The Staging Transformation Approach to Mixing Initiative

Robert Capra, Michael Narayan, Saverio Perugini,
Naren Ramakrishnan, and Manuel A. Pérez-Quiñones
Department of Computer Science
Virginia Tech, Blacksburg, VA 24061

Abstract

Mixed-initiative interaction is an important facet of many conversational interfaces, flexible planning architectures, intelligent tutoring systems, and interactive information retrieval systems. Software systems for mixed-initiative interaction must enable us to both operationalize the mixing of initiative (i.e., support the creation of practical dialogs) and to reason in real-time about how a flexible mode of interaction can be supported (e.g., from a meta-dialog standpoint). In this paper, we present the *staging transformation* approach to mixing initiative, where a dialog script captures the structure of the dialog and dialog control processes are realized through generous use of program transformation techniques (e.g., partial evaluation, currying, slicing); this allows control to be cast as the process of moving from one dialog script to another. In this approach, operationalizing mixed-initiative interaction becomes the task of finding a suitable program transformation to stage the interaction between the two participants. We highlight the advantages of this approach and present its realization in various modalities for information seeking dialogs. We also outline how high-level reasoning capabilities about dialogs can be provided in the staging transformation framework.

1 Introduction

Mixed-initiative interaction [Allen *et al.*, 1999] is an important facet of many conversational interfaces, flexible planning architectures, intelligent tutoring systems, and interactive information retrieval systems. In conversational interfaces, MII provides the ability to accommodate the shift of control between agents participating in a dialog. In planning architectures such as TRIPS [Allen *et al.*, 2001], MII naturally arises from considerations of how to improve the effectiveness of the planning system and the desire to achieve contextual goals. MII for intelligent tutoring systems [Graesser *et al.*, 2001] helps accommodate the diverse needs of participants and enables the software system to react or act proactively, as appropriate. Finally, in interactive information retrieval [Brunner *et al.*, 1992], the goal is to foster a dialog between the information seeker and the retrieval engine, in order to overcome the mismatch between how the information space is represented and how the information seeker desires to navigate or browse it. In all of these applications, MII is crucial for fostering an active dialog between the participants, and its use encourages an interactive approach to problem solving; in particular, it avoids the tendency to seek ‘one-shot’ solutions, is tolerant of failures, permits changes in focus, and encourages an evolving understanding of the underlying context.

MII is a multi-faceted concept that can be viewed along several dimensions. For instance, James Allen [Allen *et al.*, 1999] has outlined four levels of mixing initiative that range from an agent merely being able to provide unsolicited information to agents using negotiation to determine who will have the initiative. [Chu-Carroll and Brown, 1997] differentiate between task initiative and dialog initiative. [Novick and Sutton, 1997] describe three elements of MII: choice of task, choice of speaker, and choice of outcome. To systematize the study of MII, researchers have proposed many organizing principles for dialog management systems – one such high-level classification, proposed by Allen, is based on task complexity [Allen *et al.*, 2001].

In Allen’s assessment, as task complexity increases, so does the sophistication of the techniques required to process the dialog. At the lower levels of task complexity (finite-state scripts and frame-based systems), the opportunities and mechanisms to support MII are limited. Allen notes that many finite-state systems are fixed *system*-initiative and that frame-based systems can be fixed *user*-initiative. At the higher levels of task complexity (agent-based and plan-based models [Kitano and Ess-Dykema, 1991]), Allen observes that “the tasks are too complicated to represent as a series of parameterized contexts. In fact, these tasks require the system to maintain an explicit model of the tasks and/or world, reason about these models, ... [and] one also needs to start explicitly modeling the collaborative problems solving process that the system and user engage in”([Allen *et al.*, 2001], p.29).

The experiences reported in this paper primarily grew out of work in information personalization [Ramakrishnan, 2000] and web-based dialog processing [Perugini and Ramakrishnan, 2003], where one of the agents is an information system, and the task complexities of dialogs fall in the lower half of Allen’s classification. At the same time, we have found that having an explicit representation of the dialog can be crucial for supporting and sustaining a responsive and personalized interaction with the user. Our focus has been on representa-

