6-2016

A Language-Based Model for Specifying and Staging Mixed-Initiative Dialogs

Saverio Perugini
University of Dayton, sperugini1@udayton.edu

Joshua W. Buck
University of Dayton, jbuck1@udayton.edu

A Language-based Model for Specifying and Staging Mixed-initiative Dialogs

Saverio Perugini **Joshua W. Buck**
Department of Computer Science
University of Dayton
Dayton, Ohio 45429 USA
{saverio,jbuck1}@udayton.edu

ABSTRACT

Specifying and implementing flexible human-computer dialogs, such as those used in kiosks, is complex because of the numerous and varied directions in which each user might steer a dialog. The objective of this research is to improve dialog specification and implementation. To do so we developed a model for specifying and staging mixed-initiative dialogs. The model involves a dialog authoring notation, based on concepts from programming languages, for specifying a variety of unsolicited reporting, mixed-initiative dialogs in a concise representation that serves as a design for dialog implementation. Guided by this foundation, we built a dialog staging engine which operationalizes dialogs specified in this notation. The model, notation, and engine help automate the engineering of mixed-initiative dialog systems. These results also provide a proof-of-concept for dialog specification and implementation from the perspective of theoretical programming languages. The ubiquity of dialogs in domains such as travel, education, and health care with the increased use of interactive voice-response systems and virtual environments provide a fertile landscape for further investigation of these results.

ACM Classification Keywords

F.3.2. Semantics of Programming Languages: Partial Evaluation; H.5.2. Information Interfaces and Presentation (e.g. HCI): User Interfaces

Author Keywords

currying; human-computer dialogs; lambda calculus; mixed-initiative dialogs; mixed-initiative interaction; partial evaluation; task modeling

INTRODUCTION

From automated teller machines (ATMs), airport and train kiosks, and smart phone apps to installation wizards and intelligent tutoring or training, human-computer dialogs are woven into the fabric of our daily interactions with computer systems. While supporting flexibility in dialog is essential to deliver a personalized experience to the user, it makes the implementation challenging due to the numerous and varied directions in which a user might desire to steer a dialog, all of which must be supported by an implementation. This problem is difficult since dialogs range in complexity from those modeled after a simple, fixed, predefined series of questions and answers to those that give the user a great deal of control over the flow of the dialog, where the user and system act as equal participants by sharing and exchanging initiative, called *mixed-initiative* dialog [14, 19, 26, 44].

Consider the mixed-initiative dialog between a user and a flight reservation agent in Figure 1 that illustrates the rich interaction possible through dialog and the complexities involved in its implementation. The agent begins by soliciting a departure airport (line 1) and the user responds directly by saying 'Chicago' (line 2). In *fixed* dialogs, exchanges between the participants proceed in this manner, where the system cannot deviate from its pre-defined script and the user, therefore, must respond to the prompts in the order in which they are presented. Fixed dialogs are easy to implement because the top-down, control flow of the program (see Figure 2) reflects the only possible path through the dialog.

Dialogs become flexible, but more difficult to implement, when a user is permitted to deviate from the system's hard-wired, one-size-fits-all motif. For instance, note that in line 4, the user, rather than responding directly to the agent's solicitation for departure time (line 3), provides a destination instead. To enable such *unsolicited* responses [1], the implementation must support multiple paths to dialog completion. "A central problem for mixed-initiative dialogue management is coping with utterances that fall outside of the expected sequence of the dialogue" [42]. Intractable approaches, due to the combinatorial explosion in paths through a dialog, are to re-order the prompts in Figure 2 multiple ways to model all possible orderings/combinations of departure time, departure airport, and destination airport, or to dynamically branch to label `L2` (from label `L1`) followed by a jump back to `L1` in processing the input 'Brussels' when soliciting for departure time. "Authoring a dialogue is like writing a movie script with many different endings" [23]. What we desire is the ability to support *all* possible orderings/combinations from a script or task model enumerating/modeling only *one*.

Lines 6–8 demonstrate a sub-dialog where the user must determine an airline in another line of inquiry before she can respond to the solicitation for departure time (line 3),

EICS'16, June 21–24, 2016, Brussels, Belgium

ACM 978-1-4503-4322-0/16/06. . . $15.00
DOI: http://dx.doi.org/10.1145/2933242.2933262

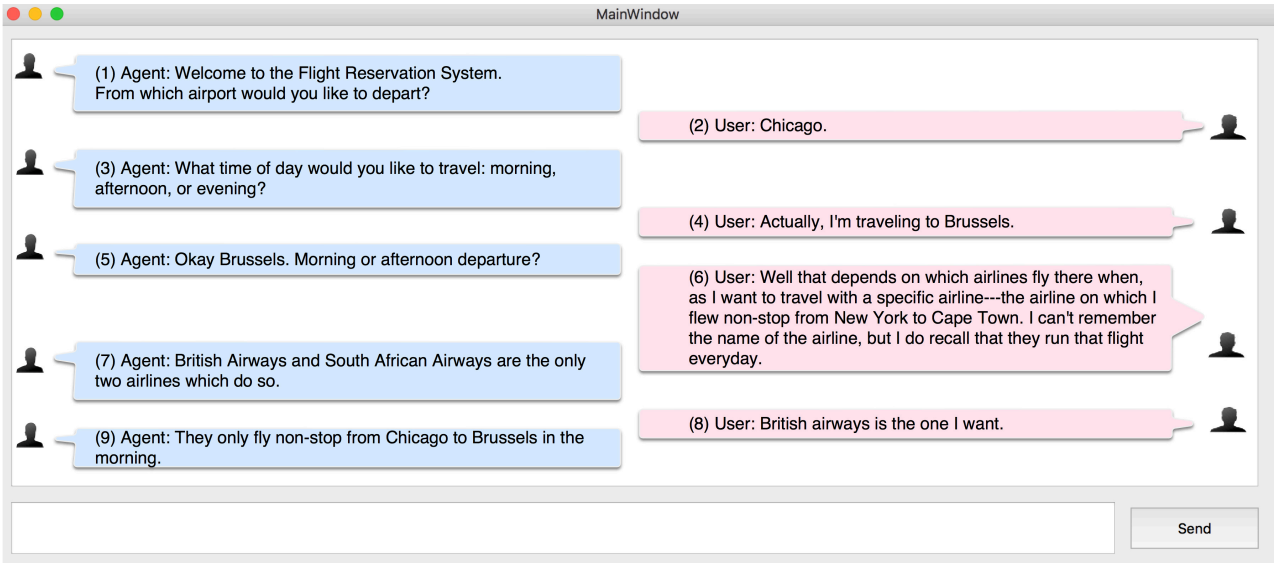


Figure 1. A mixed-initiative dialog.

```
L1:; prompt for departure time
   ; process departure time response
   ; prompt for departure airport
L2:; process departure airport response
   ; prompt for destination airport
   ; process destination airport response
```

Figure 2. A fixed-dialog script simplified for presentation.

which involves similar complexity in the control flow. Thus, "[d]eveloping a mixed-initiative dialog system is a complex task" [20] and "involves a very intensive programming effort" [17]. We address this problem through the development of a model for specifying and implementing mixed-initiative dialogs. The fundamental aspect of our model is our novel use of program transformations (e.g., partial evaluation) and other concepts from programming languages (e.g., functional currying) to specify and stage mixed-initiative dialogs, supporting this 'model one path, yet support many paths' theme. Our model involves a language-based dialog authoring notation and a dialog management engine that can stage dialogs represented with it. We seek to generalize the specification of dialogs, and improve and automate the engineering of task-based, dialog systems supporting this type of flexibility in human-computer interaction.

SPECIFYING MIXED-INITIATIVE DIALOGS

Fixed- and Mixed-initiative Dialogs

Consider a dialog to purchase gasoline using a credit card. The customer must first swipe the card, then choose a grade of octane, and finally indicate whether he desires a receipt. Such a dialog is a *fixed* dialog due to the fixed order of the questions from which the user is not permitted to deviate in his responses [1].

An *enumerated specification* is a set of episodes, and an *episode* is an ordered list of questions to be posed and answered from the start of the dialog to completion. Intuitively, an enumerated specification is a set of all possible ways to complete a dialog. Formally, a dialog specification is a set of *totally ordered sets*. We use a *Hasse* diagram, a graphical depiction of a *partially ordered set*, to represent a dialog specification. A relation R with the set S over whose Cartesian product R is defined is a *strict partially ordered set* (or *poset*) if R is an irreflexive, asymmetric, and transitive relation. This means that some of the elements of S may be unordered based on the relation R. On the other hand, a set S is a *strict totally ordered set* according to a relation R if and only if for every two elements $(x, y) \in S$, xRy or yRx. Every totally ordered set is also a partially ordered set, but the reverse is not necessarily true.

An enumerated specification of this gasoline dialog is $\{\prec$credit-card grade receipt$\succ\}$, and Table 1 (column a) illustrates the Hasse diagram that specifies it. A Hasse diagram is read bottom-up. Here, the set S of the poset is the set of the questions posed in the dialog and R of the poset is the 'must be answered before' relation denoted with an upward arrow between the source and target of the arrow.

