. Design methodology, implementation details, and two case studies are presented.

Keywords: Multimodal interfaces, DialogXML, web interaction on mobile devices, dialog processing engines, mixed-initiative interaction.

[†]**Corresponding Author:** Manuel A. Pérez-Quiñones (perez@cs.vt.edu). Tel: 540-231-2646. Fax: 540-231-6075.

1 Introduction

Computing power today is increasingly moving away from the desktop computer to mobile computing devices such as PDAs, tablet PCs, and 3G phones. While posing capacity limitations (e.g., screen real estate, memory), such devices also present possibilities for multimodal interaction via gestures, speech, and handwriting recognition. One area that is witnessing tremendous growth in multimodal interaction is web browsing on mobile devices.

Many factors have contributed to the growing interest in multimodal web interaction. Chief among them is the maturing of commercial speech recognition engines [42] and technologies such as SALT (Speech Application Language Tags) and X+V (XHTML plus Voice) that have ushered in web documents that can talk and listen rather than passively display content. Second, speech permits natural ways to perform certain types of tasks and help compensate for deficiencies in traditional hyperlink access (which can get cumbersome on small form-factor devices). More importantly, speech-enabled websites help improve accessibility for the more than 40 million visually impaired people in the world today. As a result, using speech leads to the possibility of a conversational user interface [25] that combines the expressive freedom of voice backed by the information bandwidth of a traditional browser.

Our goal in this paper is to present DialogXML – a markup language to specify and realize expressive web dialogs on mobile devices (We use the word 'dialog' here to mean the turn taking that occurs independent of the modality of interaction; therefore 'dialog' should not be considered synonymous with 'voice'). We focus on database-driven websites (i.e., sites where interaction is derived from a relational navigation schema) although the basic elements of the approach can be adapted to less structured forms of websites. As the name indicates, DialogXML allows a user interface designer to specify the underlying dialog of human-website interaction independent of the modalities used to realize it. Furthermore, DialogXML supports mixed-initiative dialogs without anticipating the points at which the shift of initiative can happen. This enables us to think of multimodal interaction not just as a way to navigate existing site structure via voice [15] but as a way to support a flexibility of information access not possible with clicking hyperlinks alone.

The rest of this paper is organized as follows. In Section 2, we present our view of multimodal interaction as a flexible mixed-initiative dialog between the user and the website. We carefully define the type of mixed-initiative interaction studied here and the challenges in realizing it in a software framework. Section 3 covers background work in dialog systems, paying specific attention to dialog processing algorithms and markup languages. Our specific approach is given in Section 4 which first defines an intermediate representation for dialogs and then details the DialogXML markup language that is designed around this intermediate representation. Algorithms to simply dialogs based on user input and implementation details are also provided here. Section 5 details evaluation studies and Section 6 provides a summary discussion.

2 Motivating Example

Consider the following dialogs between an information seeker (Sallie) and an automated political information system.

Dialog 1

1 **System:** Welcome. Are you looking for a Representative or a Senator?
2 **Sallie:** Senator.
3 **System:** Democrat or Republican or an Independent?
4 **Sallie:** Republican.
5 **System:** What State?
6 **Sallie:** Minnesota.
7 **System:** That would be Norm Coleman. First elected in 2002, Coleman ...
(conversation continues)

Dialog 2

1 **System:** Welcome. Are you looking for a Representative or a Senator?
2 **Sallie:** Senator.
3 **System:** Democrat or Republican or an Independent?
4 **Sallie:** Not sure, but represents the state of Indiana.
5 **System:** Well, then it is either a Democrat or a Republican, there is no Independent from Indiana.
6 **Sallie:** I see. Who is the Republican Senator?
7 **System:** That would be Richard G. Lugar. First elected in 1976, Lugar ...
(conversation continues)

It is helpful to contrast these dialogs from a conversational initiative standpoint. In the first dialog, Sallie responds to the questions in the order they are posed by the system. Such a dialog is called a *system-initiated* dialog as the initiative resides with the system at all times. The second dialog is system-initiated till Line 4, where Sallie's input becomes unresponsive and provides some information that was not solicited. We say that Sallie has taken the initiative of conversation from the system. Nevertheless, the conversation is not stalled, the system registers that Sallie answered a different question than was asked, and refocuses the dialog in Line 5 to the issue of party (this time, narrowing down the available options from three to two). Sallie now responds to the initiative and the dialog progresses to complete the specification of a political official. Such a conversation where the two parties exchange initiative is called a *mixed-initiative interaction* [4].

What would be required to support such a flexibility of interaction at a website? It is clear that system-initiated modes of interaction are easiest to support and are the most prevalent in web browsing today. For instance, a webpage displaying a choice of hyperlinks presents such a view, so that clicking on a hyperlink corresponds to Sallie responding to the initiative. The reader can verify that the first dialog above can be supported by a three-level tree-structured HTML site presenting options for branch of congress, party, and state. But how can we support the second dialog, allowing Sallie to take the initiative at a website? This is where speech comes in. If Sallie can talk into the browser, she can provide unsolicited information using voice when she is unable to make a choice among the presented hyperlinks. In addition, if the system can process such an out-of-turn input, it can continue the progression of dialog and tailor future webpages so that they accurately reflect the information gathered over the course of the interaction. We have designed many such multimodal web interfaces, one example is shown in Fig. 1 and corresponds to *Dialog 2* above.

Flexibility of turn management, i.e., out-of-turn interaction, as demonstrated above is only a simple facet of mixed-initiative interaction. More sophisticated forms of mixing initiative might involve *intention recognition*, where the system aims to understand the intent behind the user's input, not the specifics (so that alternative options can be pursued), *planning*, where the system breaks down a complex task across a sequence of smaller subdialogs that have to be completed in turn, and *negotiation*, where the system and user negotiate about dialog continuation options.

Mixed-initiative interaction is easily realizable in multimodal contexts, where the user is given different input paradigms for interaction. For instance, in the mobile web browsing context, users have a choice of hyperlink versus speech input. While hyperlink access must, by definition, be responsive to a current solicitation, speech input can either provide responsive or unsolicited information. It is this potentially unsolicited nature of vocal input that effectively helps us realize mixed-initiative interaction.

Multimodal interfaces have been traditionally studied for their role in resolving ambiguity and reducing recognition errors [30] but we think of them here as ways to support the mixing of initiative. The key requirement to support the taking of initiative (by the user) is to allow the specification of unsolicited information.

A conversational engine supporting structured (hyperlink) as well as unstructured (speech) input of all forms is an ambitious undertaking and beyond the scope of this paper. Here, we restrict our attention to only cases where speech is used to specify information that would have normally been solicited further in the dialog (e.g., in *Dialog 2*

(a) Sallie clicks on ‘Senator’.

(b) Sallie says ‘Indiana’.

(c) Sallie clicks on ‘Republican’.

(d) The dialog is complete.

Figure 1: A mixed-initiative interaction with a multimodal web interface. Note that after step (b), ‘Independents’ is removed as a possible choice, indicating that there are no Independent politicians in Indiana.

above, Sallie supplies information about state in Line 4, whereas it is normally specified after party). We argue that even this simple 'out-of-turn interaction' capability helps support powerful interaction scenarios. We illustrate how out-of-turn speech input can seamlessly co-exist with a directed form of interaction such as hyperlink access.

2.1 Out-of-turn Interaction

We define out-of-turn interaction as any interaction where the user supplies input not requested at the current state of the dialog (or web page) but which would normally be requested at a later point in the dialog.

Why is out-of-turn interaction relevant?

Contextual information access and delivery is critical to the user experience, and has typically been studied along dimensions such as location awareness (e.g., situating information according to geographical location), user profiles (e.g., remembering personal details), and personalization (e.g., recommending content based on prior behavior). Out-of-turn interaction is another approach to demonstrating context, because it exposes dependencies and relationships in a way that reconciles the user's mental model of the task and the system's modeling of the problem. For instance, using out-of-turn interaction capability, the user notices that choice of state (Indiana) affects future choices of party. It is hence imperative that we accommodate partial input from the user in various ways, not just in some pre-designated order.

How do users know what to say?

The question of what can the user say and reasonably expect it to be processed is a well-studied issue in speech interfaces. As Yanankelovich [45] points out, 'the functionality of [voice based] applications is hidden, and the boundaries of what can and cannot be [said] are invisible.' The semantics of out-of-turn interaction are more specific than free-form speech input, because it merely allows a dialog's interaction sequence to be realized in a different order. There are various solutions to this problem, many of them involving the careful design of prompting strategies.

How can we support out-of-turn interactions?

A typical approach to implementing mixed-initiative dialogs involves anticipating points where the user might take the initiative and hardwiring mechanisms to 'trap' and process her utterances. Unfortunately this approach becomes cumbersome in task-oriented dialogs (e.g., making a flight reservation) where there are *subdialogs* where the scope of mixing initiative must be restricted at various levels. What is needed is an approach where we can specify the structural organization of the dialog independent of how the mixing of initiative is supposed to happen.

Can we specify mixed-initiative dialogs independent of the modality of interaction?

In addition to separating dialog structure from how the mixing of initiative can/should happen, we must contend with the multiple modalities of interaction possible. The recent popularity of devices such as Blackberrys and the availability of traditional applications (e.g. address book, email, calendar) on these devices have reinvigorated research on dialog notations, especially with an eye toward separating UI details from dialog structure. In the multimodal web context, a useful goal is to ensure that the notation is independent of the particular modality (e.g., voice versus clicking) used for interaction. Is this possible? And if so, have we learned something in the process that might apply to the representation of dialogs independently of presentation? Can we apply the same principles for multi-platform dialog representations?

