A Tool for Staging Mixed-initiative Dialogs[*]

Joshua W. Buck and **Saverio Perugini**
Department of Computer Science
University of Dayton
300 College Park
Dayton, Ohio 45469–2160 USA
Tel: +001 (937) 229–4079, Fax: +001 (937) 229–2193
E-mail: {jbuck1,saverio}@udayton.edu

Abstract

We discuss and demonstrate a tool for prototyping dialog-based systems that, given a high-level specification of a human-computer dialog, stages the dialog for interactive use. The tool enables a dialog designer to evaluate a variety of dialogs without having to program each individual dialog, and serves as a proof-of-concept for our approach to mixed-initiative dialog modeling and implementation from a programming language-based perspective.

Introduction

Mixed-initiative interaction is a flexible interaction strategy where the user and system engage as equal participants in an activity and take turns exchanging initiative, thereby sharing initiative. The strategy has been proposed and studied as a link to bridge artificial intelligence (AI) and human-computer interaction (Hearst, 1999). Mixed-initiative interaction has been studied and applied to keep the human in the learning feedback loop of an intelligent search, particularly, in planning, scheduling, and constraint satisfaction (Ferguson and Allen, 1998; Fleming and Cohen, 1999, 2001; Frank et al., 2001; Pu and Lalanne, 2002; Smith et al., 2005; Wolfman et al., 2001). This paper specifically addresses mixed-initiative dialog (Glass and Seneff, 2003; Lee et al., 2010; Walker and Whittaker, 1990). While "[c]reating an actual dialog system involves a very intensive programming effort" (Guinn, 1999), our tool simplifies that effort so that dialog designers can evaluate a variety of human-computer dialogs, including mixed-initiative dialogs (see figure 1).

A Language-based Approach to Mixed-initiative Dialog Modeling

We approach this problem by casting it in a programming language-based context, using notions from *lambda calculus* (Friedman and Wand, 2008). While not the focus of this paper, we introduce the fundamentals of the approach here to help orient the reader to the overall idea. As the user progresses through a dialog, we think of the steps that she takes as the evaluation of a function. Changing the evaluation method of the function (or transforming the function) then corresponds to different interaction policies (Rudnicky et al., 1999) for the dialog (i.e., ways of mixing initiative). The overall idea is that different function evaluation strategies correspond to different interaction policies for the dialog (i.e., system initiated vs. mixed-initiative) or ways of mixing initiative. For instance, consider a course scheduling dialog where the user must make selections for department, time, and number of credits, among others. We can model this dialog with a function $schedule$ with type signature $(department \times time \times credits) \rightarrow course$, simplified for purposes of succinct presentation. Evaluating this function using the *complete, eager* (Friedman and Wand, 2008) or *applicative-order evaluation* (Abelson and Sussman, 1996) function application evaluation method common in most programming languages (e.g., C, Java, Scheme) requires the user to supply arguments for the complete list of parameters to the function, in one stroke. In interaction terms, this evaluation method corresponds to a user completing, for example, an online, web-based form with multiple input fields (as, e.g., generated by Ajax) and clicking 'submit' once he has provided responses for all of the required fields. Optional fields correspond to (optional) parameters with default values (e.g., as in Python; Sebesta 2015). In general, this evaluation strategy corresponds to providing (possibly) multiple responses in a single utterance. A single utterance with only one response (if the function is a unary function), which is common in confirmation dialog boxes in application software, can also be modeled with this method.

Currying this function corresponds to spreading the interaction required to complete the dialog across a fixed series of multiple, progressive user-system dialog turns (i.e., system solicitation followed by user response, and so on). Currying the function $schedule$ above transforms it into one with type signature