Our authoring notation for dialog specification in a compressed manner is based on concepts from programming languages. In this notation a dialog is specified by an expression of the form $\frac{X}{T}$, where x represents a program transformation or language concept and τ represents a list of terms, where each term represents either a question (of the dialog) or a sub-dialog expression (introduced below) in the dialog being specified. Each expression represents a set of episodes (i.e., an enumerated specification). The main thematic idea is that the set of episodes specified by an expression of this form correspond to all possible ways that a function parameterized by the terms (e.g., dialog questions) in the denominator can be partially applied, and re-partially applied, and so on, progressively,

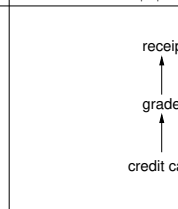
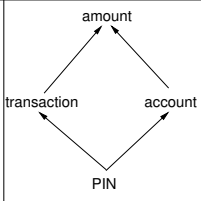
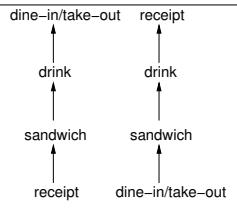
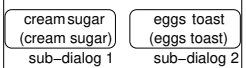
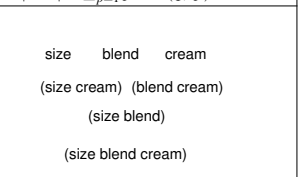
	⟵ (most rigid) fixed dialogs			.....	complete, mixed-initiative dialogs (most flexible) ⟶
ID	a	b	c	d	e
Enumerated Specification	{≺credit-card grade receipt≻}	{≺PIN transaction account amount≻, ≺PIN account transaction amount≻}	{≺receipt sandwich beverage dine-in/take-out≻, ≺dine-in/take-out sandwich beverage receipt≻}	{≺cream sugar eggs toast≻, ≺cream sugar toast eggs≻, ≺(cream sugar) toast eggs≻, ≺(cream sugar) eggs toast≻, ≺sugar cream eggs toast≻, ≺sugar cream toast eggs≻, ≺eggs toast cream sugar≻, ≺eggs toast sugar cream≻, ≺toast eggs cream sugar≻, ≺toast eggs sugar cream≻, ≺sugar cream (eggs toast)≻, ≺cream sugar (eggs toast)≻, ≺(eggs toast) (cream sugar)≻, ≺(cream sugar) (eggs toast)≻}	{≺(size blend cream)≻, ≺(size blend) cream≻, ≺cream (size blend)≻, ≺(blend cream) size≻, ≺size (blend cream)≻, ≺(size cream) blend≻, ≺blend (size cream)≻, ≺size blend cream≻, ≺size cream blend≻, ≺blend size cream≻, ≺blend cream size≻, ≺cream blend size≻, ≺cream size blend≻}
Size	$\lvert C\rvert = 1$				$\lvert PE^{\star}\rvert = \sum_{p=1}^{q=3} p! \times S(q,\ p) = 13$
Hasse diagram					
PL Not.	$\frac{C}{\text{credit-card grade receipt}}$	$\frac{C}{\text{PIN}\ \frac{SPE'}{\text{transaction account}}\ \text{amount}}$	$\frac{C}{\text{receipt sandwich drink dine-in/take-out}} \cup \frac{C}{\text{dine-in/take-out sandwich drink receipt}}$	$\frac{SPE'}{\frac{PE^{\star}}{\text{cream sugar}}\ \frac{PE^{\star}}{\text{eggs toast}}}$	$\frac{PE^{\star}}{\text{size blend cream}}$
Implementation	$[\![\texttt{mix}]\!]([\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{size} = \ldots], \text{blend} = \ldots], \text{cream} = \ldots]$				$[\![\texttt{mix}]\!][\texttt{f}, \text{size}, \text{blend}, \text{cream}]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{size} = \ldots], \text{blend} = \ldots, \text{cream} = \ldots]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{blend} = \ldots, \text{cream} = \ldots], \text{size} = \ldots]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{blend} = \ldots], \text{cream} = \ldots, \text{size} = \ldots]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{cream} = \ldots, \text{size} = \ldots], \text{blend} = \ldots]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{cream} = \ldots], \text{size} = \ldots, \text{blend} = \ldots]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{size} = \ldots, \text{blend} = \ldots], \text{cream} = \ldots]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{size} = \ldots], \text{cream} = \ldots], \text{blend} = \ldots]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{size} = \ldots], \text{blend} = \ldots], \text{cream} = \ldots]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{blend} = \ldots], \text{size} = \ldots], \text{cream} = \ldots]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{blend} = \ldots], \text{cream} = \ldots], \text{size} = \ldots]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{cream} = \ldots], \text{size} = \ldots], \text{blend} = \ldots]$, $[\![\texttt{mix}]\!]([\![\texttt{mix}]\!]([\![\texttt{mix}]\!][\texttt{f}, \text{cream} = \ldots], \text{blend} = \ldots], \text{size} = \ldots]$

Table 1. A spectrum of dialogs from fixed (column a) to complete, mixed-initiative dialogs (column e), encompassing a variety of unsolicited reporting, mixed-initiative dialogs, in three representations: enumerated specification (second row), Hasse diagram (third row), and our notation (fourth row). The last (fifth) row gives the expression, calling partial evaluation ($[\![\texttt{mix}]\!]$), used to stage each dialog.

according to the semantics of the transformation operator or language concept in the numerator.[1]

We use the concept of **C***urrying* [13] to specify a fixed dialog, where only one fixed episode is permitted. Currying transforms a function $f_{uncurried}$ with type signature $(p_1 \times p_2 \times \cdots \times p_n) \rightarrow r$ to a function $f_{curried}$ with type signature $p_1 \rightarrow (p_2 \rightarrow (\cdots \rightarrow (p_n \rightarrow r)\cdots))$, such that $f_{uncurried}(a_1, a_2, \cdots, a_n) = (\cdots((f_{curried}(a_1))(a_2))\cdots)(a_n)$. Currying $f_{uncurried}$ and running the resulting $f_{curried}$ function has the same effect as progressively partially applying $f_{uncurried}$, resulting in a dialog spread across multiple stages of interaction (i.e., questions and answers), but still in a fixed, prescribed order (e.g., Q: 'Credit or debit?' A: 'Credit,' Q: 'What grade octane?' A: '93,' Q: 'Receipt?' A: 'Yes'). For instance, a curried function representing the gasoline dialog $gasoline_{curried}$ has type signature $payment\_type \rightarrow (grade\_octane \rightarrow (receipt? \rightarrow dialog\_complete))$; evaluating it to completion requires three distinct steps or applications: $((gasoline_{curried}(\texttt{Visa}))(\texttt{93}))(\texttt{yes})$. A specification of the gasoline dialog in our notation is $\frac{C}{\text{credit-card grade receipt}} = \{\prec\text{credit-card grade receipt}\succ\}$.

Flexible dialogs typically support multiple completion paths. For instance, consider a dialog for ordering coffee. The participant must select a size and blend, and indicate whether room for cream is desired. Since possible responses to these questions are completely independent of each other, the dialog designer may wish to permit the participant to communicate the answers in any combinations and in any order. For example, some customers may prefer to use a $\frac{C}{\text{size blend cream}} = \{\prec\text{size blend cream}\succ\}$ episode:

SYSTEM: What size would you like?
USER: Small.
SYSTEM: Which blend would you like?
USER: Dark.
SYSTEM: Room for cream?
USER: No.

Others may prefer a $\frac{C}{\text{blend cream size}} = \{\prec\text{blend cream size}\succ\}$ episode:

SYSTEM: Which blend would you like?
USER: Light.
SYSTEM: Room for cream?
USER: Yes.
SYSTEM: What size would you like?
USER: Large.

Note that, in this notation, the order of the terms in the denominator matters (i.e., $(\frac{C}{\text{a b c}} = \{\prec\text{a b c}\succ\}) \neq (\frac{C}{\text{b a c}} = \{\prec\text{b a c}\succ\})$). Still others might prefer to use a $\prec$(size blend) cream$\succ$ episode, where answers to the questions enclosed in parentheses must be communicated in a single utterance (i.e., all at once):

SYSTEM: What size and which blend would you like?
USER: Small, french roast.
SYSTEM: Room for cream?
USER: No.

[1] This notation was introduced in [7] and revised in [30]. Here, we enrich it with additional concepts and modify its semantics.

We use the concept of **I***nterpretation* [13] to specify a dialog where *all* the responses to all dialog questions must be communicated in a single utterance (e.g., Q: 'What size and which blend would you like?' A: 'Small, dark roast.'), such as $\frac{I}{\text{size blend}} = \{\prec(\text{size blend})\succ\}$, because interpreting a function requires that *all* arguments be supplied at the time of the call, corresponding to a complete evaluation. Our notation is expressive enough to capture such dialogs involving sub-dialog(s) [1] by nesting these expressions in the denominator. For instance, we model this dialog as $\frac{C}{\frac{I}{\text{size blend}}\text{cream}} = \prec(\text{size blend})\ \text{cream}\succ$.

To accommodate all dialog completion paths we specify this dialog with the enumerated specification shown in the cell at the second row of column e in Table 1. Note that this specification indicates that answers to the set of questions in the dialog may be communicated in utterances corresponding to all possible set partitions of the set of questions, and using all possible permutations of those partitions. The Hasse diagram for this dialog is also given in column e. The absence of arrows between the size, blend, cream, (size blend), (size cream), (blend cream), and (size blend cream) elements indicates that the times at which each of those utterances may be communicated are unordered. Note that a specification of a dialog in our notation is a compressed representation capturing its requirements. Moreover, the compression is lossless (i.e., the episodes in the enumerated specification may be reconstructed from the expression).

Giving the user more flexibility in how to proceed through a dialog increases the number of episodes in its enumerated specification. This coffee-ordering dialog is a *mixed-initiative* dialog [1]. There are multiple tiers of mixed-initiative interaction; the tier considered in this article is called *unsolicited reporting*—an interaction strategy where, in response to a question, at any point in the dialog, the user may provide an unsolicited response to a forthcoming question. When all possible permutations (i.e., orders) of all possible partitions (i.e., combinations) of responses to questions are supported, we call the dialog a *complete*, mixed-initiative dialog. We use the program transformation *partial evaluation* [22] to specify *complete, mixed-initiative dialogs*. We first give the details of partial evaluation, and then illustrate how to specify this complete, mixed-initiative dialog in our notation using partial evaluation.

Partial Evaluation

We use the symbol `mix` from [22] to denote the partial evaluation operation because partial evaluation involves a `mix`ture of interpretation and code generation. The `mix` operator accepts two arguments: a function to be partially evaluated and a static assignment of values to any subset of its parameters. The semantics of the expression $[\![\texttt{f}]\!]\,3$ in the notation from [22] are 'invoke `f` on 3' or `f(3)`. Consider a function `pow` that accepts a `base` and an `exponent`, in that order, as arguments and returns the `base` raised to the `exponent`. The semantics of the expression $[\![\texttt{mix}]\!][\texttt{pow}, \texttt{exponent} = 2]$ are 'partially evaluate `pow` with respect to `exponent` equal to two,' an operation which returns $\texttt{pow}_{\texttt{exponent}=2}$ that accepts only a `base` (i.e., a squaring function). Therefore, $\overbrace{[\![[\![\texttt{mix}]\!][\texttt{pow}, \texttt{exponent} = 2]]\!]}^{\text{a partial evaluation, } \texttt{pow}_{\texttt{exponent}=2}} 3 = \overbrace{[\![\texttt{pow}]\!][3,2]}^{\text{a complete evaluation}} = 9$.

