Available online at www.sciencedirect.com

ScienceDirect

Speech Communication 50 (2008) 716–729

SPEECH COMMUNICATION

www.elsevier.com/locate/specom

Automating spoken dialogue management design using machine learning: An industry perspective

Tim Paek [a,*], Roberto Pieraccini [b]

[a] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
[b] SpeechCycle, 26 Broadway, 11th Floor, New York, NY 10004, USA

Received 27 June 2007; received in revised form 6 January 2008; accepted 14 March 2008

Abstract

In designing a spoken dialogue system, developers need to specify the actions a system should take in response to user speech input and the state of the environment based on observed or inferred events, states, and beliefs. This is the fundamental task of dialogue management. Researchers have recently pursued methods for automating the design of spoken dialogue management using machine learning techniques such as reinforcement learning. In this paper, we discuss how dialogue management is handled in industry and critically evaluate to what extent current state-of-the-art machine learning methods can be of practical benefit to application developers who are deploying commercial production systems. In examining the strengths and weaknesses of these methods, we highlight what academic researchers need to know about commercial deployment if they are to influence the way industry designs and practices dialogue management.
© 2008 Elsevier B.V. All rights reserved.

Keywords: Dialogue management; Machine learning; Reinforcement learning; Industry

1. Introduction

Automated systems that interact with users using speech recognition as the primary modality go by various names, such as spoken dialogue systems, interactive voice response systems (IVR), voice user interfaces (VUI), or simply, speech applications. Whatever the name, all of these systems rely on the fundamental task of *dialogue management*, which concerns what action or response a system should take in response to user input. The action taken may depend on a myriad of factors, from features of the speech recognizer (e.g., the confidence score of an utterance), to features of the dialogue interaction (e.g., the number of repairs taken so far), to features of the application domain (e.g., company guidelines for customer service), and to the response and status of external backends, devices, and data repositories. In many system architectures, a distinct *dialogue manager* exists to oversee and control the entire conversational interaction and execute any number of functions (e.g., Allen et al., 1998; Polifroni and Seneff, 2000). Some of these functions include updating new user input, resolving ambiguities, managing speech recognition grammars, communicating with external backends, and so forth. Ultimately, the dialogue manager prescribes the *next* action for each turn of an interaction. Because actions taken by the system directly impact users, the dialogue manager is largely responsible for how well the system performs and is perceived to perform by users—i.e. the user experience.

Given the importance of dialogue management, both as a research problem as well as a commercial enabler of advanced applications, researchers have recently been turning to machine learning methods to formalize and optimize the action selection process. In particular, one approach that has been gaining momentum is reinforcement learning (Sutton and Barto, 1998). In this approach, the dynamic interaction of a spoken dialogue is represented as a fully

* Corresponding author. Tel.: +1 425 703 8647; fax: +1 425 706 7329.
E-mail address: timpaek@microsoft.com (T. Paek).

0167-6393/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.specom.2008.03.010

or partially observable Markov decision process (MDP) and an optimal policy is derived which prescribes what actions the system should take in various states of the dialogue so as to maximize a reward function. The idea of having a system that can learn interactively from the rewards it receives is appealing given the alternative: crafting system responses to all possible user input, which, unfortunately, characterizes the status quo. In commercial settings, application developers, together with VUI designers, typically hand-craft dialogue management strategies using rules and heuristics. At best, these rules are based on the accumulated knowledge and trial-and-error experience of industry practitioners. At worst, they are based on intuition and limited experience. Either way, because it is extremely challenging to anticipate every possible user input, hand-crafting dialogue management strategies is an error-prone process that needs to be iteratively refined and tuned, which of course requires much time and effort. The reinforcement learning approach promises to give application developers a tool for automating the design of dialogue management strategies by having the system learn these strategies from feedback data. This casts dialogue management as an optimization problem, which, once solved, could remove the “art” from the process and facilitate rapid application development.

In this paper, we present an industry perspective on machine learning for spoken dialogue management in general, and on reinforcement learning in particular. We offer this perspective in light of the commercial success of speech applications. What began in academic institutions and industry laboratories more than 50 years ago has recently blossomed into a thriving business (Pieraccini and Lubensky, 2005). Hundreds of commercial systems are being deployed each year, adhering to industry-wide standards and protocols, such as VoiceXML, CCXML, MRCP, etc. Unfortunately, as researchers have started to point out, dialogue systems in industry have been evolving on a parallel path with those in academic research. Unless a “synergistic convergence” of architectures, abstractions and methods is reached from both communities, ideas and technologies in academic research run the risk of being overlooked by industry practitioners (Pieraccini and Huerta, 2005). This paper endeavors to bridge the gap between the two communities so that more research on spoken dialogue management can benefit commercial applications, which in turn would allow the research to impact thousands, if not millions, of customers on a daily basis.

