Mixed-Initiative Interaction = Mixed Computation

Naren Ramakrishnan
Dept. of Computer Science
Virginia Tech, Blacksburg
VA 24061, USA
naren@cs.vt.edu

Robert Capra
Dept. of Computer Science
Virginia Tech, Blacksburg
VA 24061, USA
rcapra@vt.edu

Manuel A. Pérez-Quiñones
Dept. of Computer Science
Virginia Tech, Blacksburg
VA 24061, USA
perez@cs.vt.edu

ABSTRACT

We show that partial evaluation can be usefully viewed as a programming model for realizing mixed-initiative functionality in interactive applications. Mixed-initiative interaction between two participants is one where the parties can take turns at any time to change and steer the flow of interaction. We concentrate on the facet of mixed-initiative referred to as 'unsolicited reporting' and demonstrate how out-of-turn interactions by users can be modeled by 'jumping ahead' to nested dialogs (via partial evaluation). Our approach permits the view of dialog management systems in terms of their support for staging and simplifying interactions; we characterize three different voice-based interaction technologies using this viewpoint. In particular, we show that the built-in form interpretation algorithm (FIA) in the VoiceXML dialog management architecture is actually a (well disguised) combination of an interpreter and a partial evaluator.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments; F.3.2 [**Semantics of Programming Languages**]: Partial Evaluation; H.4 [**Information Systems Applications**]: Communications Applications; H.5.2 [**User Interfaces**]: Interaction Styles

Keywords

Mixed-initiative interaction, partial evaluation, interaction sequences, dialog management, VoiceXML.

1. INTRODUCTION

Mixed-initiative interaction [8] has been studied for the past 30 years in the areas of artificial intelligence (AI) planning [17], human-computer interaction [5], and discourse analysis [6]. As Novick and Sutton point out [13], it is 'one of those things that people think that they can recognize when they see it even if they can't define it.' It can be broadly viewed as a flexible interaction strategy between participants where the parties can take turns at any time to change and steer the flow of interaction. Human conversations are typically mixed-initiative and, interestingly, so are interactions with some modern computer systems.

Consider the two dialogs in Figs. 1 and 2 with a telephone pizza delivery service that has voice-recognition capability (the line numbers are provided for ease of reference). Both these conversations involve the specification of a {size,topping,crust} tuple to complete the pizza ordering procedure but differ in important ways. In Fig. 1, the caller responds to the questions in the order they are posed by the system. The system has the initiative at all times (other than, perhaps, on line 0) and such an interaction is thus referred to as *system-initiated*. In Fig. 2, when the system prompts the caller about pizza size, he responds with information about his choice of topping instead (sausage; see line 3 of Fig. 2). Nevertheless, the conversation is not stalled and the system continues with the other aspects of the information gathering activity. In particular, the system registers that the caller has specified a topping, skips its default question on this topic, and repeats its question about the size (see line 5 of Fig. 2). The caller thus gained the initiative for a brief period during the conversation, before returning it to the system. Such a conversation that mixes system-initiated and user-initiated modes of interaction is said to be *mixed-initiative*.

1.1 Tiers of Mixed-Initiative Interaction

It is well accepted that mixed-initiative provides a more natural and personalized mode of interaction. A matter of debate, however, are the qualities that an interaction must possess to merit its classification as mixed-initiative [13]. In fact, determining who has the initiative at a given point in an interaction can itself be a contentious issue! The role of intention in an interaction and the underlying task goals also affect the characterization of initiative. We will not attempt to settle this debate here but a few preliminary observations will be useful.

One of the basic levels of mixed-initiative is referred to as *unsolicited reporting* in [3] and is illustrated in Fig. 2. In this facet, a participant provides information out-of-turn (in our case the caller, about his choice of topping). Furthermore, the out-of-turn interaction is not agreed upon in advance by the two participants. Novick and Sutton [13] stress that the unanticipated nature of out-of-turn interactions is important and that mere turn-taking (perhaps in a hardwired order) does not constitute mixed-initiative. Finally, notice that in Fig. 2 there is a resumption of the original questioning task

PEPM '02, Jan. 14-15, 2002 Portland, OR, USA
2002 ACM ISBN 1-58113-455-X/02/01...$5.00.

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 medium, please.
4 **System:** What topping would you like on your pizza?
5 **Caller:** Pepperoni.
6 **System:** What type of crust do you want?
7 **Caller:** Uh, deep-dish.
8 **System:** So that is a medium pepperoni pizza with deep-dish crust. Is this correct?
9 **Caller:** Yes.
(conversation continues to get delivery and payment information)

Figure 1: Example of a system-directed conversation.

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?
10 **Caller:** Yes.
(conversation continues to get delivery and payment information)

Figure 2: Example of a mixed-initiative conversation.

once processing of the unsolicited response is completed. In other applications, an unsolicited response might shift the control to a new interaction sequence and/or abort the current interaction.

Another level of mixed-initiative involves *subdialog invocation*; for instance, the computer system might not have understood the user's response and could ask for clarifications (which amounts to it having the initiative). A final, sophisticated, form of mixed-initiative is one where participants negotiate with each other to determine initiative (as opposed to merely 'taking the initiative') [3]. An example is given in Fig. 3.

In addition to models that characterize initiative, there are models for designing dialog-based interaction systems. Allen et al. [2] provide a taxonomy of such software models — finite-state machines, slot-and-filler structures, frame-based methods, and more sophisticated models involving planning, agent-based programming, and exploiting contextual information. While mixed-initiative interaction can be studied in any of these models, it is beyond the scope of this paper to address all or even a majority of them.

Instead, we concentrate on the view of (i) a dialog as a task-oriented information assessment activity requiring the filling of a set of slots, (ii) where one of the participants in the dialog is a computer system and the other is a human, and (iii) where mixed-initiative arises from unsolicited reporting (by the human), involving out-of-turn interactions. This characterization includes many voice-based interfaces to information (our pizza ordering dialog is an example) and web sites modeling interaction by hyperlinks [15]. In Section 2, we show that partial evaluation can be usefully viewed as a programming model for such applications. Section 3 presents three different voice-based interaction technologies and analyzes them in terms of their native support for mixing initiative. Finally, Section 4 discusses other facets of mixed-initiative and mentions other software models to which our approach can be extended.