0	**Caller:**	≺calls Joe's Pizza on the phone≻
1	**System:**	Thank you for calling Joe's pizza ordering system.
2	**System:**	What size pizza would you like?
3	**Caller:**	I'd like a sausage pizza, please.
4	**System:**	Okay, sausage.
5	**System:**	What size pizza would you like?
6	**Caller:**	Medium.
7	**System:**	What type of crust do you want?
8	**Caller:**	Deep-dish.
9	**System:**	So that is a medium sausage pizza with deep-dish crust. Is this correct?
(conversation continues to get delivery and payment information)		

```
pizzaorder (size, topping, crust) {
  if (unfilled (size) ) {
    /* prompt for size */
  }
  if (unfilled (topping) ) {
    /* prompt for topping */
  }
  if (unfilled (crust) ) {
    /* prompt for crust */
  }
}
```

Figure 1: (left) Example of a mixed-initiative form-filling dialog. The user takes the initiative in line 3 and specifies a topping when prompted for size; the system registers the out-of-turn input in line 4 and repeats the question about size in line 5. (right) Modeling the pizza dialog as a program parameterized by slot variables that are passed by reference.

tions of dialogs that allow us to both operationalize the mixing of initiative (i.e., permit the creation of practical dialog processing engines) and to reason in real-time about how a flexible mode of interaction can be supported (e.g., from a meta-dialog standpoint).

From the perspective of operationalization, one of the important aspects of MII elusive to capture in a representation is that of control, e.g., who can say what at a given point in the dialog, what should be done if a shift of initiative happens, and so on. Traditionally, mechanisms for handling control in MII have relied on extensive use of state and anticipating and tracking all the ways in which transfers of control can be effected. Researchers such as Bridge [Bridge, 2002] have explored principles of conversational analysis to derive rules that guide the mixing of initiative and control of dialog flow. From the perspective of flexibility of reasoning, the representation has to be malleable enough to permit, for instance, the dynamic insertion of a clarification subdialog before resuming other aspects of information assessment. This requires that the representation also support higher-level reasoning capabilities, such as reflection and goal revision.

We approach mixed-initiative dialog processing from a traditional viewpoint of thinking of dialogs as program scripts representing salient aspects of information communication. However, we do not use the sequentiality of program scripts to capture dialog control; rather the scripts merely capture the structure of the dialog, and dialog control processes are realized through generous use of program transformation techniques (e.g., partial evaluation, currying, slicing) [Jones, 1996; Jones *et al.*, 1993]. This allows control to be cast as the process of transforming one dialog script into another. We refer to this approach as the *staging transformation* framework and argue that it can be used as a powerful organizing principle for mixed-initiative dialogs. In this approach, operationalizing MII becomes the task of applying program transformation techniques (repeatedly) to a representation. In addition, program transformation techniques can also be used for analysis of dialog specifications, providing reasoning and meta-dialog functionality.

The use of staging transformations helps systematically extend the MII capabilities of dialog processing systems that utilize finite-state, frame-based, and sets-of-context approaches. In finite-state and frame-based systems, staging transformations allows designers to specify what informational elements are important to collect in a specified order and what elements may be provided in any sequence (at any time). In the sets-of-context designs [Allen *et al.*, 2001], staging transformations operate at a higher level of dialog organization than informational elements and allow us to use program transformation techniques hierarchically to achieve greater functionality.

2 Basic Approach

Consider how we can support mixed-initiative functionality in a slot filling dialog, e.g., a pizza ordering service (see Fig. 1, left). The traditional approach is to have explicit mechanisms that recognize when a shift of initiative has happened and distinguish between regular slot filling (where the user has provided a responsive input) and out-of-turn processing (for cases when the user has provided an out-of-turn, but in-vocabulary input). By anticipating and modeling the shift of initiative as a transfer of control, the software design makes extensive use of state for tracking the flow of the dialog.

In the staging transformation approach, a form-filling dialog is represented as a program script soliciting input from users (Fig. 1, right), and a program transformation technique is utilized to repeatedly process inputs provided by the users [Ramakrishnan *et al.*, 2002]. One such program transformation technique is *partial evaluation*, an approach to specializing programs given some, but not all, of their input [Jones *et al.*, 1993]. For instance, given the value of `exponent` set to 2 as shown in Fig. 2 (left), the loop can be unrolled and the value of the `prod` variable forward propagated to yield the program in Fig. 2 (right). Partial evaluators are available for C, Scheme, and many other languages.