This paper divides into three sections. In the first section, we describe how commercial applications are built and delineate the kinds of requirements, specification, and tuning that is typically used for dialogue management. In particular, we highlight the important role of VUI design and specification, drawing examples from real applications. In the second section, we investigate the pursuit of automated spoken dialogue management and attempt to distinguish between the hype and reality of automatic learning. In particular, we review current state-of-the-art reinforcement learning methods and consider to what extent dialogue management can be automated by these methods and whether these methods can be of practical benefit to application developers. Finally, in the last section, we draw upon the issues raised in the previous sections to discuss how researchers may be able to effectively influence the design and practice of dialogue management in industry, and in so doing, allow their research to touch the lives of multitudes of customers who interact with deployed commercial systems.

2. Commercial development and deployment

The development of a commercial spoken dialogue system (SDS) is a complex process. Dialogue management is only a part of it, though arguably the most expensive. Development starts with the collection of *requirements*, which describe what the system will and will not do. Before starting any development activity, requirements need to be properly and thoroughly defined to contain the scope of the final product. For example, for a banking application, the requirements will define which type of transactions the system is going to perform, such as account balances, fund transfer, and other inquiries. Moreover, customers often have strict business rules that need to define the way certain operations are performed. For instance, certain transactions may not be performed in the absence of proper user identification or validation, or the sequence of certain operations may be constrained. Only after the requirements have been defined and jointly agreed upon by the customer and the vendor who is going to develop the application, can a proper development phase begin.

Development processes are different from vendor to vendor, but they all include a functional specification and a detailed design of the interaction with the voice user interface (VUI). The functional *specification* is a high level definition of the different phases of the interaction, and is typically defined by a workflow where each block encapsulates atomic logical components defined in a hierarchical structure. For instance, in the simple banking application mentioned above, a functional module would be defined for account balance and one for fund transfer. Each high level module would be expanded into smaller components, such as requesting the source account, the destination account, and the amount of money to transfer.

The VUI design phase is mostly concerned with user experience. As such, it has to deal with the way the system speaks and interprets user speech input. Commercial systems are built by logically sequencing predetermined system utterances—called *prompts*—and the interpretation of the user responses based on context-specific grammars or statistical language models. The actual interaction logic is determined by a graph—called the *call-flow*—where the arcs are associated with conditional statements based on user speech input and other variables, and the nodes are associated with system actions.

Along with the VUI design, there has to be a parallel activity devoted to the development and tuning of the grammars and language models. Most often when new applications are built, corpora of utterances are not available to design or train the grammars and the language models. Thus, an effort has to be made to anticipate all possible user input for every specific dialogue context. This is no easy task. The anticipation of user utterances cannot be done independently of the design of each individual prompt. VUI designers and application developers have to work together in order to develop not only the prompts and grammars, but also the semantic structure, which defines the values returned by the grammars, and the conditions on the arcs of the call-flow.

One critical aspect of dialogue management is how a SDS should respond to errors. Because speech recognition is not perfect, there is always a non-zero chance that a user utterance will be misinterpreted. Hence, error handling is an integral part of the VUI design. One common way to alleviate errors is to use techniques aimed at establishing a confidence level for the speech recognition result, and to use that for deciding when to ask the user for confirmation, or reject the hypothesis completely and re-prompt the user. Too many confirmations as well as too many re-prompts would annoy users. So, it is important to reduce the number of confirmations and rejections to a minimum that also preserves a reasonable level of accuracy.

Backend integration is a large part of the engineering of a SDS. Most applications need to interact with external content management systems and tools, such as databases, customer relationship managers (CRMs), computer telephony integration (CTI) with call-center agents (e.g. screen pops), diagnostic tools, etc. The absence of a common interface for these external interactions often makes the backend integration for each new customer a costly and laborious engineering task. Moreover, the VUI needs to be gracefully integrated with the external backend in order to provide a reasonable user experience. For instance, we all have experienced, while talking to a customer representative, the communication disruption that occurs because of delays in the response of the backend, and how a good agent can leverage that time for advancing the interaction on alternative paths. A high performance commercial SDS should be able to behave in a similar fashion as part of its dialogue management.