	Only a single response per utterance	**Multiple responses per utterance**
Only one utterance	Confirmation dialog boxes common in application software; interpretation (I)	Online forms with multiple fields; interpretation (I)
Totally-ordered utterances	Purchasing gasoline with a credit card; buying beverages from a vending machine; currying (C) **(a)**	Providing a telephone, credit card, or PIN number through voice; partial function application n ($PFA_n^\star$)
Partially-ordered utterances	ATMs, and airport or train kiosks; single-argument partial evaluation (SPE') **(b)**	Ordering a coffee or pizza; partial evaluation ($PE^\star$) **(e)**

Table 2. Sample dialogs involving permutations or partitions of responses to questions. Parenthesized concept mnemonic in each cell indicates the language-based concept in our notation used to specify the dialog(s) in that cell. Bolded parenthesized letters (a), (b), and (e) connect these dialogs to those in Table 1.

Given a ternary function `f` with integer parameters `x`, `y`, and `z`: $\texttt{f}_{\texttt{y}=2} = [\![\texttt{mix}]\!][\texttt{f}, \texttt{y} = 2]$ and $[\![\texttt{f}]\!][1,2,3] = [\![[\![\texttt{mix}]\!][\texttt{f}, \texttt{y} = 2]]\!][1,3]$. In general, $[\![[\![\texttt{mix}]\!][f, input_{static}]]\!] input_{dynamic} = [\![f]\!][input_{static}, input_{dynamic}]$.

Partial evaluation accepts a function of any arity as input and is a *closed* operator over its domain (i.e., it takes a function as input and returns a function as output). Here, we are interested in a progressive series of applications of it that terminates at a fixpoint. Therefore, we superscript a concept mnemonic X in the numerator with a $\star$, where applicable, to indicate a progressive series of applications of the corresponding function ending at a fixpoint. For instance, the expression $\frac{PE^\star}{\text{size blend cream}}$ which denotes the set of all six permutations of {size, blend, cream} and all permutations of all set partitions of {size, blend, cream} or, in other words, all thirteen, possible episodes to complete the dialog given in Table 1 (second row, column e). Repeatedly applying $[\![\texttt{mix}]\!]$ as shown the last row of column e in Table 1 realizes these episodes. Table 1 represents a space from fixed to complete, mixed-initiative dialogs, encompassing a wide variety of unsolicited reporting, mixed-initiative dialogs. Table 2 identifies some practical, everyday dialogs that fall into the cross product of permutations and partitions of responses to questions.

Additional Language Concepts

There is a combinatorial explosion in the number of possible dialogs between the fixed and complete, mixed-initiative ends of the spectrum in Table 1. Specifically, the number of dialogs possible in this space is $2^{|PE_q^\star|} - 1 = \sum_{r=1}^{|PE_q^\star|} \binom{|PE_q^\star|}{r}$ (i.e., all possible subsets, save for the empty set, of all episodes in a complete, mixed-initiative dialog $PE_q^\star$), where $PE_q^\star$ represents the enumerated specification of a complete, mixed-initiative dialog given q, the number of questions posed in the dialog. We use additional concepts from lambda calculus [13], namely partial function application (PFA_1), partial function application n (PFA_n), and single-argument partial evaluation (SPE), to enrich our notation for specifying these dialogs. Partial function application, `papply1`, takes a function and its first argument and returns a function accepting the remainder of its parameters. The function `papplyn`, on the other hand, takes a function f and all of the first n of m arguments to f where $n \leqslant m$, and returns a function accepting the remainder of its $(m-n)$

	ID	⟵ …dialogs between fixed dialogs and complete, mixed-initiative dialogs (Δ)… ⟶				
PL Ntd		f	g	h	i	j
		$\frac{PFA_n}{\text{size blend cream}}$	$\frac{PFA_n^\star}{\text{size blend cream}}$	$\frac{SPE}{\text{size blend cream}}$	$\frac{SPE'}{\text{size blend cream}}$	$\frac{PE}{\text{size blend cream}}$
Enum. Spec.		{≺(size blend cream)≻, ≺(size (blend cream)≻, ≺(size blend) cream≻}	{≺(size blend cream)≻, ≺(size (blend cream)≻, ≺(size (blend) cream≻, ≺size blend cream≻}	{≺size (blend cream)≻, ≺(blend (size cream)≻, ≺cream (size blend)≻}	{≺(size blend cream≻, ≺(size cream blend≻, ≺(blend size cream≻, ≺(blend cream size≻, ≺(cream blend size≻, ≺(cream size blend≻}	{≺(size blend cream)≻, ≺(size (blend cream)≻, ≺(blend (size cream)≻, ≺(cream (size blend)≻, ≺(size (blend) cream≻, ≺(size (cream) blend≻, ≺(blend cream) size≻}
Size		$\lvert PFA_n\rvert = q = 3$	$\lvert PFA_n^\star\rvert = 2^{q-1=3-1=2} = 4$	$\lvert SPE\rvert = q = 3$	$\lvert SPE'\rvert = q! = 3! = 6$	$\lvert PE\rvert = \sum_{p=1}^{q=3}\binom{q}{p} = 7$
Implementation		⟦mix⟧[f, size = …, blend = …, cream = …], ⟦mix⟧[⟦mix⟧[f, size = …], blend = …, cream = …], ⟦mix⟧[⟦mix⟧[f, size = …, blend = …], cream = …]	⟦mix⟧[f, size = …, blend = …, cream = …], ⟦mix⟧[⟦mix⟧[f, size = …], blend = …, cream = …], ⟦mix⟧[⟦mix⟧[f, size = …, blend = …], cream = …], ⟦mix⟧[⟦mix⟧[⟦mix⟧[f, size = …], blend = …], cream = …]	⟦mix⟧[⟦mix⟧[⟦mix⟧[f, size = …], cream = …], blend = …], ⟦mix⟧[⟦mix⟧[f, blend = …], size = …, cream = …], ⟦mix⟧[⟦mix⟧[f, cream = …], size = …, blend = …]	⟦mix⟧[⟦mix⟧[⟦mix⟧[f, size = …], cream = …], blend = …], ⟦mix⟧[⟦mix⟧[⟦mix⟧[f, size = …], blend = …], cream = …], ⟦mix⟧[⟦mix⟧[⟦mix⟧[f, blend = …], size = …], cream = …], ⟦mix⟧[⟦mix⟧[⟦mix⟧[f, blend = …], cream = …], size = …], ⟦mix⟧[⟦mix⟧[⟦mix⟧[f, cream = …], size = …], blend = …], ⟦mix⟧[⟦mix⟧[⟦mix⟧[f, cream = …], blend = …], size = …]	⟦mix⟧[f, size = …, blend = …, cream = …], ⟦mix⟧[⟦mix⟧[f, size = …], cream = …, blend = …], ⟦mix⟧[⟦mix⟧[f, blend = …], size = …, cream = …], ⟦mix⟧[⟦mix⟧[f, cream = …], size = …, blend = …], ⟦mix⟧[⟦mix⟧[f, size = …, blend = …], cream = …], ⟦mix⟧[⟦mix⟧[f, size = …, cream = …], blend = …], ⟦mix⟧[⟦mix⟧[f, blend = …, cream = …], size]

Table 3. Specifications of dialogs in our notation (second row) and as enumerated specifications (third row). The last (fourth) row gives the expression, calling partial evaluation (⟦mix⟧), used to stage each dialog.

parameters. In single-argument partial evaluation, the input function may be partially evaluated with only one argument at a time. These concepts correspond to higher-order functions that each take a function and arguments for some subset of its parameters. All of these functions return a function. Like `mix`, these functions are general in that they accept a function of any arity as input, and the functions `curry`, `papply1`, `papplyn`, `smix` (single-argument partial evaluation) are *closed* operators over their domain. We can also superscript PFA_1, PFA_n, and *SPE* with a $\star$ symbol. For instance, repeatedly applying `papplyn` to a ternary function `f` as `(apply (papplyn (papplyn f small) mild) no)` realizes the episode ≺size blend cream≻ in addition to the ≺size (blend cream)≻, ≺(size blend) cream≻, and ≺(size blend cream)≻ episodes which are realized with only a single application of `papplyn`. The second row of Table 3 shows specifications of dialogs for ordering coffee in our authoring notation using only one concept mnemonic (and these dialogs are situated in the middle of the space depicted in Table 1). The third row gives the enumerated specification each expression represents.

Spectrum of Dialogs

These language-based concepts (and combinations of them) within the context of an expression in our notation help specify dialogs between the fixed and complete, mixed-initiative ends of the dialog spectrum shown in Table 1 and, thus, help bring structure to this space. For instance, consider a specification for an ATM dialog where PIN and amount must be entered first and last, respectively, but the transaction type (e.g., deposit or withdrawal) and account type (e.g., checking or savings) may be communicated in any order (see Table 1, column b): {≺PIN transaction account amount≻, ≺PIN account transaction amount≻}. We model this dialog, which contains an embedded, mixed-initiative sub-dialog (i.e., {≺transaction account≻, ≺account transaction≻}) as $\frac{C}{\text{PIN}\ \frac{SPE'}{\text{transaction account}}\ \text{amount}}$.

Alternatively, consider a dialog for ordering lunch where requesting a receipt or indicating whether you are dining-in or taking-out can be communicated either first or last, but specification of sandwich and beverage must occur in that order: {≺receipt sandwich beverage dine-in/take-out≻, ≺dine-in/take-out sandwich beverage receipt≻}. This dialog contains an embedded, fixed sub-dialog (i.e., {≺sandwich beverage≻}) and, unlike the prior examples, cannot be captured by a single poset or expression (see Table 1, column c). To specify such dialogs in our notation we use a union of expressions, called a *compound expression*: $\frac{C}{\text{receipt sandwich drink dine-in/take-out}} \cup \frac{C}{\text{dine-in/take-out sandwich drink receipt}}$.