Fig. 3 describes how partial evaluation can be used to stage the dialog of Fig. 1. At each step of Fig. 3, the first applicable prompt is played and the user's input is utilized by the partial evaluator to produce a revised dialog script. We can think of dialog management as being organized into tuples of three steps each: (i) examine the structure of the dialog script and play a prompt, (ii) collect the utterance from the user, and (iii) apply a program transformation to the script with the user's utterance. The two participants in the dialog are thus the program script interactor (in this case, a *single stepper*) and the user. The partial evaluator plays the role of a *stager* who me-

```
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 2: 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).

```
pizzaorder (size, topping, crust) {
   if (unfilled (size) ) {
      /* prompt for size */
   }
   if (unfilled (topping) ) {
      /* prompt for topping */
   }
   if (unfilled (crust) ) {
      /* prompt for crust */
   }
}
```

```
pizzaorder (size, crust) {
   if (unfilled (size) ) {
      /* prompt for size */
   }
   if (unfilled (crust) ) {
      /* prompt for crust */
   }
}
```

```
pizzaorder (crust) {
   if (unfilled (crust) ) {
      /* prompt for crust */
   }
}
```

$\boxed{\theta}$

Figure 3: Staging the dialog of Fig. 1 (left) using partial evaluation. θ denotes the empty dialog.

diates between the two participants. Thus, partial evaluation is used to support a form of *unsolicited reporting* [Allen *et al.*, 1999].

This approach has two advantages. First, it loses the distinction between a responsive input and an out-of-turn input – both of them require the same processing, namely partial evaluation! Second, since the script is divorced from the program transformation, the parts of the system soliciting input are different from the parts of the system that process a received input. This facet allows us to retain the sequential nature of directed dialog prompting in the representation and yet realize non-sequential behavior by an appropriate sequence of program transformations (in this case, partial evaluations).

3 Application Prototypes and Systems

In this section, we outline two different software implementations of our staging transformation framework and also relate it to the dialog processing engine underlying the VoiceXML framework [McGlashan et al., 2001]. The first implementation, available in Haskell and Scheme, is purely functional and can be used as the substrate for many applications. The second describes an interaction instrument and an instantiation of our framework for website interactions. We then relate our approach to the VoiceXML implementation and finally describe ongoing work using the new SALT (Speech Application Language Tags) standard, which allows the creation of multimodal interfaces where the user can interact via both web hyperlinks and voice.

3.1 Functional Stagers

To support the construction of practical dialogs, we view stagers as mechanisms for functionally specifying the set of interaction sequences that are to be supported. For instance, we can specify the pizza dialog as:

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

which indicates that the dialog comprised of prompts *s* (for size), *t* (for topping), and *c* (for crust) is being staged with a partial evaluator. This specification allows all 3! orderings of the pizza attributes (13, if we allow multiple attributes per utterance) to be achieved, without explicitly programming for them. However, it uses an all-or-nothing stager that cannot enforce (i.e., require) a particular ordering. Using interpretation (I) instead of PE allows us to enforce an ordering, but this enforces only one ordering at a time, causing the dialog to become a directed dialog. Yet another option is to use a curryer (C) for a stager, whose legal inputs can only be prefixes of the original list of arguments; thus only four interaction sequences are supported. There are a number of other program transformations that can potentially be used as stagers, such as slicing (SL) [Binkley and Gallagher, 1996], and deforestation (D). Given a dialog script as input, they enforce/allow different sets of orderings and interactively transform the dialog script based upon user input.

Consider a simple variant of the pizza dialog – 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* [Allen *et al.*, 1999]. 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