A SDS which is deeply integrated with a backend pose serious issues for testing and quality assurance. In fact, *testing*, which is the next activity after a prototype system has been completely developed, requires the simulation of all possible ranges of return values and behaviors for the backend systems. The combination of all possible backend and user inputs and behaviors often makes testing a very cumbersome task which requires a thorough plan for its completion. The best practice for testing is to plan for it from the beginning of the design process as opposed to doing it as an afterthought (McConnell, 2004). Whatever dialogue management strategies are used, they need to be devised with the understanding that they can and will be thoroughly tested.

Testing of a sophisticated SDS is often incomplete, as it is for any complex software. Bugs always seem to appear after deployment. This, and the need for a deep understanding of the system behavior in the presence of real users, calls for thorough monitoring which typically lasts for the whole lifetime of the commercial application. Monitoring requires the possibility of accessing different types of logs and recordings of the dialogue transactions. If the system is properly structured, one can analyze the logs for a large number of interactions and report the system behavior at different levels of resolution, i.e. at the transactional level, the functional level, the VUI level, and the speech processing level. Besides statistical analyses of the behavior of the system, it is always useful to listen to interaction recordings and understand why the behavior of the system does not exactly match what was predicted and specified during design. However, listening to interaction recordings is a costly operation, and the selection of significant interactions to listen to is extremely important. Monitoring a deployed system results in a set of recommendations aimed at the improvement of the design, the correction of possible bugs, and the beginning of a new development cycle.

The above description of the full cycle of development of a commercial SDS should give an idea of its complexity and cost. The point we wish to make with respect to the main topic of this paper is that dialogue management is only one part of the development cycle. Although breakdowns of the full development cost into single components are rarely published, a rough estimation leads to the conclusion that, with the current platforms and tools, the design and development of the dialogue manager is no more than 15–20% of the total cost. Although automating that portion might reduce some cost, because of its tight coupling with other parts of development, which often require thorough monitoring, it may be difficult to achieve sustainable levels of automation. This is not to argue that automating dialogue management is not worth the effort, but rather that it must be done in appreciation of all the other aspects of developing, deploying and re-deploying commercial applications.

2.1. VUI completeness

Automating dialogue management design, or at least parts of it, may indeed be worth the effort in light of an architectural requirement for almost all commercial applications called *VUI completeness*. As explained earlier, any interface exposed by an enterprise to its customer base needs to be thoroughly tested and its quality assessed before putting it online. That means that the behavior of the application needs to be validated in all possible situations and conditions. In short, VUI completeness entails that *all* possible combinations of user inputs and conditions need to be anticipated. No “unpredicted” behavior can be left unexplored in a production system. Think, as an analogy, of the effect that an untested production web site would have on customers of companies such as Amazon or EBay. Furthermore, think of the consequences on their businesses if the behavior of those