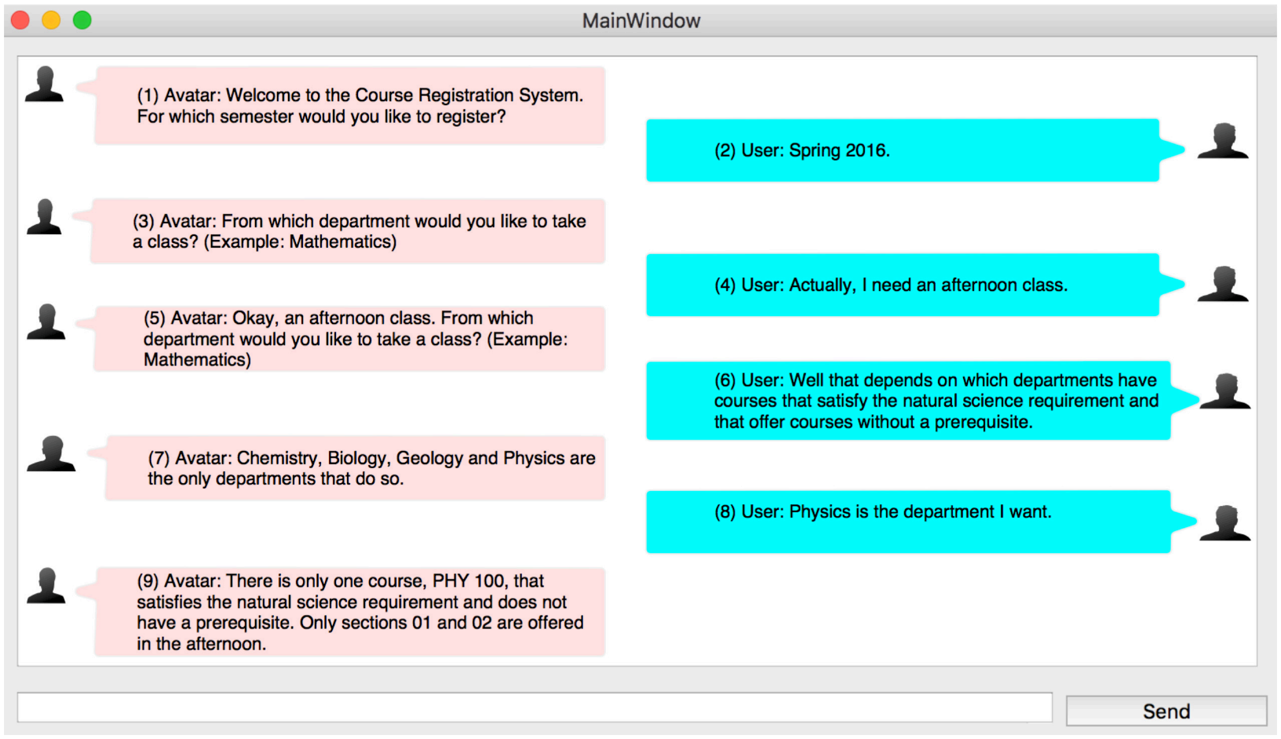
$$department \rightarrow (time \rightarrow (credits \rightarrow course))^{1},$$

such that

$$schedule_{uncurried}(\texttt{biology}, \texttt{morning}, 3) =$$

[*]We thank the four anonymous reviewers of this paper for their helpful comments and suggestions that helped to improve it.

[1]Parentheses included for purposes of clarity.



(1) Avatar: Welcome to the Course Registration System. For which semester would you like to register?

(2) User: Spring 2016.

(3) Avatar: From which department would you like to take a class? (Example: Mathematics)

(4) User: Actually, I need an afternoon class.

(5) Avatar: Okay, an afternoon class. From which department would you like to take a class? (Example: Mathematics)

(6) User: Well that depends on which departments have courses that satisfy the natural science requirement and that offer courses without a prerequisite.

(7) Avatar: Chemistry, Biology, Geology and Physics are the only departments that do so.

(8) User: Physics is the department I want.

(9) Avatar: There is only one course, PHY 100, that satisfies the natural science requirement and does not have a prerequisite. Only sections 01 and 02 are offered in the afternoon.

Figure 1: A mixed-initiative dialog.

$$((schedule_{curried}(\texttt{biology}))(\texttt{morning}))(3).$$

In general, *currying* (Scott, 2009) transforms a function $f_{uncurried}$ with type signature

$$f_{uncurried}: \ (p_1 \times p_2 \times \cdots \times p_n) \rightarrow r$$

into a function $f_{curried}$ with type signature

$$f_{curried}: p_1 \rightarrow (p_2 \rightarrow (\cdots \rightarrow (p_n \rightarrow r)\cdots)),$$

such that

$$f_{uncurried}(a_1, a_2, \cdots, a_n) = \\ (\cdots((f_{curried}(a_1))(a_2))\cdots)(a_n).$$

(Uncurrying a curried function inverses the curry operation.) The function resulting from currying a function modeling a dialog corresponds to the interaction common in wizards for installing application software (Hamidi, Andritsos, and Liaskos, 2014) or in ATMs or airport and train kiosks, where the user is required to complete a *fixed*, system-initiated dialog (Allen, 1999) that involves supplying only one response per utterance. (Fixed dialogs have been referred to as *strongly-typed interactions* (Lee et al., 2010), underscoring a connection to the language concept of *strong typing* (Sebesta, 2015).)

Partially applying the function $schedule_{uncurried}$ corresponds to a fixed dialog where the user is afforded the flexibility to provide more than one response per utterance. This mode of interaction is common when providing a telephone, credit card, or PIN number through a voice modality. *Partial function application* states that for any function $f(p_1, p_2, \ldots, p_n)$,

$$f(a_1, a_2, \ldots, a_m) = g(p_{m+1}, p_{m+2}, \ldots, p_n),$$

where $m \leqslant n$, such that

$$g(a_{m+1}, a_{m+2}, \ldots, a_n) = \\ f(a_1, a_2, \ldots, a_m, a_{m+1}, a_{m+2}, \ldots, a_n).$$

(Note that currying $f_{uncurried}$ and progressively applying the resulting $f_{curried}$ function has the same effect as partially applying $f_{uncurried}$.)