2. PROGRAMMING A MIXED-INITIATIVE APPLICATION

Before we outline the design of a system such as Joe's Pizza (ref. Figs. 1 and 2), we introduce a notation [7, 11] that captures basic elements of initiative and response in an interaction sequence. The notation expresses the local organization of a dialog [14] as adjacency pairs; for instance, the dialog in Fig. 1 is represented as:

(Ic Rs) (Is Rc) (Is Rc) (Is Rc) (Is Rc)
0 1 2 3 4 5 6 7 8 9

The line numbers given below the interaction sequence refer to the utterance numbers in Fig. 1. The letter I denotes who has the initiative — caller (c) or the system (s) — and the letter R denotes who provides the response. It is easy to see from this notation that the dialog in Fig. 1 consists of five turns and that the system has the initiative for the last four turns. The initial turn is modeled as the caller having the initiative because he or she chose to place the phone call in the first place. The system quickly takes the initiative after playing a greeting to the caller (which is modeled here as the response to the caller's call). The subsequent four interactions then address three questions and a confirmation, all involving the system retaining the initiative (Is) and the caller in the responding mode (Rc).

(with apologies to O. Henry)
Husband: Della, Something interesting happened today that I want to tell you.
Wife: I too have something exciting to tell you, Jim.
Husband: Do you want to go first or shall I tell you my story?

Figure 3: Example of a mixed-initiative conversation where initiative is negotiated.

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, medium, and deep-dish.
4 **System:** So that is a medium sausage pizza with deep-dish crust. Is this correct?
5 **Caller:** Yes.
(conversation continues to get delivery and payment information)

Figure 4: Example of a mixed-initiative conversation with a frequent customer.

Likewise, the mixed-initiative interaction in Fig. 2 is represented as:

(Ic	Rs)	(Is	(Ic	Rs)	Rc)	(Is	Rc)	(Is	Rc)
0	1	2,5	3	4	6	7	8	9	10

In this case, the system takes the initiative in utterance 2 but instead of responding to the question of size in utterance 3, the caller takes the initiative, causing an 'insertion' to occur in the interaction sequence (dialog) [11]. The system responds with an acknowledgement ('Okay, sausage.') in utterance 4. This is represented as the nested pair (Ic Rs) above. The system then re-focusses the dialog on the question of pizza size in utterance 5 (thus retaking the initiative). In utterance 6 the caller responds with the desired size (medium) and the interaction proceeds as before, from this point.

There are various other possibilities for mixing initiative. For instance, if a user is a frequent customer of Joe's Pizza, he might take the initiative and specify all three pizza attributes on the first available prompt, as shown in Fig. 4. Such an interaction would be represented as:

(Ic	Rs)	(Is	Rc)	(Is	Rc)
0	1	2	3	4	5

Notice that even though this dialog consists of only three turns it constitutes a complete instance of interaction with the pizza ordering service.

The notation aids in understanding the structure and staging of interaction in a dialog. By a post-analysis of all interaction sequences described in this manner, we find that utterances 0 and 1 have to proceed in order. Utterances dealing with selection of {size,topping,crust} can then be nested in any order and provide interesting opportunities for mixing initiative. Finally, the utterances dealing with confirmation of the user's request can proceed only after choices of all three pizza attributes have been made.

While the notation doesn't directly reflect the computational processing necessary to achieve the indicated structure, it can be used to express a set of requirements for a dialog system design. There are 13 possible interaction sequences (discounting permutations of attributes specified in a given utterance): 1 possibility of specifying everything in one utterance, 6 possibilities of specification in two utterances, and 6 possibilities of specification in three utterances. If we include permutations, there are 24 possibilities (our calculations do not consider situations where, for instance, the system doesn't recognize the user's input and reprompts for information). Of these possibilities, all but one are mixed-initiative sequences.

Many programming models view mixed initiative sequences as requiring some special attention to be accommodated. In particular, they rely on recognizing when a user has provided unsolicited input[1] and qualify a shift-in-initiative as a 'transfer of control.' This implies that the mechanisms that handle out-of-turn interactions are often different from those that realize purely system-directed interactions. Fig. 5 (left) describes a typical software design. A dialog manager is in charge of prompting the user for input, queueing messages onto an output medium, event processing, and managing the overall flow of interaction. One of its inputs is a dialog script that contains a specification of interaction and a set of slots that are to be filled. In our pizza example, slots correspond to placeholders for values of size, topping, and crust. An interpreter determines the first unfilled slot to be visited and presents any prompts for soliciting user input. A responsive input requires mere slot filling whereas unsolicited inputs would require out-of-turn processing (involving a combination of slot filling and simplification). In turn, this causes a revision of the dialog script. The interpreter terminates when there is nothing left to process in the script. While typical dialog managers perform miscellaneous functions such as error control, transferring to other scripts, and accessing scripts from a database, the architecture in Fig. 5 (left) focusses on the aspects most relevant to our presentation.

Our approach, on the other hand, is to think of a mixed-initiative dialog as a program, all of whose arguments are passed by reference and which correspond to the slots comprising information assessment. As usual, an interpreter in the dialog manager queues up the first applicable prompt to an output medium. Both responsive and unsolicited inputs by a user now correspond (uniformly) to values for arguments; they are processed by partially evaluating the program with respect to these inputs. The result of partial evaluation is another dialog (simplified as a result of user input) which is handed back to the interpreter. This novel

[1] We use the term 'unsolicited input' here to refer to expected but out-of-turn inputs as opposed to completely unexpected (or out-of-vocabulary) inputs.