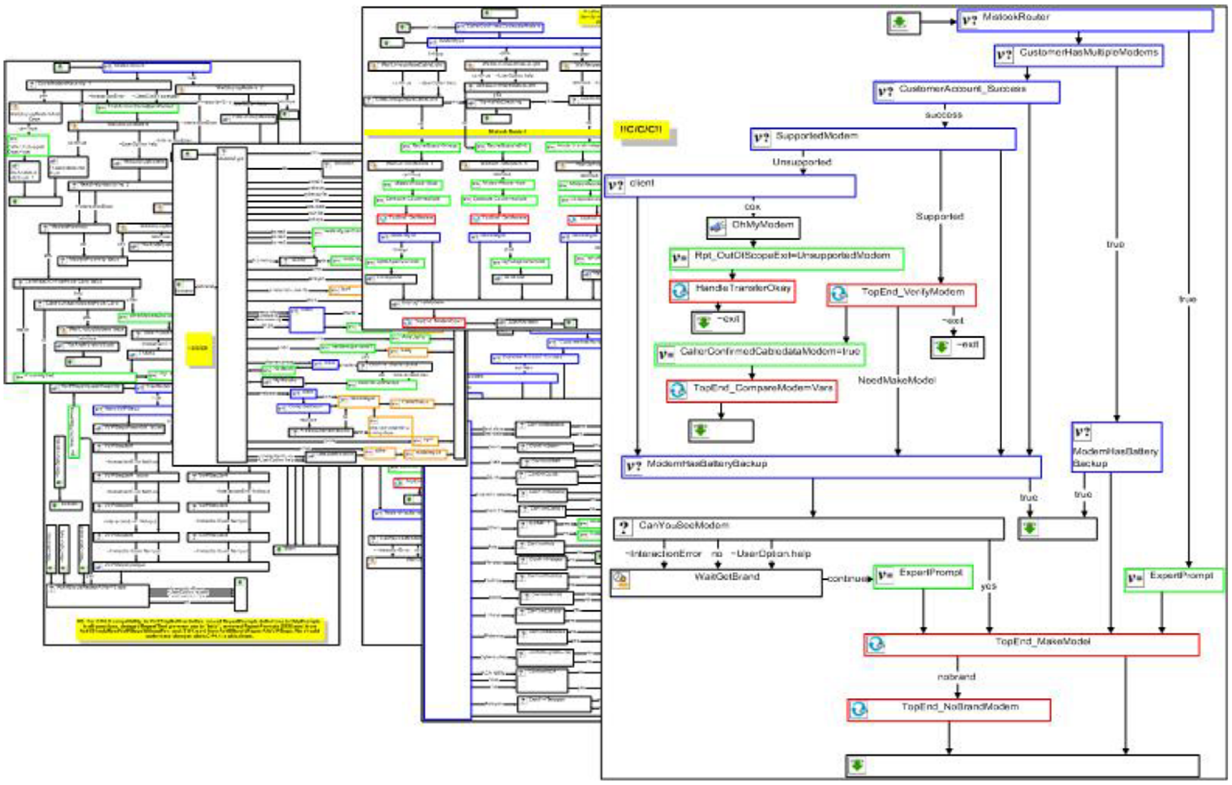

Fig. 1. A few screenshots taken from over three hundred pages of the call-flow graph of a typical troubleshooting spoken dialogue system. The figure is meant to give an idea of the complexity of today's dialogue systems; the actual details of the call-flows represented here are outside the scope of this paper.

web sites were not completely specified, predicted, and compliant with company guidelines. This architectural requirement of *VUI completeness* demands that all possible outcomes be anticipated so that "no unpredictable user input should ever lead to an unforeseeable behavior" (Pieraccini and Huerta, 2005). According to VUI completeness, the only acceptable outcomes are (1) the user fulfills the task, or (2) a fallback strategy is employed (e.g., handing off calls to a human operator).

VUI completeness is carried out through a VUI specification document, a standard procedure in industry, which is then reviewed and handed over to a team of developers who implement the application using their choice of platform. The specification process for any non-trivial commercial application is an arduous task. To give an idea of the complexity intrinsic in the functional design of a commercial SDS, we can safely say that the call-flow of the most sophisticated customer-care applications deployed today—3rd generation dialogue systems (Acomb et al., 2007)—include several thousands of nodes, prompts and hundreds of grammars. We show a portion of the call-flow graph for a troubleshooting application[1] in Fig. 1. The sheer size of the call-flow poses huge scalability problems for its commercial production and deployment. The only way to ensure VUI completeness for such a system is by having a direct mapping between the formalisms and abstractions used by the VUI specification designers and the programming model available to application developers. This is the reason why most dialogue managers, and sophisticated authoring tools, follow the same abstractions utilized in the VUI specification, and why more sophisticated dialogue management representations have not found much utilization in commercial settings (Pieraccini and Huerta, 2005). Although a call-flow graph, such as Fig. 1, limits the types of dialogue interactions possible to whatever can be represented by a finite-state controller (Pieraccini and Huerta, 2005), it is easy to understand, easy to adjust, and directly matches the conceptual requirements of the VUI specification process.

The burden of VUI completeness is certainly a selling point for automating dialogue management. Although it would seem as if any technology or method that could lighten this burden would be welcome by application developers, this is not the case. Application developers do not seek to fulfill VUI completeness as just an architectural requirement. It is a guarantee to their customers. Any technology that could facilitate VUI completeness must not only persuade VUI specification designers and application developers to trust it, but they, in turn, must be able to convince their customers to trust it. We return to this issue in Section 4.

