University of Dayton
eCommons

Computer Science Faculty Publications | Department of Computer Science

2004

Staging Transformations for Multimodal Web Interaction Management

Michael Narayan
Virginia Polytechnic Institute and State University

Christopher Williams
Virginia Polytechnic Institute and State University

Saverio Perugini
University of Dayton, sperugini1@udayton.edu

Naren Ramakrishnan
Virginia Polytechnic Institute and State University

Follow this and additional works at: http://ecommons.udayton.edu/cps_fac_pub

Part of the Databases and Information Systems Commons, Graphics and Human Computer Interfaces Commons, OS and Networks Commons, Other Computer Sciences Commons, Programming Languages and Compilers Commons, Systems Architecture Commons, and the Theory and Algorithms Commons

eCommons Citation

Narayan, Michael; Williams, Christopher; Perugini, Saverio; and Ramakrishnan, Naren, "Staging Transformations for Multimodal Web Interaction Management" (2004). *Computer Science Faculty Publications.* Paper 33.
http://ecommons.udayton.edu/cps_fac_pub/33

This Article is brought to you for free and open access by the Department of Computer Science at eCommons. It has been accepted for inclusion in Computer Science Faculty Publications by an authorized administrator of eCommons. For more information, please contact frice1@udayton.edu, mschlangen1@udayton.edu.

Staging Transformations for Multimodal Web Interaction Management

Michael Narayan, Christopher Williams, Saverio Perugini, and Naren Ramakrishnan
Department of Computer Science
Virginia Tech, Blacksburg, VA 24061, USA
Email: {mnarayan, chwilli4, sperugin, naren}@vt.edu
Project website: http://pipe.cs.vt.edu

ABSTRACT

Multimodal interfaces are becoming increasingly ubiquitous with the advent of mobile devices, accessibility considerations, and novel software technologies that combine diverse interaction media. In addition to improving access and delivery capabilities, such interfaces enable flexible and personalized dialogs with websites, much like a conversation between humans. In this paper, we present a software framework for multimodal web interaction management that supports mixed-initiative dialogs between users and websites. A mixed-initiative dialog is one where the user and the website take turns changing the flow of interaction. The framework supports the functional specification and realization of such dialogs using staging transformations – a theory for representing and reasoning about dialogs based on partial input. It supports multiple interaction interfaces, and offers sessioning, caching, and co-ordination functions through the use of an interaction manager. Two case studies are presented to illustrate the promise of this approach.

Categories and Subject Descriptors

H.5.4 [**Hypertext/Hypermedia**]: Navigation; H.5.2 [**User Interfaces**]: Interaction Styles; F.3.2 [**Semantics of Programming Languages**]: Partial Evaluation

General Terms

interaction management in web applications, multimodal interfaces, novel browsing paradigms.

Keywords

program transformations, partial evaluation, mixed-initiative interaction, out-of-turn interaction, web dialogs.

1. INTRODUCTION

Web interaction management is a well-studied topic and has produced many innovative solutions to support contextual interactions with websites [5, 10]. Today's web systems feature a diverse range of interactive functionality, from preserving state across sessions (e.g., shopping carts at amazon.com) to gracefully accommodating users' interruptive activities such as pressing back buttons and cloning windows ([38]; e.g., in form-based services). With the shift of web access from the desktop to mobile devices such as PDAs, tablet PCs, and 3G phones [4, 12, 17, 18, 32], and the advent of novel multimodal interfaces, the importance of interaction management has only become accentuated. We posit that the logical

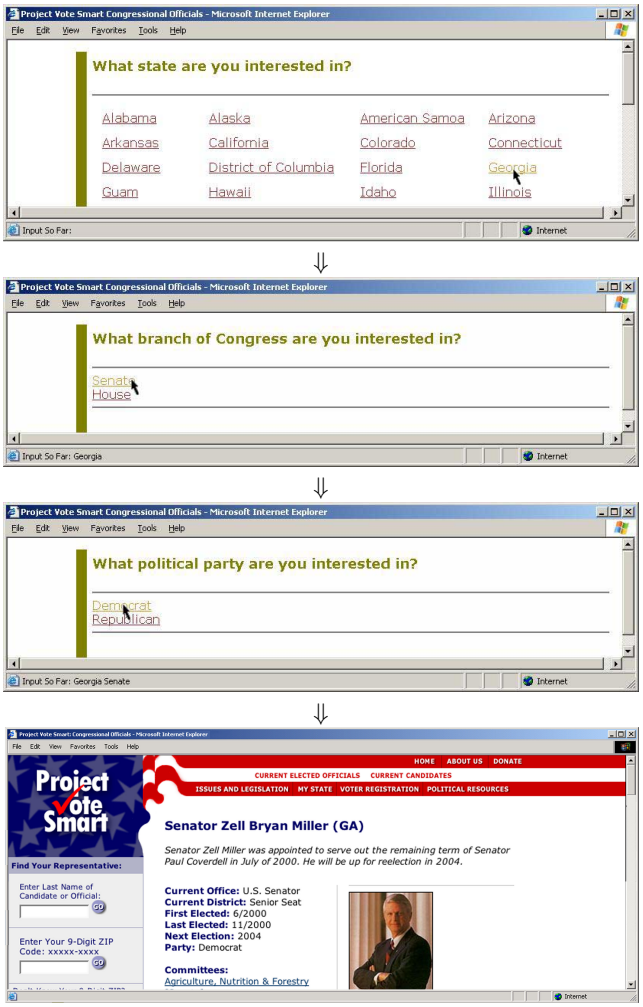


Figure 1: A site-initiated dialog in a US congressional site to reach the webpage of the Democratic Senator from Georgia.

Copyright is held by the author/owner(s).
WWW2004, May 17–22, 2004, New York, NY USA.
ACM 1-58113-844-X/04/0005.

culmination of interaction management research will be to enable flexible and personalized dialogs with websites [34], much like a conversation between humans.

Viewing web interactions as dialogs can be very instructive, and suggests useful metaphors. Imagine the interaction between a user and a website to be a conversation between two participants. The conversation typically begins as *user-initiated* (since the user chose to visit the site). The site then presents a choice of hyperlinks from which the user is expected to make a selection. This step of the conversation is *site-initiated* since by clicking on a(ny) hyperlink, the user will be responding to the initiative taken by the website. After following a link, the user might decide to pursue further links (the initiative thus residing with the site), or press the back button (hence retaking the initiative). Such an interaction where the user and the site exchange initiative is called a *mixed-initiative* dialog.

Mixed-initiative interaction (MII) has been well studied by the speech interfaces, artificial intelligence planning, and discourse analysis communities [2] but has only recently begun to be investigated in web interactions [35]. This is because the mechanisms for the user to take the initiative in web interactions used to be limited. With the emergence of the multimodal web and the availability of multiple paradigms for interaction, these opportunities are now expanding; thus setting the stage for MII to play a prominent role in web interactions.

The primary development we are alluding to is, of course, the emergence of the speech-enabled web [28]; technologies such as SALT (Speech Application Language Tags; [1]) and X+V (XHTML plus Voice; [6]) are ushering in documents that can talk and listen rather than passively display content. The maturing of commercial speech recognition engines [41], the naturalness of speech for conversational interaction, and its role in improving accessibility (i.e., for visually impaired people) have been key factors in the emergence of this niche segment of multimodal browsing.

Speech is commonly perceived merely as a vocal substitute to hyperlink access (i.e., 'say' a hyperlink label instead of clicking on it) [21, 39]. Our approach, on the other hand, is to think of speech as a way for the user to take the initiative in web interactions, and hence *augment* hyperlink usage. While hyperlink access must, by definition, be responsive to the initiative taken by the site, speech interaction can be used to either respond to or take the initiative. Our view of the multimodal web is hence one that enables flexible dialogs, with rich opportunities for mixed-initiative interaction. We focus our studies here on speech and hyperlink as the modes of interaction although our framework will apply more generally to new input mechanisms.

Motivating Example

Project Vote Smart (PVS; vote-smart.org) is a hierarchically organized site for information about US Congressional officials. The first level in this site corresponds to choice of state, the second level corresponds to branch of congress, the third level for party, and so on[1]. Fig. 1 and Fig. 2 depict two dialogs of interaction with Vote Smart; while both culminate in the webpage of Senator Miller, they involve fundamentally different interaction sequences.