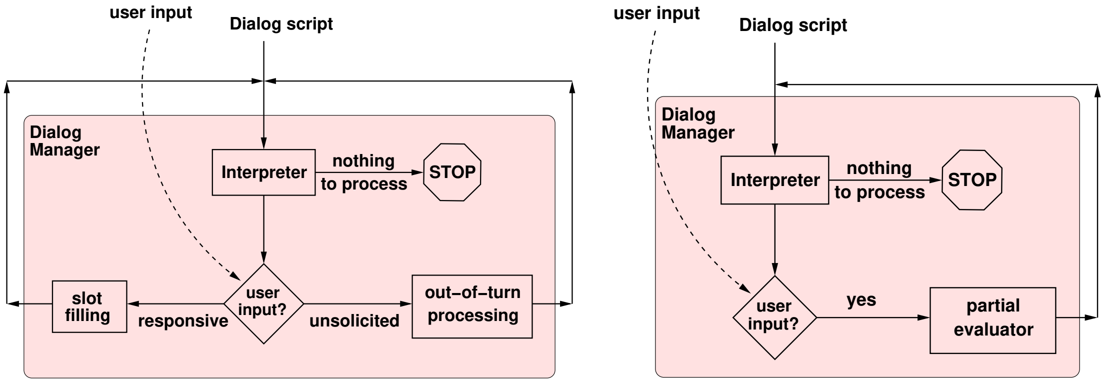

Figure 5: Designs of software systems for mixed-initiative interaction. (left) Traditional system architecture, distinguishing between responsive and unsolicited inputs. (right) Using partial evaluation to handle inputs uniformly.

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

Figure 6: Modeling a dialog script as a program parameterized by slot variables that are passed by reference.

design is depicted in Fig. 5 (right) and a dialog script represented in a programmatic notation is given in Fig. 6. Partial evaluation of Fig. 6 with respect to user input will remove the conditionals for all slots that are filled by the utterance (global variables are assumed to be under the purview of the interpreter). The reader can verify that a sequence of such partial evaluations will indeed mimic the interaction sequence depicted in Fig. 2 (and any of the other mixed-initiative sequences).

Partial evaluation serves two critical uses in our design. The first is obvious, namely the processing of out-of-turn interactions (and any appropriate simplifications to the dialog script). The more subtle advantage of partial evaluation is its support for staging mixed-initiative interactions. The mix-equation [9] holds for every possible way of splitting inputs into two categories, without enumerating and 'trapping' the ways in which the computations can be staged. For instance, the nested pair in Fig. 2 is supported as a natural consequence of our design, not by anticipating and reacting to an out-of-turn input.

Another way to characterize the system designs in Fig. 5 is to say that Fig. 5 (left) makes a distinction of responsive versus unsolicited inputs, whereas Fig. 5 (right) makes a more fundamental distinction of fixed-initiative (interpretation) versus mixed-initiative (partial evaluation). In other words, Fig. 5 (right) carves up an interaction sequence into (i) turns that are to be handled in the order they are modeled (by an interpreter), and (ii) turns that can involve mixing of initiative (handled by a partial evaluator). In the latter case, the computations are actually used as a *representation of interactions.* Since only mixed-initiative interactions involve multiple staging options and since these are handled by the partial evaluator, our design requires the *least* amount of specification (to support all interaction sequences). For instance, the script in Fig. 6 models the parts that involve mixing of initiative and helps realize all of the 13 possible interaction sequences. At the same time it does not model the confirmation sequence of Fig. 2 because the caller cannot confirm his order before selecting the three pizza attributes! This turn must be handled by straightforward interpretation.

To the best of our knowledge, such a model of partial evaluation for mixed-initiative interaction has not been proposed before. While computational models for mixed-initiative interaction remain an active area of research [8], such work is characterized by keywords such as 'controlling mixed-initiative interaction,' 'knowledge representation and reasoning strategies,' and 'multi-agent co-ordination.' There are even projects that talk about 'integrating' mixed initiative interaction and partial evaluation to realize an architecture for planning and learning [17]. We are optimistic that our work has the same historical significance as the relation between explanation-based generalization (a learning technique in AI) and partial evaluation established by van Haremelen and Bundy in 1988 [16].

2.1 Preliminary Observations

It is instructive to examine the situations under which a concept studied in a completely different domain is likened to partial evaluation. Establishing a resemblance to partial evaluation is usually done by mapping from an underlying idea of specialization or customization in the original domain. This is the basis in [16] where specialization of domain theories is equated to partial evaluation of programs. The motivating theme is one of re-expressing the given program (domain theory) in an efficient but less general form, by recognizing that parameters have *different rates of variation* [9]. This theme has also been the primary demonstrator in the partial evaluation literature, where inner loops

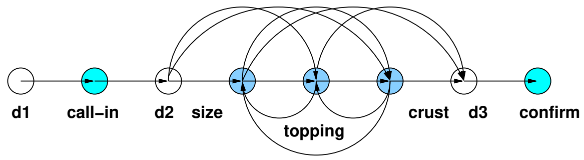

Figure 7: Modeling requirements for unsolicited reporting as traversals of a graph.

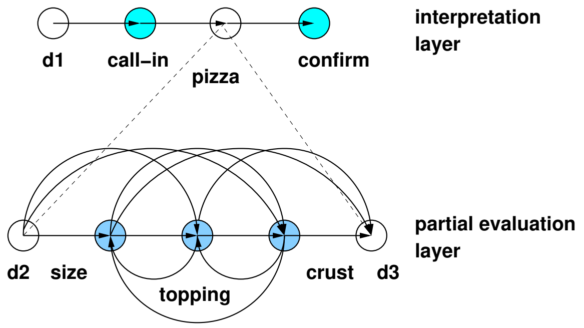

Figure 8: Layering interaction sequences for unsolicited reporting.

in heavily parameterized programs are optimized by partial evaluation.

Our model is grounded on the (more basic) view of parameters as involving *different times of arrival*. By capturing the essence of unsolicited reporting as merely differences in arrival time (specification time) of aspects, we are able to use partial evaluation for mixed-initiative interaction. The property of partial evaluation that is pertinent here is not just that it is a specialization technique, but also that a sequence of such specializations corresponds to a valid instance of interaction with the dialog script. Moreover, the set of valid sequences of specializations is exactly the set of interactions to be supported. The program of Fig. 6 can thus be thought of as a compaction of all interaction sequences that involve mixing initiative.

2.1.1 Decomposing Interaction Sequences

An important issue to be addressed is the decomposition of interaction sequences into parts that should be handled by interpretation and parts that can benefit from partial evaluation. We state only a few general guidelines here. Fig. 7 describes the set of valid interaction sequences for the pizza example as traversals of a graph. Nodes in the graph correspond to stages of specification of aspects. Thus, taking the outgoing edge from the call-in node implies that this stage of the interaction has been completed (the need for the dummy nodes such as d2 and d3 will become clear in a moment). The rules of the traversal are to find paths such that all nodes are visited and every node is visited exactly once. It is easy to verify that with these rules, Fig. 7 models all possibilities of mixing initiative (turns where the user specifies multiple pizza aspects can be modeled as a sequence of graph moves).