3 Spoken Systems Research

A spoken dialog system can be defined as one in which computers and humans interact on a turn-by-turn basis, and in which spoken natural language plays an important part in the communication [25]. Although speech is the primary (or only) means of interaction in the projects surveyed here, these projects provide useful insight into how they have tackled problems such as dialog representation and dialog management. The survey is necessarily brief; it only covers those efforts that have direct relevance to the types of applications considered here (multimodal web browsing) or can help contrast the specific solution proposed in this paper.

3.1 Dialog Management

In the view of [25], a dialog manager plays role of a mediator between the user and an external application (for example an information system, like a database, or a knowledge base to support problem solving). It must determine if sufficient information has been obtained from the user in order to enable communication with an external application, perform the actual communication, and then return the results back to the user. In doing so, it must tackle a variety of problems:

- *Natural language processing:* The recognized utterance must be parsed to determine what the user said. The strategy could be as simple as keyword matching, or the dialog manager may use sophisticated algorithms to parse and understand the utterance.
- *Badly formed, or unrecognized utterances:* A dialog manager may use such strategies as speech act analysis [44], discourse context, user models, and dialog structure in order to interpret these. In the case of failure, the system may fall back on methods such as spelling mode or touch-tone input.
- *Verification strategy:* A dialog manager must also decide upon a verification strategy for utterances. Verification may be explicit ('Did you say this?') or implicit as in having the system incorporate the user's utterance in the next prompt it plays.
- *Intention recognition:* This involves determining what the user is trying to do based on the currently recognized utterance.
- *Dialog control:* The dialog control strategy is a further critical issue and varies with the complexity of the application. This involves deciding what the user may say or not say, and what to do at the next turn.

3.1.1 General Strategy

According to Souvignier et al. the general strategy followed by a dialog manager in a spoken system is one of slot-filling [41]. In a system-driven dialog only values for specific slots are accepted, whereas in a mixed-initiative dialog, the user is allowed to supply much information as he wants, possibly with values for slots not in the system prompt. The task of the dialog manager thus is to fill as many slots to meet the user's dialog goal, while keeping the dialog as short as possible. The dialog manager consults the database after each turn to determine if more information is required. If so, it determines which item is to be requested, and may use such heuristics as "solicit input for the slot with the maximum disambiguation potential" in the process. In their paper, Souvignier et al. also mention the high level dialog description language [7] which allows designers to specify task-specific aspects like slot-definitions, questions, and verification strategies in a declarative way.

- *Consistency within the turn:* The utterance may not contain two different values for the same slot, unless one of them is negated. Also, utterances with values for different slots that have no match in a consulted database are rejected.
- *Consistency with system prompt:* Corrections to items not occurring in the system prompt are not accepted.
- *Consistency with system belief:* Interpretations that do not refer to a valid database entry after being combined with current system belief are rejected.

Figure 2: Consistency criteria that may be applied by the dialog manager [41].

3.1.2 Dialog History

A dialog consists of information that is built-up over several turns, so later utterances might contain a reference to earlier ones. This dialog history can be used in various ways in dialog systems – ranging from consistency checks to ensure that the user isn't contradicting himself, to switching language models based on the history. An example given in [41] is a directory system, where the user says "Give me his e-mail." In such a case, the system must look up the dialog history and attempt to fill up the slot 'his.' Fig. 2 shows three types of consistency checks that may be applied within a dialog (reproduced from [41]).

The authors give the example of their PADIS-XXL system, which maintains multiple system beliefs by keeping track of N-best lists, holding values from different dialog turns. At any given time, the active part of database is restricted to one which contains the N-best lists. However this strategy is only applicable to speech recognizers that are able to provide such lists of recognized utterances together with associated confidences (e.g., statistically-based such as using hidden Markov models).

The dialog manager can also learn as it interacts with a number of users. For example, in [23] the authors propose a reinforcement learning approach to learning the optimum dialog sequence, by observing real users interact with the system.

3.2 Dialog Modeling

As the complexity of spoken dialog increases, it becomes useful to model the dialog manager's behavior analytically. In this section, we review two analytical models for dialog management that support flexible mixed-initiative dialogs.

3.2.1 Construct Algebra

Abella and Gorin [1] propose an analytical model for spoken dialog managers using an object oriented approach. They propose encoding task knowledge into objects. The objects are organized into an inheritance hierarchy that defines the relationships that exist among these objects. This *task representation* of the dialog is an analytical representation of the structure of the dialog.

The dialog manager now uses a collection of *dialog motivators* that exploit the task knowledge encoded in this inheritance hierarchy to control the progress of the dialog. A set of methods and operators operate on the constructs, and together form a *construct-algebra*. The authors identify six relations (equality, restriction, containment, generalization, symmetric generalization, and containment generalization) that apply to Constructs, and three main operations (union, projection, union of heads). This construct-algebra has been employed in two very different spoken systems — How May I Help You? [16] and Voice Post Queries [11]. With the analytical model for the dialog defined, the system uses the dialog control strategy shown in Fig. 3 until a complete construct is obtained, which means there is no further need for a dialog.

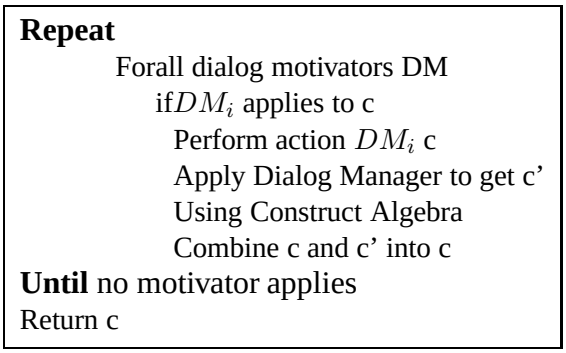

Figure 3: Dialog control strategy from [1].

The authors give us a few examples of motivators - the 'disambiguation' motivator determines if there are conflicts in the users input, while 'missing information' decides what piece of information should be solicited in order to complete the transaction. 'Database querying' is a motivator that decides when the system has acquired enough information to be able to query the database for some requested information. Construct-Algebra is thus a high-level framework for a dialog manager and can manage dialogs involving mixed-initiative interaction.

3.2.2 History, Domain Objects and Task Hierarchy (HOT)

The HOT framework [17] is specifically targeted at the development of mixed-initiative dialog systems. In order to develop a dialog system with this framework, a developer must specify a task hierarchy (which includes the set of tasks together with their parameters organized in a hierarchy), a set of domain specific objects, and the interaction modalities associated with the tasks. A task is a unit of work for a dialog system, and thus the task-hierarchy forms a plan for the dialog. Five relationships can be specified between the tasks in the hierarchy. They are subtasking, ordering, cardinality, inheritance, and subdialog. The task at the root of the hierarchy provides ways to manage the dialog, handle conflicts, and disambiguate the user's utterance. At each turn, the user's input is scored to determine what task the user intends to work on. This task is said to 'receive focus.' The *focus model* controls which task may receive focus, hence identifies the opportunities for mixed-initiative interaction. Barring conflicts, the framework commences execution of the task in focus. Within the root task, the developer must specify such dialog aspects as how to respond if there are no more tasks eligible for focus, and the various back-ends the dialog system uses.

Task functionality may be categorized as pre-backend and post-backend [17]. Pre-backend functionality includes assessment and confirmation of various parameters to be sent to the back-end, and post-backend functionality acts upon the backend outcome (requesting further constraints, communicating the outcome). At every turn, the HOT framework maintains the dialog history, which includes system prompt, user input, and the results of each turn. According to the authors, the dialog-history serves as a task-stack, as it conveys the set of tasks that have been performed.

3.3 Markup Languages and Standards

While research has demonstrated proof-of-concept voice and multimodal applications, large-scale adoption of these technologies requires industry standards for such technologies and robust implementations of these standards. The World Wide Web Consortium (W3C) is overseeing the standards efforts of industry-leaders through its numerous working groups. While advances have been made in speech recognition technology, the recognition of free-form speech is still error-prone, and brings with it the much larger problem of interpreting the user's utterance. Thus a common thread that runs through all attempts at voice-enabled standards is the use of speech-grammars such as

the Speech Recognition Grammar Specification [19] in order to constrain the speech-recognizer. The problem of understanding the user's utterance becomes easier, as the developer only has to look for words that are part of the specified grammar.

3.3.1 Voice standards

VoiceXML [14] is a popular standard for creating voice-response applications[1]. A VoiceXML document can be viewed as a finite-state machine (FSM), where the user must be in one of the states (corresponding to dialogs). Transitions in this FSM are represented by URI references to other dialogs which may be within the same VoiceXML document or links to other documents. VoiceXML provides support for two kinds of dialogs - forms and menus. Forms provide information to the user, collect the resulting user input and interpret the meaning of the input using XML grammars. Menus allow the user to navigate through a series of alternative choices, and allow the user to transition to other dialogs which may be in the same, or in different documents. Subdialogs allow the user to transition to another dialog, and return when finished (like a function call). A sub-dialog might confirm a user's action or handle specific tasks. Developers can create libraries of reusable sub-dialogs for performing recurring tasks.