Lastly, applying the program transformation partial evaluation (Jones, 1996) to $schedule_{uncurried}$ corresponds to permitting the user to communicate answers to the set of questions in the dialog in utterances corresponding to all possible set partitions of the set of questions, and using all possible permutations of those partitions. *Partial evaluation* generalizes the idea of partial function application from any prefix of the parameter list to any subset of the parameter list and formally states that for any function $f(p_1, p_2, \ldots, p_n)$,

$$f(a, b, \ldots, r) = g(p_1, p_2, \ldots, p_n - a, b, \ldots, r),$$

where $\{a, b, \ldots, r\} \subseteq \{p_1, p_2, \ldots, p_n\}$, such that

$$g(s, t, \ldots, z) = f(a, b, \ldots, z).$$

Table 1 summaries these associations from programming language concepts to human-computer dialogs.

We developed a notation employing these function evaluation strategies for specifying mixed-initiative dialogs. A specification involving multiple functions using different evaluation strategies corresponds to a dialog involving multiple sub-dialogs. We refer the reader interested in additional details to (Perugini, 2015).

System Design and Implementation

While the implementation details are beyond the scope of this paper, we make some remarks. Figure 2 provides a conceptual overview of the design and execution of our dialog system construction tool. The left side of the figure depicts dialog specification and the right side illustrates dialog staging.

At present, the dialog designer must specify the dialog in a textual format using a high-level description language that is a less formal version of our language-based, dialog authoring notation that is then compiled into an XML document (see right side of figure 4). While the visual dialog builder interface through which the designer specifies the dialog to be implemented (see figure 4) is still in development, we use it in scenario 0 below to obviate the need to cover the formal syntax and details of our dialog authoring language. A specification in either format is then compiled into an XML document (see right side of figure 4). The XML document is then parsed into a variety of data structures (see figure 3) for use by the dialog staging engine, which processes user responses and stages the turns of the dialog (i.e., structures the interaction). We use the term *staging* to refer to the process by which the progressive turns of the dialog are structured or layered. For instance, the system starts by soliciting a response from the user to a particular question and then awaits a reply. The user in turn provides a response and the system makes another solicitation. The user then might respond to an unsolicited, yet forthcoming, question that the system accepts, processes, and uses to determine the next prompt to present. Dialog management, conducted through the staging engine here, performs system-action prediction (Lee et al., 2010)—deciding what to prompt for and/or accept next based on the discourse history and the current utterance. In this sense, staging operationalizes the dialog modeled by the specification and can be thought of as an interpreter (in the programming language sense of the word) for the specification.

Though not shown explicitly in figure 2, the dialog designer can also interact directly with the generation engine and the staging engine, in addition to a visual dialog builder, which is why the designer is depicted at the center of the dialog toolkit, especially since the staging engine supports server commands for interacting with user clients. Figure 3 illustrates the data structures used to represent the human-computer dialog demonstrated in the following designer and user scenarios.

Designer and User Scenarios

We use the following six scenarios from student course scheduling at a university to demonstrate our tool.

Scenario 0: Dialog Design

Using a drag-and-drop, split-screen, visual dialog builder (see figure 4), a dialog designer specifies a dialog by opening a new blank dialog project. In a new blank dialog template, a designer may specify aspects of the dialog such as a welcome message and generic (confirmation) responses to user utterances such as ‘Okay, you answered . . . .’ The designer can then create the first solicitation. Creating a new solicitation launches a set of boxes in the visual panel (left side of figure 4), by default, prompting the designer for a solicitation (typically a question) and three valid responses to it that the dialog (client) user may provide in response to that solicitation. The first solicitation in a course registration dialog might be ‘For which semester would you like to register?’ The three valid responses specified by the designer to this solicitation might be ‘Fall,’ ‘Spring,’ and ‘Summer.’ Using the tool, the designer can add and remove solicitations and responses at any time from the dialog specification she is authoring.

Once the dialog designer has specified more than one solicitation, she can establish a total or partial ordering between responses and solicitations by drawing arrows using the visual tool. For instance, the dialog designer might draw a relationship arrow from the ‘Mathematics’ response of the third solicitation to the fifth solicitation ‘For which class would you like to register?’ with the mathematics courses as possible responses. This scenario illustrates how a particular response can lead to a different path through a dialog through the inclusion of relationship arrows to associate certain responses to forthcoming solicitations. If no relationships are specified, the default interaction policy is implicit in the visual, top-to-bottom order of the solicitations in the left pane (i.e., a fixed dialog). In other words, if a response is listed without a defined relationship to another solicitation, the next solicitation by default is the solicitation immediately below the given response. Relationships that collectively induce a loop in the dialog specification—such as a relationship between a response and itself—without at least one path out of the loop are not permissible.

As mentioned above, the dialog designer can also create the dialog specification using our dialog authoring notation in a text format. Note that the right pane of the interface shown in figure 4 is a textual XML representation of the dialog specification. The two panes of the split-screen shown in figure 4 are synchronized in real time to be accurate reflections of each other, and the dialog designer always has the option of using either or both, at her discretion. The dialog designer can also dynamically change the interaction policy of a dialog or any sub-dialogs (i.e., evaluation order of the solicitations), as well as perform other operations, by interacting directly with the staging engine though a console, as mentioned above.