Expressing our requirements in a graph such as Fig. 7 reveals that the signature bushy nature of mixing initiative is restricted to only a subgraph of the original graph. We can demarcate the starting (d2) and ending points (d3) of

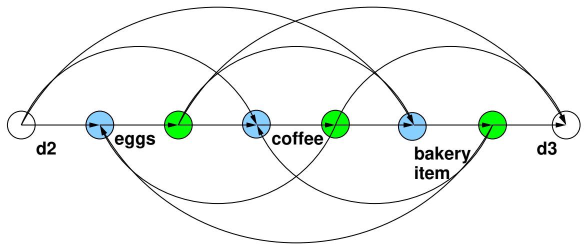

Figure 9: Modeling subdialog invocation.

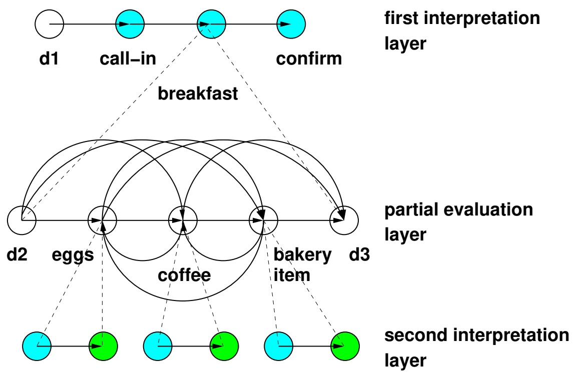

Figure 10: Layering interaction sequences for subdialog invocation.

the subgraph and layer it in the context of a larger interaction sequence as shown in Fig. 8. The nodes in the top layer dictate a strict sequential structure, thus they should be modeled by interpretation. The nodes in the bottom layer encapsulate and exhibit the bushy characteristic; they are hence candidates for partial evaluation. The key lesson to be drawn from Fig. 8 is that partial evaluation effectively supports the mixed-initiative subgraph without maintaining any additional state (for instance after a node has been visited, this information is not stored anywhere to ensure that it is not visited again). In contrast, the interpretation layer has an implicit notion of state (namely, the part of the interaction sequence that is currently being interpreted). The layered design can be implemented as two alternating pieces of code (for interpretation and partial evaluation, respectively) where the interpreter passes control to the partial evaluator for the segments involving pizza attribute specification and resumes after these segments have been evaluated.

For some applications, the alternating layer concept might need to be extended to more than two layers. Consider a hotel telephone service for ordering breakfast. Assume that ordering breakfast involves specifications of {eggs, coffee, bakery item} tuples. The user can specify these items in any order, but each item involves a second clarification aspect. After the user has specified his 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 mixed-initiative was introduced in Section 1.1 as subdialog invocation. The set of interaction sequences that address this requirement can be represented as shown in Fig. 9 (only the mixed-initiative parts are shown). In

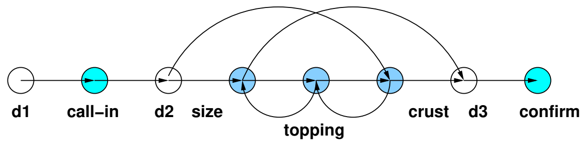

Figure 11: A requirement for mixing initiative that cannot be captured by partial evaluation.

this case, it is not possible to achieve a clean separation of subgraphs into interpretation and partial evaluation in just two layers.

One solution is to use three layers as shown in Fig. 10. If we implement both interpretation layers of this design by the same code, some form of scope maintenance (e.g., a stack) will be necessary when moving across the bottom two layers. Pushing onto the stack preserves the context of the original interaction sequence and is effected for each step of partial evaluation. Popping restores this context when control returns to the partial evaluator. The semantics of graph traversal remain the same. Once the nodes in the second and third layers are traversed, interpretation is resumed at the top layer to confirm the breakfast order. It is important to note that once again, the semantics of transfer of control between the interpreter and partial evaluator are unambiguous and occur at well defined branch points.

The above examples highlight the all-or-nothing role played by partial evaluation. For a dialog script parameterized in terms of slot variables, partial evaluation can be used to support all valid possibilities for mixing initiative, but it cannot restrict the scope of mixing initiative in any way. In particular this means that, unlike interpretation, partial evaluation cannot enforce any individual interaction sequence! Even a slight deviation in requirements that removes some of the walks in the bushy subgraph will render partial evaluation inapplicable.

For instance, consider the graph in Fig. 11 that is the same as Fig. 7 with some edges removed. Mixing initiative is restricted to visiting the size, topping, and crust nodes in a strict forward or reverse order. Partial evaluation cannot be used to restrict the scope of mixing initiative to just these two possibilities of specifying the pizza attributes. We can model this by interpretation but this requires anticipation, akin to the design of Fig. 5 (left).

This is the essence of partial evaluation as a programming model; it makes some tasks extremely easy but like every other programming methodology, it is not a silver bullet. In a realistic implementation for mixed-initiative interaction, partial evaluation will need to be used in conjunction with other paradigms (e.g., interpretation) to realize the desired objectives.

In this paper, we concentrate on only the unsolicited reporting facet of mixed initiative for which the decomposition illustrated by Fig. 8 is adequate. In other words, these are the applications where the all-or-nothing role of partial evaluation is sufficient to realize mixed-initiative interaction.

Our modeling methodology is concerned only with the interaction staging aspect of dialog management, namely determining what the next step(s) in the dialog can or should be. We have not focused on aspects such as response generation. In the pizza example, responses to successful partial evaluation (or interpretation) can be easily modeled as side-

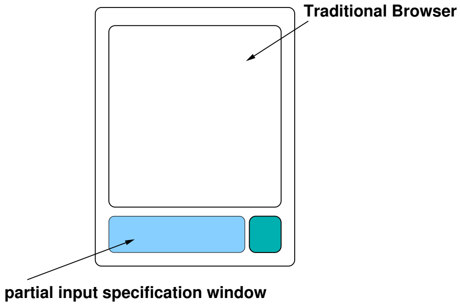

Figure 12: Sketch of an interface for mixed-initiative interaction with web sites.

effects. In other cases, we would need to take into account the context of the particular interaction sequence. The same argument holds for what is known as tapered prompting [4]. If the prompt for size needs to be replayed (because the user took the initiative and specified a topping), we might want to choose a different prompt for a more natural dialog (i.e., instead of repeating 'What size pizza would you like?,' we might prompt as 'You still haven't specified a size. Please choose among small, medium, or large.'). We do not discuss these aspects further except to say that they are an important part of a production implementation of dialog based systems.