The dialog in Fig. 1 is site-initiated since the user progressively clicks on presented hyperlinks (Georgia: Senate: Democrat) to specify values for relevant politician attributes. All such browsing interactions are responsive to the current solicitation; we sometimes refer to such an interaction sequence as an *in-turn* sequence.

Fig. 2 describes a web session with capabilities for speech in-

[1]Since this writing, the Vote Smart site has been restructured into a flat faceted classification; nevertheless the observations and central ideas of this paper still apply.

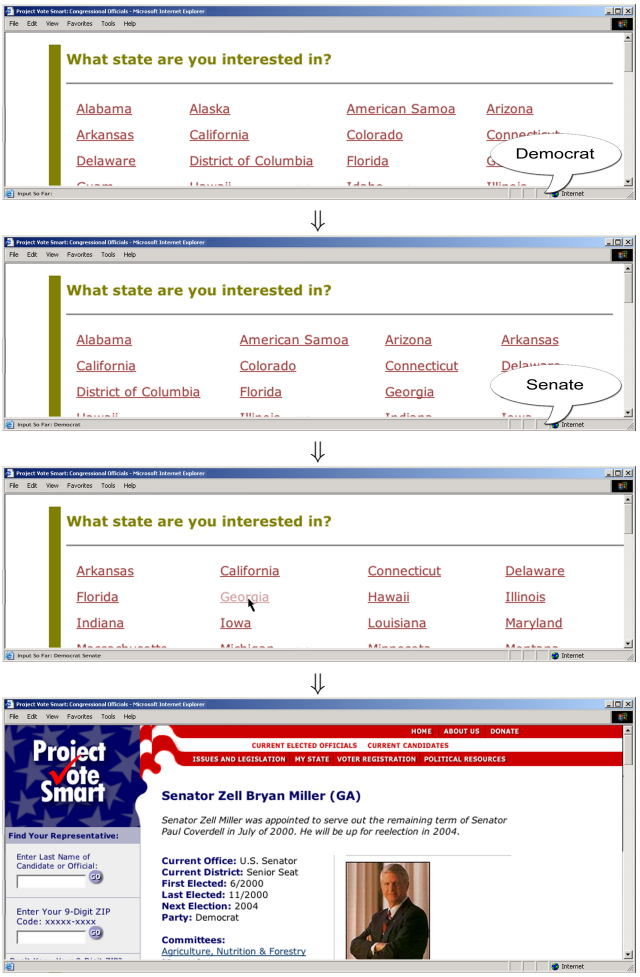


Figure 2: A mixed-initiative dialog in a US congressional site to reach the webpage of the Democratic Senator from Georgia.

put. At any stage in the dialog the user has the option of either pursuing a hyperlink or speaking some utterance. In the top panel of Fig. 2, when the website solicits for choice of state, the user responds with his choice of party (Democrat) instead, by speaking out-of-turn. The user has hence taken the initiative. Such an interaction is called an *out-of-turn* interaction since the user is specifying party information at the first level (when it is normally specified at the third level). As a result of this out-of-turn input, states which do not have any Democratic politicians are pruned out (e.g., Alaska). Notice that the website continues to solicit for state, since it remains unspecified. In conversational terms, we say that the website has reclaimed the initiative, and is repeating its prompt for input. At this point the user again speaks out-of-turn, this time with branch information (Senate). More states are pruned out (e.g., Alabama and Arizona). After this step, the user reverts back to in-turn mode, and responds to the site's solicitation by clicking on the 'Georgia' hyperlink; leading again, directly, to Senator Miller's webpage. While there can be only one purely in-turn interaction sequence to arrive at a webpage—unless, of course, the designer has hardwired multiple paths to a leaf content page—there could be several involving out-of-turn interactions. Notice that in a single out-of-turn interaction, more than one aspect could be specified.

Out-of-turn interaction is most appropriate when the user has a focused information-seeking goal. Browsing is more suitable when

the user's information seeking is exploratory. By enabling the user to supply an out-of-turn input, at possibly multiple times, we facilitate information-finding tasks requiring both exploratory and focused behavior.

The aforementioned example is admittedly a simple form of mixing initiative (called *unsolicited reporting* [2]; for more complicated flavors, see [2, 46]) but provides a powerful mechanism to interact with websites. In particular, the capability for out-of-turn interaction obviates the need to express interaction sequences directly in the browsing structure (i.e., as in parallel faceted browsing classifications [24]). From an information seeking standpoint, out-of-turn interaction is a flexible and unintrusive way to bridge any mental mismatch between users and websites, by increasing the opportunities for communicating partial input.

Contributions of this Paper

The goal of this paper is to flexibly support out-of-turn dialogs with multimodal websites; we present a cross-platform web service architecture that factors multimodal interaction management into the three facets of interaction interfaces, a transformation engine, and an interaction manager. Our main contributions can be summarized alongside each of these facets:

1. Interaction interfaces support hyperlink and speech modes of input and allow users to employ them uniformly in a multimodal session;
2. The transformation engine uses *staging transformations* [13]—a functional approach to dialog management—to specify, reason about, and implement web dialogs without side-effects. In particular, staging transformations allow us to automatically enable existing sites for multimodal interaction, without manual re-engineering;
3. The transformation engine further uses the staging notation to implicitly capture state information in a web dialog, seamlessly supporting save, restore, and caching functionalities;
4. Both categorical and non-categorical modeling is supported in the staging transformations framework, allowing it to be applicable to the vast majority of existing sites.
5. The interaction manager built around these concepts helps realize sessioning, caching, and co-ordination functionality; and, most importantly,
6. the integration of the above ideas supports both in-turn and out-of-turn modes of interaction in a unified manner, obviating the need to ever distinguish between them.

Finally, we argue that the factoring presented here can be easily extended to support novel modalities of interaction in emerging application domains.

2. BASIC APPROACH

To understand the staging transformations philosophy, let us revisit the dialogs of Figs. 1 and 2. We begin with the provocative observation that both in-turn and out-of-turn interactions can be supported by the same dialog programming model!

To see how, it is helpful to think of modeling the Vote Smart website as the program of Fig. 3 (left) where the nesting of conditionals reflects the hierarchical hyperlink structure and each program variable denotes a hyperlink label. For an in-turn sequence, the top series of transformations in Fig. 3 depicts what we want to happen. For the interaction of Fig. 2, the bottom series of transformations depicts what we want to happen. Notice that both sequences start and end with the same representation, but take different paths.

The first sequence of transformations corresponds to *interpreting* the program in the order in which it is written, i.e., when the user clicks on 'Georgia,' that variable is set to one and all other state variables (e.g., 'Alabama') are set to zero, and the program is interpreted. This leads to a simplified program that now solicits for branch of congress. The second sequence of transformations involves 'jumping ahead' to nested program segments and simplifying them even before outer portions are evaluated. Such a non-sequential evaluation is well known in the programming languages literature to be *partial evaluation* ([26]; see Fig. 4), a technique for specializing programs given some (but not all) of their input. Thus, when the user says 'Democrat' out-of-turn, the program is partially evaluated with this variable set to one (and 'Republican' set to zero). The simplified program continues to solicit for state at the top level, but some states are now removed since the corresponding program segments involve dead-ends. Notice that since partial evaluation can be used for interpretation, it can support the first interaction sequence as well.

This is the essence of our staging transformations framework: writing a program to model the structure of the dialog and using a program transformer to *stage* it. In addition, we must adopt a way to map users' partial inputs to assignments of values to program variables.

There are many ways to model dialog structure as programs and the choice of representation is dependent on both the structural characteristics of the website and the interaction scenarios that must be supported. For instance, we can take advantage of the levelwise property of the politicians website and model the dialog as shown in Fig. 5. In this representation, the user's input is captured as assignment of values to only three categorical variables, whereas the representation of Fig. 3 used many more variables, but all boolean.