Scenario 1: Fixed Dialog with

Table 1: Metaphors from programming language evaluation strategies to a variety of human-computer dialogs.

Evaluation strategy	Responses per utterance	Ordering of utterances	Example(s)	Scenario
complete, eager evaluation	single	only one utterance	confirmation dialog boxes common in application software	–
complete, eager evaluation	multiple	only one utterance	web forms with multiple fields	2
currying	single	totally-ordered	wizards for installing application software or ATMs or airport and train kiosks	1
partial function application	multiple	totally-ordered	communicating a telephone, credit card, or PIN through voice	4
partial evaluation	single	partially-ordered	online test/quiz or survey	5
partial evaluation	multiple	partially-ordered	making a flight, rental car, or hotel reservation with a human agent	3

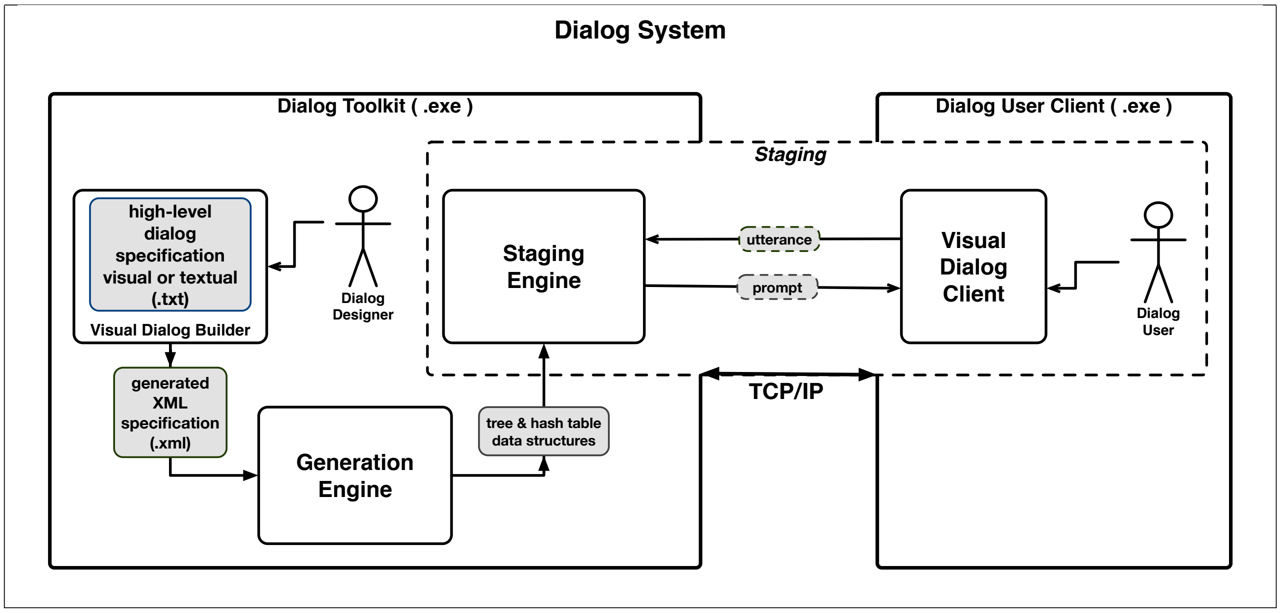


Figure 2: Conceptual design and execution of our dialog system.

Single Response per Utterance

The system solicits for a semester for which to register and the user responds with 'spring 2016.' In a fixed dialog, the user must respond to this question. The next solicitation is for a department in which to register for a course. The user replies with 'mathematics department' and the system solicits for the course level: undergraduate or graduate. The user responds with 'undergraduate' and the system prompts for a course. The user replies '168' and the system responds with 'Okay, mathematics 168. The course is offered from 3:35pm to 4:50pm on Tuesdays and Thursdays. Confirm registration?' If the user responds with 'yes,' the dialog is complete. This scenario illustrates one complete path through the dialog, where the user responds to all of the system solicitations in the order in which they are presented. This scenario corresponds to the third row of table 1 (i.e., currying). Figure 1 illustrates the interface through which the user, in this scenario and all following scenarios, interacts with the system while participating in the dialog.

Scenario 2: Fixed Dialog with All Responses in a Single Utterance

We restart the dialog system. This time, when prompted for a semester, the user responds with 'spring 2016 mathematics 168.' Note that the system accepts utterances containing responses embedded among other non-essential words. For instance, an equivalent response is 'I would like to register for mathematics 168 in spring 2016, please.' The system processes all of the responses in this utterance and responds with 'Okay, mathematics 168 in spring 2016. The course is offered from 3:35pm to 4:50pm on Tuesdays and Thursdays. Confirm registration?' If the user responds with 'yes,' the dialog is complete. This scenario illustrates a complete path through the dialog where the user provides responses to all of the solicitations in a single utterance. This scenario corresponds to the second row of table 1 (i.e., complete, eager evaluation).