2.1.2 Implementation Technologies

We now turn our attention to implementing our partial evaluation model for existing information access and delivery technologies. As stated earlier, our model is applicable to voice-based interaction technologies as well as web access via hyperlinks. In [15], we study the design and implementation of web site personalization systems that allow the mixing of initiative. In contrast to a voice-based delivery mechanism, (most) interactions with web sites proceed by clicking on hyperlinks. For instance, a pizza ordering web service might provide hyperlinks for choices of size so that clicking on a link implies a selection of size. This might refresh the browser to a new page that presents choices for topping, and so on. Since clicking on links implies a response to the initiative taken by the web page author, a different communication mechanism needs to be provided to enable the user to take the initiative.

Our suggested interface design is shown in Fig. 12. An extra window is provided for the user to type in his specification aspects. This is not fundamentally different from a location toolbar in current browsers that supports the specification of URL. Consider that a web page presents hyperlinks for choices of pizza size. Using the interface in Fig. 12, the user can either click on her choice of size attribute in the top window (effectively, responding to the initiative), or can use the bottom window to specify, say, a topping out-of-turn. To force an interpretation mode for segments of an interaction sequence, the bottom window can be made inactive. Modeling decisions, implementation details, and experimental results for two web applications developed in this manner are described in [15]; we refer the reader to this reference for details.

Our original goal for this paper was to study these concepts for voice-based interaction technologies and to see if our model can be used for the implementation of a voice-

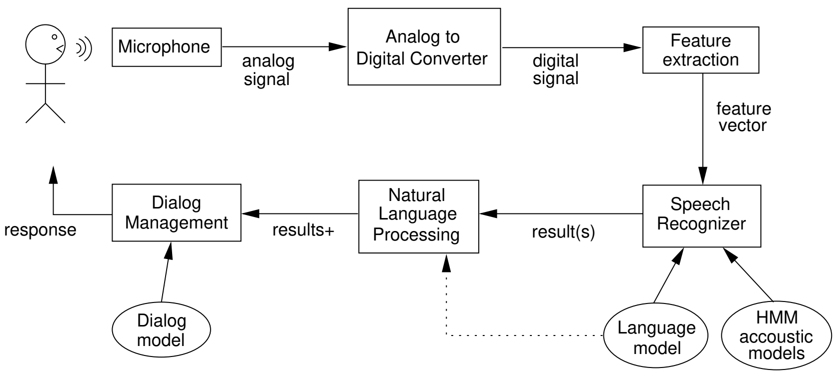

Figure 13: Basic components of a spoken language processing system.

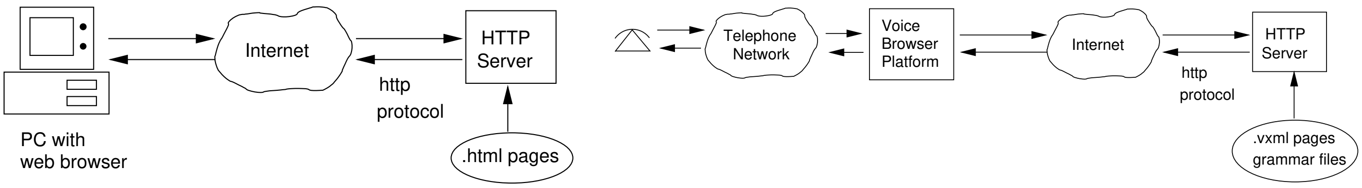

Figure 14: (left) Accessing HTML documents via a HTTP web server. (right) Accessing VoiceXML documents via a HTTP web server.

based mixed-initiative application. A variety of commercial technologies are available for developing voice-based applications; as a first choice of a representational formalism, we proceeded to use the specification language of the VoiceXML dialog management architecture [4]. The idea was to describe dialogs in VoiceXML notation and use partial evaluation to realize mixed-initiative interaction. After some initial experimentation we realized that VoiceXML's form interpretation algorithm (FIA), which processes the dialogs, provides mixed-initiative interaction using a script very similar to the one we presented for use with a partial evaluator (see Fig. 6)! In other these, there is no real advantage to partially evaluating a VoiceXML specification! This pointed us to the possibility that perhaps we can identify an instantiation of our model in VoiceXML's dialog management architecture and especially, the FIA. The rest of the paper takes this approach and shows that this is indeed true.

We also identify other implementation technologies where we can usefully implement a voice-based mixed-initiative system using our model. We merely identify the opportunities here and hope to elaborate on a complete implementation of our model in a future paper.

3. SOFTWARE TECHNOLOGIES FOR VOICE-BASED MIXED-INITIATIVE APPLICATIONS

Before we can study the programming of mixed-initiative in a voice-based application, it will be helpful to understand the basic architecture (see Fig. 13) of a spoken language processing system. As a user speaks into the system, the sounds produced are captured by a microphone and converted into a digital signal by an analog-to-digital converter. In telephone-based systems (the VoiceXML architecture covered later in the paper is geared toward this mode), the microphone is part of the telephone handset and the analog-to-digital conversion is typically done by equipment in the telephone network (in some cellular telephony models, the conversion would be performed in the handset itself).

The next stage (feature extraction) prepares the digital speech signal to be processed by the speech recognizer. Features of the signal important for speech recognition are extracted from the original signal, organized as an attribute vector, and passed to the recognizer.