The representation of Fig. 3 is suitable in a setting where we would like to provide dynamic feedback to the user during the course of an interaction. For instance, even after just supplying 'Democrat' at the outset, the user is provided feedback such as the fact that there are no Democratic politicians (senators or otherwise) in Alaska. The representation of Fig. 5 does not explicitly capture such dependencies and is more appropriate in a database-driven website where a lookup happens only after all relevant information is collected. Furthermore, this representation assumes that there is a way to map user's inputs to the relevant categories (e.g., when the user says 'Alaska,' it must be inferred that he is talking about a state). There is no automatic means to obtain this categorical assignment by a program analysis. Whereas, in the representation of Fig. 3, we would merely need to infer that 'Democrat' excludes the possibility of being 'Republican,' a feature that can be captured mechanically through the application of the program transformation (see end of Section 3.1).

Staging transformations thus provide a functional, implementation-neutral way to specify web dialogs; to fully realize our vision of a flexible multimodal web interaction framework, we require:

- A greater variety of stagers to support practical web dialogs;
- A theory for reasoning about dialogs based on user input;
- A robust transformation engine based on commercial web technologies;
- Interaction interfaces for capturing and communicating user input; and
- an interaction manager providing session management, coordination, and caching functionality.

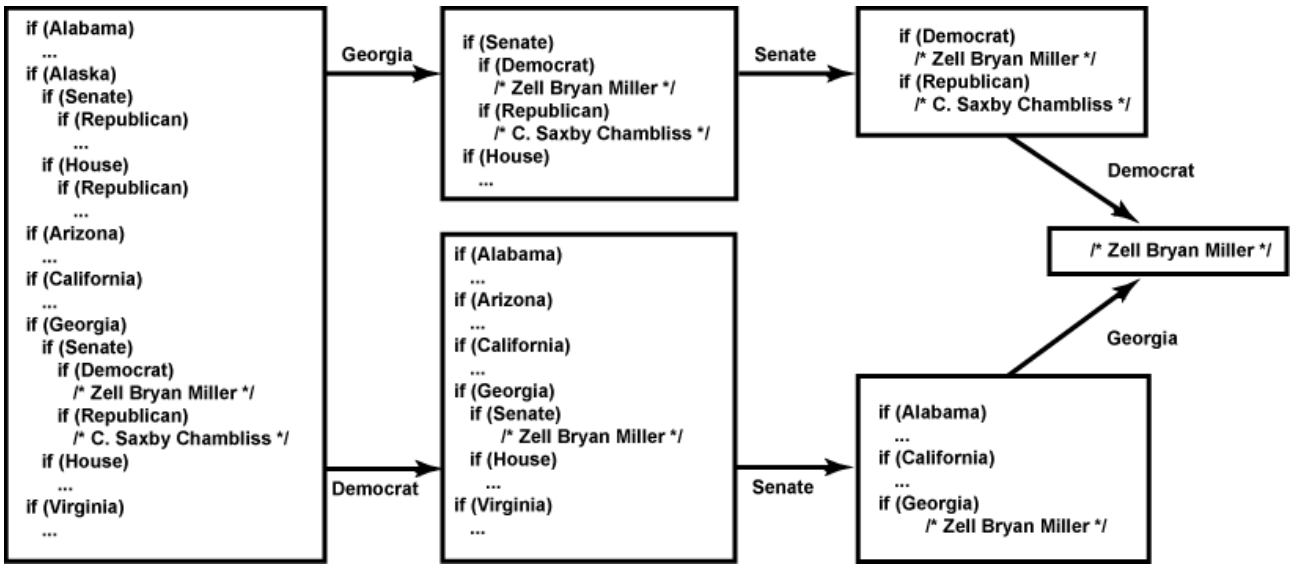


Figure 3: Staging dialogs using program transformations. The top series of transformations mimic an in-turn dialog with the user specifying (Georgia: Senate: Democrat), in that order. The bottom series of transformations correspond to a mixed-initiative dialog where the user specifies (Democrat: Senator: Georgia), in that order.

```
int pow(int base, int exponent) {
  int prod = 1;
  for (int i = 0; i < exponent; i++)
    prod = prod * base;
  return prod;
}
```

```
int pow2(int base) {
  return (base * base);
}
```

Figure 4: Illustration of the partial evaluation technique. A general purpose `pow`er function written in C (left) and its specialized version (with `exponent` statically set to 2) to handle squares (right). Automatic partial evaluators (e.g., C-Mix) use techniques such as loop unrolling and copy propagation to specialize given programs.

```
politicians (state, branch, party) {
  if (unfilled (state) )
    /* prompt for state */

  if (unfilled (branch) )
    /* prompt for branch */

  if (unfilled (party) )
    /* prompt for party */
}
```

```
politicians (state, branch) {
  if (unfilled (state) )
    /* prompt for state */

  if (unfilled (branch) )
    /* prompt for branch */
}
```

```
politicians (state) {
  if (unfilled (state) )
    /* prompt for state */
}
```

θ

Figure 5: Representing the politicians site using categorical variables and staging the dialog of Fig. 2 using partial evaluation. θ denotes the empty dialog.

3. SYSTEM OVERVIEW AND DESIGN

3.1 Staging Transformations

Dialog Notation

We begin by introducing a notation to represent the structure of dialogs as well as the program transformations for staging them. The notation is easiest to understand when only categorical variables are involved. A purely in-turn sequence in the politicians site would be denoted by:

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

where the I indicates that an interpreter is used as the staging transformer. Similarly,

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

denotes a dialog staged with a partial evaluator (PE). 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 allows all 3! orderings of the politician attributes (13, if we allow multiple attributes per utterance) to be achieved, without explicitly programming for them. It is the most flexible of stagers. However, it is an all-or-nothing stager that cannot enforce (i.e., require) a particular ordering. With the above notation, hence just replacing the I in a dialog with a PE dramatically changes a website from one that is meant for browsing to one that supports out-of-turn interaction!

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 $\prec c\ a\ b\ d \succ$.

Another useful stager is the currier (C), which corresponds to the standard definition of currying from the programming languages literature. Programmatically, currying is a specialization of a function when some partial prefix of its arguments are known. From a dialog viewpoint, a currier allows for multiple utterances to be made at one time, with the restriction that these utterances must fill in the dialog arguments in consecutive order, starting at the beginning of the dialog.

To support non-categorical modeling such as shown in Fig. 3 we require the ability to model webpages with multiple links, where only one link can be pursued at a given time. The alternator stager (A) addresses this requirement; while it can be implemented as a program transformation, it is not particularly interesting to look at it as such, since it has little meaning in other contexts.

Let us investigate how to specify a browsing interaction with the politicians website. Since the partial inputs correspond to hyperlink labels, the dialog representation must involve a hierarchical composition over these labels:

$$\frac{A}{\text{ga}\ \frac{A}{\text{s}\ \frac{A}{\text{r d}}\ \frac{I}{\text{h…}}}\ \frac{I}{\text{ak…}}\ \frac{I}{\text{al…}}\ \cdots}$$

At the top-level of this dialog, there is a choice of multiple subdialogs, all of which involve the specification of some state label. We see that from the first page, there are links to pages for Georgia ('ga'), Arkansas ('ak'), and so on. Notice that the order in which we list these links does not matter since, from the viewpoint of the A stager, only one of the corresponding subdialogs can be entered. Looking at the link for Georgia, which is expanded in more detail, we see that this subdialog consists of a further choice among senate ('s') versus house ('h) politicians, and so on. The I's emphasize that the levels must be entered in strict sequential order, reinforcing the browsing paradigm. If we replace the Is with PEs, we will effectively specify a mixed-initiative dialog.

Transformation Rules

With this notation in place, it is now possible to present the rules that govern the behavior of the stagers when processing user input. Notice that this is rather tricky as it might require a global restructuring of the representation. Consider a breakfast dialog given by:

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

where e_1, e_2 are egg specification aspects, c_1, c_2 support coffee specification, and b_1, b_2 specify a bakery item. The top-level PE signifies that the three subdialogs can be entered in any order; the C's denote that once entered each subdialog involves a second clarification aspect. After the user has specified his eggs (e_1), a clarification of 'how do you like your eggs?' (e_2) might be needed. Similarly, when the user is talking about coffee (c_1), a clarification of 'do you take cream and sugar?' (c_2) might be required, and so on. Now, assume we stage this dialog using the 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:

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

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. We formally capture such restructurings by using transformation rules to describe what happens to a {dialog script, stager} pair when a given input is received.