Lastly, consider the dialog containing two embedded, complete, mixed-initiative sub-dialogs whose enumerated specification is shown in the second row of column d in Table 1. Here, the user can specify coffee and breakfast choices in any order, and can specify the sub-parts of coffee and breakfast in any order, but cannot mix the atomic responses of the two. For instance, the episode ≺cream eggs sugar toast≻ is not permitted because, if the user specifies 'cream' as the first utterance, the system must not accept an indication as to whether sugar is desired or not as the second utterance to be faithful to the dialog specification; by specifying 'eggs' in the second utterance, the user is pursuing the breakfast sub-dialog before completing the coffee sub-dialog pursued first and that interaction is not supported in the dialog specification. This dialog is represented as $\frac{SPE'}{\frac{PE^\star}{\text{cream sugar}}\ \frac{PE^\star}{\text{eggs toast}}}$.

While the star ($\star$) superscript permits repeated applications (but does not require them), the prime ($'$) superscript requires repeated applications of the operator until a fixpoint is reached. For instance, the episode ≺size (blend cream)≻ is specified by $\frac{SPE^\star}{\text{size blend cream}}$, but not by $\frac{SPE'}{\text{size blend cream}}$. In dialogs containing two or more terms in the denominator, where at least one of the terms is a sub-dialog (e.g., dialogs $\frac{\cdots}{\text{a}\ \frac{C}{\text{b c}}}$ and $\frac{\cdots}{\frac{C}{\text{a b}}\ \frac{PE^\star}{\text{c d}}}$, but not $\frac{\cdots}{\frac{I}{\text{a b}}}$), each of the *I*, PFA_n, $PFA_n^\star$, *PE*, and $PE^\star$ concept mnemonics is not a candidate for the numerator. This is because those concepts require (in the case of *I*) or support multiple responses per utterance and it is not possible to complete multiple sub-dialogs in a single utterance or complete a sub-dialog and an individual question in a single utterance. Only the PFA_1 and *SPE* concept mnemonics suffice for two categories of dialogs containing sub-dialogs: those with no more than two terms in the denominator, where one of the terms is a sub-dialog (e.g., $\frac{PFA_1}{\text{a}\ \frac{PE^\star}{\text{b c}}}$, $\frac{PFA_1}{\frac{PE^\star}{\text{a b}}\ \frac{PE^\star}{\text{c d}}}$, $\frac{SPE}{\text{a}\ \frac{PE^\star}{\text{b c}}}$, and $\frac{SPE}{\frac{PE^\star}{\text{a b}}\ \frac{PE^\star}{\text{c d}}}$) and those with more than two terms in the denominator where only the first term is a sub-dialog (e.g., $\frac{PFA_1}{\frac{PE^\star}{\text{a b}}\ \text{c d e f}}$ and $\frac{SPE}{\frac{PE^\star}{\text{a b}}\ \text{c d e f}}$). This is because when used as the numerator in an expression whose denominator contains more than two terms, one of which is a sub-dialog not in the first position, PFA_1 and *SPE* require multiple responses in the second and final utterance. Hence, *C* is the only concept mnemonic that can always be used in the numerator of an expression containing any arbitrary number of sub-dialogs in the denom-

inator. However, C only supports fixed orders of responses. Thus, we need a mnemonic for a concept that restricts utterances to one response and only permits one sub-dialog to be pursed at a time, but also permits all possible completion orders. Such a concept could be used to specify a dialog with more than two terms in the denominator, any of which can be a sub-dialog, that can be completed in any order. The concept represented by the mnemonic SPE' is ideal for this purpose (see column i in Table 3). Note that $\frac{C}{\frac{PE^\star}{\text{a b}}\ \frac{PE^\star}{\text{c d}}\ \frac{PE^\star}{\text{e f}}} \neq \frac{SPE'}{\frac{PE^\star}{\text{a b}}\ \frac{PE^\star}{\text{c d}}\ \frac{PE^\star}{\text{e f}}}$, but $\frac{C}{\frac{PE^\star}{\text{a b}}\ \frac{PE^\star}{\text{c d}}\ \frac{PE^\star}{\text{e f}}} \subset \frac{SPE'}{\frac{PE^\star}{\text{a b}}\ \frac{PE^\star}{\text{c d}}\ \frac{PE^\star}{\text{e f}}}$; the episode $\prec$(c d) f e a b$\succ$ is supported by the latter, but not by the former where there is a fixed-order on the sub-dialogs.

The row labeled 'Size' in Tables 1 and 3 provides formulas for the number of episodes in dialogs specifiable using only one concept mnemonic. Note that $I \cup C \cup PFA_1 \cup PFA_n \cup PFA_n^\star \cup SPE \cup SPE' \cup PE \subset PE^{\star 2}$ indicating that partial evaluation subsumes all other concepts in this model. The implication of this is that any dialog specified using this notation can be realized through partial evaluation (see last row of Tables 1 and 3).

We denote the space of dialogs possible given q, the number of questions posed in a dialog, with the symbol $\mathscr{U}_q$. Let x denote a concept mnemonic in this model (e.g., C or $PE^\star$). We use the symbol $\mathscr{X}_q$ to denote a *class* of dialogs (e.g., $\mathscr{C}_q$ or $\mathscr{PE}_q^\star$), where a class is a set of dialogs where each dialog in the set can be specified with only the concept mnemonic corresponding to the class. The number of dialogs possible given a value for q is $|\mathscr{U}_q| = 2^{|PE_q^\star|} - 1$ (i.e., all subsets, save for the empty set, of episodes in a complete, mixed-initiative dialog). Of those dialogs, there are $2^{|PE^\star|_q - 3q! - q - 5}$ dialogs that cannot be specified with a single concept (e.g., dialogs b, c, and d in Table 1) whose class we refer to as Δ. For instance, $\mathscr{U}_3$ = 8,191 ($= 2^{|PE_3^\star|} - 1 = 2^{13} - 1$) and Δ_3 = 8,165[3] ($= 2^{|PE_3^\star|} - 1 - 3q! - q - 5 = 8,192 - 3(3!) - 3 - 5$). However, we can specify each dialog in Δ using our authoring notation as a *compound expression* (e.g., dialog c in Table 1, or $\frac{I}{\text{x y z}} \cup \frac{PFA_1}{\text{x y z}} = \{\prec$(x y z)$\succ, \prec$x (y z)$\succ\}$) or with *sub-dialogs* through nesting (e.g., dialogs b and d in Table 1), or both (e.g., $\frac{C}{\text{size}\ \frac{SPE}{\text{blend cream}}} \cup \frac{C}{\text{blend}\ \frac{SPE}{\text{cream size}}} \cup \frac{C}{\text{cream blend size}}$).

An attractive consequence of this language-based notation for dialog specification is that the (nested) structure of the expression, and the language concepts used therein, provide a design pattern for staging (i.e., implementing) the dialog.

STAGING MIXED-INITIATIVE DIALOGS

Our notation for specifying mixed-initiative dialogs lends itself to two methods of dialog implementation: using i) partial evaluation [22] or ii) a set of rewrite rules [2] to stage the interaction. We use an example to illustrate how dialogs can be staged with partial evaluation. Consider the ternary Scheme function shown within a dotted border in Figure 3.[4] Note that it only models one dialog episode: $\prec$size blend cream$\succ$. We define this function without the intent of ever invoking it, and rather only with the intent of progressively transforming it automatically with partial evaluation to stage the interaction of a mixed-initiative dialog. Thus, we only use this function as a malleable data object, and when it has been completely consumed through transformation, the dialog is complete.

The top half of Figure 3 demonstrates how the $\prec$size blend cream$\succ$ episode is staged. This function can be used to realize a completely different episode than the one which it naturally reflects. For instance, the bottom half of Figure 3 demonstrates how the $\prec$cream blend size$\succ$ episode is staged, with the *same* function. While the control flow models only one episode (in this case, $\prec$size blend cream$\succ$), through partial evaluation we can stage the interaction required by thirteen distinct episodes. In general, by partially evaluating a function representing only one episode, we can realize $\sum_{p=1}^{q} p! \times S(q,\ p)$ distinct episodes (i.e., $|PE_q^\star|$), where q is the number of questions posed in a dialog, and $S(m,n)$ is size of the set of all partitions of a set of size m into exactly n non-empty subsets, where n is a positive integer and $n \leqslant m$ (i.e., the *Stirling number* of a set of size m [25]). This 'model one episode, stage multiple' feature is a significant result of our approach to dialog modeling and management, and the main theme around which our model for specifying and staging mixed-initiative dialogs is centered.

The dialog $\frac{I}{\text{size blend cream}} = \{\prec$(size blend cream)$\succ\}$ can be staged with partial evaluation as $[\![\texttt{mix}]\!][\text{f}, \text{size} = \ldots, \text{blend} = \ldots, \text{cream} = \ldots]$. Similarly, the dialog $\frac{PFA_1}{\text{size blend cream}} = \{\prec$size (blend cream)$\succ\}$ can be staged with partial evaluation as $[\![\texttt{mix}]\!][[\![\texttt{mix}]\!][\frac{PFA_1}{\text{size blend cream}}, \text{size} = \ldots], \text{blend} = \ldots, \text{cream} = \ldots]$. The last row of Tables 1 and Table 3 details how dialogs specified using only one concept mnemonic in an expression are staged by partial evaluation, which subsumes all of the other concepts based on the supplied arguments. For instance, $PFA_n^\star$ is achieved by progressively partially evaluating with any prefix of arguments (see last row, column g of Table 3).