Most modern speech recognizers use Hidden Markov Models (HMMs) and associated algorithms to represent, train, and recognize speech. HMMs are probabilistic models that must be trained on an input set of data. A common technique is to create sets of acoustic HMMs that model phonetic units of speech in context. These models are created from a training set of speech data that is (hopefully) representative of the population of users who will use the system. A language model is also created prior to performing recognition. The language model is typically used to specify valid combinations of the HMMs at a word- or sentence-level. In this way, the language model specifies the words, phrases, and sentences that the recognizer can attempt to recognize. The process of recognizing a new input speech signal is then accomplished using efficient search algorithms (such as Viterbi decoding) to find the best matching HMMs, given the constraints of the language model. The output of the speech recognizer can take several different forms, but the basic result is a text string that is the recognizer's best guess of what the user said. Many recognizers can provide additional information such as a lattice of results, or an N-best ranked list of results (in case the later stages of processing wish to reject the recognizer's top choice). A good introduction to speech recognition is available in [10].

The stages after speech recognition vary depending on the application and the types of processing required. Fig. 13 presents two additional phases that are commonly included in spoken language processing systems in one form or another. We will broadly refer to the first post-recognition stage as natural language processing (NLP). NLP is a large field in its own right and includes many sub-areas such as parsing, semantic interpretation, knowledge representation, and speech acts; an excellent introduction is available in Allen's classic [1]. Our presentation in this paper has assumed NLP support for slot-filling (i.e., determining values for slot variables from user input).

Slot-filling is commonly achieved by defining parts of a language model and associating them with slots. The language model could be specified as a context-free grammar or as a statistically-based model such as n-grams. Here we focus on the former: in this approach, slots can be specified within the productions of a context-free grammar (akin to a attribute grammar) or they can be associated with the non-terminals in the grammar.

We will refer to the next phase of processing as simply 'dialog management' (see Fig. 13). In this phase, augmented results from the NLP stage are incorporated into the dialog and any associated logic of the application is executed. The job of the dialog manager is to track the proceedings of the dialog and to generate appropriate responses. This is often done within some logical processing framework and a dialog model (in our case, a dialog script) is supplied as input that is specific to the particular application being designed. The execution of the logic on the dialog model (script) results in a response that can be presented back to the user. Sometimes response generation is separated out into a subsequent stage.

3.1 The VoiceXML Dialog Management Architecture

There are many technologies and delivery mechanisms available for implementing Fig. 13's basic components. A popular implementation can be seen in the VoiceXML dialog management architecture. VoiceXML is a markup language designed to simplify the construction of voice-response applications [4]. Initiated by a committee comprising AT&T, IBM, Lucent Technologies, and Motorola, it has emerged as a standard in telephone-based voice user interfaces and in delivering web content by voice. We will hence cover this architecture in detail.

The basic idea is to describe interaction sequences using a markup notation in a VoiceXML *document.* As the VoiceXML specification [4] indicates, a VoiceXML document constitutes a conversational finite state machine and describes a sequence of interactions (both fixed- and mixed-initiative are supported). A web server can serve VoiceXML documents using the HTTP protocol (Fig. 14, right), just as easily as HTML documents are currently served over the Internet (Fig. 14, left). In addition, voice-based applications require a suitable delivery platform, illustrated by a telephone in Fig. 14 (right). The voice-browser platform in Fig. 14 (right) includes the VoiceXML interpreter which processes the documents, monitors user inputs, streams messages, and performs other functions expected of a dialog management system. Besides the VoiceXML interpreter, the voice-browser platform typically includes a speech recognizer, a speech synthesizer, and telephony interfaces to help realize these aspects of voice-based interaction.

Dialog specification in a VoiceXML document involves organizing a sequence of *forms* and *menus.* Forms specify a set of slots (called field item variables) that are to be filled by user input. Menus are syntactic shorthands (much like a `case` construct); since they involve only one field item variable (argument), there are no opportunities for mixing initiative. We do not discuss menus further in this paper. An example VoiceXML document for our pizza application is given in Fig. 15.

As shown in Fig. 15, the pizza dialog consists of two forms. The first form (`welcome`) merely welcomes the user and transitions to the second. The `place_order` form involves four `fields` (slot variables) — the first three cover the pizza attributes and the fourth models the confirmation variable (recall the dialogs in Section 1). In particular, prompts for soliciting user input in each of the fields are specified in Fig. 15.

Interactions in a VoiceXML application proceed just like a web application except that instead of clicking on a hyperlink (to goto a new state), the user talks into a microphone. The VoiceXML interpreter then determines the next state to move to. Any appropriate responses (to user input) and prompts are delivered over a speaker. The core of the interpreter is a so-called form interpretation algorithm (FIA) that drives the interaction. In Fig. 15, the fields provide for a fixed-initiative, system-directed interaction. The FIA simply visits all fields in the order they are presented in the document. Once all fields are filled, a check is made to ensure that the confirmation was successful; if not, the fields are cleared (notice the `clear namelist` tag) and the FIA will proceed to `prompt` for the inputs again, starting from the first unfilled field — `size`.

The form in Fig. 15 is referred to as a directed one since the computer has the initiative at all times and the `fields` are filled in a strictly sequential order. To make the interaction mixed-initiative (with respect to `size`, `crust`, and `topping`), the programmer merely has to specify a *form-level grammar* that describes possibilities for slot-filling from a user utterance. An example form-level grammar file (`sizetoppingcrust.gram`) is given in Fig. 16. The productions for `sizetoppingcrust` cover all possibilities of filling slot variables from user input, including multiple slots filled by a given utterance, and various permutations of specifying pizza attributes. The grammar is associated with the dialog script by including the line:

```
<grammar src="sizetoppingcrust.gram"
 type="application/x-jsgf"/>
```

just before the definition of the first `field` (size) in Fig. 15.

The form-level grammar contains productions for the various choices available for size, topping, and crust and also qualifies all possible parses for a given utterance (modeled by the non-terminal `sizetoppingcrust`). Any valid combination of the three pizza aspects uttered by the user (in any order) is recognized and the appropriate slot variables are instantiated. To see why this also achieves mixed-initiative, let us consider the FIA in more detail.