The dialog model of VoiceXML can be used to create simple dialogs and interaction sequences. System initiated dialogs are created by activating only the grammar corresponding to the currently executing dialog. Simple mixed-initiative applications are created by activating other grammars in addition to the grammar corresponding to the currently executing dialog. A simple example of a mixed-initiative dialog would be for the document to activate a *form-level* grammar. This would allow the user to specify any item in the form irrespective of whether the system is currently soliciting an utterance for it. VoiceXML uses the Form Interpretation Algorithm (FIA) to interpret the voice dialogs described by the document. The FIA permits multiple slots to be filled as a result of a user utterance, hence it permits out-of-turn and in-turn interaction at the same time.

```
<vxml version="1.0">
 <form>
  <field name="branch">
     <prompt>Would you like information about
               a Representative or a Senator?</prompt>
     <grammar src="politics.gram"
        type="application/x-jsgf"/>
  </field>
  <block>
     <submit next= "http://newpage/process.jsp"/>
  </block>
 </form>
</vxml>
```

Figure 4: A simple VoiceXML document adapted from [10]. Notice that a grammar file is associated with a prompt, yielding a fixed-initiative application.

Figure 4 is a simple example of a VoiceXML document. It permits the user to select a choice of branch of congress, and when the user's utterance is recognized, submits the form to the next page in the application. The document references an external JSpeech grammar file [18] (politics.gram) which contains the list of words for recognition. A more complicated example of a VoiceXML document, that permits mixed-initiative, results when we specify a form-level grammar (not shown here) in the document.

[1] At the time of this writing, a new version (2.1) of the VoiceXML standard is being proposed [37].

3.3.2 Multimodal standards

As explained earlier, the interaction in today's web browser can be characterized as predominantly single mode (mouse) and system initiative. W3C, through its Multimodal Interaction Activity group has been developing standards to extend this user interface to permit multiple modes of input, such as voice, keypad, keyboard, mouse or gestures. The work of this group is still in its infancy, but they have already defined requirements for such standards. Some work has already been done in multimodal standards. For example, Synchronous Multimedia Integration Language (SMIL) is one technology that has emerged from under the auspices of the W3C. It permits integration of input received from a variety of modalities. Two specifications for multimodal web interfaces are also currently available through the W3C – X+V (XHTML + Voice) and SALT (Speech Application Language Tags).

XHTML+Voice (X+V)

XHTML+Voice (X+V) [8] is a specification proposed by IBM, Motorola, and Opera Software. It combines the best features of the leading standards in web-presentation and telephony - XHTML and VoiceXML. VoiceXML includes several features as a standalone language that are not required for multimodal use with XHTML. As a result, the designers of X+V modularized VoiceXML 2.0, creating platform specific *modules* to match application deployment environments. [8]. X+V makes XHTML the host language, and uses the modularized version of VoiceXML consisting of only those modules that are relevant for web-based multimodal presentation. The VoiceXML modules omitted from X+V include such modules as *VoiceXML standalone*, *VoiceXML forms*, and *VoiceXML menus* neither of which are necessary for multimodal use. Thus a number of VoiceXML elements such as object, menu, vxml, meta, transfer, and disconnect are absent in X+V. In adopting most of VoiceXML's features, the major design rationale for X+V has been to reduce the learning curve for the large pool of web developers who are already proficient in VoiceXML. The XML-events specification [24] is used for event management in X+V. The document imports namespaces from all three specifications (XHTML, VoiceXML and XML-events).

SALT

SALT is a specification designed by the SALT Forum led by Microsoft Corp. that is also targeted at multimodal devices. It offers a lower level API than does X+V [34] and integrates with the Document Object Model [6] of HTML. The SALT specification contributes a set of XML tags to HTML. They include `<listen>` for speech input, `<prompt>` for speech output, `<dtmf>` for touch-tone input, and `<smex>` for platform messaging to enable platform call-control and telephony features. Each SALT tag has the ability to trigger events, and developers must write code using scripting to handle these events. Developers apply this event-driven model comprising scripting, HTML, SALT, and SRGS (Speech Recognition Grammar Specification; [19]) in order to write speech-enabled web applications. As a result of the lower level API, SALT applications may provide developers with control over interaction at a finer level of granularity with the caveat that the scripting necessary for dialog control may get messy.

An important distinction between the two technologies X+V and SALT is that X+V has the notion of a dialog. Just like VoiceXML, X+V allows developers to create simple dialogs within a XHTML web page. The Form Interpretation Algorithm (FIA) is used to interpret the dialog. Web developers can write forms interpreted by the FIA and do not need to provide any high level dialog management logic within the page. SALT have absolutely no notion of dialog control, and all dialogs must be explicitly scripted by the developers. SALT and X+V are otherwise functionally identical.

Support for mixed-initiative interaction in both SALT and X+V is identical to VoiceXML, and hence is not discussed in this survey. The reader is referred to the W3C specification documents for SALT [39] and X+V [8] for more information on how they support MII. See also Table 1 for a comparison.

Table 1: Summary table of three XML based speech technologies.

Criterion	**VoiceXML**	**X+V**	**SALT**
Support for web interfaces	None	Through XHTML	Through HTML
Standardization status at W3C	Recommendation	Note	Submitted
Support for dialogs	FIA	FIA	Scripted
Level of the API	High-level	High-level	Low-level
Event handling	Dialog-based	XML-events	Scripts and DOM
Mixed-initiative interaction	Supported	Supported	Supported
Design goals	Ease of use	VoiceXML compatibility	Flexibility

4 Basic Approach

While technologies such as SALT and X+V enable the augmentation of speech into browsers, they either operate at a lower level of specification than the applications considered here, hence significantly increasing programming effort, or are otherwise limited in their expressive power for creating and managing dialogs [34]. A multimodal web application must build on these technologies to provide flexible dialog capabilities.

Besides the issues raised in Table 1, several considerations emerge in thinking about a software system design for multimodal web interaction. First, it is important to have uniform processing of hyperlink and voice interaction and, when voice is used, to introduce minimal overhead in handling responsive versus unsolicited input. Observe that hyperlink access can only be used to respond to the initiative whereas voice input can be used both for responding and for taking the initiative. Furthermore, a user may combine these modes of initiative in a given utterance – e.g., in Fig. 1, if the user speaks "Republican Senator Minnesota" at the outset, he is responding to the current solicitation as well as providing two unsolicited pieces of information. Uniform processing of input modalities irrespective of medium (hyperlink or voice) or initiative (responsive or unsolicited) is thus important to support a seamless multimodal interface. Second, it is beneficial to have a representation of the dialog at all times, in order to determine how the user's input affects remaining dialog options. For instance, in Line 5 of *Dialog 2* above, Independents are removed as a possible party choice; in addition to pruning the hyperlink structure (shown in Fig. 1(c)), we must dynamically reconfigure the speech recognizer to only await the remaining legal utterances. Third, it must be possible for the site designer to exert fine-grained control over what types of out-of-turn interactions are to be supported. And finally, it is desirable to be able to automatically re-engineer existing websites for multimodal out-of-turn interaction, without manual configuration.

4.1 Dialog Representation

We have designed a framework taking into account all these considerations [40]. It is based on *staging transformations* [12] – an approach that represents dialogs by programs and uses program transformations to simplify them based on user input. As an example, Fig. 5 (left) depicts a representation of the dialog from Section 2 in a programmatic notation. The hierarchical nature of the website is represented as a nested program of conditionals, where each variable corresponds to a hyperlink that is present in the site. We can think of this program as being derived from a depth-first traversal of the site. For *Dialog 1* of Section 2, the sequence of transformations in Fig. 5 depicts what we want to happen. For *Dialog 2* of Section 2, the sequence of transformations in Fig. 6 depicts what we want to happen.

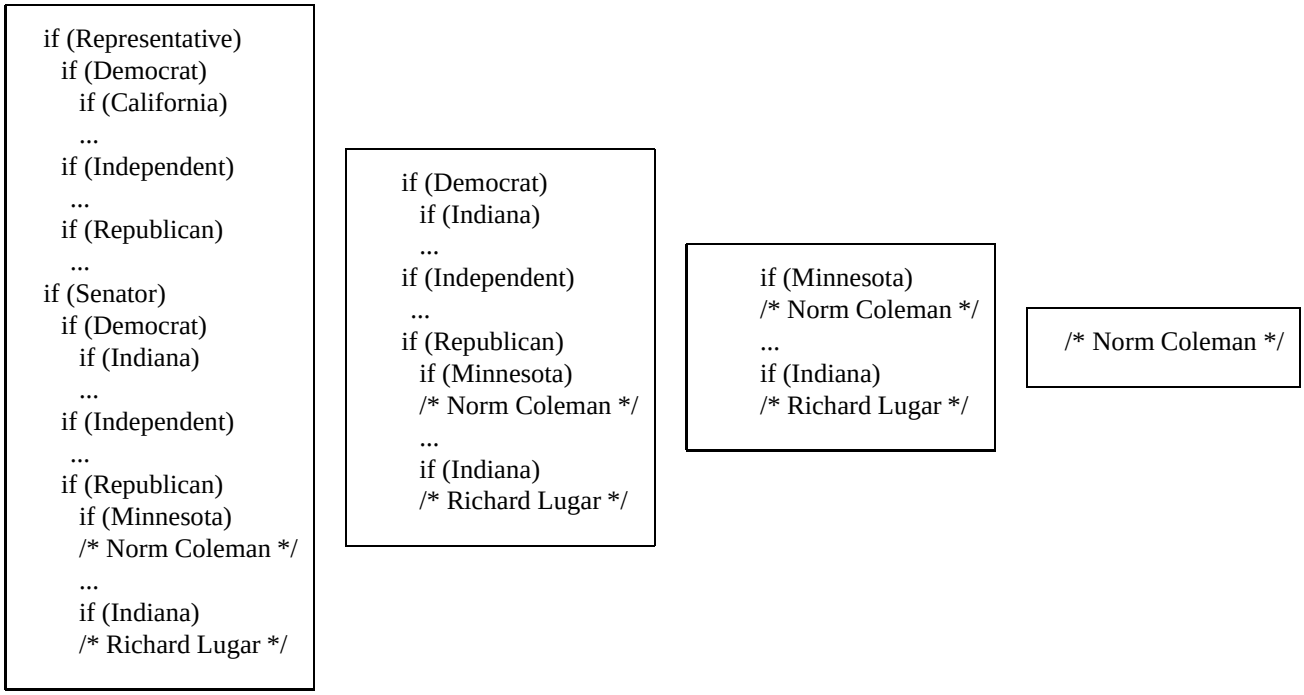

Figure 5: Staging a system-initiated dialog using program transformations. The user specifies ('Senator,' 'Republican,' 'Minnesota'), in that order. This sequence of transformations corresponds to *Dialog 1* of Section 2.