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

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

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

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

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

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

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

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

Figure 4: Reduction rules for currying and partial evaluation stagers.

item. Notice that the dialog staged by PE is itself a sequential dialog comprised of stagings by a curryer (C). It should be clear that arbitrarily complex hierarchies of dialog subunits can be specified where the scope of mixing initiative can be tailored to a desired level of granularity (useful in the sets-of-context designs). 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 $c\ a\ b\ d$.

For this form of specification to be effective, we must define the semantics of repeated applications of program transformations to dialog specifications. Fig. 4 outlines a set of transformation rules that describe what happens to a {dialog script, program transformer} pair when a given input is received. For illustration purposes, these rules are summarized for only the PE and C stagers. 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 pizza topping, or egg specification such as 'sunny side up,' or 'scrambled.' 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. 4 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. 5. Each $\Longrightarrow$ in Fig. 5 describes a transformation based on user input and each $\longrightarrow$ describes a simplification of the dialog structure. The rule numbers from Fig. 4 are shown alongside the reductions.

Before we describe implementations of this framework, it is helpful to note that the use of a program transformer such as a PE simplifies some tasks (like turn management in MII) but certain others, such as state maintenance, revisiting 'filled' parts of a dialog, and imposition of orderings have to be handled outside the staging transformation framework; these are situations where we must violate the functional tenet. Nevertheless, by using program transformers to 'compress' specifications of interaction sequences, our approach aids in the automated software engineering of mixed-initiative dialogs, namely bringing specification of dialog structure closer to realization. In Section 4 we also describe how simple reasoning and 'stateful' capabilities can be provided using other forms of transformation techniques.

$$\frac{PE}{\frac{C}{e_1\;e_2}\;\frac{C}{c_1\;c_2}\;\frac{C}{b_1\;b_2}}\cdot c_1 \Rightarrow \frac{C}{\frac{C}{c_1\;c_2}\cdot c_1\;\frac{PE}{\frac{C}{e_1\;e_2}\;\frac{C}{b_1\;b_2}}} \Rightarrow \frac{C}{\frac{C}{c_2}\;\frac{PE}{\frac{C}{e_1\;e_2}\;\frac{C}{b_1\;b_2}}} \qquad \text{Rule 5, Rule 4}$$

$$\frac{C}{\frac{C}{c_2}\;\frac{PE}{\frac{C}{e_1\;e_2}\;\frac{C}{b_1\;b_2}}}\cdot c_2 \Rightarrow \frac{C}{\frac{C}{c_2}\cdot c_2\;\frac{PE}{\frac{C}{e_1\;e_2}\;\frac{C}{b_1\;b_2}}} \Rightarrow \frac{C}{\frac{PE}{\frac{C}{e_1\;e_2}\;\frac{C}{b_1\;b_2}}} \rightarrow \frac{PE}{\frac{C}{e_1\;e_2}\;\frac{C}{b_1\;b_2}} \qquad \text{Rule 6, Rule 2, Rule 1}$$

$$\frac{PE}{\frac{C}{e_1\;e_2}\;\frac{C}{b_1\;b_2}}\cdot b_1 \Rightarrow \frac{C}{\frac{C}{b_1\;b_2}\cdot b_1\;\frac{PE}{\frac{C}{e_1\;e_2}}} \Rightarrow \frac{C}{\frac{C}{b_2}\;\frac{PE}{\frac{C}{e_1\;e_2}}} \qquad \text{Rule 5, Rule 4}$$

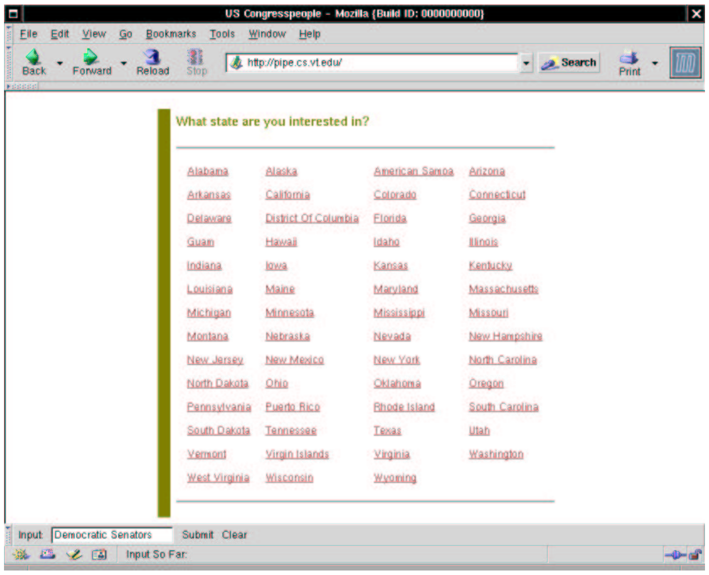
$$\frac{C}{\frac{C}{b_2}\;\frac{PE}{\frac{C}{e_1\;e_2}}}\cdot e_1 \Rightarrow \frac{C}{\frac{C}{b_2}\;\frac{PE}{\frac{C}{e_1\;e_2}}} \qquad \text{Rule 7}$$

$$\frac{C}{\frac{C}{b_2}\;\frac{PE}{\frac{C}{e_1\;e_2}}}\cdot b_2 \Rightarrow \frac{C}{\frac{C}{b_2}\cdot b_2\;\frac{PE}{\frac{C}{e_1\;e_2}}} \Rightarrow \frac{C}{\frac{PE}{\frac{C}{e_1\;e_2}}} \rightarrow \frac{PE}{\frac{C}{e_1\;e_2}} \rightarrow \frac{C}{e_1\;e_2} \qquad \text{Rule 6, Rule 2, Rule 1, Rule 1}$$

$$\frac{C}{e_1\;e_2}\cdot e_1 \Rightarrow \frac{C}{e_2} \qquad \text{Rule 4}$$

$$\frac{C}{e_2}\cdot e_2 \Rightarrow \theta \qquad \text{Rule 2}$$

Figure 5: 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.


US Congresspeople - Mozilla {Build ID: 0000000000}

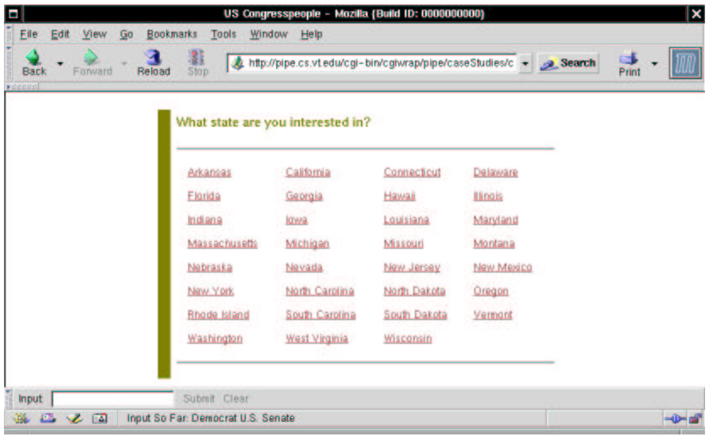
http://pipe.cs.vt.edu/

What state are you interested in?

Alabama	Alaska	American Samoa	Arizona
Arkansas	California	Colorado	Connecticut
Delaware	District Of Columbia	Florida	Georgia
Guam	Hawaii	Idaho	Illinois
Indiana	Iowa	Kansas	Kentucky