Fig. 17 reproduces the salient aspects of the FIA relevant for our discussion. Compare the basic elements of the FIA to the stages in Fig. 5 (right). The Select phase corresponds to the interpreter, the Collect phase gathers the user input, and actions taken in the Process phase mimic the partial evaluator. Recall that 'programs' (scripts) in VoiceXML can be modeled by finite-state machines, hence the mechanics of partial evaluation are considerably simplified and just amount to filling the slot and tagging it as filled. Since the

```
<?xml version="1.0"?>
<vxml version="1.0">
<!-- pizza.vxml
     A simple pizza ordering demo to illustrate some basic elements
     of VoiceXML.  Several details have been omitted from this demo
     to help make the basic ideas stand out. -->
  <form id="welcome">
    <block name="block1">
      <prompt> Thank you for calling Joe's pizza ordering system. </prompt>
      <goto next="#place_order" />
    </block>
  </form>

  <form id="place_order">
    <field name="size">
      <prompt> What size pizza would you like? </prompt>
    </field>

    <field name="topping">
      <prompt> What topping would you like on your pizza? </prompt>
    </field>

    <field name="crust">
      <prompt> What type of crust do you want? </prompt>
    </field>

    <field name="verify">
      <prompt>
        So that is a <value expr="size"/> <value expr="topping"/> pizza
        with <value expr="crust"/> crust.
        Is this correct?
      </prompt>
      <grammar> yes | no </grammar>
    </field>

    <filled>
      <if cond="verify=='no'">
         <clear namelist="size topping verify crust"/>
         <prompt> Sorry.  Your order has been canceled. </prompt>
      <else/>
         <prompt>Thank you for ordering from Joe's pizza.</prompt>
      </if>
    </filled>

  </form>
</vxml>
```

Figure 15: Modeling the pizza ordering dialog in a VoiceXML document.

```
#JSGF V1.0;

grammar sizetoppingcrust;

public <sizetoppingcrust> =
   <size> {this.size=$} [<topping> {this.topping=$}] [<crust> {this.crust=$}] |
   <size> {this.size=$} <crust> {this.crust=$} <topping> {this.topping=$} |
   <topping> {this.topping=$} [<crust> {this.crust=$}] [<size> {this.size=$}] |
   <topping> {this.topping=$} <size> {this.size=$} <crust> {this.crust=$} |
   <crust> {this.crust=$} [<size> {this.size=$}] [<topping> {this.topping=$}] |
   <crust> {this.crust=$} <topping> {this.topping=$} <size> {this.size=$};

<size> = small | medium | large;
<topping> =  sausage | pepperoni | onions | green peppers;
<crust> = regular | deep dish | thin;
```

Figure 16: A form-level grammar to be used in conjunction with the script in Fig. 15 to realize mixed-initiative interaction.

```
While (true)
{
        // SELECT PHASE
        Select the first form item with an unsatisfied guard condition
             (e.g., unfilled)
          If no such form item, exit

        // COLLECT PHASE
        Queue up any prompts for the form item
        Get an utterance from the user

        // PROCESS PHASE
        foreach (slot in user's utterance)
        {
               if (slot corresponds to a field item) {
                    copy slot values into field item variables
                    set field item's 'just_filled' flag
               }
        }
        // some code for executing any 'filled' actions triggered
}
```

Figure 17: Outline of the form interpretation algorithm (FIA) in the VoiceXML dialog management architecture. Adapted from [4].

```
#JSGF V1.0;

grammar sizetoppingcrust;

public <sizetoppingcrust> = <word>*;

<word> = <size> {this.size=$} |
       <crust> {this.crust=$} |
       <topping> {this.topping=$};

<size> = small | medium | large;
<topping> =  sausage | pepperoni | onions | green peppers;
<crust> = regular | deep dish | thin;
```

Figure 18: A alternative form-level grammar to realize mixed-initiative interaction with the script in Fig. 15.

FIA repeatedly executes while there are unfilled form items remaining, the processing phase (Process) is effectively parameterized by the form-level grammar file. In other words, the form-level grammar file not only enables slot filling, *it also implicitly directs the staging of interactions for mixed-initiative.* When the user specifies 'peperroni medium' in an utterance, not only does the grammar file enable the recognition of the slots they correspond to (topping and size), it also enables the FIA to simplify these slots (and mark them as 'filled' for subsequent interactions).

The form-level grammar file shown in Fig. 16 (which is also a specification of interaction staging) may make VoiceXML's design appear overly complex. In reality, however, we could have used the vanilla form-level grammar file in Fig. 18. While helping to realize mixed-initiative, the new form-level file (as does our model) also allows the possibility of utterances such as 'pepperoni pepperoni,' or even, 'pepperoni sausage!' Suitable semantics for such situations (including the role of side-effects) can be defined and accommodated in both the VoiceXML model and ours. It should thus be obvious to the reader that VoiceXML's dialog management architecture is actually implementing a mixed evaluation model (for conversational finite state machines), comprising interpretation and partial evaluation.

The VoiceXML specification [4] refers to the form-level file as a 'grammar file,' when it is actually also a specification of staging. Even though the grammar file serves the role of a language model in a voice application, we believe that recognizing its two functionalities is important in understanding mixed-initiative system design. A case in point is our study of personalizing interaction with web sites [15] (see also Fig. 12). There is no requirement for a 'grammar file,' as there is usually no ambiguity about user clicks and typed-in keywords. Specifications in this application thus serve to associate values with program variables and do not explicitly capture the staging of interactions. The advantageous of partial evaluation for interaction staging are thus obvious.

3.2 Other Implementation Technologies

VoiceXML's FIA thus includes native support for slot filling, slot simplification, and interaction staging. All of these are functions enabled by partial evaluation in our model. Table 1 contrasts two other implementation approaches in terms of these aspects. In a purely slot-filling system, native support is provided for simplifying slots from user utterances but extra code needs to be written to model the control logic (for instance, 'the user still didn't specify his choice of size, so the question for size should be repeated.'). Several commercial speech recognition vendors provide APIs that operate at this level. In addition, many vendors support low-level APIs that provide basic access to recognition results (i.e., text strings) but do not perform any additional processing. We refer to these as recognizer-only APIs. They serve more as raw speech recognition engines and require significant programming to first implement a slot-filling engine and, later, control logic to mimic all possible opportunities for staging. Examples of the two latter technologies can be seen in the commercial telephone-based speech recognition market (from companies such as Nuance, SpeechWorks, and IBM). The study presented in this paper suggests a systematic way by which their capabilities for mixed-initiative interaction can be assessed. Table 1 also shows that in the latter two software technologies, our partial evaluation model can be implemented to achieve mixed-initiative interaction.