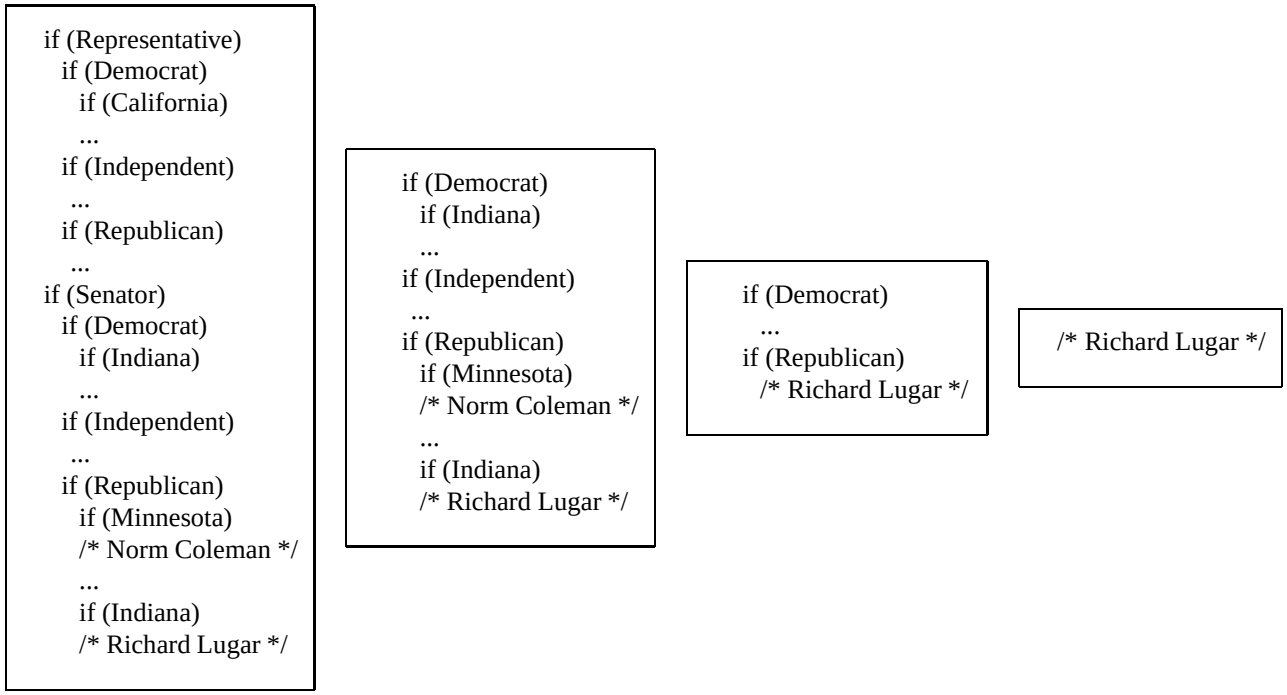

Figure 6: Staging a mixed-initiative dialog using program transformations. The user specifies ('Senator,' 'Indiana,' 'Republican'), in that order. This sequence of transformations corresponds to *Dialog 2* of Section 2.

The first sequence of transformations corresponds to simply interpreting the program in the order in which it is written. Thus, when Sallie clicks on 'Senator' she is specifying the values for the top-level of nested conditionals ('Senator' is set to one, and 'Representative' is set to zero). This leads to a simplified program that now solicits for choice of party. The sequence of Fig. 6, on the other hand, corresponds to 'jumping ahead' to nested segments and simplifying out inner portions of the program before outer portions are even specified. This transformation is well known to be *partial evaluation*, a technique popular to compiler writers and implementors of programming systems [20]. In Fig. 6 when the user says 'Indiana' at the second step, the program is partially evaluated with respect to this variable (and variables for other states set to zero); the simplified program continues to solicit for party, but one of the choices is pruned out since it leads to a dead-end. Notice that a given program when used with an interpreter corresponds to a system-initiated dialog but morphs into a mixed-initiative dialog when used with a partial evaluator!

This is the essence of the staging transformation framework: using a program to model the structure of the dialog and specifying a program transformer to stage it. Notice that this achieves our crucial goal of separating dialog structure from the specification of mixing of initiative. We write the first dialog as:

$$\frac{I}{\text{branch party state}}$$

where the I denotes an interpreter, i.e., the top portion of the specification indicates the stager and the bottom portion indicates the slots over which the stager operates. Similarly, the second dialog is represented as:

$$\frac{PE}{\text{branch party state}}$$

where the PE denotes a partial evaluator. An interpreter permits only inputs that are responsive to the current solicitation and proceeds in a strict sequential order; it results in the most restrictive dialog. A PE, on the other hand, allows utterances of any combination of available input slots in the dialog. It is the most flexible of stagers.

We will introduce a third stager, called a *curryer* (C) that permits utterance of only valid prefixes of the input arguments. The dialog represented by

$$\frac{C}{\text{branch party state}}$$

allows utterance of either 'branch,' or ('branch,' 'party'), or ('branch,' 'party,' 'state'). In other words, if we are going to take the initiative at a certain point, we must also answer the currently posed question.

These stagers can be composed in a hierarchical fashion to yield dialogs comprised of smaller dialogs, or subdialogs. This allows us to make fine-grained distinctions about the structure of dialogs and the range of valid inputs. In this sense,

$$\frac{PE}{a\ b\ c\ d}$$

is not the same as

$$\frac{PE}{\frac{PE}{a\ b}\ \frac{PE}{c\ d}}$$

The former allows all $4!$ permutations of $\{a, b, c, d\}$ whereas the latter precludes utterances such as $\prec c\ a\ b\ d \succ$.

At this point, the reader will notice that we have already separated the slots in the dialog from how the dialog will be staged. We can easily prototype new dialogs by using the basic elements above, for instance a pizza dialog:

$$\frac{PE}{s\ t\ c}$$

which indicates that the dialog is comprised of prompts *s* (for size), *t* (for topping), and *c* (for crust). Consider a more complicated dialog, involving ordering breakfast over a hotel service. Let us suppose that this involves specification

of a {eggs, coffee, bakery item} tuple. The user can specify these items in any order, but each item involves a second clarification aspect. After the user has specified the choice of eggs, a clarification of 'how do you like your eggs?' might be needed. Similarly, when the user is talking about coffee, a clarification of 'do you take cream and sugar?' might be required, and so on. This form of MII is known as *subdialog invocation* [5]. We specify such dialogs using *multi-layered* compositions of constructs, and which can be subsequently staged using a diverse mix of stagers. So, the breakfast dialog is defined as:

$$\frac{PE}{\frac{C}{e_1\ e_2}\ \frac{C}{c_1\ c_2}\ \frac{C}{b_1\ b_2}}$$

where e_1, e_2 are egg specification aspects, c_1, c_2 are associated with coffee specification, and b_1, b_2 specify the bakery item. Notice that the dialog staged by PE is itself a sequential dialog comprised of stagings by a curryer (C).

The staging transformations framework also specifies a set of rules that dictate how a (dialog, stager) pair is to be simplified based on user input. Notice that this is not as straightforward as it looks as it might require a global restructuring of the representation. Assume that we stage the breakfast dialog using the interaction sequence $\prec\ c_1\ e_1\ c_2\ \cdots \succ$; the occurrence of e_1 is invalid according to the dialog specification above, but we will not know that such an input is arriving at the time we are processing c_1. So in response to the input c_1, the dialog must be restructured as follows:

$$\frac{PE}{\frac{C}{e_1\ e_2}\ \frac{C}{c_1\ c_2}\ \frac{C}{b_1\ b_2}} \Longrightarrow \frac{C}{\frac{C}{c_2}\ \frac{PE}{\frac{C}{e_1\ e_2}\ \frac{C}{b_1\ b_2}}}$$

By replacing the top-level PE stager with a C, it becomes clear that the only legal input now possible must have c_2. Once the coffee subdialog is completed, the top-level stager will revert back to a PE. Such dialog restructurings are necessary if we are to remain faithful to the original specification.