Given a specification expression, an alternate implementation approach involves the use of rewrite rules to stage the interaction [2]. The concepts I and C are primitive in that any dialog modelable with our notation can be represented using *only* the I or C concept mnemonics in an expression. In particular, to specify any dialog in the spectrum shown in Table 1 we can simply translate each episode in its enumerated specification as a sub-expression with either an I or C in the numerator and the entire specification as a union of those sub-expressions. For instance, $\{\prec$(x y z)$\succ, \prec$x y z$\succ, \prec$y z x$\succ, \prec$z x y$\succ, \prec$x (y z)$\succ\} = \frac{I}{\text{x y z}} \cup \frac{C}{\text{x}\ \frac{I}{\text{y z}}} \cup \frac{C}{\text{y}\ \frac{I}{\text{z x}}} \cup \frac{C}{\text{z}\ \frac{I}{\text{x y}}} \cup \frac{C}{\text{x}\ \frac{I}{\text{y z}}}$. Therefore, we defined *rewrite rules*, not shown here, akin to those in [30], and can progressively apply them after every utterance, rather than partial evaluation itself, to transform the representation of the dialog, to stage it. For instance, the above dialog $\frac{PFA_1}{\text{size blend cream}}$ can be staged with term rewriting as $\frac{PFA_1}{\text{size blend cream}} = \frac{C}{\text{size}\ \frac{I}{\text{blend cream}}}$ (first rewrite), and $[\frac{C}{\text{size}\ \frac{I}{\text{blend cream}}}, \text{size} = \ldots] = \frac{C}{\frac{I}{\text{blend cream}}} = \frac{I}{\text{blend cream}}$ (second rewrite), and $[\frac{I}{\text{blend cream}}, \text{blend} = \ldots, \text{cream} = \ldots] = \ \sim$ (i.e., dialog

[2] When the denominator is irrelevant to the discussion at hand we drop it and simply use only the concept mnemonic to refer to a set of episodes.

[3] $\mathscr{U}_4 = 3.7 \times 10^{21}$ and $\Delta_4 = \sim 3.7 \times 10^{21}$.

[4] An expression of the form <...> is used to represent a list of valid choices (e.g., <sizes> could represent the list '(small medium large)). Moreover, the functions being partially evaluated in Figure 3 omit else (exceptional) branches for purposes of succinct exposition.

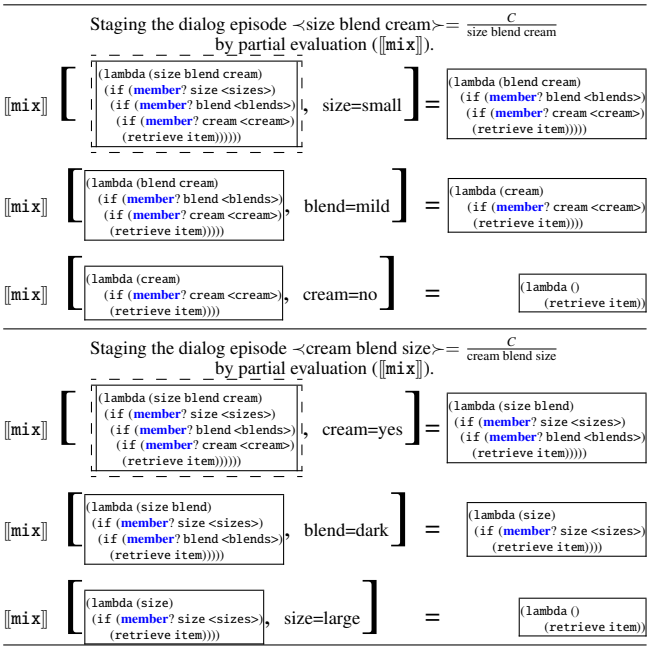


Figure 3. Staging dialog episodes by partial evaluation, explicitly illustrating the intermediate output of each partial evaluation. Dotted boxes reinforce that both series of transformations, top half and bottom half, start with the same function.

complete). Similarly, $[\frac{SPE'}{\frac{PE}{a\,b}\,\frac{PE}{c\,d}}, d = \ldots] = \frac{SPE'}{\frac{PE}{c}\,\frac{PE}{a\,b}} = \frac{SPE'}{\frac{C}{c}\,\frac{PE}{a\,b}} = \frac{C}{{}_{c}\frac{PE}{a\,b}}$, and $[\frac{C}{{}_{c}\frac{PE}{a\,b}}, c = \ldots] = \frac{C}{{}_{c}\frac{PE}{a\,b}} = \frac{PE}{a\,b}$, and $[\frac{PE}{a\,b}, b = \ldots] = \frac{PE}{a} = \frac{C}{a} = \frac{I}{a}$, and finally $[\frac{I}{a}, a = \ldots] = \sim$.

While complete, mixed-initiative dialogs can be staged efficiently using this approach, they represent only a fraction of all possible dialogs. Most dialog specifications contain less episodes than those that can be modeled by an expression with a $PE^{\star}$ in the numerator. However, since partial evaluation can be used to partially apply a function with respect to *any* subset of its parameters (i.e., it supports the partial application of a function with all possible orders and combinations of its arguments), we can stage any unsolicited reporting, mixed-initiative dialog in this space using only partial evaluation. For instance, note that the last row of Tables 1 and 3 demonstrates how to stage dialogs conforming to only a single language concept. However, while partial evaluation subsumes all other language concepts considered here, it does not discriminate against any of the possible partial assignments of arguments to parameters of the function being partially evaluated. A specification expression containing a concept mnemonic other than $PE^{\star}$ represents a particular type of restriction on partial evaluation (corresponding to restrictions on the ways of mixing initiative). Implementing dialogs with partial evaluation that cannot be specified with a single concept (e.g., dialogs b and d in Table 1) or with a non-compound expression (e.g., dialog c in Table 1) requires additional attention. To be faithful to a specification, we require a controller, we call a *stager*, to coordinate the judicious invocation of partial evaluation, with respect to the different orders and combinations of arguments that reflect the permissible episodes of a dialog, to realize or 'stage' the progressive interaction of the dialog (in all dialogs

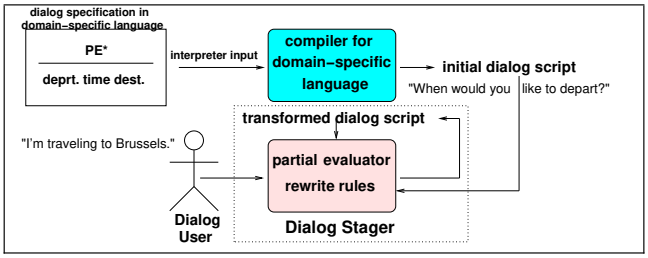


Figure 4. Conceptual design of prototype implementation.

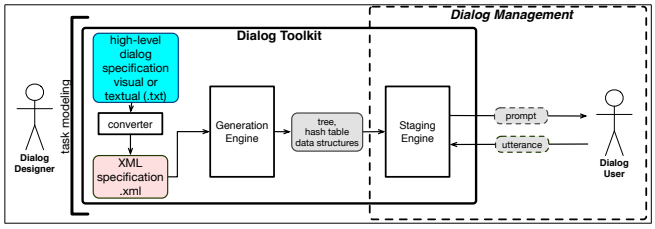


Figure 5. Dialog toolkit and resulting dialog system design and execution.

except complete, mixed-initiative dialogs—those conforming entirely to the $PE^{\star}$ concept).

Grounded in these theoretical principles, we prototyped this model for mixed-initiative dialogs, as a proof-of-concept, by building a system in Scheme which, given a specification of a mixed-initiative dialog in our notation, automatically generates a stager to execute the dialog (see Figure 4). The system includes a compiler (i.e., translator) from our dialog authoring notation to a stager in Scheme. Running the resulting stager enables the interaction depicted in Figure 1.

IMPLEMENTING MIXED-INITIATIVE DIALOGS

Guided by these principles, we built a cross-platform implementation of our model for mixed-initiative dialogs, including a dialog staging engine, using XML, C++, and Qt. Given an implementation-neutral representation of a specification of a mixed-initiative dialog in our notation, our system realizes the dialog. While a majority of the implementation details are beyond the scope of this paper, we make some remarks to convey the implementation strategy. Figure 5 provides an overview of the approach.

XML Specification of Dialog

The specification of a dialog is represented as an XML document using a variety of attributes to capture necessary information. The tree structure of the XML document mirrors the structure of the dialog specification expression. The `eval` attribute, whose value is the concept mnemonic (e.g., C or $PE^{\star}$) corresponding to the desired *interaction policy* [39] (e.g., system initiated or mixed-initiative), restricts the scope of responses and supports sub-dialogs. The XML specification is automatically generated from the specification expression, and a list of valid responses, synonyms, and other pertinent (contextual) information, in an ASCII text format.

Data Structures for Dialog Representation

The dialog generation engine converts the XML document into a tree. Figure 6 illustrates a dialog tree, where nodes are

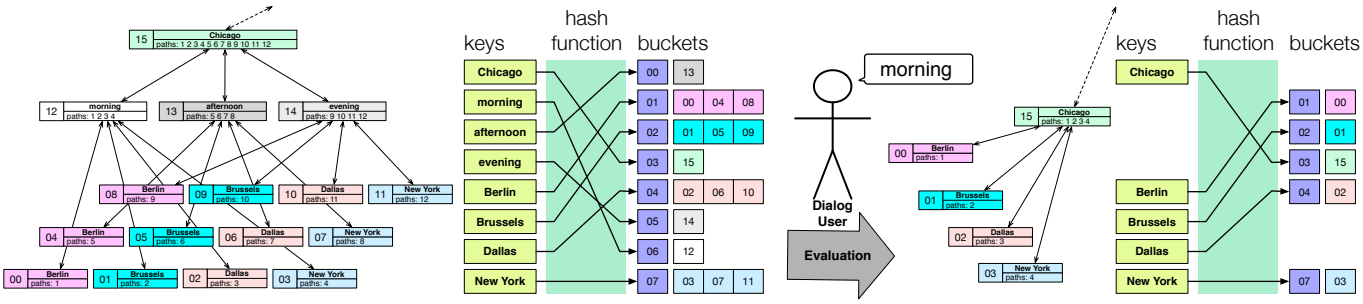


Figure 6. Conceptual transformation of data structures that represent a dialog in processing a user response (here, 'morning').

annotated with path vectors, and hash table from the flight reservation scenario. The edges of the tree[5] represent system solicitations and node labels represent valid user responses. For instance, in Figure 6, the bidirectional edge connecting 'Chicago' to 'afternoon' represents the solicitation 'What time of day would you like to travel?' When the user supplies a (solicited or unsolicited) response, the staging engine (discussed below) must identify all of the paths from the root to leaves containing a node(s) labeled with that response in approximately constant time. We build a response-to-nodes hash table, where each key corresponds to a valid response (e.g., 'morning') and each value is a pointer to all nodes[6] in the tree labeled with the key string for the purpose of identifying all of the nodes that need to be accessed when processing a user response. For instance, when the term 'morning' is accessed through the hash table, the result is a pointer to node 12 in Figure 6 that has the value 'morning.' We also annotate the nodes with vectors to support processing responses efficiently. To identify all paths that involve a node labeled with the user response efficiently, we assign an unsigned integer to each leaf node, and associate a *path vector*, with each non-leaf node, which contains the unsigned integers from each of its descendant leaves. In Figure 6, the node that represents the response 'morning' has four valid paths (i.e., [1,2,3,4]) that represent the four remaining ways to complete the dialog (i.e., by making a choice among the remaining possibilities for destination: Berlin, Brussels, Dallas, and New York).