Scenario 3: Mixed-initiative Dialog,

Unsolicited Responses

Assume we undo the user utterance in the scenario 2 above. This causes the system to solicit for a semester again. This time, the user does not provide a semester but rather a department in which to register for a course: 'mathematics department.' The system accepts and processes this unsolicited response instead and again solicits for a semester. This demonstrates the ability to provide responses for solicitations that the system has not made yet but will at some point in the dialog—a degree of mixed-initiative interaction known as *unsolicited reporting* (Allen, 1999). The user completes the dialog in one more turn with the utterance: 'mathematics 168 spring 2016.' This scenario corresponds to the fifth and sixth rows of table 1 (i.e., partial evaluation).

Scenario 4: Fixed Dialog with Multiple Responses per Utterance

Assume the dialog designer desires to permit the user to respond to multiple solicitations in a single utterance, but also wants to enforce an ordering on the set of solicitations of the dialog. Here the user could respond first with 'spring 2016 mathematics department' and then with '168.' Responses given in a single utterance are unordered as they are processed at the same time. This scenario illustrates a dialog supporting multiple responses per utterance, but with a fixed order on its solicitations, corresponding to the fourth row of table 1 (i.e., partial function application).

Scenario 5: Unsolicited, Single-response, Utterances

In contrast, the dialog designer might desire to permit the user to respond to solicitations in any order, but with only one response per utterance. In such a dialog, the user could reply with 'mathematics 168,' but could not provide multiple responses in any single utterance; responses such as 'spring 2016 mathematics department' are not permitted. This scenario corresponds to the fifth row of table 1 (i.e., partial evaluation).

Practical Considerations

Our dialog engine automatically infers dependencies between possible dialog responses in an effort to reduce dialog length. For instance, if the user responds with 'mathematics 168,' the system automatically infers that the user wants to register for an undergraduate course in the mathematics department and, thus, it is unnecessary to prompt for course level and department. The user then only needs to select the semester in which to register for the course and confirm the registration.

Our dialog engine uses error recovery strategies (Lee et al., 2010), including (explicit) confirmation, regarding mismatching and/or conflicting user responses. For instance, assume the user desires to register for a physics course. If the user responds first with 'mathematics department' and then with 'physics 168' in the next utterance, the dialog system responds with 'physics 168 is offered in the physics department, not the mathematics department. Would you like to change the department from mathematics to physics? If not, please choose a mathematics course.' This feature obviates the need to undo responses up to the response involving invalid or inconsistent information (e.g., department) and dynamically adjusts the response (e.g., to 'physics department'), if confirmed by the user. Also note that if the user makes an utterance that does not contain any valid responses for the dialog, such as 'I would like to book a flight from New York to Chicago please,' the system responds with 'I am sorry. I did not understand your response. Please reply to the prompt.' We designed our dialog engine to be thread-safe and the data structures that it manipulates to be stateless to allow multiple users to participate in individual instances of the same dialog concurrently.

Related Research

One approach to dialog modeling and implementation is from an artificial intelligence (AI) perspective (Allen et al., 2001), and involves selectively enumerating only those flows of control within the implementation which have been gleaned to be essential through learning algorithms. For instance, researchers have used constraint satisfaction to determine an ordering of dialog prompts (Donaldson and Cohen, 1997) and constraint propagation to implement subdialog invocation (Frank et al., 2001). There are a variety of other learning-based approaches (DeVault, Leuski, and Sagae, 2011; Misu et al., 2012). In contrast, using language-based concepts, including partial evaluation (Jones, 1996), 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. It achieves flexible dialog in a cleaner, less conditionally complex, design.

While there is an array of literature on developing (mixed-initiative) dialog systems and dialog management and construction frameworks (Hochberg, Kambhatla, and Roukos, 2002), we mention a few research projects due to their implementation-oriented nature: the DARPA *Communicator*-funded CMU *Sphinx*[2] project (Rudnicky et al., 1999), *RavenClaw* (Bohus and Rudnicky, 2003, 2009; Rudnicky and Xu, 1999) the *TuTalk*[3] project (Jordan, Ringenberg, and Hall, 2006), the *Atlas* dialog management system (Freedman, 2000), and the *Radiobot-CFF* (Leuski and Traum, 2011) and *NPCEditor* (Leuski and Traum, 2011) systems.