3. Automating dialogue management design

Developing and deploying a commercial application that fulfills VUI completeness is clearly a demanding process, even with streamlining of that process through industry

[1] © 2007 SpeechCycle, Inc.

standards, tools, and best practices. Given that tuning is already performed using machine learning techniques, such as setting language model weights or confidence thresholds, the natural next step to consider is whether dialogue management overall can also be learned and tuned towards the optimization of the automation target and the user experience using similar techniques. This might circumvent the hand-crafted nature of devising, testing, and tuning dialogue management strategies, which typically requires a fair amount of expertise on the part of the development team. In this section, we evaluate the reinforcement learning approach to dialogue management, while at the same time broadening our discussion to machine learning in general whenever appropriate. We first provide relevant background on reinforcement learning techniques and then compare the strengths and weaknesses of these techniques with respect to the development and deployment issues raised in the previous section. In so doing, we discuss challenges that need to be overcome before reinforcement learning techniques can become commonplace in commercial applications as well as long-term opportunities for academic research.

3.1. Reinforcement learning

Reinforcement learning addresses the problem of how an agent should act in dynamic environments so as to maximize a reward function (Sutton and Barto, 1998). Dialogue management can be construed as a reinforcement learning problem in that a SDS needs to take sequential actions, and those actions should be "optimal" in some way, such as maximizing a reward function. A central debate in the reinforcement learning literature concerns the use of models. Model-free approaches do not explicitly represent the dynamics of dialogue, but instead directly approximate a value function that measures the desirability of each environment state. These approaches offer near-optimal solutions that depend on systematic exploration of all actions in all states (Watkins and Dayan, 1992). On the other hand, model-based approaches explicitly represent a model of the dynamics of the dialogue to compute an estimate of the expected value of each action. With a model, the SDS can reduce the number of steps it takes to learn a policy by simulating the effects of its actions at various states (Sutton and Barto, 1998). Perhaps for this reason, and for the fact that it is possible to derive a policy that is optimal with respect to the data, spoken dialogue researchers have by and large pursued model-based reinforcement learning methods (see e.g., Levin et al., 1998; Singh et al., 2002).

The framework underlying model-based reinforcement learning is that of the MDP, which can be characterized by a tuple (S, A, T, R), where:

- S is a finite set of states;
- A is a finite set of actions;
- T is a state-transition function such that $T(s', a, s) = p(s'—s, a)$; and
- $R : S \times A \mapsto \Re$ is a local reward function.

The objective of the SDS is to maximize its expected cumulative reward, which for the infinite time horizon, can include a discount factor to ensure that rewards accrued later are counted less than those accrued earlier:

$$E\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right) \tag{1}$$

where γ is a geometric discount factor, $0 \leqslant \gamma < 1$. The discount factor can be used to model processes that can terminate at any time with probability $1 - \gamma$, such as a user hanging up.

Unfortunately, an MDP requires complete knowledge of S, which may be intractably large if S encodes all relevant dialogue history variables (Young, 2002). Furthermore, keeping track of *unobservable* states such as the user's intentions and beliefs, which can be inferred from observations, such as the user's utterance, has been shown to improve performance (e.g., Roy et al., 2000; Williams and Young, 2005; Zhang et al., 2001). If a SDS cannot observe all states $s \in S$, then the MDP is considered a partially observable MDP (POMDP), and can be characterized by the tuple (S, A, T, R, O, Z), where

- S, A, T, R constitute an MDP;
- O is a finite set of observations that can be received from the environment; and
- Z is the observation function such that $Z(o, s, a) = p(o|s, a)$.

For the dialogue system, Because the dialogue system never knows with certainty what the current state is, it maintains a belief state $b(s)$, or a probability distribution over S. The local reward is then computed as the expected reward ρ over belief states:

$$\rho(b, a) = \sum_{s \in S} R(s, a) \cdot b(s) \tag{2}$$

and the objective of the dialogue system is again to maximize its expected cumulative reward, as in Eq. (1).

Once a spoken dialogue has been formalized as above, a number of algorithms can be exploited to learn an optimal or near-optimal *policy* from data (Kaelbling et al., 1996), where an optimal policy π: $S \mapsto A$ is a mapping from states to actions. With a POMDP, deriving a policy is more complicated (see Kaelbling et al., 1998 for a survey) as the policy itself becomes a mapping from n-dimensional belief states to actions. A POMDP policy can be represented in several ways. Perhaps the most pertinent for dialogue management is that of a finite-state controller, which can be learned for relatively simple interactions with small state spaces and actions (Hansen, 1998). Given that some application developers may already be familiar with this kind of representation, it has been investigated for dialogue management (Williams et al., 2005).