Staging Engine

The staging engine processes user responses and stages the turns of the dialog. Upon user entry, the prompt that corresponds to the current root node is displayed to the user and the engine awaits a response. The current root node represents the user's place in the dialog. Any edges below the current root node represent solicitations which have yet to be made to the user, but will at some point in the dialog, unless the user provides an unsolicited response to any of those solicitations first. The staging engine captures an utterance from the user as a string of text which it parcels into a set of responses to the current or forthcoming solicitation(s). Words that are not discerned as English or lookup keys are ignored. Synonyms, if given, are then replaced with the corresponding keys. The string is then parsed for node labels, which are the keys to the hash table. Note that the interaction flow policy (e.g., C or $PE^{\star}$) determines the scope of responses to any given solicitation. Once a response is made and determined to be within scope, the resulting path vector(s) is accessed through the hash table. We compute the path vector of the new root node as the union of the intersection of each of the path vectors of all nodes labeled by the user response with the path vector of the current root node. As the user provides (additional) responses, the path vector of the root node shrinks in size commensurate with the reduction in the remaining paths leading to dialog completion.

Consider an example of this process from the flight reservation scenario. Assume the first user utterance is 'Chicago.' Figure 6 illustrates an example tree and hash table for that single departure airport. Assume the second utterance is 'morning.' Since the path vector of the node labeled 'morning' is [1,2,3,4], and since it has at least one path in common with the path vector of the current root node labeled 'Chicago' [1,2,3,4,5,6,7,8,9,10,11,12], it is a valid response. The path vector of the new root node labeled 'morning' is computed as the intersection of the path vector for the node labeled 'morning' and the path vector for current root node labeled 'Chicago'; that intersection is [1,2,3,4]. If the next utterance is 'New York', then the path vector of the new root node only contains one path, [3], which means that there is only one leaf node left, though multiple paths to it might still exist. (In this example however, it is the end of the dialog.) If there is more than one node labeled 'morning' left in the dialog tree with at least one path in common with the path vector of the current root node, then the path vector of the new root node is the union of the intersection of each node labeled by the user response with the path vector of the current root node.

The purpose of the path vector mechanism is to determine which nodes and edges have been (effectively) removed from the current (state of the) tree. The path vector associated with the current root node only contains the unsigned integers from each of its descendant leaves; we can think of these integers as enumerating the paths remaining in the dialog. If the path vector of a node does not have at least one such integer in common with the path vector of the current root node, then all paths from that node are ignored. The hash table and path vectors precomputed in the generation phase both

[5]Though not shown in Figure 6, the presence of crosslinks (i.e., encoded with `id` and `refid` attributes in XML) to model dependencies between responses (e.g., there is only evening flight from Dallas to Chicago and, thus, departure time need not be solicited) make this structure a directed acyclic graph.

[6]Note that some responses/keys (e.g., 'Berlin') label multiple nodes in the tree.

provide a fast evaluation and obviate the need to traverse the tree or extract or prune entire paths when processing a user response. If multiple responses are given in a single utterance, the process above is repeated for each response.

Sub-dialogs: Staging dialogs (in the Δ class) that involve sub-dialogs requires additional consideration. Once a sub-dialog is started, it must be completed before responses outside of its scope are available for use again. Thus, staging such dialogs involves not only supporting the particular interaction policy for the sub-dialog, which often is different from the parent dialog, but also coordinating entry to and exit from sub-dialogs. To support this requirement, the dialog label of each node are compared to each other. The dialog label is the unique identifier that is assigned to the first node in a dialog or sub dialog. Every node in a dialog or sub dialog has the same dialog label. Once a sub-dialog is started, the current node becomes the first unanswered node in the sub-dialog temporarily. Then, only other responses with the same dialog label as the current node are valid. Once the sub-dialog is completed, the current node becomes the first node in the parent dialog leading to solicitations for which a response has not been supplied.

Practical Considerations: We have implemented other features into our dialog engine, but due to space constraints, we only make some cursory remarks. Using first-class continuations [13] as the theoretical basis, we have implemented undo and redo operations available to the user between utterances. We also have added weights to the edges of the dialog tree and applied search and other optimization algorithms (e.g., shortest path in the flight reservation scenario) to support the user in *meta*dialog inquiry. The data structures were designed to be immutable so that multiple staging engine threads could safely access them concurrently.

EVALUATION

Evaluating models for mixed-initiative dialog is itself an unsolved problem for a variety of reasons including the extremely limited nature of existing data and the ambiguity of the very definition of initiative [17]. One way to capture the efficacy of a model is to evaluate how well the model fits data. In the context of our model, this means evaluating the frequency of dialogs that can be captured by our notation and how well it captures each. Given any value for q, the number of questions per episode, every dialog in the space $\mathscr{U}_q$ can be specified using our dialog authoring notation. Since the specification expression of a dialog serves as a design pattern for implementing it, the number of sub-expressions in the specification is an evaluation metric for how well the notation captures the specification. A complete, mixed-initiative dialog can be captured by one expression: e.g., $\frac{PE^{\star}}{\text{size blend cream}}$. If we remove only one—$\prec$(size blend cream)$\succ$—of the thirteen episodes from this dialog, specifying it requires five sub-expressions: $\frac{SPE'}{\text{size blend cream}} \cup \frac{SPE}{\text{size blend cream}} \cup \frac{C}{\frac{I}{\text{size blend}}\text{ cream}} \cup \frac{C}{\frac{I}{\text{blend cream}}\text{ size}} \cup \frac{C}{\frac{I}{\text{size cream}}\text{ blend}}$. We specified each of the 8,191 dialogs in $\mathscr{U}_3$ using our notation and computed the frequency that could be captured by 1, 2, ..., and 13 sub-expressions. Our results are shown in Figure 7 (e.g., there are 46 dialogs that can be specified with one expression, and 2,977 that can be specified with four sub-expressions).

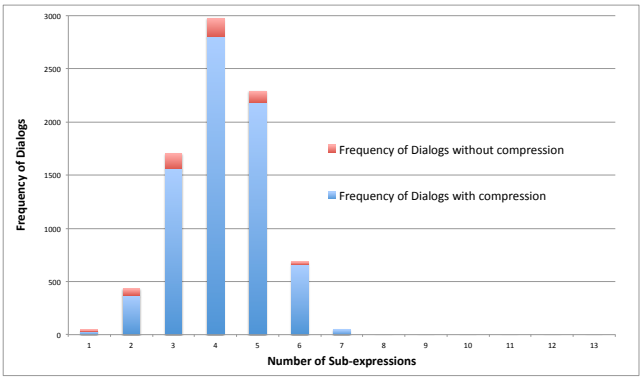


Figure 7. Histogram illustrating the frequency of dialog specifications in $\mathscr{U}_3$ (y-axis) that can be represented with 1–13 sub-expressions (x-axis).

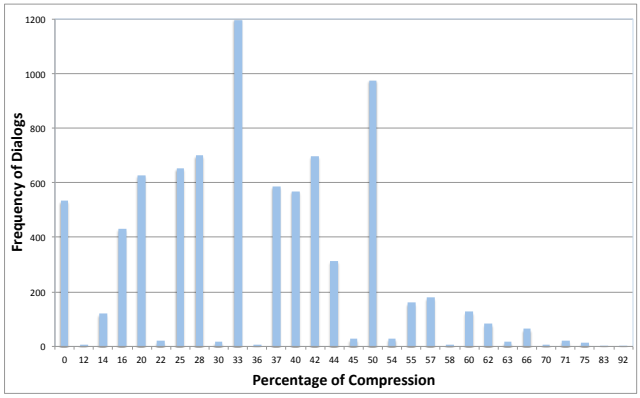


Figure 8. Histogram illustrating the frequency of the dialog specifications in $\mathscr{U}_3$ (y-axis) that can be compressed to the observed percentages (x-axis).

Since there are no dialogs in $\mathscr{U}_3$ that require greater than seven sub-expressions to model, and there are dialogs in the space with greater than seven episodes (e.g., the maximum number of episodes in any one dialog is thirteen for q=3), the use of our notation provides a compressed dialog specification. However, what is not illustrated in Figure 7 is the number of episodes in each dialog that can be represented with a particular number of sub-expressions or, in other words, the magnitude of the results given in Figure 7. For instance, if all 46 dialogs that can be represented with only one sub-expression only contain one episode, then there is no compression. To measure the efficacy of the compression, we computed the frequency of dialogs which can be specified at the observed compression percentages. For instance, 533 dialogs of the 8,191 could not be compressed at all (i.e., there is a one-to-one relation between the number of episodes and the number of sub-expressions). However 1,197 dialogs can be compressed 33% (e.g., a dialog that involves nine episodes which can be specified with six sub-expressions), and 975 can be compressed 50%. Figure 8 presents these compression results: over 20% of the dialogs (1,692/8,192) can be compressed 50% or more. While we cannot characterize the dialog specifications comprehensively beyond q=3 because it is not possible to enumerate and simulate [21, 26, 29, 33] all of them, we can say intuitively that the results for $q > 3$ are better than q=3 because the opportunities for compression increase as the number of questions

posed in an episode increases. Therefore, both the number of sub-expressions required to specify a dialog as well as the percentage of dialogs being compressed to a high degree increase.