Why is the staging transformations approach useful? While our actual dialog processing engine is implemented atop a database system (see below), staging transformations constitutes a powerful representational basis to design dialogs: it has a uniform vocabulary for denoting specification aspects (e.g., each of the slots above could be filled via hyperlink clicks or voice), it helps us enforce acceptability criteria on the input (by analyzing the structure of the representation), and its use of stagers helps us specify the mixing of initiative in a very precise manner. Notice that these advantages are broader than dialog management functionality, and also emphasize the ease of dialog specification.

4.2 DialogXML

In order to make the staging notation machine-readable, we have defined an XML representation of dialogs called DialogXML. DialogXML provides elements for defining the slots associated with a dialog, the textual prompts associated with each slot, the accompanying vocal prompts and any tapering of them over the course of interaction, confirmatory characteristics (whether the user's response needs confirmation), and constructs for combining basic dialog elements to create complex dialogs. While DialogXML borrows ideas from some tags in the VoiceXML standard [14], the structure of the DialogXML document is more closely modeled after the idea of stagers[2].

The root of a DialogXML document is the **dialog-spec** element, which may have one or more **dialog** elements. One of the dialog has attribute id="top", and this is the first dialog that is interpreted. A dialog may be a leaf dialog, in which case it contains a set of **dialog-item**s, or a branch dialog, in which case it composes other dialogs using the **dialog-ref** element. An optional attribute 'next' identifies the next dialog in the chain to be interpreted. Thus an abstract view of the DialogXML document is one of a linked list of hierarchical dialogs. Every dialog is associated with a stager, which refers to a program transformation operation that controls the flow of that dialog. The various elements together with their attributes are defined in Table 2.

[2]It has been recently brought to our attention that there is a similar technology with the same name [29]. This work, however, is an extension of the VoiceXML standard for voice interfaces and does not address multimodal interaction.

Table 2: Elements of DialogXML.

Element	Description	Attributes
dialog-spec	This is the top-level element of a dialog specification. The **dialog** element is a child of this element.	• **welcome:** The welcome prompt played to the user the first time she views the page
dialog	This is the basic unit of dialog composition. These elements can be of two types - leaf dialogs contain **dialog-item** elements, while node dialogs compose other dialogs using **dialog-ref** element.	• **id**: an identifier for a dialog • **stager**: the stager for the dialog (pe/currier/interpreter) • **next**: the identifier for the dialog to follow this dialog; • **type**: whether a terminal dialog (leaf/node)
dialog-item	This element is the basic unit of interaction, and represents a variable that the system is soliciting an input from the user for.	• **name** The name of the slot/variable to be filled. • **confirm:** Whether the variable needs confirmation.
vprompt	The tapered voice prompt associated with each **dialog-item**. One or more **vprompts** can be associated with a **dialog-item** and are played in order of the id attribute.	• **id:** The identifying number for the tapered prompt. Prompts are played by order. • **text** The text associated with the voice prompt
tprompt	The text prompt associated with each **dialog-item**. Each **dialog-item** may have only one text prompt	• **text** The string for the text prompt.
dialog-ref	A reference to a **dialog-item** element. This element may be used as a child of the element **dialog** if the dialog is of type branch.	• **link:** The value for the id attribute that this dialog-ref is pointing to. Must be a valid id for a dialog-item.

```
<?xml version="1.0" encoding="UTF-8"?>
<dialog-spec>
<dialog id="top" stager="pe" next="" type="leaf">
        <dialog-item name="topping" confirm="yes"/>
        <dialog-item name="size"/>
        <dialog-item name="crust" confirm="yes"/>
</dialog>
</dialog-spec>
```

Figure 7: DialogXML document for a pizza dialog.

Fig. 7 is a simple, barebones example of the DialogXML notation used by the out-of-turn interaction management system. It is equivalent to the mathematical representation $\frac{P}{t\ s\ c}$. It is a leaf dialog, and does not have a next dialog.

A more complete example of the same DialogXML includes the various text and voice prompts that are played when the dialog is staged. An expanded version of the DialogXML for the same pizza scenario is shown in Fig. 8. Note that each dialog-item can include multiple voice prompts. These are tapered voice prompts that are used in case there is an error recognizing the user's utterance.

4.3 Software Architecture

Dialogs defined in DialogXML can be rendered in either an application web portal (suitable for small form-factor handhelds) or a voice portal (suitable for access via cell phones). Both realizations analyze the current structure of the dialog at each step, and deliver appropriate content to a client multimodal browser. In the case of the web portal, the content is delivered as HTML augmented with SALT (Speech Application Language Tags) markup and in the case of the voice portal, the content is delivered using VoiceXML. Based on user input, the portal simplifies the dialog and presents personalized content, including facilities for continuing the dialog.

An integrated software framework for this purpose is shown in Fig. 9. Operationally it can be divided into four modules: (i) seeding dialog representations, (ii) staging dialogs, (iii) input and output processing, and (iv) database connectivity. The idea of seeding dialog representations is to take a dialog specified in our DialogXML notation and create an internal representation, suitable for staging. The staging transformation framework is then used to handle dialog management. The embodiment of these dialogs must contend with voice realizations, hence a significant portion of the framework is devoted to generating grammars for speech output and validating voice input. The framework currently uses the SALT and Speech Recognition Grammars standards to handle voice interaction. Finally, the database operations module manages and streamlines the delivery of content. Every end-application is organized as a database that the user initially selects for exploration via mixed-initiative interaction. Each record in the database identifies a unique interaction sequence according to the various addressable attributes.

When the interaction begins, the dialog manager uses the parsed DialogXML and metadata from the database to initialize the representation. The dialog manager must then decide what content to present, the items to offer the user for selection, and the speech prompts to play for the current slot. In addition, the dialog manager must determine what aspects the user may specify out-of-turn. Through an analysis of the dialog representation and a set of SQL queries, it determines this information and generates a HTML page that contains relevant SALT XML objects and references to a suitably generated SRGS grammar, or a VoiceXML document. The grammar identifies all the legally speakable utterances for the displaed document.

User input from any modality (current hyperlink and speech) is uniformly handled by the Utterance Validator module of the system. Upon receiving an utterance from the user, the module first determines whether the utterance contains fillings for multiple slots, and whether the utterance is valid. If part or whole of the utterance was invalid,

```
<?xml version="1.0" encoding="UTF-8"?>
<dialog-spec welcome="Welcome to Joe's Pizza">
<dialog id="top" stager="pe" next="" type="leaf">
    <dialog-item name="topping" confirm="yes">
        <tprompt text="What topping would you like for your pizza?"/>
        <vprompt id="1" text="What topping would you like?"/>
        <vprompt id="2" text="I'm sorry, I didn't understand you.
                What topping would you like again?"/>
        <vprompt id="3" text="I'm having trouble understanding you.
                Please try again, or click on the
                topping of your choice"/>
    </dialog-item>

    <dialog-item name="size">
        <tprompt text="What size pizza would you like?"/>
        <vprompt id="1" text="What size would you like?"/>
        <vprompt id="2" text="My fault, please tell me what size you'd like."/>
    </dialog-item>

    <dialog-item name="crust"/>
        <tprompt text="Please select a crust for your pizza"/>
        <vprompt id="1" text="Tell me what crust you would like."/>
        <vprompt id="2" text="I didn't understand. What crust do you want?"/?
        <vprompt id="3" text="My fault again. Please select a crust."/>
    </dialog-item>
</dialog>
</dialog-spec>
```

Figure 8: Complete DialogXML for the example pizza site.

then the system accepts the valid utterances and rejects the invalid utterances. An appropriate prompt is displayed and played to the user. Having tokenized the user's utterance into its constituent fillings, the dialog manager then calls the staging transformer with the values for the fillings in the order they were received. After the representation is simplified, the dialog manager applies a suite of *dialog motivators* [1] (discussed below) to the dialog. If the dialog is not completed, content creation and speech grammar generation resumes.

These aspects are discussed in detail next.

4.3.1 Seeding Dialog Representations

Recall that the program 'scripts' in the staging transformations approach do not define the dialog flow, they merely capture the structure of the dialog. Dialog control is operationalized using program-transformation techniques, which are said to *stage* the dialog. A set of rules are used to systematically reduce the dialog representation using program transformation sequences. These rules are reproduced from [12] in Fig. 10. For purposes of brevity, rules involving interpretation (I) are not shown.