3.2. Strengths and weaknesses

The formalization in the previous subsection identifies the key concepts for utilizing a fully or partially observable MDP for dialogue management. Here, we consider the strengths and weaknesses of these concepts with respect to practical deployment, and discuss the challenges that need to be overcome. For reference, we summarize those strengths and weakness that we cover in Table 1, as well as the challenges and opportunities in Table 2.

3.2.1. Objective function

The appeal of reinforcement learning for speech research may be because dialogue management is cast into the same kind of statistical framework for optimization as speech recognition and spoken language understanding (Young, 2002). Unfortunately, whereas speech recognition and spoken language understanding is generally based on a maximum likelihood approach that essentially minimizes word or concept error rates, in dialogue management, the objective function is less clear. Eq. (1) states that the SDS should maximize its expected cumulative reward. However, that objective function could also be based on post hoc measures such as usability scores (Singh et al., 2002; Walker et al., 2001), and construed to reflect whatever qualities application developers may want the SDS to possess.

The usual practice is to accept the expected cumulative reward, Eq. (1), as the objective function, and adjust the reward function R to modify system behavior. However, the implications of an objective function for modeling dialogue have not been well investigated. First, it is unclear whether every dialogue can be viewed as an optimization problem with a specifiable objective function. Moreover, it is unclear how the choice of the expected cumulative reward, as opposed to any other objective function, affects the types of dialogue interaction that can be optimized.

Given an explicit objective function, a promising avenue for research is to optimize speech recognition and/or spoken language understanding using the same objective function as dialogue management. Just as spoken language understanding does not require correctly recognizing all the words, taking appropriate actions, in certain contexts, may not require correctly identifying all the words and

Table 1
A summary of strengths and weakness of the reinforcement learning approach with respect to practical deployment that we discuss in Section 3.2

	Strengths	Weaknesses
Objective function	• Optimization framework • Explicitly defined and adjustable • Can serve as evaluation metric	• Unclear what dialogues can and cannot be modelled using a specifiable objective function • Not easily adjusted
Reward function	• Overall behavior very sensitive to changes in reward	• Mostly hand-crafted and tuned • Not easily adjusted
State space and transition function	• Once modelled as MDP or POMDP, well-studied algorithms exist for deriving policy	• State space small due to algorithmic limitations • Selection is still mostly manual • No best practices • No domain-independent state variables • Markov assumption • Not easily adjusted
Policy	• Guaranteed to be optimal with respect to the data	• Removes control away from developers • Not easily adjusted
Evaluation	• Can be rapidly trained and tested with user models	• Real user behavior may differ • Creating user model may be more work than deploying prototype • Hand-crafted policy should be tuned to same objective function

Table 2
A summary of a research opportunities for the reinforcement learning approach that we discuss in Section 3.2

	Research opportunities
Objective function	• Open up black boxes and optimize ASR/SLU using same objective function as dialogue manager
Reward function	• Adapt reward function/policy based on user type or behavior (similar to adapting mixed-initiative) • Inverse reinforcement learning • Learn performance function for use as reward function
State space and transition function	• Learn what state space variables are important for local/long-term reward • Apply more efficient POMDP methods • Identify domain-independent state variables • Identify best practices
Policy	• Online policy learning (explore vs. exploit) • Identify domain-independent, re-usable error handling mechanisms
Evaluation	• Close gap between user model and real user behavior

concepts; e.g., in more conversational settings where maintaining a perception of competence outweighs everything.

For practical deployment, requiring an objective function to be explicitly specified may be both a strength and a weakness. It can be a strength in that the objective function can serve as an evaluation metric for controlled experiments. For example, it can be used to measure the effect of adding or removing features from the SDS. On the other hand, it can be a weakness in that most application developers have little to no experience with optimization or even statistics, and would likely be hard-pressed to specify an objective function. They may opt with the default setting, not understanding how it governs dialogue management, and later be puzzled as to why the SDS behaves as it does.

In general, applying any machine learning technique to dialogue management involves committing to an objective function. Although it is possible to select an appropriate objective function for optimizing the performance of a SDS with respect to a data set, it is harder to incorporate other goals that application developers have for dialogue management such as representational clarity to ensure VUI completeness, scalability, maintainability, portability, code reuse, and ultimately the usability of the interface. Because dialogue management design is just one part of the full development process, as discussed in Section 2, deciding what aspects of the interaction should be optimized for dialogue management is less important than making sure that whatever objectives they and their customers agree upon in their VUI specification document are fully met.