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

$$\frac{PE}{(a : T)}$$

This expression represents a dialog that consists of the dialog script a and is being staged with a partial evaluator. $(a : T)$ indicates that a is a simple prompt, not a subdialog. The expression can be read, 'This dialog is being staged using a PE and consists of the prompt a, of type T.' Similarly,

$$\frac{C}{(y : PE|C|A|T)}$$

means that the dialog is staged using a currier. $(y : PE|C|A|T)$ indicates that the dialog script is either a single prompt (T), or a subdialog itself being staged using a partial evaluator, a currier, or an alternator. The expression $\langle x \bullet a \rangle$, as above, denotes the transformation of x when given the input a, as mandated by the

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

$$\langle \frac{PE|C|A}{(a:T)} \bullet a = \theta \rangle \tag{2}$$

$$\langle \frac{PE}{(x:PE|C|A|T)^*(a:T)(y:PE|C|A|T)^*} \bullet a \rangle = \frac{PE}{xy} \tag{3}$$

$$\langle \frac{C}{(a:T)(x:PE|C|A|T)^*} \bullet a \rangle = \frac{C}{x} \tag{4}$$

$$\langle \frac{A}{(x:PE|C|A|T)^*(a:T)(y:PE|C|A|T)^*} \bullet a \rangle = \theta \tag{5}$$

$$\langle \frac{PE}{(x \vdash A)^*(y \vdash A)(z:PE|C|A|T)^*} \bullet a \rangle = \frac{PE}{x\langle y \bullet a \rangle z} \qquad \langle y \bullet a \rangle \neq y \tag{6}$$

$$\langle \frac{PE}{(x:PE|C|A|T)^*(y:PE|C|A)(z:PE|C|A|T)^*} \bullet a \rangle = \frac{C}{\langle y \bullet a \rangle \frac{PE}{xz}} \qquad \langle y \bullet a \rangle \neq y \tag{7}$$

$$\langle \frac{C}{(x:PE|C|A)(y:PE|C|A|T)^*} \bullet a \rangle = \frac{C}{\langle x \bullet a \rangle y} \qquad \langle x \bullet a \rangle \neq x \tag{8}$$

$$\langle \frac{A}{x_1^*(y_1:PE|C|A)x_2^* \ldots x_n^*(y_n:PE|C|A)x_{n+1}^*} \bullet a \rangle = \frac{A}{\langle y_1 \bullet a \rangle \langle y_2 \bullet a \rangle \ldots \langle y_n \bullet a \rangle} \qquad \langle y_i \bullet a \rangle \neq y_i \wedge \langle x_i \bullet a \rangle = x_i \tag{9}$$

$$\langle \frac{PE}{(x:PE|C|A|T)^*} \bullet * \rangle = \frac{PE}{x} \tag{10}$$

$$\langle \frac{C}{(x:PE|C|A|T)^*} \bullet * \rangle = \frac{C}{x} \tag{11}$$

$$\langle \frac{A}{(x:PE|C|A|T)^*} \bullet * \rangle = \frac{A}{x} \tag{12}$$

Figure 6: Reduction rules for simplifying dialog specifications.

transformation rules. Finally $x \vdash A$ indicates that the subdialog x contains an alternator somewhere inside it (i.e., at a level below the top-level stager). Regular expressions are used, when possible, to simplify presentation.

These rules are shown in Figure 6, numbered for convenience. Rule 1 represents a simplification rule that should be repeatedly applied to the result of every reduction until no further simplification is possible. The rest of the rules represent the reductions that should take place when some input is given. These rules are listed in order of precedence, so that the first applicable one fires.

Rule 2 generates the empty dialog (θ) when the given input matches the only remaining prompt, irrespective of the stager. Rules 3–5 test if the given input is legal under the current top-level stager and generate a pruned dialog. Rule 6 is specifically designed for dialogs involving non-categorical variables, and is discussed further below. Rule 7 handles the type of transformation such as in the breakfast dialog below. Similar transformations for a top-level C and A stagers are given by rules 8 and 9. Rule 10 will fit any input to a dialog script that is being staged with a partial evaluator; respectively for rules 11 (currier) and rule 12 (alternator). These three rules represent the transformation that occurs when no input can be filled, with the transformation simply generating the original {dialog script, interaction stager} pair. The one other aspect of these rules that needs to be mentioned is that the result of concatenating θ to any $(x : PE|C|T)$ is x, which will occur when a dialog or subdialog is completed.

The function of rule 6 is worth explaining in more depth. In general, once a subdialog is entered, the user should have to complete the subdialog before continuing with the rest of the dialog. This requirement is enforced by rule 7. When using the alternator however, this restriction is not always desirable. In some situations, the dialog designer may want to only allow the user to pick one choice from a list of possible inputs to a dialog, but may still want the top-level dialog to continue without entering the subdialog. This is especially useful in non-categorical websites. Rule 6 enables this behavior by checking for a list of alternators on the far left of the list of subdialogs (when using a PE stager), and uses this information as a cue to not enter these subdialogs. This allows the dialog designer to choose either behavior as he desires.

We now demonstrate the staging of the interaction in Fig. 2 using the transformation rules (see Fig. 7). The beginning dialog is similar to the modeling of the website shown earlier, but to trigger the application of rule 6 the subdialogs are written in the form: $\frac{PE}{\text{<page><linktext>}}$ with the link text appearing to the *right* of the page description (the meaning is still preserved since the stager is a PE). These substructures thus represent a link label and the page that the label leads to (when dealing with a website that is not in tree form, the page could be a reference to a page defined elsewhere). Recall that this 'page' is typically a subtree in the original website.

In order to preserve space, subdialogs that will not be entered are not completely shown. In some cases though, they are assumed to contain certain attributes that prevent them from being removed from the dialog. For example, it is assumed the Alabama ('al') has some politicians who are Democrats ('d'), and thus remains in the dialog after the utterance 'd.' On the other hand, Alaska ('ak'), does not, so it is simplified out of the dialog when the user says 'd'. While Alabama does have Democratic politicians, none of them are senators, and is thus simplified out of the dialog after the user specifies Senator ('s').

After every input, the appropriate transformation rules are applied, and the resulting {dialog script, interaction stager} pair is simplified. The simplified dialog is then used as the model to ac-

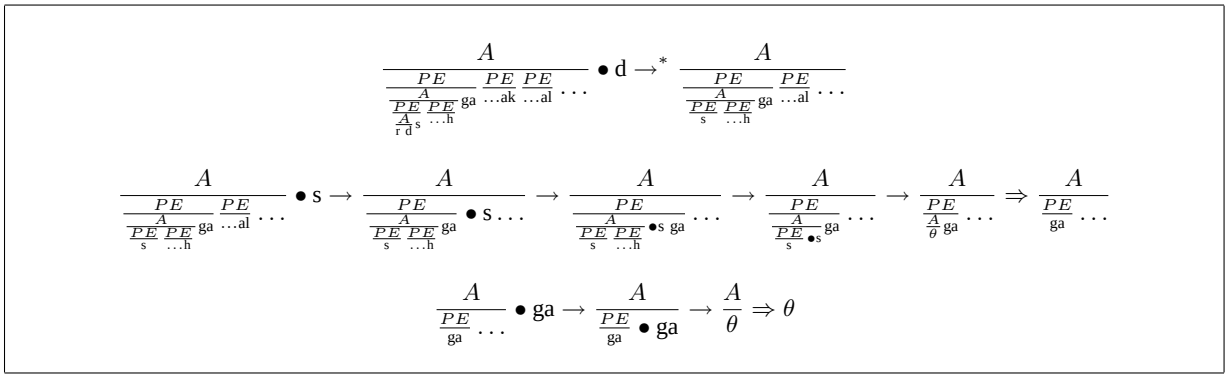


Figure 7: Staging the interaction of Fig. 2 using transformation rules.

program	backward (6, vol)	forward (1, h)
(1) read (r, h);	read (r, h);	read (r, h);
(2) cArea = π*r^2;	cArea = π*r^2;	
(3) sArea = 2*cArea+2*r*π*h;		sArea = 2*cArea+2*r*π*h;
(4) vol = cArea*h;	vol = cArea*h;	vol = cArea*h;
(5) print (sArea);		print (sArea);
(6) print (vol);	print (vol);	print (vol);

Figure 8: Illustration of program slicing. (left) A program which takes the radius and height of a cylinder as input and computes and prints its surface area and volume. (center) A static backward slice (of left) w.r.t. (6, vol). (right) A static forward slice (of left) w.r.t. (1, h) (variable key: r = radius; h = height; cArea = circle area; sArea = surface area; vol = volume).

cept the next piece of input. This process is carried out until the dialog is reduced to θ, indicating that the dialog has completed. Each $\rightarrow$ in Fig. 7 describes a transformation based on user input, and each $\Rightarrow$ describes a simplification of the dialog structure.