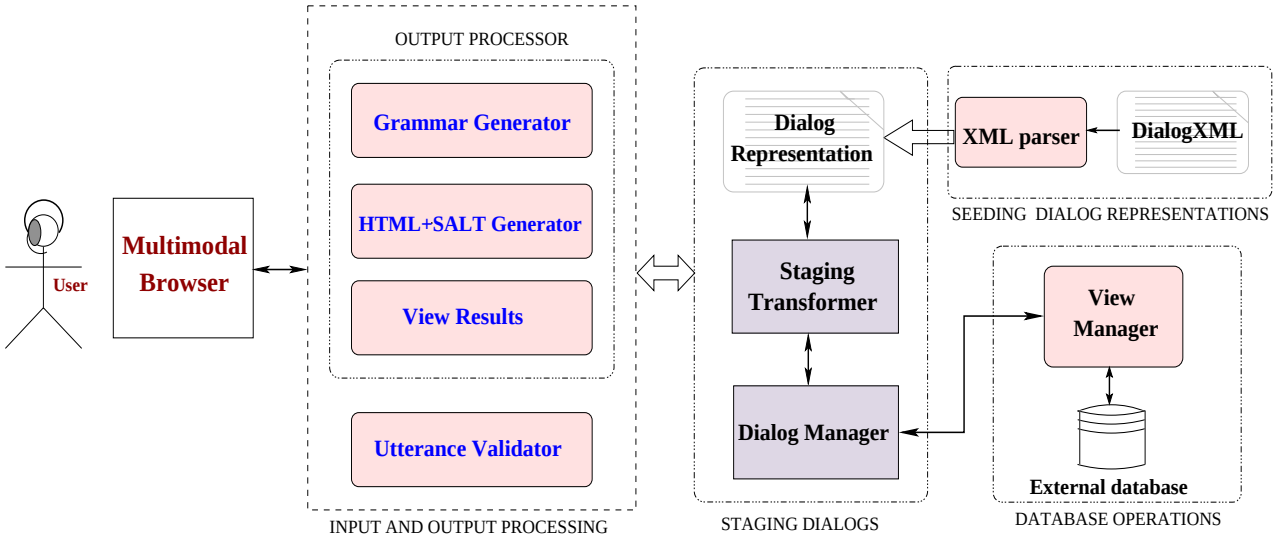

Figure 9: Mixed-initiative interaction framework architecture.

In order to facilitate the description of these rules, the following notation is used.

$$\frac{PE}{X}$$

This expression represents a dialog script X being staged with a partial evaluator. Likewise,

$$\frac{C}{X}$$

represents dialog script X being staged with a curryer. In each of these notations, the X could be either:

$$(a : T)$$

meaning that the dialog script is a simple script consisting of one attribute a to be input from the user, and that a is of a primitive input type T. An example of a primitive input type is a party name, or a branch of congress specification. Such a dialog script, given an appropriate input a, would be simplified by rule 2 to θ, the empty dialog. In addition, X could be:

$$(x : PE|C|T)$$

meaning that the dialog script could be one of three options: (i) a complex script being staged by a partial evaluator (PE), (ii) a complex script being staged by a curryer (C), or (iii) a simple script with one attribute as described above. We also use the Kleene star operator $*$ to represent repetitions of these basic types. Precedence matters when applying the rules in Fig. 10 so they are shown in order of decreasing precedence.

As an example of these rules in operation, let us stage the breakfast dialog presented earlier and assume that the user's input arrives in the order: $\{c_1, c_2, b_1, e_1, b_2, e_1, e_2\}$. Note that the first occurrence of e_1 is an invalid input. The sequence of transformations are shown in Fig. 11. Each $\Longrightarrow$ in Fig. 11 describes a transformation based on user input and each $\longrightarrow$ describes a simplification of the dialog structure. The rule numbers from Fig. 10 are shown alongside the reductions.

4.3.2 An Efficient Staging Algorithm

In order to stage dialogs, the system uses an interpretation of the staging transformation rules described in Fig. 10 tailored to operations performed on the XML DOM representation. The algorithm that is used to process the user's

$$\frac{PE|C}{(x:PE|C|\theta)} = x \tag{1}$$

$$[\frac{PE|C}{(a:T)} \text{ given } a] = \theta \tag{2}$$

$$[\frac{PE}{(x:PE|C|T)^*(a:T)(y:P|C|T)^*} \text{ given } a] = \frac{PE}{xy} \tag{3}$$

$$[\frac{C}{(a:T)(x:PE|C|T)^*} \text{ given } a] = \frac{C}{x} \tag{4}$$

$$[\frac{PE}{(x:PE|C|T)^*(y:PE|C)(z:PE|C|T)^*} \text{ given } a \text{ filling } y] = \frac{C}{[y \text{ given } a]\frac{PE}{xz}} \tag{5}$$

$$[\frac{C}{(x:PE|C)(y:PE|C|T)^*} \text{ given } a \text{ filling } x] = \frac{C}{[x \text{ given } a]y} \tag{6}$$

$$[\frac{PE}{(x:PE|C|T)} \text{ given any input}] = \frac{PE}{x} \tag{7}$$

$$[\frac{C}{(x:PE|C|T)} \text{ given any input}] = \frac{C}{x} \tag{8}$$

Figure 10: Reduction rules for currying and partial evaluation stagers. Adapted from [12].

utterance is specified in Fig. 12. This algorithm is possibly executed multiple times for multiple user utterances. The algorithm fires the complete-dialog event (explained later) when the user's dialog is complete.

When the user selects a database to explore, the system retrieves the DialogXML corresponding to the database and parses the document. The DialogXML can be viewed as a linked list of trees, with each tree corresponding to a hierarchically staged dialog. The list of dialogs are staged in the sequence they are parsed. As a result, the system first stages the first dialog in the list. The system must now identify what question to ask the user, and what the user may say in response to the question. This is done using a recursive pre-processing step that is depicted in Fig. 13. As a result of this step, the system now knows what item to offer the user as part of the system-initiated dialog. This item, together with legal values that be used as filling for that item, is sent to the HTML+SALT generator which generates valid HTML for the page. The HTML page also contains SALT markup identifying a URI that contains the SRGS grammar [19] that defines what the user may say. The set of items that can be said, together with a data-structure representation of the grammar structure is sent to the Grammar generator, which generates the grammar file on demand. Details about grammar generation are covered later.

Upon recognition of the user's utterance, the SALT-compliant browser submits the recognized utterance to the Utterance Validator using the HTTP POST request. The validator tokenizes the utterance into its constituent slot/fillings, sends them to the stagers, which employ the dialog staging rules to process the representation sequentially. Multiple passes through the dialog-processing algorithm may be made if multiple utterances were received from the user. The dialog sequence ends when the null dialog (θ) is reached, or if the user has explicitly requested that the dialog be ended, and the collect results operator be invoked. In such a case, the system transfers control to the view-results module, which displays details about the collected results.

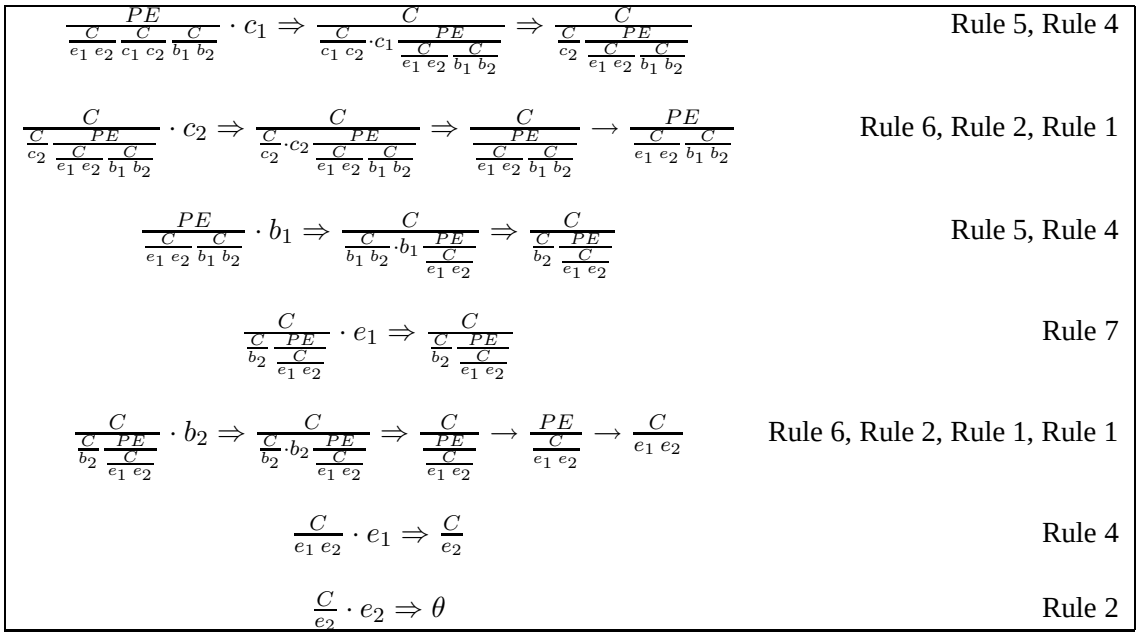

Figure 11: Input induced transformations on the breakfast dialog. The transformation where rule 7 is applied indicates an input that could not fill any currently applicable slot, and would presumably lead to an error message, but does not result in a simplified dialog.

4.3.3 Dialog motivators

The only aspect of the architecture to be covered are the dialog motivators and the grammar generators. Dialog motivators are useful nuggets of processing that help streamline the dialog at every user utterance. We use four main motivators:

1. **prune-dialog:** This motivator decides if the internal dialog representation can be pruned as a result of the previous utterance by the user. For example, if a user is interested in 'New York' and 'Senators' there is no longer a need to prompt for party, since both senators from New York are from the Democratic Party. Thus the party slot can be automatically filled with 'Democraat' and removed from the dialog representation.

For a filling received for slot *nodeb*
In *current-dialog*, locate the node (`nodeb`)
If *nodeb* not found, the utterance was invalid. **return**.
locate the parent(*nodep*) of `nodeb`
delete *nodeb*
if *nodep* has more children, **return**
else, *current-dialog* needs to be changed.
    find the first ancestor(*anc*) of `current-dialog`, deleting all dialogs without children
    current-dialog=anc;
    if *anc* not found, the dialog is complete. Trigger **dialog-complete**.
return

Figure 12: Staging transformation algorithm.

```
while there are more dialog-elements in the current tree
begin
    for the current dialog
        call the stager in the current dialog on the elements in the current dialog.
        this returns the following information stored in global variables-
                —the set of items that can be said
                —the item that must be presented to the user
                —data-structure representation of the grammar to be used for validation
end
```

Figure 13: Processing to generate grammar.