Conclusion

Human-computer dialog implementation is a challenging problem. Published literature has highlighted this problem, and discussed how much of a laborious, time-consuming process it can be (Glass and Seneff, 2003; Guinn, 1999; Hochberg, Kambhatla, and Roukos, 2002). Casting this problem on a programming languages landscape has suggested the use of a variety of function evaluation strategies, corresponding to different dialog interaction flow policies (i.e., staging methods) and ways of mixing initiative. It

[2]`http://cmusphinx.sourceforge.net/`

[3]An acronym for *Tutorial Talk.*

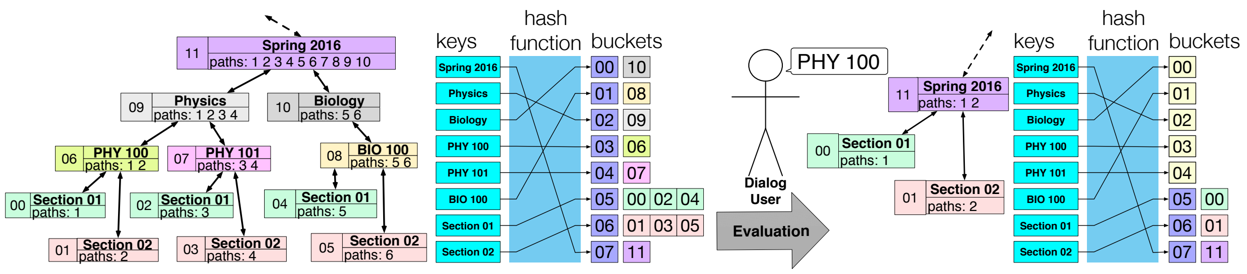


Figure 3: Data structures used to represent a human-computer dialog in our implementation.

has also supported our construction of the tool for automatic dialog implementation addressed here. Our tool can aid human-computer dialog designers in the following ways: i) no programming is required by the designer; only dialog specification is required, and the dialog system is automatically implemented and ii) the resulting dialog system can be reconfigured (e.g., different interaction policies can be applied/realized) (Glass and Seneff, 2003; Jordan, Ringenberg, and Hall, 2006; Polifroni, Chung, and Seneff, 2003) through minor modifications to the dialog specification. These benefits might help a dialog designer prototype and evaluate (Jordan, Ringenberg, and Hall, 2006), through studies with users (Feng et al., 2006), a variety of human-computer dialogs. For instance, evaluating several competing dialog designs with users and monitoring the paths taken by users to dialog completion and mining the results for the most-frequently used can help suggest refinements to the dialog specification (Spiliopoulou, 2000). Evaluating the least-frequently used paths might help the designer identify and isolate aspects of the dialog specification that could benefit from further attention.

This work can be extended in multiple directions. Lifting the problem of mixed-initiative dialog modeling, management, and implementation to a language-based landscape opens up opportunities for enriching not only the supportable dialogs but also the flexibility in interaction afforded in each as we explore additional language concepts and study their analogs in mixed-initiative interaction. Specifically, we plan to explore the use of *computational reflection* (Maes, 1987), and particularly *introspection* (i.e., read-only reflection), to support the user in *meta*-dialog inquiries, a higher-order degree of mixed-initiative interaction where the user may ask the system questions about the status of the dialog versus the typical reverse—only the system asking questions (Kronenberg and Regel-Brietzman, 2001). For instance, a user might inquire: 'How many more questions are you going to require me to answer?' We also intend to explore *meta*-dialog operations, such as 'Let us save the current status of this dialog and start over.' The capture, storage, invocation of *first-class continuations* (e.g., as provided by the `call/cc` facility in Scheme) (Friedman and Wand, 2008) can support *meta*-dialog operations such as the capture and reuse of frequently accessed dialog branches as macros as discussed by Quan et al. (2003), though in a slightly different context.

Coroutines and threads, and the actor model of concurrency with its message-passing motif, especially as used in the functional languages Elixir (Thomas, 2014) and Erlang (Armstrong, 2013), may help us specify and stage complex, problem-solving dialogs, where the system or user must engage each other in multiple (sub-)dialogs in parallel to solve the task at hand.

Our long-term goal is to develop a catalog of concepts of programming languages that are helpful metaphors for specifying and staging mixed-initiative dialogs, and formalizing their synergistic effects on dialog management and system design, and empirically studying the desirability of the interactions they support and enable.

References

Abelson, H., and Sussman, G. 1996. *Structure and Interpretation of Computer Programs*. Cambridge, MA: The MIT Press, second edition.

Allen, J.; Byron, D.; Dzikovska, M.; Ferguson, G.; Galescu, L.; and Stent, A. 2001. Towards conversational human-computer interaction. *AI Magazine* 22(4):27–37.

Allen, J. 1999. Mixed-initiative interaction. *IEEE Intelligent Systems* 14(5):14–16.