For the first interaction, all of the transformations are not shown for readability. The second interaction is shown in more detail. First rule 9 is applied, which removes Alabama from the dialog, since it is no longer applicable. This is followed by rule 6 to begin simplifying the Branches page, followed by rule 9, which removes House ('h') from the dialog. The next transformation is based on rule 2, which removes the need to say Senate from the dialog, since it has already been said. The resulting dialog is then simplified as per rule 1 (denoted by $\Rightarrow$), to yield the new dialog.

Building a Robust Transformation Engine

We have presented a formal theory for reasoning about hierarchical staging notation and for simplifying dialogs; to target this theory for real-time interaction management in websites, we represent interactions with websites as XML documents and implement the transformations using XSLT technology. The XML documents summarize the hyperlink structure, the hierarchical staging notation, and indirectly the vocabulary comprising of legal patial input. While XML documents obey a tree-structured model, notice that we can use id's and refid's to factor crosslinks, subdialogs in other sites, and hence effectively model DAGs as well.

Recall that the stagers serve a dual role in our framework: they enforce acceptability criteria on user input (i.e., by distinguishing valid from invalid inputs) and they also capture the underlying program transformation that must be performed, for valid input. Therefore, in implementing the framework, we first verify user input for legality (e.g., when the user says 'Democrat' out-of-turn, we consult the dialog specification to determine if this is acceptable) and then perform the desired transformation to accommodate this partial input. Interestingly, all of I, PE, C, and A staging functionality can be implemented in terms of a more general program transformation called *slicing*.

Program slicing [8] is a technique used to extract statements, which *may* affect or be affected by the values of variables of interest computed at some point of interest, from a program. A slice of a program is taken w.r.t. a (a point of interest, a variable of interest) pair, referred to as the *slicing criterion*. The point of interest is specified with a line number from the program. The resulting slice consists of all program statements which may affect or be affected by the value of the variable at the specified point.

Fig. 8 illustrates simple program slicing. Slices such as that shown in Fig. 8 (center), are called 'backward slices' [8]. The slice is *backward* since this is the direction in which dependencies are followed to their sources in the program. Contrast this with a *forward slice* [25] which consists of the program statements affected by the value of a particular variable at a particular statement (see Fig. 8—right). Backward slices contain data and control predecessors, while forward slices consist of data and control successors. For an introduction to program slicing and applications, we refer the interested reader to [8].

The transformation of websites, for I, C, PE, and A stagers, can be modeled as a forward slice followed by a backward slice (w.r.t. corresponding program variables). Intuitively, given valid input, a forward slice is performed w.r.t. the corresponding program variable to determine the terminal webpages that are reachable from that point. These webpages are collected and back-propagated via backward slicing, so that only those paths that reach these pages are retained. Notice that these two operations implicitly capture exclusions among program variables; e.g., when the user says 'Democrat' the slices will remove any program segments that involve Republicans. XSLT's support for pattern-oriented programming is particularly advantageous here since both forward and backward slicing can be captured in the form of ancestor or descendant axis types in location paths. Such a combination of forward and back-

ward slicing is closely related to other operators for pruning information hierarchies, namely Sacco's zoom operator [40].

3.2 Interaction Interfaces

To exercise our staging transformation framework, we developed two different input interfaces. The first (*Extempore*), leverages users' familiarity with toolbar interfaces, and provides a way to supply out-of-turn textual input. The second (*SALTII*, for SALT Interaction Interface, pronounced 'salty') utilizes a rapidly emerging technology for integrating speech into web browsing. Both interfaces employ a common JavaScript toolkit which handles communication with the interaction manager (see next section). The toolkit also is designed to reduce the development cost for future interaction interfaces (e.g., PDAs and 3G phones).

Extempore

The Extempore toolbar was developed using the XML User interface Language (XUL) and JavaScript for the cross-platform Mozilla web browser. It was designed to be non-invasive and to become active only when the user is visiting a website capable of out-of-turn interaction. By displaying a lightweight, text-based interface, Extempore leverages users' prior knowledge to provide a familiar and easy method of interaction. We will see an illustration of Extempore in a new case study depicted in Fig. 10. It is important to note that Extempore is embedded in the web browser, and not the site's webpages. It also is not a site-specific search tool that returns a flat list of results (akin to the Google toolbar).

SALTII

The SALTII interface is built using the SALT XML-based markup language, allowing one to embed speech tags in HTML to realize webpages capable of speech input and output. The current SALTII implementation requires the SALT voice recognition plugin for Microsoft Windows Internet Explorer 6.0. Using this interface, users could potentially carry out an entire dialog with speech alone, using speech for not only out-of-turn interaction, but in-turn as well. This interaction interface is patterned after speech recognition technologies such as VoiceXML.

Future Interfaces

The JavaScript toolkit for the above interfaces was designed with future development in mind and factors the entire system to roughly five simple functions. Since it is developed using the vcXML-RPC JavaScript library, the functionality of the interaction manager can be remotely accessed, enabling shared context scenarios [14]. This toolkit has been made available at the project's website (http://pipe.cs.vt.edu) for developers of other interaction interfaces.

3.3 Interaction Manager

The interaction manager primarily co-ordinates communication between the transformation engine and the interaction interfaces. Recall that the staging transformations framework treats in-turn inputs no differently from out-of-turn inputs, so it is desirable that the interaction manager also preserve this uniformity. We first outline the overall process by which interaction is established and managed, followed by descriptions of the four constitutent subsystems (see Fig. 9).

Preparing for Out-of-Turn Interaction

To situate the interaction manager as a dialog facilitator of both in-turn and out-of-turn inputs, we have investigated a variety of mechanisms, ranging from those that involve the full participation of the website, to proxy-based bypass schemes. The former requires a DNS re-direction so that HTTP GET requests are forwarded to the interaction manager (notice that out-of-turn inputs are received directly from the interaction interfaces). This solution also has the attractive property that mixed-initiative interaction can be enabled at as fine or coarse a level of granularity as desired (e.g., it can be enabled for only certain subtrees). The proxy-based approach is a less configurable solution and must be targeted very carefully, to avoid loss of functionality. We adopt the former approach in this paper. Once such an initial handshake is established, the interaction manager is responsible for providing concurrent access to the transformation engine, from potentially multiple interaction interfaces.

The interaction manager, now placed in the loop, evaluates if out-of-turn interaction is possible, activates the interaction interfaces as appropriate, and mediates all interactions from this point. Notice that intermediate dialog states might not correspond to any of the site's existing webpages (especially after some out-of-turn interaction), so the interaction manager must mediate the dialog to the fullest. For websites such as PVS, where the content sought lies at the leaf level, notice that the interaction manager need revert back to the original site only to display the leaf page(s).

Content Handling

Content handling determines the feasibility of out-of-turn interaction, caches dialog states, and ensures currency of site representations. It is also responsible for retrieving, caching, and updating content from websites.

To determine the feasibility of out-of-turn interaction, the content handler uses a simple HTTP GET request for a well-formed XML document located at a defined URL. This document, if it exists, is meant to supply the representation of the site, and when annotated with stager tags, helps initialize the dialog representation. For web sites that utilize dynamically generated content and structure, this URL could link to a script which retrieves a snapshot of the current structure and content at the time of request. The content file is then stored in a local cache database to facilitate fast transformation computations, for a duration that is either set by the content provider or a system default, the former of which is suitable for dynamically generated sites. The content handler also initiates the activation of the Extempore toolbar or the SALT tags, as appropriate. From this point, the content handler is responsible for ensuring the currency of the representation and re-retrieving the file as appropriate.