2. **complete-dialog:** This motivator decides if a dialog is complete. A dialog is complete if a unique record in the database VIEW being used has been identified, or if there are no more items left to solicit input for in the dialog. In such cases, the unnecessary slots are removed from the representation. Notice that **complete-dialog** is a specialization of **prune-dialog**.

3. **confirm-dialog:** This motivator applies if the item has been designated in the DialogXML as one for which confirmation must be sought. In a real application, confirmation would be sought for utterances that have been recognized with a low value of confidence; however SALT does not provide us with the hooks to learn about confidence values of recognized utterances, hence we specify the need for confirmation in the DialogXML markup.

4. **collect-results:** This motivator applies if the user explicitly requested (via a 'Show me results' utterance) that the dialog be terminated in order to view a flat listing (of the relevant remaining records).

4.3.4 Grammar generation

Grammar generation in the voice portal uses Nuance's GSL grammar format and requires no special handling. In the web portal, grammar generation involves a careful division of labor between the browser, utterance validator, and the embedded speech grammars themselves. For instance, the system generates JavaScript to handle some types of interaction on the client side within the browser itself. Confirmation of the user's utterance, 'What may I say?,' and 'Show me something else' type of questions are examples of interactions handled by JavaScript. The embedded SRGS grammars are used for encapsulating site-specific logic and are faithful to the C and I stager specifications. Generating grammar fragments corresponding to the PE stager will result in an exponential enumeration of utterance possibilities, so we use a less restrictive grammar and allow the invalid utterances to be caught by the Utterance Validator instead.

4.4 Implementation Details

The web portal instantiation is implemented as a Java web application using JSPs and Servlets. The application runs inside the Tomcat servlet engine. The system uses the Apache HTTPd web-server with a WARP connector to connect to the Tomcat servlet engine, and functions like a proxy-server. A PostgreSQL database server serves the example databases we use in this paper. The web-application connects to the database server using the Java Database Connectivity (JDBC) API. It uses a meta-data API to learn about the structure of the data present in the database. An SQL query initially helps compute a VIEW that serves as the starting point for the dialog. This VIEW is used for all future interactions and helps reinforce that a user is always working with a personalized 'view' of the

information space. The use of VIEWs can be used to increase system efficiency as they can be shared across many users. We tested the system using the Microsoft SALT plug-in for the Internet Explorer 6.0 browser on a system running Windows XP.

5 Case Studies and Evaluation

Several applications have been created using the out-of-turn interaction framework presented above. The first is an interface to the Project VoteSmart website (http://www.vote-smart.org) and an example interaction has been already described in Fig. 1. This interface provides details on about 540 politicians comprising the U.S. Congress and is modeled after the Project VoteSmart site (http://vote-smart.org). In addition to party, branch of Congress, and state information, we modeled district and seat information, in the form of a PE over five specification aspects.

Two other applications, covered in [40], involve a tour application for Washington D.C visitors and an interface to the fuel economy guide at the environmental protection agency (EPA – http://www.fueleconomy.gov). See [40] for more details.

The evaluation of out-of-turn interaction interfaces is a topic of interest in itself. Two important issues are: does the support for out-of-turn interaction make some tasks easier and do users actually utilize out-of-turn interaction for such tasks? The answer to the first question is clearly in the affirmative. For instance, in the example from the introduction, being able to say 'Indiana' out-of-turn is advantageous because it shows the inapplicability of one of the party choices. Without this capability, Sallie would have to manually browse through each party and determine if she obtains the information she was seeking.

To investigate whether users utilize out-of-turn interaction for such tasks, we are conducting several user studies, with several applications. Preliminary results with the US Congress application are described in [31]. The study involved 25 users who were assigned eight information-finding tasks. Four of the tasks were *non-oriented*, meaning they could be completed with or without out-of-turn interaction; the remaining four of them were *out-of-turn-oriented*, meaning they would be cumbersome to complete without the capability of out-of-turn input. However, users were not told which tasks fell under which category. On analyzing the results, we learnt that all users employed out-of-turn interaction when presented with an out-of-turn oriented task, indicating that they are adept at discerning when such interaction is necessary. Many users successfully inferred that out-of-turn interaction is suitable when we have a specific goal in mind and not so much when we are conducting exploratory browsing. See [31] for more details of this study.

6 Discussion

The above applications have highlighted the key features of our software framework. The staging transformers have been primarily responsible for dialog control, specifying what the user may say at any given time. The dialog manager has streamlined the dialog, pruning it when necessary, and triggering the appropriate actions. Our modularized implementation approach makes it easy to construct speech-enabled interfaces to database-driven sites.

By using a combination of stagers (PE, I, and C), our dialog representation is independent of in-turn versus out-of-turn interaction and, most importantly, the modality used for interaction. This, in of itself, is a significant contribution of our work. The user interface research community has been exploring the separation of representation issue for many years. For example, 'user interface management systems' (UIMS), a term coined by Kasik [21] (circa 1982), refers to the study of software architectures for building interactive systems [28]. One of the explicitly stated goals of the UIMS community was 'achieving dialog independence' [33]. The focus of the research was mainly to automatically generate high quality user interfaces from abstract representations. This approach did not find wide acceptance because most of the generation tools available were based on heuristics that made the generation process unpredictable and hard to control [27].

Lately, abstract user interface specifications (also known as model-based approaches) have received renewed attention due to a multiplicity of devices and the variety of interaction possibilities with them. While some approaches have created domain-specific languages such as SISL [9] and WebML [13], others are still exploring the general idea of having a universal specification language for building interactive applications. Szekely [43] presents a retrospective of the model-based approaches and how they have evolved over time.

Still more recent work has focused on user interface representations for multiplatform user interfaces. The goal is to specify the user interface in parts or models (e.g. presentation model, data model, dialog model) so that moving the user interface to another platform requires minimal modifications. Despite a lot of attention to this area, the research community has not yet converged upon a dialog representation that is independent of the task-details of the user interface. For example, TERESA [26] uses enabling operators as part of a task model representation; UIML [2] uses an event-based model that is closely tied to the presentation; XIML [35] uses a separate dialog model, but it is derived from the task model. USIXML [22], a recent addition to the XML-based UI representations has a separate representation for dialogues but it is not clear how well their approach could handle out-of-turn interaction as in our work.

Furthermore, most existing abstract notations require some form of explicit representation of dialog flow. Our use of staging transformers (e.g., the PE stager) avoids such explicit enumeration of dialog flows and promises to serve as a model for building multi-platform user interfaces [3]. This is one of our ongoing directions of future work.

In general, our work helps demonstrate the viability of our view of the speech-enabled web – namely, that of a flexible dialog between the user and the system which allows the user to take the initiative in controlling the flow of the dialog. Rosenfeld et al. [38] have argued that speech interfaces will become increasingly ubiquitous and will be able to support smaller form-factors without comprising usability. The applications presented here validate this viewpoint and help illustrate the importance of using voice to supplement interaction in mobile devices.

It is helpful to contrast our representation-based approach with other ways of specifying dialogs, notably VoiceXML. While they share some similarities, our DialogXML notation is purely declarative and captures only the structure of the dialog. Control is implicitly specified using program transformations, which makes the process of dialog specification less cumbersome for the designer. Furthermore, while VoiceXML permits mixed-initiative dialog sequences too, it does so more as a result of how its form interpretation algorithm (FIA) is organized. Using a combination of program transformation constructs and hierarchically composed dialogs, we are able to specify the nature of out-of-turn interaction in a manner not precisely expressable in VoiceXML (see [36] for more details). Furthermore, in [36], we have shown that barring a few side-effects, the FIA is a partial evaluator in disguise, and hence mixed-initiative applications realizable using VoiceXML are a proper subset of what can be achieved with the many stagers available in DialogXML.

The design emphasis of VoiceXML has been to enable creation of reusable dialog elements which can be composed to create complete voice applications. Developers can create complex dialogs using constructs such as looping and subdialogs in conjunction with the FIA. The VoiceXML interpreter interprets the VoiceXML document and implements the FIA.

The successful implementation of a dialog-based system [41, 46] requires many more facets such as language understanding, task modeling, intention recognition, and plan management, which are beyond the scope of this work. We are now exploring several directions such as natural language speech input and extending the specification capability of DialogXML. We are also conducting usability studies for our multimodal interfaces and carefully assessing the role of speech as an out-of-turn interaction medium. Especially important is addressing the veritable ‘how do users know what to say?’ problem [45] for multimodal web browsing.

This work is an initial exploration into the use of multimodal interfaces to websites. As the use of browsers that support technologies such as SALT, X+V grows, the importance of software frameworks to support multimodal interaction will only increase.

Acknowledgments

This work is supported in part by US National Science Foundation grants IIS-0049075, IIS-0136182, and by a grant from IBM to explore the use of VoiceXML within their WebSphere product. We acknowledge the helpful contributions of Michael Narayan (who designed the formal rules underlying staging transformations), Robert Capra (for input on speech-enabled interfaces), and Saverio Perugini (who implemented a toolbar version of out-of-turn interfaces [32]).

References

[1] A. Abella and A. Gorin. Construct Algebra: Analytical Dialog Management. In *Proceedings of the 37th Annual Meeting of the Association of Computational Linguistics (ACL'99)*, pages 191–200, June 1999.