4. DISCUSSION

Our work makes contributions to both partial evaluation and mixed-initiative interaction. For the partial evaluation community, we have identified a novel application where the motivation is the staging of interaction (rather than speedup). Since programs (dialogs) are used as specifications of interaction, they are *written to be partially evaluated*; partial evaluation is hence not an 'afterthought' or an optimization. An interesting research issue is: Given (i) a set of interaction sequences, and (ii) addressable information (such as arguments and slot variables), determine (iii) the smallest program so that every interaction sequence can be staged in a model such as Fig. 5 (right). As stated earlier, this requires algorithms to automatically decompose and 'layer' interaction sequences into those that are best addressed in the interpreter and those that can benefit from representation and specialization by the partial evaluator.

For mixed-initiative interaction, we have presented a programming model that accommodates all possibilities of staging, without explicit enumeration. The model makes a distinction between fixed-initiative (which has to be explicitly programmed) and mixed-initiative (specifications of which can be compacted for subsequent partial evaluation). We have identified instantiations of this model in VoiceXML and slot-filling APIs. We hope this observation will help system designers gain additional insight into voice application design strategies.

It should be recalled that there are various facets of mixed-initiative that are not addressed in this paper. Besides subdialog invocations, VoiceXML's design can support dialogs such as shown in Fig. 19. *Caller 1*'s request, while demonstrating initiative, implies a dialog with an optional stage (which cannot be modeled by partial evaluation). Such a situation has to be trapped by the interpreter, not by partial evaluation. *Caller 2* does specify a staging, but his staging poses constraints on the computer's initiative, not his own. Such a 'meta-dialog' facet [5] requires the ability to jump out of the current dialog; VoiceXML provides many elements for describing such transitions. Extending our programming model to cover these facets is an immediate direction of future research.

VoiceXML also provides certain 'impure' features and side-effects in its programming model. For instance, after selecting a size (say, medium), the caller could retake the initiative in a different part of the dialog and select a size again (this time, large). This will cause the new value to override any existing value in the `size` slot. In our model, this implies the dynamic substitution of an earlier, 'evaluated out,' stage with a functional equivalent. Obviously, the dialog manager has to maintain some state (across partial evaluations) to accomplish this feature or support a notion of despecialization. This suggests new directions for research in program transformation.

It is equally possible to present the above feature of VoiceXML as a shortcoming of its implementation of mixed initiative. Consider that after selection of a size, the scope of any future mixing of initiative should be restricted to the remaining slots (topping and crust). The semantics of graph traversal presented earlier capture this requirement. Such an effect is cumbersome to achieve in VoiceXML and would

Software Technology	Support for Slot Simplification	Support for Interaction Staging
VoiceXML	√	√
Slot Filling Systems	√	×
Recognizer-Only APIs	×	×

Table 1: Comparison of software technologies for voice-based mixed-initiative applications.

1	**System:**	Thank you for calling Joe's pizza ordering system.
2	**System:**	What size pizza would you like?
3	**Caller 1:**	What sizes do you have?
3	**Caller 2:**	Err.. Why don't you ask me the questions in topping-crust-size order?

Figure 19: Other mixed-initiative conversations that are supported by VoiceXML.

probably require transitioning to progressively smaller forms (with correspondingly restrictive form-level grammars). Our model provides this feature naturally; after size has been partially evaluated 'out,' the scope of future partial evaluations is automatically restricted to involve only topping and crust.

Our long-term goal is to characterize mixed initiative facets, not in terms of initiative, interaction, or task models but in terms of the opportunities for staging and the program transformation techniques that can support such staging. This means that we can establish a taxonomy of mixed-initiative facets based on the transformation techniques (e.g., partial evaluation, slicing) needed to realize them. Such a taxonomy would also help connect the facets to design models for interactive software systems. We also plan to extend our software model beyond slot-and-filler structures, to include reasoning and exploiting context.

5. NOTES

The work presented in this paper is supported in part by US National Science Foundation grants DGE-9553458 and IIS-9876167. After this paper was submitted, a new version (version 2.00) of the VoiceXML specification was released [12]. Our observations about the instantiation of our model in the VoiceXML dialog management architecture also apply to the new specification.

6. REFERENCES

[1] J. Allen. *Natural Language Understanding*. Benjamin Cummings, 1995. Second Edition.

[2] J. Allen, D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. Towards Conversational Human-Computer Interaction. *AI Magazine*, 2001. to appear.

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

[4] L. Boyer, P. Danielsen, J. Ferrans, G. Karam, D. Ladd, B. Lucas, and K. Rehor. Voice eXtensible Markup Language: VoiceXML. Technical report, VoiceXML Forum, May 2000. Version 1.00.

[5] 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.

[6] M. Coulthard. *An Introduction to Discourse Analysis*. Longman, London, 1977.

[7] E. Goffman. Replies and Responses. *Language in Society*, Vol. 5:pages 257–313, 1976.

[8] S. Haller and S. McRoy. *Computational Models for Mixed Initiative Interaction (Papers from the 1997 AAAI Spring Symposium)*. Technical Report SS-97-04, AAAI/MIT Press, 1997.

[9] N. Jones, C. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall International, 1993.

[10] D. Jurafsky and J. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2000.

[11] S. Levinson. *Pragmatics*. Cambridge University Press, 1983. Cambridge Textbooks in Linguistics.

[12] 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 2.00.

[13] D. 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.

[14] Pérez-Quiñones, M.A. and Sibert, J.L. A Collaborative Model of Feedback in Human-Computer Interaction. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'96)*, pages 316–323. Vancouver, BC, Canada, 1996.

[15] N. Ramakrishnan and S. Perugini. The Partial Evaluation Approach to Information Personalization. *ACM Transactions on Information Systems*, August 2001. Communicated for publication. Also available as Technical Report cs.IR/0108003, Computing Research Repository (CoRR) at `http://xxx.lanl.gov/abs/cs.IR/0108003`.

[16] F. van Harmelen and A. Bundy. Explanation-Based Generalisation = Partial Evaluation. *Artificial Intelligence*, Vol. 36(3):pages 401–412, 1988.

[17] M. Veloso, J. Carbonell, A. Pérez, D. Borrajo, E. Fink, and J. Blythe. Integrating Planning and Learning: The PRODIGY Architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 7(1):pages 81–120, 1995.