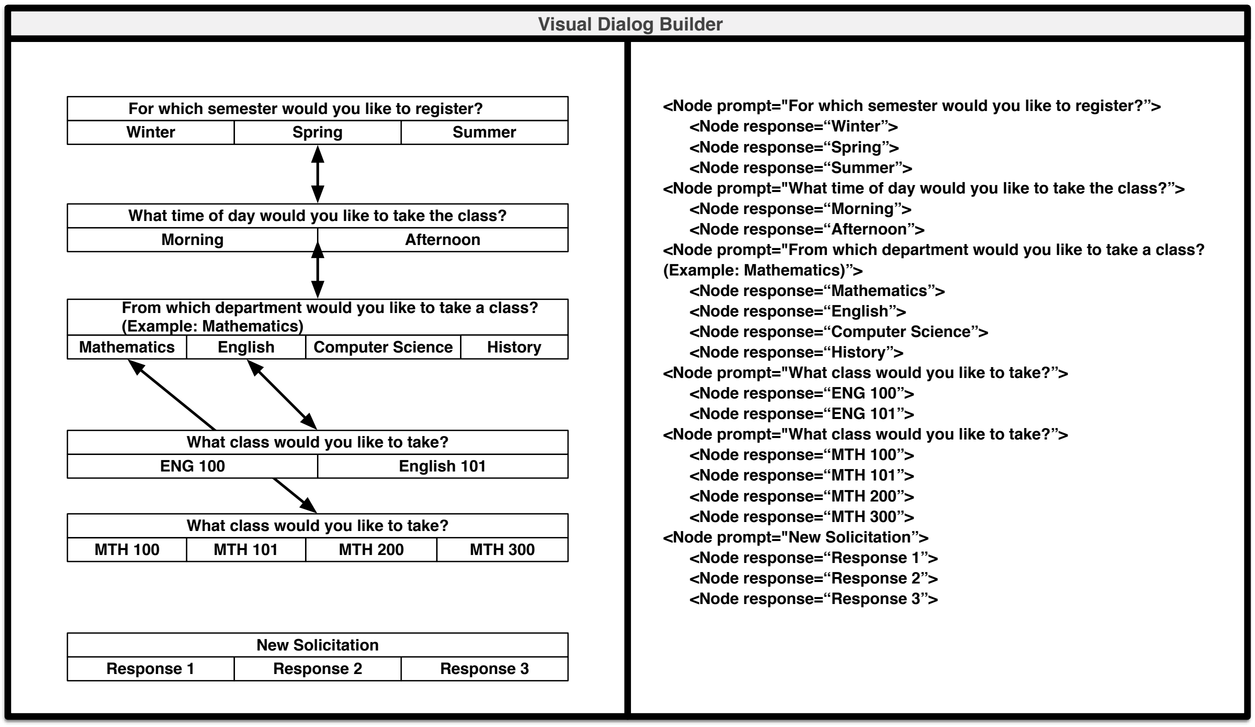
Armstrong, J. 2013. *Programming Erlang: Software for a Concurrent World*. Dallas, TX: Pragmatic Bookshelf, second edition.

Bohus, D., and Rudnicky, A. 2003. RavenClaw: Dialog management using hierarchical task decomposition and an expectation agenda. In *Proceedings of the Sixth Annual INTERSPEECH Conference*. International Speech Communication Association.

Bohus, D., and Rudnicky, A. 2009. The ravenclaw dialog management framework: Architecture and systems. *Computer Speech and Language* 23(3):332–361.

DeVault, D.; Leuski, A.; and Sagae, K. 2011. Toward learning and evaluation of dialogue policies with text examples. In *Proceedings of the Twelfth Association for Computational Linguistics (ACL) SIGDIAL Workshop on Discourse and Dialogue*, 39–48. Portland, OR: Association for Computational Linguistics.

Donaldson, T., and Cohen, R. 1997. A constraint satisfaction framework for managing mixed-initiative discourse.


Visual Dialog Builder
For which semester would you like to register?
Winter
Spring
Summer
What time of day would you like to take the class?
Morning
Afternoon
From which department would you like to take a class? (Example: Mathematics)
Mathematics
English
Computer Science
History
What class would you like to take?
ENG 100
English 101
What class would you like to take?
MTH 100
MTH 101
MTH 200
MTH 300
New Solicitation
Response 1
Response 2
Response 3

```
<Node prompt="For which semester would you like to register?">
    <Node response="Winter">
    <Node response="Spring">
    <Node response="Summer">
<Node prompt="What time of day would you like to take the class?">
    <Node response="Morning">
    <Node response="Afternoon">
<Node prompt="From which department would you like to take a class?
(Example: Mathematics)">
    <Node response="Mathematics">
    <Node response="English">
    <Node response="Computer Science">
    <Node response="History">
<Node prompt="What class would you like to take?">
    <Node response="ENG 100">
    <Node response="ENG 101">
<Node prompt="What class would you like to take?">
    <Node response="MTH 100">
    <Node response="MTH 101">
    <Node response="MTH 200">
    <Node response="MTH 300">
<Node prompt="New Solicitation">
    <Node response="Response 1">
    <Node response="Response 2">
    <Node response="Response 3">
```

Figure 4: Sketch of a split-screen, visual/textual user interface for dialog specification. An incomplete/partial dialog is shown here (i.e., what is shown here is not the final representation of the dialog).

In *Proceedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interactions*, number SS-97-04, 37–43. Menlo Park, CA: AAAI Press.