[2] M.F. Ali and M. Abrams. Simplifying Construction of Multi-Platform User Interfaces Using UIML. In *Proceedings of UIML2001*, 2001.

[3] M.F. Ali, M.A. Pérez-Quiñones, and M. Abrams. Building Multi-Platform User Interfaces with UIML. In A. Seffah and H. Javahery, editors, *Multiple User Interfaces: CrossPlatform Applications and Context-Aware Interfaces*, pages 95–118. John Wiley & Sons, Ltd, 2004.

[4] J. Allen, D. Byron, M. Dzikovska, G. Ferguson, and L. Galescu. Towards Conversational Human-Computer Interaction. *AI Magazine*, Vol. 22(4):pages 27–37, 2001.

[5] J. Allen, C. Guinn, and E. Horvitz. Mixed-Initiative Interaction. *IEEE Intelligent Systems*, Vol. 14(5):pages 14–23, Sep-Oct 1999.

[6] V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood. Document Object Model (DOM) Level 1 Specification Version 1.0. W3C Recommendation Document, October 1998.

[7] H. Aust and M. Oerder. Dialogue Control in Automatic Inquiry Systems. In *Proceedings of the ESCA Workshop on Spoken Dialogue Systems*, pages 121–124, 1995.

[8] J. Axelsson, C. Cross, H. Lie, G. McCobb, T. Raman, and L. Wilson (eds.). XHTML+Voice Profile 1.0. W3C Note, December 2001.

[9] T. Ball, C. Colby, P. Danielsen, L.J. Jagadeesan, R. Jagadeesan, K. Laufer, P. Mataga, and K. Rehor. SISL: Several Interfaces, Single Logic. *International Journal of Speech Technology*, Vol. 3(2):pages 91–106, 2000.

[10] L. Boyer, P. Danielsen, J. Ferrans, G. Karam, D. Ladd, B. Lucas, and K. Reho (eds.). Voice eXtensible Markup Language VoiceXML Version 1.0. W3C Note, May 2000.

[11] B. Buntschuh, C. Kamm, G. Di Fabbrizio, A. Abella, M. Mohri, S. Narayanan, I. Zeljkovic, R. Sharp, J. Wright, S. Marcus, J. Shaffer, R. Duncan, and J.Wilpon. VPQ: a Spoken Language Interface to Large Scale Directory Information. In *Proceedings of the Fifth International Conference on Spoken Language Processing (ICSLP'98)*, page 877, 1998.

[12] R. Capra, M. Narayan, S. Perugini, N. Ramakrishnan, and M.A. Pérez-Quiñones. The Staging Transformation Approach to Mixing Initiative. In G. Tecuci, editor, *Proceedings of the IJCAI'03 Workshop on Mixed-Initiative Intelligent Systems*, pages 23–29. IJCAI Press, 2003.

[13] S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. In *Proceedings of the 9th International World Wide Web Conference*, pages 137–157, 2000.

[14] J. Ferrans, B. Lucas, K. Rehor, B. Porter, A. Hunt, S. McGlashan, S. Tryphonas, D.C. Burnett, J. Carter, and P. Danielsen. Voice eXtensible Markup Language: VoiceXML. W3C Recommendation, Mar 2004. Version 2.00.

[15] S. Goose, M. Newman, C. Schmidt, and L. Hue. Enhancing Web Accessibility via the Vox Portal and a Web Hosted Dynamic HTML-VoxML Converter. *Computer Networks*, Vol. 36(1–6):pages 583–592, June 2000.

[16] A. Gorin, G. Riccardi, and J. Wright. How May I Help You? *Speech Communication*, Vol. 23:pages 113–127, October 1997.

[17] J. Hochberg, N. Kambhatla, and S. Roukos. A Flexible Framework for Developing Mixed-Initiative Dialog Systems. In *Proceedings of the Third SIGdial Workshop on Discourse and Dialogue*, July 2002.

[18] A. Hunt. JSpeech Grammar Format (JSGF). W3C Note, June 2000.

[19] A. Hunt and S. McGlashan. Speech Recognition Grammar Specification Version 1.0. W3C Recommendation, Mar 2004.

[20] N.D. Jones. An Introduction to Partial Evaluation. *ACM Computing Surveys*, Vol. 28(3):pages 480–503, September 1996.

[21] D.J. Kasik. A User Interface Management System. In *Proceedings SIGGRAPH82: Computer Graphics*, pages 99–118. ACM Press: Boston, MA, 1982.

[22] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, M. Florins, and D. Trevisan. USIXML: A User Interface Description Language for Context-Sensitive User Interfaces. In *Developing User Interfaces with XML: Advances on User Interface Description Languages, a Satellite Workshop of Advanced Visual Interfaces*, pages 55–62, 2004.

[23] D. Litman, M. Kearns, S. Singh, and M. Walker. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. *Journal of Artificial Intelligence Research*, 16:pages 105–133, 2002.

[24] S. McCarron, S. Pemberton, and Raman T. (eds.). XML Events An Events Syntax for XML. W3C Candidate Recommendation, February 2003.

[25] M. McTear. Spoken Dialogue Technology: Enabling the Conversational User Interface. *ACM Computing Surveys*, Vol. 34(1):pages 90–169, March 2002.

[26] G. Mori and F. Paterno. Design and Development of Multidevice User Interfaces Through Multiple Logical Descriptions. *IEEE Transactions on Software Engineering*, Vol. 30(8):pages 507–520, 2004.

[27] B. Myers, S.E. Hudson, and R. Pausch. Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, Vol. 7(1):3–28, 2000.

[28] B.A. Myers. User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, Vol. 2(1):pages 64–103, 1995.

[29] E. Nyberg, T. Mitamura, P. Placeway, M. Duggan, and N. Hataoka. DialogXML: Extending VoiceXML for Dynamic Dialog Management. In *Proceedings of the Human Language Technology Conference (HLT'02)*, March 2002.

[30] S.L. Oviatt. Taming Recognition Errors with a Multimodal Interface. *Communications of the ACM*, Vol. 43(9):pages 45–51, 2000.

[31] S. Perugini, M.E. Pinney, N. Ramakrishnan, M.A. Pérez-Quiñones, and M.B. Rosson. Taking the Initiative with Extempore: Exploring Out-of-turn Interactions with Websites. Technical Report cs.HC/0312016, Computing Research Repository, 2003.

[32] S. Perugini and N. Ramakrishnan. Personalizing Web Sites with Mixed-Initiative Interaction. *IEEE IT Professional*, Vol. 5(2):pages 9–15, Mar-Apr 2003.

[33] G.E. Pfaff, editor. *User Interface Management Systems: Proceedings of the Seeheim Workshop*. Springer-Verlag, Berlin, 1985.

[34] S. Potter and J. Larson. VoiceXML and SALT. *Speech Technology Magazine*, Vol. 7(3), May/June 2002.

[35] A. Puerta and J. Eisenstein. XIML: A Common Representation for Interaction Data. In *Proceedings of the Seventh International Conference on Intelligent User Interfaces (IUI2002)*, pages 214–215, 2002.

[36] N. Ramakrishnan, R. Capra, and M.A. Pérez-Quiñones. Mixed-Initiative Interaction = Mixed Computation. In P. Thiemann, editor, *Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM)*, pages 119–130. ACM, 2002.

[37] K. Rehor, A. Lee, D. Burke, R.J. Auburn, P. Baggia, M. Oshry, D.C. Burnett, S. McGlashan, E. Candell, H. Kilic, B. Porter, and M. Bodell. Voice eXtensible Markup Language: VoiceXML. W3C Candidate Recommendation, June 2005. Version 2.1.

[38] R. Rosenfeld, D.R. Olsen, and A. Rudnicky. Universal Speech Interfaces. *ACM Interactions*, Vol. 8(6):pages 34–44, 2001.

[39] Speech Application Language Tags SALT, July 2002. Submission to W3C.

[40] A. Shenoy. A Software Framework for Out-of-turn Interaction in a Multimodal Web Interface. Master's thesis, Department of Computer Science, Virginia Tech, June 2003.

[41] B. Souvignier, A. Kellner, B. Reuber, H. Schramm, and F. Seide. The Thoughtful Elephant: Strategies for Spoken Dialog Systems. *IEEE Transactions on Speech and Audio Processing*, Vol. 8(1):pages 51–62, January 2000.

[42] S. Srinivasan and E.W. Brown. Is Speech Recognition Becoming Mainstream? *IEEE Computer*, Vol. 35(4):pages 38–41, 2002.

[43] P. Szekely. Retrospective and Challenges for Model-Based Interface Development. In *Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces*, pages xxi–xliv, 1996.

[44] D. Traum, L. Schubert, M. Poesio, N. Martin, M. Light, C. Hwang, P. Heeman, G. Ferguson, and J. Allen. Knowledge Representation in the TRAINS-93 Conversation System. Technical Report TN96-4, University of Rochester, 1996.

[45] N. Yankelovich. How do Users Know What to Say? *ACM Interactions*, Vol. 3(6):pages 32–43, Nov-Dec 1996.

[46] V. Zue, S. Seneff, J. Glass, J. Polifroni, C. Pao, T. Hazen, and L. Hetherington. JUPITER: A Telephone-based Conversational Interface for Weather Information. *IEEE Transactions on Speech and Audio Processing*, Vol. 8(1):pages 85–96, January 2000.