Louisiana	Maine	Maryland	Massachusetts
Michigan	Minnesota	Mississippi	Missouri
Montana	Nebraska	Nevada	New Hampshire
New Jersey	New Mexico	New York	North Carolina
North Dakota	Ohio	Oklahoma	Oregon
Pennsylvania	Puerto Rico	Rhode Island	South Carolina
South Dakota	Tennessee	Texas	Utah
Vermont	Virgin Islands	Virginia	Washington
West Virginia	Wisconsin	Wyoming	

Input: Democratic Senators

Input So Far:


US Congresspeople - Mozilla {Build ID: 0000000000}

http://pipe.cs.vt.edu/cgi-bin/cgiwrap/pipe/caseStudies/c

What state are you interested in?

Arkansas	California	Connecticut	Delaware
Florida	Georgia	Hawaii	Illinois
Indiana	Iowa	Louisiana	Maryland
Massachusetts	Michigan	Missouri	Montana
Nebraska	Nevada	New Jersey	New Mexico
New York	North Carolina	North Dakota	Oregon
Rhode Island	South Carolina	South Dakota	Vermont
Washington	West Virginia	Wisconsin	

Input:

Input So Far: Democrat U.S. Senate

Figure 6: (top) Starting webpage for personalizing information about US congressional officials. (bottom) Response of the system to an out-of-turn interaction by the user.

3.2 MII in Web Site Interactions

We now turn our discussion to studying MII in website interactions. The predominant way in which we interact with web sites is to click on presented hyperlinks, where we are merely responding to the choices put forth by the website. Due to the statelessness of the HTTP access protocol, and a fundamental lack of interaction instruments for MII, web access does not naturally lend itself to supporting mixed-initiative interaction. As a result, site designers have typically anticipated all the forms of interactions possible and hardwired them as navigation flows in the website structure. Using our staging transformation framework and an out-of-turn toolbar we have developed (called Extempore), we can provide a more personalized experience for the user without hardwiring all the possible interaction sequences. Consider the US Congressional portion of the Project Vote Smart web site (www.votesmart.org). The site provides information about political individuals; users interact with the site by specifying choices of state, party, branch of congress, and seat, in that order. With MII support, we allow the user to specify these attributes in any order and have the site automatically transformed to reflect the partial information provided.

Fig. 6 describes a sample session with Extempore. The site initially presents choice of a state to the user who, however, prefers to specify politicians in terms of party and branch of congress (i.e., 'Democratic Senators'). The result of the staging transformation removes many states, restricting the options to only those states that have democratic senators. We also employed query expansion techniques based on containment and functional dependency to provide a more personalized experience. For instance, if the user says 'Senior seat,' he is referring to a Senator, not a Representative. So, we can partially evaluate w.r.t. these additional variables. Saying 'North Dakota' and 'Representative' in the 2003 political landscape defines a unique member of Congress (no party information is needed), and so on. It is important to consider such facets in order to deliver a compelling personalized experience. The net effect of such considerations will be the initialization of multiple program variables based on the user's input and thus the site created at every stage will reflect an accurate summary of the remaining dialog options. For more details, please see [Perugini and Ramakrishnan, 2003].

3.3 Relationships to the VoiceXML Architecture

VoiceXML is a language that is designed to support simple system-directed and mixed-initiative dialogs for telephone-based interactions [Potter and Larson, 2002]. VoiceXML markup tags describe 'prompts,' 'forms,' and 'fields' that constitute a dialog. Dynamics of interaction are handled by a form interpretation algorithm (FIA) that (i) *selects* the first unfilled field in the active form, (ii) *collects* information by playing any prompts associated with the field and getting an utterance from the user, and finally (iii) *processes* the utterance using the specified speech recognition grammar to determine what slots can be filled based on the utterance, and copying the recognized values into variables associated with each appropriate field item [McGlashan et al., 2001].

For example, a greatly simplified representation of a VoiceXML form to handle a pizza ordering dialog can be given as:

```
form=place_order
        field=size, prompt="What size?"
        field=topping, prompt="What topping?"
        field=crust, prompt="What crust?"
endform
```

Combined with the grammar given by:

```
size = small | medium | large;
topping = sausage | onions | pepperoni;
crust = regular | thin;
```

the FIA would produce a system-directed, single-step dialog in the order shown (first prompting for size, then topping, then crust). If the user provided out-of-turn input (e.g., saying 'sausage' when asked for the size), the system would reject the utterance and re-prompt for the size without filling the topping field. If, however, we specified that the form should use a *form-level grammar*:

```
sizetoppingcrust = (size|topping|crust)*;
size = small | medium | large;
topping = sausage | onions | pepperoni;
crust = regular | thin;
```

then the FIA would support a mixed-initiative mode of filling the fields in the dialog. In [Ramakrishnan *et al.*, 2002], we showed that when the form-level grammar contains all permutations of specifying the pizza aspects, then the FIA reduces to a partial evaluation transformer. However, the FIA's implementation is not a purely functional one, since it doesn't trap the over-riding of slots even after they are filled; in our framework progressive dialog simplification happens as a natural consequence. Since VoiceXML's architecture uses grammars for both utterance recognition and staging specification, it is difficult to abstract staging from the dialog structure at the level that is possible in our approach.

3.4 Multimodal Interfaces

The emergence of multimodal interfaces, motivated by accessibility and mobile computing considerations, provides additional opportunities for investigating MII. Using multiple modes of communication can increase conversational bandwidth, reduce ambiguity, increase common ground, and streamline dialog. Speech Application Language Tags (SALT) is one effort to integrate voice input as part of multimodal interaction with existing web applications and architectures. While SALT shares many ideas of dialog specification with VoiceXML, these two technologies differ considerably in their execution models [Potter and Larson, 2002]. SALT uses an event-driven model typical of web applications, leaving the specific implementation of dialog control to designers. Thus, it provides a lower-level interface to controlling the flow of the dialog than does the built-in FIA of VoiceXML. This means that our staging transformations are an ideal candidate for supporting dialog management in this setting. From this point of view, SALT tags permit the capturing of out-of-turn input, and web pages embedded with SALT tags can be transformed (by stagers) to reflect the processing of such input. In our approach, most such transformations are handled by a specially configured proxy server but some of them can be conducted at the browser itself, using embedded code.

4 Reasoning about Dialogs

Our discussion thus far has concentrated on operationalizing MII but our emphasis on representations and using program transformations allows us to support many forms of meta-dialog reasoning as well. We broadly classify these forms of reasoning into (i) reasoning about scripts, (ii) reasoning about scripts and stagers, and (iii) reasoning about scripts, stagers, and interaction sequences. The first form of reasoning uses program transformations such as slicing [Binkley and Gallagher, 1996] to perform analyses on the structure of the dialog script. Slicing is a program transformation technique that allows the incorporation of partial information in a more expressive form than is possible in PE, where it is restricted to static values of program arguments. Forward slicing aids in what-if analyses and accommodates user requests such as 'what are the options available if I selected pepperoni?' Backward slicing is useful when the user is concerned that he might be going down the wrong path in the dialog (e.g., 'will this dialog give me a contact phone number at the end?').