RELATED RESEARCH

Our work lies in the *dialog management* area of dialog-based systems. The dialog management component plays a central role in the architecture of a traditional dialog system, and is primarily concerned with controlling the flow of the dialog, while maintaining discourse history, sometimes referred to as system-action prediction, and coordinating with other (typically input/output) components of the system (e.g., automatic speech recognition, spoken language understanding, and presentation of results). In this paper, we focus on the dialog management independent of the input and output modalities (e.g., text or voice) and mechanisms that can be used in our framework.

There are two main approaches to dialog management: task-based and data-driven. Our research targets task-based dialog systems whose goal is to support the user in satisfying clearly-defined goals by completing highly-structured tasks. Therefore, we compare to and distinguish our work from other task-based approaches. While the data-driven approaches are not directly comparable to our task-based approach, they are complementary to our work. We focus rather on frameworks for (automatic) construction for their relatedness to our work [11, 23, 33].

The task-based approach involves modeling a collection of tasks to be supported by the system, using a modeling notation or language, and discerning how the user can be most effectively afforded (the desired) interaction flexibility in completing those tasks. Finite state automata (FSA), and other transition networks, context-free grammars (CFG), and events have been used as general task structures to model dialog [16]. While these models are sound, and can be used to prove mathematical proprieties, tasks often need to be over-specified to model a rich and flexible form of human-computer interaction. Moreover, since dialogs can contain arbitrarily nested sub-dialogs, FSA are less effective as general discourse structures [12]. Similarly, CFGs might be appropriate if the evolution of a dialog was something known a priori [12].

Sometimes the task-based approach is referred to as *knowledge-based* because it often relies on a dialog designer with domain-specific knowledge (e.g., travel, health care) [26] to model the dialog (and because the dialog itself helps provide the user access to a knowledge base in the targeted domain). These structures and this approach, therefore, can be time-consuming and expensive to use because the design process must be repeated when developing a similar application for a new domain, thus inhibiting domain portability. Therefore, a formidable challenge in the task-based approach to dialog management is determining the level of granularity at which to factor the system architecture to most effectively navigate the delicate balance between which dialog/task-modeling notation to use and operationalizing that model to factor domain-dependent and -independent aspects from each other to promote domain portability; this is one way of distinguishing frameworks for the construction of dialog-based systems.

One level of decomposition involves using specialized task structures for modeling complex tasks as a collection of subtasks [18, 44] and realizing the modeled dialog using a dialog engine to capture the control logic and manage the dialog flow. The specialized task structures are typically variations of hierarchical structures for modeling interactions (i.e., task modeling) [40]. "The task hierarchy constitutes a plan for the dialog" [20]. Discourse modeling, uses data and knowledge structures (e.g., scripts, plans, and goals) [41]. There has been some work on integrating discourse models with user models [24].

The dialog task specifications used are chosen and designed to capture the aspects of the dialog specific to the targeted domain and the dialog engine is domain-independent and, thus, reusable, and acts as an interpreter, in the programming languages sense, for the given dialog specification. This approach attempts to provide a clean separation of the domain-dependent and -independent aspects (e.g., control logic and dialog flow) [3] as well as separation of other relevant concerns [8]. "In principle no operation to do with domain information should take place within the dialog manager" [39]. This approach is used in the *RavenClaw* dialog management framework [5, 6]. RavenClaw uses an agenda-based approach to task modeling [39, 40]. Our framework is an instantiation of this 'separation of task model and dialog engine' approach to dialog management (see Figures 4 and 5).

To address the costly manual design and construction of task structures by domain experts, techniques for mining knowledge sources, such as dialog corpora [4, 45, 46] and websites [11, 14, 33], for automatic modeling have been developed. While there are multiple dialog management frameworks that instantiate this approach to task-based dialog systems, some emphasize automatic construction [23] and use logic-based, language approaches (i.e., reactive planning) [12].

Rather than agenda [40], rule-oriented [12], and the myriad of other task structures and task modeling approaches used for task-based dialog management, we use programming language theory. We designed a notation based on lambda calculus that serves as an authoring notation for specifying dialogs and also suggests implementation ideas. This is our main contribution and distinguishes our model from other knowledge/task-based approaches which use hierarchical task/agenda models. Using program transformations [31], including partial evaluation [22], and language concepts, to specify dialogs and to intensionally model multiple paths through a dialog without extensionally hardcoding each into the control flow of the implementation, is a fundamentally different approach to dialog modeling, management, and implementation.

Program transformations and other languages concepts have been used for similar purposes. For instance, researchers [35] have used first-class continuations [13] to maintain state in web dialogs, and program slicing [37] and source-to-source rewrite rules [38] to restructure web interactions. Using first-class continuations, researchers have developed an approach

to automatically restructure batch programs for interactive use on the web [15]. Researchers have explored the idea of using currying and continuations to postpone, save, and resume dialogs in application software [34]. The common theme of these efforts, and our research, is the appeal to concepts from programming languages to engineer a rich and expressive form of a human-computer interaction. The novel use of these language concepts provides the theoretical basis for elegant implementation solutions, without which might require developers to enumerate code in an ad hoc manner to trap and accommodate special situations.

Since our approach factors the domain-dependent (i.e., task structures) and the -independent (i.e., control logic) aspects from each other in the dialog manager, all of the peripheral/auxiliary techniques for domain-knowledge acquisition (automatic or otherwise) or other aspects for automatic dialog system construction are applicable in our approach and can be integrated into it. For instance, any of the automatic knowledge acquisition mining techniques from dialog copra or human-human conversations dovetail with our approach. Our dialog toolkit also includes a preprocessor, dialog mining component, not discussed here due to space limitations, that given (observable or other) dialog episodes can identify opportunities for mixing initiative (i.e., it mines a minimal specification of the dialog in our language-based notation).

While prior research projects have approached engineering interactive computing systems from the perspective of (functional) programming languages [15, 28, 34, 35], only few have sought to marry human-computer dialogs with concepts from programming languages [7, 32, 36]. Due to the conceptual analogs between natural languages and programming languages, viewing human-computer dialog modeling, management, and implementation from the perspective of programming language theory suggests a natural, yet under-explored, approach to dialog representation and reasoning. The concepts from programming languages are not just helpful metaphors for dialog specification, but also lend insight into operationalizing dialogs.

DISCUSSION

Dialog is essential to providing a rich form of human-computer interaction [9]. We summarize the contributions of our research as: we i) developed a language-based model for specifying and staging mixed-initiative, human-computer dialogs, ii) generalized and automated the activity of building a dialog system, and iii) evaluated its descriptive and staging capabilities by demonstrating that it can succinctly capture and stage a wide variety of dialogs, including those involving sub-dialogs. While "[c]reating an actual dialog system involves a very intensive programming effort" [17] and "complete automation in creating ... dialog applications remains an extremely difficult problem" [11], given a specification of a dialog in our dialog authoring notation, from among a variety of mixed-initiative dialogs, our system automates the implementation of the dialog. Designers of task-based dialog systems can use our dialog authoring notation and staging engine as a dialog modeling and implementation toolkit to explore, prototype, and evaluate [23] a variety of unsolicited reporting, mixed-initiative dialogs.

While the use of simulation for evaluation of dialog systems is common [21, 26, 29, 33], the application of our results will benefit from a formal usability evaluation. We intend to conduct studies with users to evaluate the interface through which users experience the human-computer dialog (i.e., Figure 1) as well as the interface for task modeling used by dialog designers to specify the dialog as part of future work. Evaluating the interface through which dialog participants experience the dialog will help us discern whether mixed-initiative dialogs resulting from our language-based model have desirable qualities (i.e., How effective and efficient are they? Does mixed-initiative dialog help the user in an information-seeking activity and how, e.g., time-to-task completion, satisfaction? For which types of dialogs or tasks is mixed-initiative interaction most effective?). We desire "computational agents carrying out our dialog theory to produce conversations with desirable qualities" [17]. To this end, we plan to conduct a study similar to [11] and, in a more broad context, using the results of [43].

Usability (i.e., the speed of use and ease of use) from the designers' perspective is also an important issue that we plan to address in a formal evaluation study as part of future work. We are exploring the idea of using a visual graphic design tool with a split-screen using drag-and-drop elements (e.g., solicitations and responses) on one side and the corresponding XML dialog specification on the other side, which are synchronized in situ, allowing the dialog designer to use either or both at her discretion for task modeling and dialog specification. Through this tool, a dialog designer can craft the solicitations and responses of a dialog, establish relationships supporting fixed or flexible dialog completion orders, and customize the features and layout of the end-user client used for processing this dialog during staging. A split-screen user interface is terser than a purely textual modality, and may be easier to use.

The advent of virtual, immersive environments in cyberlearning has attracted the attention of researchers [10] and provides a new landscape and opportunity to research models for engineering flexible human-computer dialogs [27]. We are currently studying the use of our model in a university course schedule application in an immersive, virtual environment through a verbal modality. Applications on platforms, such as smart phones, gaming consoles, airport kiosks, ATM machines, interactive, voice-response systems, and cyberlearning environments, whose success relies on flexible, mixed-initiative dialog can benefit from a model for engineering dialogs in a more systematic and simplified way. We envisage the long-term practical implications of our work involving the incorporation of stagers based on partial evaluation and rewrite rules into these platforms whose ubiquity in service-oriented domains, such as education, health care, and travel provide a fertile landscape for further exploration of our model for mixed-initiative interaction.

ACKNOWLEDGMENTS

The research was supported in part by grants from the Ohio Board of Regents, University of Dayton Research Council, and the University of Dayton College of Arts and Sciences.

We thank John Cresencia, Shuangyang Yang, and Brandon Williams at the University of Dayton for assisting in the implementation of the dialog modeling toolkit, and for helpful discussions and insight.