Notice that caching is trivially implemented by associating intermediate dialog states to content files generated over the course of an interaction. A more sophisticated solution is to develop a caching policy that exploits the structure of program transformations. For instance, if a user is requesting a partial evaluation w.r.t. 'Democratic Senators,' but the cache only contains a document that has been evaluated w.r.t. 'Democrat,' we can partially evaluate this document internally w.r.t. the remaining input (namely, 'Senate'), thus removing the need to partially evaluate from the root document. While reducing storage complexity this approach also creates interesting design tradeoffs (including concerns about session and user security).

Transformation Dispatch

The transformation dispatch is responsible for handling communication with the transformation engine. It handles connecting to the transformation engine as well as notifying the interaction interfaces if such a connection cannot be made (When the transformation engine receives partial input, recall that it does not know, and need not know, whether the partial input is a result of browsing or of supplying some information out-of-turn). Finally, transformation dispatch

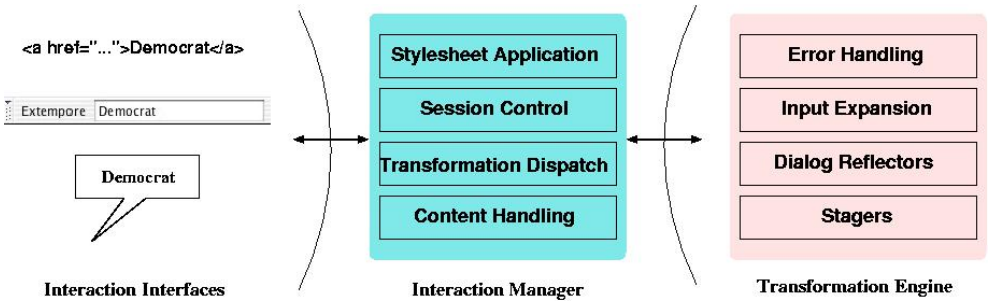


Figure 9: Multimodal web interaction framework architecture, depicting the central role played by the interaction manager.

supports the marshalling and un-marshalling of transformation requests into messages, as well as the transmission and reception of those messages.

Session Control

Session control has an interesting responsibility that differs from most web systems' concept of session management. Notice that our notion of 'state' in a dialog is just the staging representation, since it succinctly summarizes all remaining dialog options. Furthermore, the transformation engine does not explicitly manipulate state and is hence, a purely functional[2] entity. Thus the goal of session control is merely to distinguish one user's interaction from another. Due to our requirement for handling in-turn and out-of-turn inputs uniformly, session tokens (we use a ten decimal digit identifier) are required to be kept in two different places, the interaction interface and the browser itself. This two-headed session format negates the application of most modern session management packages, which are primarily concerned with tracking browsing interactions. A session manager was specifically designed to handle this issue as well as to handle the normal session management issues (e.g., back button browsing and threaded browsing).

Stylesheet Application

Stylesheet application is responsible for transforming the information returned from the transformation engine into the site's native presentation format. In addition, it must introduce suitable grammar tags into the HTML page (for the voice interface) by analyzing the remaining dialog options. Currently we support (x)HTML, WML, SALT, SVG, or any XML-based presentation format, and this is determined by the interaction interface making the request. Default stylesheets for these formats are made publicly available from the project website.

3.4 Miscellaneous Design Decisions

Input Validation

Input validation for the Extempore toolbar interface is performed directly by the reduction rules of Fig. 6 whereas input validation for the voice interface is trapped on the client side by suitable generation of an SRGS grammar (at the Stylesheet Applicator).

Orienting Users

In order for out-of-turn interaction to be effective, the user must have a basic understanding of what can be said. This is a well-acknowledged issue by the speech interfaces community; as Yankelovich [47] points out, 'the functionality of [such] 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 conversational input, because it merely allows a site's existing navigation structure to be realized in a different order.

To better orient the user in their interactions, we implemented an 'Input So Far:' feedback label (in the browser status bar) in both implementations that summarizes the partial input supplied thus far (e.g., see Fig. 2). We also support meta-dialog enquiries (e.g., 'What may I say?') via dialog reflectors. This is activated through a '?' button in Extempore and by a spoken query in SALTII, and involves traversing the current representation for determining legally specifiable inputs.

3.5 Implementation Details

The elegance of our implementation is reflected in the minimal codebase required. The transformation engine is built using C and the libxml, libxslt libraries, and is under 500 lines of code. This invokes a 120-line transformation template for out-of-turn interaction, with 50 lines for handling input expansion and dialog reflection capabilities. The transformation engine is wrapped using SOAP (Simple Object Access Protocol), effectively making it a web service from the perspective of the interaction manager. External communication thus happens through SOAP messages. Extempore is implemented in 50 lines of XUL and SALTII only requires lines of code proportional to the size of the underlying grammar. The JavaScript toolkit supporting new interfaces is about 300 lines of code. The interaction manager was developed using the PHP scripting language, and is implemented in a total of 375 lines of code, barring two external open source libraries (the NuSOAP library and the vcXML-RPC library, for communication between other layers).

4. APPLICATION CASE STUDIES

Besides the Vote Smart study used as our running example, we implemented multimodal web interfaces for selected subtrees of the ODP hierarchy (dmoz.org) and the CITIDEL repository of research papers (citidel.org). Due to space considerations, we only discuss the Vote Smart and ODP applications here. Dialogs in both applications were initialized as shown in the starting dialog representation of Fig. 7.

Figs. 1 and 2 have already depicted interactions with the multimodal web interface to Vote Smart. Here, we actually employed a four-level modeling (state, branch, party, district/seat) although we have described only the first three levels (state, branch, party) thus far in this paper, for ease of description.

Fig. 10 depicts a multimodal web interaction with the ODP HOME subtree, this time using Extempore. ODP is a human-compiled directory of the web and is a constantly evolving categorization of websites. A significant proportion of the links in ODP are actually symbolic links, where one subtree points to another, under a

[2]We use the word 'functional' in a programming languages connotation (e.g., Haskell).

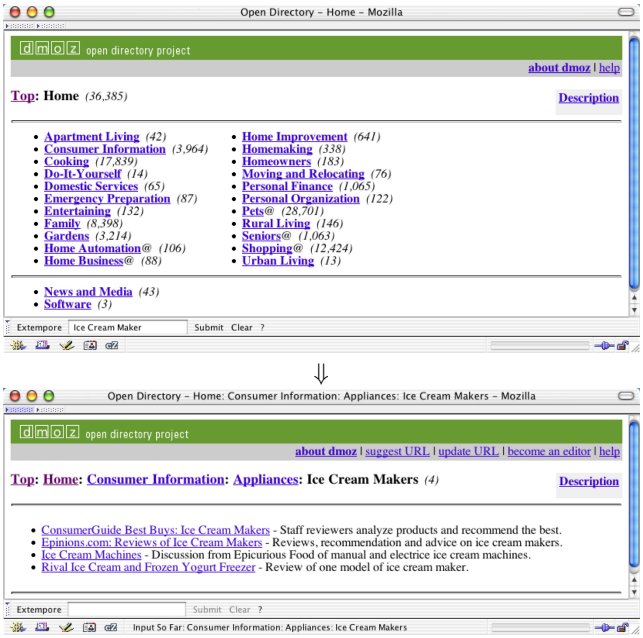


Figure 10: Out-of-turn interaction with the HOME subtree of the ODP hierarchy. In the top window, the user enters 'Ice Cream Maker' out-of-turn using the Extempore toolbar, producing the page shown below. Notice that two functional dependencies are automatically triggerred: {Ice Cream Makers} → {Appliances, Consumer Information}.

different classification. For instance, 'Recreation: Martial Arts' actually points to the 'Martial Arts' subtree physically located under 'Sports.' It can be argued that the presence of symbolic links is actually foretelling of the mental mismatch anticipated by directory compilers, and hence *in-turn* mechanisms have been hardwired to better orient users. The use of out-of-turn interaction, in effect, obviates the need for symbolic links by increasing the flexibility of communicating partial input.