Feng, J.; Hakkani-Tür, D.; Fabbrizio, G. D.; Gilbert, M.; and Beutnagel, M. 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)*, 573–576. Los Alamitos, CA: IEEE Computer Society Press.

Ferguson, G., and Allen, J. 1998. Trips: An integrated intelligent problem-solving assistant. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*. Menlo Park, CA: AAAI Press.

Fleming, M., and Cohen, R. 1999. Towards a methodology for designing and evaluating mixed-initiative AI systems. In *Proceedings of the AAAI Workshop on Mixed-initiative Intelligence*, 130–134. Menlo Park, CA: AAAI Press.

Fleming, M., and Cohen, R. 2001. A user modeling approach to determining system initiative in mixed-initiative ai systems. In Bauer, M.; Gmytrasiewicz, P.; and Vassileva, J., eds., *Proceedings of the Eighth International Conference on User Modeling (UM)*, 54–63. Sonthofen, Germany: Springer.

Frank, M.; Muslea, M.; Oh, J.; Minton, S.; and Knoblock, C. 2001. An intelligent user interface for mixed-initiative multi-source travel planning. In *Proceedings of the Sixth ACM International Conference on Intelligent User Interfaces (IUI)*, 85–86. New York, NY: ACM Press.

Freedman, R. 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.

Friedman, D., and Wand, M. 2008. *Essentials of Programming Languages*. Cambridge, MA: MIT Press, third edition.

Glass, J., and Seneff, S. 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*, 19–21. Stroudsburg, PA: Association for Computational Linguistics.

Guinn, C. 1999. Evaluating mixed-initiative dialog. *IEEE Intelligent Systems* 14(5):21–23.

Hamidi, S.; Andritsos, P.; and Liaskos, S. 2014. Constructing adaptive configuration dialogs using crowd data. In *Proceedings of the Twenty-ninth ACM/IEEE International Conference on Automated Software Engineering (ASE)*, 485–490. New York, NY: ACM Press.

Hearst, M. 1999. Mixed-initiative interaction. *IEEE Intelligent Systems* 14(5):14–16.

Hochberg, J.; Kambhatla, N.; and Roukos, S. 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*, 60–63. Stroudsburg, PA: Association for Computational Linguistics.

Jones, N. 1996. An Introduction to Partial Evaluation. *ACM Computing Surveys* 28(3):480–503.

Jordan, P.; Ringenberg, M.; and Hall, B. 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.

Kronenberg, S., and Regel-Brietzman, P. 2001. Bridging the gap between mixed-initiative dialogs and reusable subdialogs. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 276–279. Los Alamitos, CA: IEEE Computer Society Press.

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

Leuski, A., and Traum, D. 2011. NPCEditor: Creating virtual human dialogue using information retrieval techniques. *AI Magazine* 32(2):42–56.

Maes, P. 1987. Concepts and experiments in computational reflection. In *Proceedings of the International ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, 147–155. New York, NY: ACM Press.

Misu, T.; Georgila, K.; Leuski, A.; and Traum, D. 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*, 84–93. Stroudsburg, PA: Association for Computational Linguistics.

Perugini, S. 2015. Staging mixed-initiative dialogs by program generation and transformation. Technical Report arXiv:1108.0476v5 [cs.PL], Computing Research Repository (CoRR). Available from `http://lanl.arXiv.org/abs/1108.0476v5`.

Polifroni, J.; Chung, G.; and Seneff, S. 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)*, 193–196. International Speech Communication Association.

Pu, P., and Lalanne, D. 2002. Design visual thinking tools for mixed-initiative systems. In *Proceedings of the Seventh International Conference on Intelligent User Interfaces (IUI)*, 119–126. New York, NY: ACM Press.

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

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

Rudnicky, A.; Thayer, E.; Constantinides, P.; Tchou, C.; Stern, R.; Lenzo, K.; Xu, W.; and Oh, A. 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.

Scott, M. 2009. *Programming Language Pragmatics*. Amsterdam: Morgan Kaufmann, third edition.

Sebesta, R. 2015. *Concepts of Programming Languages*. Boston, MA: Addison Wesley, eleventh edition.

Smith, S.; Cortellessa, G.; Hildum, D.; and Ohler, C. 2005. Using a scheduling domain ontology to compute user-oriented explanations. In Castillo, L.; Barrajo, D.; Salido, M.; and Oddi, A., eds., *Planning, Scheduling, and Constraint Satisfaction: From Theory to Practice*. IOS Press.

Spiliopoulou, M. 2000. Web usage mining for web site evaluation. *Communications of the ACM* 43(8):127–134.

Thomas, D. 2014. *Programming Elixir: Functional, Concurrent, Pragmatic, Fun*. Dallas, TX: Pragmatic Bookshelf.

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

Wolfman, S.; Lau, T.; Domingos, P.; and Weld, D. 2001. Mixed initiative interfaces for learning tasks: Smartedit talks back. In *Proceedings of the Sixth ACM International Conference on Intelligent User Interfaces (IUI)*, 167–174. New York, NY: ACM Press.