REFERENCES

1. J.F. Allen. 1999. Mixed-Initiative Interaction. *IEEE Intelligent Systems* 14, 5 (1999), 14–16.
2. F. Baader and T. Nipkow. 1999. *Term Rewriting and All That*. Cambridge University Press, Cambridge, UK.
3. T. Ball, C. Colby, P. Danielsen, L.J. Jagadeesan, R. Jagadeesan, K. Läufer, P. Mataga, and K. Rehor. 2000. Sisl: Several Interfaces, Single Logic. *International Journal of Speech Technology* 3, 2 (2000), 93–108.
4. F. Bechet, G. Riccardi, and D. Hakkani-Tür. 2004. Mining Spoken Dialogue Corpora for System Evaluation and Modeling. In *Proceedings of the Association for Computational Linguistics (ACL) Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Stroudsburg, PA, 134–141.
5. D. Bohus and A.I. Rudnicky. 2003. RavenClaw: Dialog management using hierarchical task decomposition and an expectation agenda. In *Proceedings of the Sixth Annual INTERSPEECH Conference*. International Speech Communication Association.
6. D. Bohus and A.I. Rudnicky. 2009. The RavenClaw Dialog Management Framework: Architecture and Systems. *Computer Speech and Language* 23, 3 (2009), 332–361.
7. R. Capra, M. Narayan, S. Perugini, N. Ramakrishnan, and M.A. Pérez-Quiñones. 2003. The Staging Transformation Approach to Mixing Initiative. In *Working Notes of the IJCAI 2003 Workshop on Mixed-Initiative Intelligent Systems*, G. Tecuci (Ed.). AAAI/MIT Press, Menlo Park, CA, 23–29.
8. J. Chu-Carroll. 2000. MIMIC: An adaptive mixed initiative spoken dialogue system for information queries.. In *Proceedings of the Sixth Conference on Applied Natural Language Processing (ANLC)*. Association for Computational Linguistics, Stroudsburg, PA, 97–104.
9. A. Dix, J. Finlay, G.D. Abowd, and R. Beale. 2010. *Human-Computer Interaction* (third ed.). Prentice Hall, Harlow, England, Chapter 16: Dialog Notations and Design.
10. A. Dubrow. Seven Cyberlearning Technologies Transforming Education. *Huffington Post*, 6 April 2015. Available: `http://www.huffingtonpost.com/aaron-dubrow/7-cyberlearning-technolog_b_6988976.html` [Last accessed: 6 May 2016].
11. J. Feng, D. Hakkani-Tür, G. Di Fabbrizio, M. Gilbert, and M. Beutnagel. 2006. Webtalk: Towards Automatically Building Spoken Dialog Systems Through Mining Websites. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE Computer Society Press, Los Alamitos, CA, 573–576.
12. R. Freedman. 2000. Using a Reactive Planner as the Basis for a Dialogue Agent. In *Proceedings of the Thirteenth International Florida Artificial Intelligence Research Society Conference*. 203–208.
13. D.P. Friedman and M. Wand. 2008. *Essentials of Programming Languages* (third ed.). MIT Press, Cambridge, MA.
14. J. Glass and S. Seneff. 2003. Flexible and Personalizable Mixed-initiative Dialogue Systems. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (ACL): Human Language Technologies (NAACL-HLT) Workshop on Research Directions in Dialogue Processing*. Association for Computational Linguistics, Stroudsburg, PA, 19–21.
15. P. Graunke, R. Findler, S. Krishnamurthi, and M. Felleisen. 2001. Automatically Restructuring Programs for the Web. In *Proceedings of the Sixteenth IEEE International Conference on Automated Software Engineering (ASE)*. 211–222.
16. M. Green. 1986. A Survey of Three Dialogue Models. *ACM Transactions on Graphics* 5, 3 (1986), 244–275.
17. C.I. Guinn. 1999. Evaluating Mixed-initiative Dialog. *IEEE Intelligent Systems* 14, 5 (1999), 21–23.
18. E. Hagen and B. Grote. 1997. Planning Efficient Mixed-initiative Dialogue. In *Proceedings of the Association for Computational Linguistics (ACL) Conference on Interactive Spoken Dialog Systems on Bringing Speech and NLP Together in Real Applications (ISDS)*. Association for Computational Linguistics, Stroudsburg, PA, 53–56.
19. S. Haller and S. McRoy (Eds.). 1997. *Proceedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*. Number SS-97-04. AAAI Press, Menlo Park, CA.
20. J. Hochberg, N. Kambhatla, and S. Roukos. 2002. A Flexible Framework for Developing Mixed-initiative Dialog Systems. In *Proceedings of the Third Association for Computational Linguistics (ACL) SIGDIAL Workshop on Discourse and Dialogue*. Association for Computational Linguistics, Stroudsburg, PA, 60–63.
21. R.B. Inouye. 2004. Minimizing the Length of Non-mixed Initiative Dialogs. In *Proceedings of the Association for Computational Linguistics (ACL) Workshop on Student Research*. Association for Computational Linguistics, Stroudsburg, PA.
22. N.D. Jones. 1996. An Introduction to Partial Evaluation. *Comput. Surveys* 28, 3 (1996), 480–503.
23. P. Jordan, M. Ringenberg, and B. Hall. 2006. Rapidly Developing Dialogue Systems that Support Learning Studies. In *Proceedings of Intelligent Tutoring Systems (ITS) Workshop on Teaching with Robots, Agents, and NLP*. 1–8.

24. A. Kobsa. 1988. User Models and Dialog Models: United They Stand. *Computational Linguistics* 14, 3 (1988), 91–94.

25. D.L. Kreher and D.R. Stinson. 1999. *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press, Boca Raton, FL.

26. C. Lee, S. Jung, K. Kim, D. Lee, and G.G. Lee. 2010. Recent approaches to dialog management for spoken dialog systems. *Journal of Computing Science and Engineering* 4, 1 (2010), 1–22.

27. A. Leuski and D. Traum. 2011. NPCEditor: Creating Virtual Human Dialogue Using Information Retrieval Techniques. *AI Magazine* 32, 2 (2011), 42–56.

28. S.N. Malkov. 2010. Customizing a Functional Programming Language for Web Development. *Computer Languages, Systems and Structures* 36, 4 (2010), 345–351.

29. T. Misu, K. Georgila, A. Leuski, and D. Traum. 2012. Reinforcement Learning of Question-answering Dialogue Policies for Virtual Museum Guides. In *Proceedings of the Thirteenth Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Association for Computational Linguistics, Stroudsburg, PA, 84–93.

30. M. Narayan, C. Williams, S. Perugini, and N. Ramakrishnan. 2004. Staging Transformations for Multimodal Web Interaction Management. In *Proceedings of the Thirteenth International ACM World Wide Web Conference (WWW)*, M. Najork and C.E. Wills (Eds.). ACM Press, New York, NY, 212–223.

31. H. Partsch and R. Steinbrüggen. 1983. Program Transformation Systems. *Comput. Surveys* 15, 3 (1983), 199–236.

32. M.A. Pérez-Quiñones. 1996. *Conversational Collaboration in User-initiated Interruption and Cancellation Requests*. Ph.D. dissertation. The George Washington University.

33. J. Polifroni, G. Chung, and S. Seneff. 2003. Towards the Automatic Generation of Mixed-Initiative Dialogue Systems from Web Content. In *Proceedings of the Eighth European Conference on Speech Communication and Technology (EUROSPEECH)*. International Speech Communication Association, 193–196.

34. D. Quan, D. Huynh, D.R. Karger, and R. Miller. 2003. User Interface Continuations. In *Proceedings of the Sixteenth Annual ACM Symposium on User Interface Software and Technology (UIST)*. ACM Press, New York, NY, 145–148.

35. C. Queinnec. 2000. The Influence of Browsers on Evaluators or, Continuations to Program Web Servers. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP)*. ACM Press, New York, NY, 23–33. Also appears in *ACM SIGPLAN Notices*, 35(9), 2000.

36. N. Ramakrishnan, R. Capra, and M.A. Pérez-Quiñones. 2002. Mixed-Initiative Interaction = Mixed Computation. In *Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM)*, P. Thiemann (Ed.). ACM Press, New York, NY, 119–130. Also appears in *ACM SIGPLAN Notices*, 37(3), 2002.

37. F. Ricca and P. Tonella. 2001. Web Application Slicing. In *Proceedings of the International Conference on Software Maintenance (ICSM)*. IEEE Computer Society Press, Los Alamitos, CA, 148–157.

38. F. Ricca, P. Tonella, and I.D. Baxter. 2001. Restructuring Web Applications via Transformation Rules. In *Proceedings of the First International Workshop on Source Code Analysis and Manipulation (SCAM)*. IEEE Computer Society Press, Los Alamitos, CA, 150–160.

39. A. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Stern, K. Lenzo, W. Xu, and A. Oh. 1999. Creating natural dialogs in the Carnegie Mellon communicator system. In *Proceedings of the Sixth European Conference on Speech Communication and Technology (EUROSPEECH)*. International Speech Communication Association.

40. A. Rudnicky and W. Xu. 1999. An agenda-based dialog management architecture for spoken language systems. *IEEE Automatic Speech Recognition and Understanding Workshop* 13, 4 (1999).

41. R.C. Schank and R.P. Abelson. 1977. *Scripts, Plans, Goals and Understanding: an Inquiry into Human Knowledge Structures*. L. Erlbaum, Hillsdale, NJ.

42. D. Stallard. 2001. Dialogue management in the Talk'n'Travel system. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE Computer Society Press, Los Alamitos, CA, 235–239.

43. M. Walker, L. Hirschman, and J. Aberdeen. 2000. Evaluation for DARPA communicator spoken dialogue systems. In *Proceedings Second International Conference on Language Resources and Evaluation*. European Language Resources Association.

44. M. Walker and S. Whittaker. 1990. Mixed-initiative in dialogue: An investigation into discourse segmentation. In *Proceedings of the Twenty-eighth Annual Meeting on Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, Stroudsburg, PA, 70–78.

45. W. Wong, L. Cavedon, J. Thangarajah, and L. Padgham. 2012. Mixed-initiative Conversational System Using Question-answer Pairs Mined from the Web. In *Proceedings of the Twenty-first ACM International Conference on Information and Knowledge Management (CIKM)*. ACM Press, New York, NY, 2707–2709.

46. X. Yao, E. Tosch, G. Chen, E. Nouri, R. Artstein, A. Leuski, K. Sagae, and D. Traum. 2012. Creating conversational characters using question generation tools. *Dialogue and Discourse* 3, 2 (2012), 125–146.