Input Expansion

In building both these applications, we have realized the importance of input expansion strategies to capture semantic relatedness among dialog options. An example of a semantic constraint is when the user says 'Senior seat' out-of-turn, we can also infer the choice of 'Senate' (as opposed to 'House'). This is because the functional dependency {Senior seat} → {Senate} holds in the underlying domain. Such dependencies can be automatically inferred by simple association mining methods [31]; we identified 125 of them in the Vote Smart site and more than 3,600 in just the ODP's Home subtree. An elegant by-product of such automatic input expansion is that out-of-turn interaction can provide rapid shortcuts to desired leaves, even at the root level. For instance, Washington, D.C. has only one representative and no senators, and hence the following dependency holds: {Washington, D.C.} → {House, Democrat, District at Large}. Therefore, saying/clicking 'Washington, D.C.' at the top-level page uniquely identifies one congressperson and transports the user directly to her webpage. Mining functional dependencies for use in out-of-turn interaction is an interesting topic by itself, and one that we intend to explore fully in a future paper.

Collapsing Transformations

Another practical consideration exists when only one leaf page (e.g., congressperson) remains as the result of a transformation. For instance, in the current political landscape, the Democratic senator from Minnesota is occupying the senior seat, therefore saying 'Minnesota Democrat Senator' uniquely identifies a politician. In such a case, we collapse the remaining series of hyperlinks (involving seat) and return the webpage of the official directly to the user (relieving her from having to click through these links).

Notice that no information is lost as a result of either automatic input expansion or collapsing as a leaf webpage contains the values for the facets under which it is classified.

5. EVALUATION

There are many ways to evaluate the framework presented here. For instance, we can study user experiences with the deployed applications, characterize the framework by its support for modeling, and also investigate the ease of implementing new applications within the framework.

User experiences with out-of-turn interaction are described in [33]; 25 users were given information-finding tasks about U.S. politicians and were free to use either in-turn or mixed-initiative interactions to complete these tasks. Some of these tasks were *non-oriented* (meaning they could be performed with browsing alone, if desired) and some were *out-of-turn-oriented* (meaning they would be cumbersome to perform via plain browsing). We found that 100% of the users utilized the out-of-turn interfaces when presented with an out-of-turn-oriented task. Since the task type was not disclosed *a priori*, this result demonstrates that users are adept at discerning when out-of-turn interaction is desirable. Extempore and SALTII interfaces were utilized equally effectively.

From an information-seeking standpoint, it is easy to see that our use of out-of-turn interaction dramatically increases the number of ways to reach a given webpage. For example, in Vote Smart, the original 540 browsing sequences are now a small subset of 12,960 realizable sequences (540× 4!), where the length of each interaction sequence is constant (this is not always the case; e.g., ODP). This is a 2,300% increase in the number of sequences supported! From a representational perspective, such increase comes through *without* modeling any more than the original 540 sequences. This is in contrast to faceted browsing [24] which would cause an exponential blowup in site structure. In fact, we do not even model the original 540 distinctly as our nesting of dialog choices factors the representation. Similarly, in ODP we observe a 44,400% increase in the number of sequences supported in relation to the 1,411 original browsing sequences.

New applications, especially hierarchical sites, are easily targeted using the software framework presented here (see project website for more information). Taxonomies (e.g., gams.nist.gov), LDAP directories, database-driven indices (e.g., acm.org/dl), and bioinformatics ontologies (e.g., www.gene-ontology.org) are ideal for modeling in the staging transformations framework. The property that these information sources share is that they all foster (and sometimes require) *focused* information-seeking behavior.

In contrast, consider a site such as the Internet movie database (www.imdb.com), which is meant for *exploratory* browsing and uses connections in social networks as a navigational metaphor. Such a site is more cumbersome to model using staging transformations.

Another shortcoming of staging transformations is that the partial input suppliable by the user is primarily of a *declarative* nature, and hence does not adequately support *procedural* tasks. For instance, consider a task such as 'Find the political party of the senior senator representing the only state which has congresspeople from the Independent party.' It involves finding Vermont as the answer to the 'only state part' (via out-of-turn interaction) and then using it in

another interaction to find the political party of the senior senator from that state. When tested with participants, we found that only 50% of the users successfully completed this task [33], because the rest of them were attempting to continue the interaction after answering 'Vermont' for the first part of the question. To support such prolonged dialogs, staging tranformations must provide *constructive operators*; all staging operators considered in this paper are *destructive* in that valid partial input causes pruning of remaining dialog options.

A final technological limitation pertains to speech interaction in large sites (e.g., higher subtrees of ODP). In a grammar based approach such as used in SALT, large sites can involve a dramatic growth in vocabulary, especially when we use a PE stager at a high-level of composition. More robust statistical methods or other dialog abstraction capabilities must be investigated.

6. RELATED RESEARCH

Web interaction management emerged as a legitimate area of research ever since researchers attempted to build stateful and responsive web applications on top of stateless protocols such as HTTP [43]. Interaction management research is typically concerned with issues such as automated delivery of static as well as dynamically generated pages [36], sessioning, accommodating simultaneous users, concurrency control, stateful implementation of client-side functionality such as cloning windows and pursuing back buttons [38], and domain-specific language (DSL) support for targeted applications such as form-field interaction and database-centered services (e.g., `<bigwig>` [10] and MAWL [5]). Interestingly, a significant body of this research has involved concepts from functional programming (e.g., continuations, currying) [22, 37, 38].

Our work embraces this tradition and proposes the use of program transformations for staging web dialogs. It thus casts the problem of dialog control and management in a purely functional framework, with attendant benefits. The modeling of interaction undertaken here is reminiscent of the approach advocated by Marchionini for designing information systems [29]. It also addresses Dumais's vision of a tighter coupling between structure and search in information access [20].

The dialog notation presented here is part of a larger effort from our group to ease the specification and realization of mixed initiative dialogs [13]. This work has similar motivations to other formalisms aimed at capturing interactions with computer systems, e.g., GOMS [15], strategies and scripts for information-seeking (COR; [7]), the speech acts framework for office communication systems [45], and structured discourse theory [23]. By emphasizing turn management more than other aspects of dialogs (e.g., intention, plans, and goals), staging transformations is of reduced scope than these efforts but more targeted for web interaction paradigms.

Another pertinent area of related research can be found in the adaptive hypermedia community [3, 9, 11]. Here, an explicit user model is built (e.g., from past interactions) which is then used as the driver to support adaptive presentation and personalized interaction. Our philosophy, on the other hand, is that by enriching the expressiveness with which users can supply partial input, we can help them achieve their information-seeking goals better. Needless to say, these two views can be fruitfully integrated.

The software framework proposed here is complementary to other frameworks for interaction co-ordination [19, 27, 42], functional web adaptation [16] and re-engineering of websites [22]. However, the specific setting assumed here (i.e., out-of-turn interaction) is different from those considered in these works. Our framework is closer in motivation to systems like VoiceXML [30] which provide support for creating mixed-initiative dialogs.

7. DISCUSSION

We have described a software framework for multimodal out-of-turn interaction, thus laying the foundation for creating mixed-initiative dialogs with websites. Our usage of out-of-turn interaction is optional, unintrusive, and can be integrated into browsing experiences at the user's discretion. It also promotes a novel interpretation of multimodal paradigms. For the designer, the framework simplifies the process of integrating in-turn and out-of-turn interaction using a uniform handling of both dialog specification and implementation. Minimal modeling is required to re-engineer existing sites.

The multimodal web view realized in this paper also extends the idea of web access via voice [21, 39] and could be usefully applied in a variety of mobile browsing contexts (e.g., see [4, 12, 18]). The framework was developed with future web access paradigms in mind, beyond cell phones and PDAs. The factoring of the system architecture into three components means that content providers and developers need only concern themselves with the interfaces and modalities they wish to support. An additional possibility we are exploring is the idea of supporting interjection-style out-of-turn interaction, wherein the browser can dynamically update content *while* the user is supplying out-of-turn input. This feature is currently projected to utilize the SSU [44] software framework to provide real-time feedback to users, and alert them if something is amiss. The current model of interaction management uses a coarser level of dialog unit at which feedback is provided. Exploring these issues will undoubtedly open up new research directions and additional applications for multimodal web interfaces.