The second form of reasoning is concerned with the representation of the dialog as a whole (both stagers and scripts). In this category, user negotiation can result in the dynamic replacement of the stager and/or restructuring the dialog (e.g., 'ask me the questions in topping-crust-size order'). Another example is determining when the dialog can be pruned (e.g., 'pepperoni pizzas are available in only large form, so there is no need to prompt the user for size'). These forms of dialog revision can be handled in a manner akin to replanning functionality.

The final form of reasoning brings in data about user interaction sequences and studies them from the perspective of either improving interaction for future dialogs or dynamic control of the current dialog. For instance, in [Ramakrishnan *et al.*, 2001] we have described how prototype scenarios of interaction can be 'explained' with respect to a domain theory (of information seeking) and how templates for interaction can be derived. Using sample interaction data to optimize dialog policy for specific tasks has been recently undertaken [Singh *et al.*, 2002]; such dynamic updates to dialog structure can be formally posed in the staging transformation framework. And finally we have been recently led to investigating how dialog-based interaction is influenced by the granularity (and expressiveness) with which information can be addressed — the representation-based theme espoused in this paper provides promising directions to study such issues.

Acknowledgements: This work was supported in part by US NSF grants IIS-0049075, IIS-0136182, and by a grant from IBM to explore the use of VoiceXML within their WebSphere product.

References

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

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

[Binkley and Gallagher, 1996] D.W. Binkley and K.B. Gallagher. Program Slicing. *Advances in Computers*, Vol. 43:pp. 1–50, 1996.

[Bridge, 2002] D. Bridge. Towards Conversational Recommender Systems: A Dialogue Grammar Approach. In *Proceedings of the Mixed-Initiative Case-Based Reasoning Workshop, European Conference on Case-Based Reasoning*, pages 9–22. 2002.

[Brunner *et al.*, 1992] H. Brunner, G. Whittemore, K. Ferrara, and J. Hsu. An Assessment of Written/Interaction Dialogue for Information Retrieval Applications. *Human-Computer Interaction*, Vol. 7:pages 197–249, 1992.

[Chu-Carroll and Brown, 1997] J. Chu-Carroll and M.K. Brown. Tracking Initiative in Collaborative Dialog Interactions. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 262–270. 1997.

[Graesser *et al.*, 2001] A.C. Graesser, K. VanLehn, C.P. Rose, P.W. Jordan, and D. Harter. Intelligent Tutoring Systems with Convesational Dialogue. *AI Magazine*, Vol. 22(4):pp. 39–51, 2001.

[Jones *et al.*, 1993] N.D. Jones, C.K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. PHI, 1993.

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

[Kitano and Ess-Dykema, 1991] H. Kitano and C.V. Ess-Dykema. Toward a Plan-Based Understanding Model for Mixed-Initiative Dialogues. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 25–32. 1991.

[McGlashan et al., 2001] S. McGlashan et al. Voice eXtensible Markup Language: VoiceXML. Technical report, VoiceXML Forum, Oct 2001. Version 2.00.

[Novick and Sutton, 1997] D.G. Novick and S. Sutton. What is Mixed-Initiative Interaction? In S. Haller and S. McRoy, editors, *Procedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, pages 114–116. AAAI/MIT Press, 1997.

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

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

[Ramakrishnan *et al.*, 2001] N. Ramakrishnan, M.B. Rosson, and J.M. Carroll. Explaining Scenarios for Information Personalization. Technical Report cs.HC/0111007, Computing Research Repository (CoRR), August 2001.

[Ramakrishnan *et al.*, 2002] 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.

[Ramakrishnan, 2000] N. Ramakrishnan. PIPE: Web Personalization by Partial Evaluation. *IEEE Internet Computing*, Vol. 4(6):pages 21–31, Nov-Dec 2000.

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