8. ACKNOWLEDGEMENTS

The authors thank C. Queinnec (Université Paris) for helpful discussions and comments. This work is supported in part by US National Science Foundation grant IIS-0136182.

9. REFERENCES

[1] Speech Application Language Tags (SALT) Specification. Technical report, SALT Forum, July 2002. Version 1.0.

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

[3] E. André and T. Rist. From Adaptive Hypertext to Personalized Web Companions. *CACM*, Vol. 45(5):pages 43–46, May 2002.

[4] Y. Aridor, D. Carmel, Y. S. Maarek, A. Soffer, and R. Lempel. Knowledge Encapsulation for Focused Search from Pervasive Devices. In *Proc. WWW10*, pages 754–764, 2001.

[5] D. L. Atkins, T. Ball, G. Bruns, and K. Cox. Mawl: A Domain-Specific Language for Form-Based Services. *IEEE Transactions on Software Engineering*, Vol. 25(3):pages 334–346, May–June 1999.

[6] J. Axelsson, C. Cross, H. Lie, G. McCobb, T. Raman, and L. Wilson (eds.). Xhtml+voice profile 1.0. W3C Note, December 2001.

[7] N. J. Belkin, C. Cool, A Stein, and U. Thiel. Cases, Scripts, and Information Seeking Strategies: On the Design of Interactive Information Retrieval Systems. *Expert Systems with Applications*, Vol. 9(3):pages 379–395, 1995.

[8] D. W. Binkley and K. B. Gallagher. Program Slicing. In *Advances in Computers*, volume 43, pages 1–50. 1996.

[9] P. De Bra, P. Brusilovsky, and G.-J. Houben. Adaptive Hypermedia: From Systems to Framework. *ACM Computing Surveys*, Vol. 31(4es), December 1999. Article No. 12.

[10] C. Brabrand, A. Møller, and M. I. Schwartzbach. The <bigwig> Project. *ACM Transactions on Internet Technology*, Vol. 2(2):pages 79–14, May 2002.

[11] P. Brusilovsky. Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, Vol. 11(1–2):pages 87–110, 2001.

[12] O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Seeing the Whole in Parts: Text Summarization for Web Browsing on Handheld Devices. In *Proc. WWW10*, pages 652–662, 2001.

[13] R. Capra, M. Narayan, S. Perugini, N. Ramakrishnan, and M. A. Pérez-Quiñones. The Staging Transformation Approach to Mixing Initiative. In G. Tecuci, editor, *Working Notes of the IJCAI 2003 Workshop on Mixed-Initiative Intelligent Systems*, pages 23–29. AAAI/MIT Press, August 2003.

[14] R. Capra, M. A. Pérez-Quiñones, and N. Ramakrishnan. WebContext: Remote Access to Shared Context. In *Proc. PUI*, November 2001.

[15] S. K. Card, T. P. Moran, and A. Newell. Computer Text-Editing: An Information-Processing Analysis of a Routine Cognitive Skill. *Cognitive Psychology*, Vol. 12:pages 32–74, 1980.

[16] J. Chen, B. Zhou, J. Shi, H. Zhang, and Q. Fengwu. Function-Based Object Model Towards Website Adaptation. In *Proc. WWW10*, pages 587–596, 2001.

[17] Y. Chen, W.-Y. Ma, and H.-J. Zhang. Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices. In *Proc. WWW12*, pages 225–233, 2003.

[18] D. Cohen, M. Herscovici, Y. Petruschka, Y.S. Maarek, and A. Soffer. Personalized Pocket Directories for Mobile Devices. In *Proc. WWW11*, pages 627–638, 2002.

[19] A. Coles, E. Deliot, T. Melamed, and K. Lansard. A Framework for Coordinated Multi-Modal Browsing with Multiple Clients. In *Proc. WWW12*, pages 718–726, 2003.

[20] S. Dumais. Tightly Coupling Structure and Search. In *Proc. SIGIR Workshop on Information Reduction*, July 1997.

[21] J. Freire, B. Kumar, and D. Lieuwen. WebViews: Accessing Personalized Web Content and Services. In *Proc. WWW10*, pages 576–586, 2001.

[22] P. Graunke, R. Findler, S. Krishnamurthi, and M. Felleisen. Automatically Restructuring Programs for the Web. In *Proc. ASE*, November 2001.

[23] B. J. Grosz and C. L. Sidner. Attention, Intentions, and the Structure of Discourse. *Computational Linguistics*, Vol. 12:pages 175–204, 1986.

[24] M. A. Hearst, A. Elliott, J. English, R. Sinha, K. Swearingen, and K.-P. Yee. Finding the Flow in Web Site Search. *CACM*, Vol. 45(9):pages 42–49, September 2002.

[25] S. Horwitz, T. Reps, and D. W. Binkley. Interprocedural Slicing Using Dependency Graphs. *ACM Transactions on Programming Languages and Systems*, Vol. 12(1):pages 26–60, January 1990.

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

[27] I.-Y. Ko, K.-T. Yao, and R. Neches. Dynamic Coordination of Information Management Services for Processing Dynamic Web Content. In *Proc. WWW11*, pages 355–365, 2002.

[28] J. Lai. Conversation Interfaces. *CACM*, Vol. 43(9):pages 24–27, September 2000.

[29] G. Marchionini. *Information Seeking in Electronic Environments*. Cambridge Series on Human-Computer Interaction. Cambridge University Press, 1997.

[30] S. McGlashan, D. Burnett, P. Danielsen, J. Ferrans, A. Hunt, G. Karam, D. Ladd, B. Lucas, B. Porter, K. Rehor, and S. Tryphonas. Voice eXtensible Markup Language: VoiceXML. Technical report, VoiceXML Forum, October 2001. Version 2.0.

[31] B. Mobashier, R. Cooley, and J. Srivastava. Automatic Personalization Based on Web Usage Mining. *CACM*, Vol. 43(8):pages 142–151, August 2000.

[32] B. A. Myers and M. Beigl. Guest Editors' Introduction: Handheld Computing. *IEEE Computer*, Vol. 36(9):pages 27–29, September 2003.

[33] 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 (CoRR), December 2003.

[34] S. Perugini and N. Ramakrishnan. Personalizing Interactions with Information Systems. In *Advances in Computers*, volume 57: Information Repositories, pages 323–382. September 2003.

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

[36] J. Pokorny. Static Pages are Dead: How a Modular Approach is Changing Interaction Design. *ACM Interactions*, Vol. 8(5):pages 19–24, September–October 2001.

[37] D. Quan, D. Huynh, D. R. Karger, and R. Miller. User Interface Continuations. In *Sixteenth ACM Symposium on User Interface Software and Technology (UIST)*, November 2003.

[38] C. Queinnec. The Influence of Browsers on Evaluators or, Continuations to Program Web Servers. In *Proc. ICFP*, pages 23–33, September 2000.

[39] S. Rollins and N. Sundaresan. AVoN Calling: AXL for Voice-Enabled Web Navigation. In *Proc. WWW9*, 2000.

[40] G. M. Sacco. Dynamic Taxonomies: A Model for Large Information Bases. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 12(3):pages 468–479, May–June 2000.

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

[42] J. Steinberg and J. Pasquale. A Web Middleware Architecture for Dynamic Customization of Content for Wireless Clients. In *Proc. WWW11*, pages 639–650, 2002.

[43] J. Veitch. A Conversation with Paul Graham. *CACM*, Vol. 41(5):pages 52–54, May 1998.

[44] K. Wang. A Study of Semantic Synchronous Understanding on Speech Interface Design. In *Proc. UIST'03*, November 2003.

[45] T. Winograd and F. Flores, editors. *Understanding Computers and Cognition – A New Foundation for Design*. Addison-Wesley, Reading, PA, 1987.

[46] S. A. Wolfman, T. Lau, P. Domingos, and D. S. Weld. Mixed Initiative Interfaces for Learning Tasks: SMARTedit Talks Back. In *Proc. IUI*, pages 167–174, 2001.

[47] N. Yankelovich. How Do Users Know What To Say? *ACM Interactions*, 3(6):pages 32–43, November–December 1996.