3.2.2. Reward function

In pursuing the expected cumulative reward in Eq. (1), a local reward function R must be specified. The typical practice is to assign a small negative reward for each dialogue turn and a large positive or negative reward upon completing the interaction successfully or unsuccessfully.

The local reward function is perhaps the most handcrafted aspect of the reinforcement learning framework for dialogue management. The overall behavior of the system, as dictated by the policy, is very sensitive to changes in R, yet R is almost always set by intuition, not data. For practical deployment, application developers may find it too difficult to assign R, as that entails having a good understanding about how the relative values of $R(s, a)$ for particular states and actions influence each other as well as the overall behavior of the system. Although application developers may be fine, for the most part, to go with reasonable defaults, if they are ever asked to modify the SDS so that it behaves in a certain way, this will be hard to do without conceptually understanding R very well. They may be better off coding heuristics they understand than to try to tweak R.

Another problem is that R is typically set so that it is static. However, it may be beneficial to have R adapt to the user type and/or goal. For example, for airline ticket reservation, without knowing anything about the user, a SDS may initially take actions that minimize penalties for turns. When it becomes clear that the user is more interested in exploring prices for vacations than purchasing a ticket, it may be worthwhile to increase the reward for engaging in longer dialogues and decrease the penalties for not being in a termination state. Of course, R could just be construed to be a function of user type or goal. Depending on the size of the state space S, that may or may not be feasible. Alternatively, different policies could be learned for different user types and alternated based upon the likelihood of a different user type. This kind of approach is similar to those that have been taken with adapting the dialogue modality between system- and mixed-initiative (Litman and Pan, 2002).

Finally, a promising avenue for future research is to learn the local reward function by watching a system behave according to its optimal policy and inferring R. This line of research is called “inverse reinforcement learning” (Ng and Russell, 2000). For dialogue management, it may be possible to learn R by observing human interlocutors engage in dialogue, allowing the SDS to mimic the human agent. A similar and potentially promising approach that has been investigated by researchers (Walker, 2000; Singh et al., 2002) is to learn a performance function, such as PARADISE, for estimating the reward of a final dialogue state. The performance function can be trained on usability data, which itself can be gathered at the end of user interactions with the SDS. However, it is still up to the application developer to understand and appreciate how the performance function is computed and what effects the different components of the function have on actual system behavior. For many application developers, setting a performance function will not be any easier than setting a reward function.

3.2.3. State space and transition function

So far, the discussion has focused on local and cumulative rewards, but R is a function of both the action space A and the state space S. The set of actions that a SDS can take is typically known, and for tractability purposes, kept small. This leaves open the question of what to include and how to model S. Indeed, modeling S is perhaps the most fundamental aspect of reinforcement learning, as it affects how the dynamics of the SDS operate, how rewards are assigned, and how tractable policy learning will be. For systems that currently utilize reinforcement learning, researchers have limited the state space to just a handful of variables, which constrains the kinds of domains and interactions that can be handled by the SDS. Furthermore, because the transition function T is Markovian, state space variables must be selected so as to support the Markov assumption, which may not always be the best option (Paek and Chickering, 2005). To deal with scalability, researchers have exploited factored representations of T to reduce the number of parameters that need to be estimated (Williams et al., 2005) and introduced methods to scale POMDP dialogue managers to slot-filling problems of realistic size (Williams and Young, 2006). However, the state space still needs to be delineated up front. This is, for the most part, a manual task. Even for research focused on selecting state space variables for policy learning (Chickering and Paek, 2007; Tetreault and Litman, 2006), choosing candidate variables to test still requires manual selection.

Deciding what state space variables to include is a problem common to all machine learning techniques in general. For practical deployment, it is unclear whether application developers should accept whatever variables researchers have utilized in their systems. First of all, the research community has not established best practices for modeling the state space of a spoken dialogue nor agreed upon a domain-independent set of variables that could be utilized in any SDS. Although certain variables, such as the confidence level of the recognition result, are utilized by almost all systems in research and industry, an interesting challenge for the field is to ascertain the set of all such common variables. In extending state space variables to new domains, application developers may find that they need to model domain-dependent variables to improve the performance of the SDS (Paek and Chickering, 2005). Alternatively, after the system has been deployed, they may find that they need to add new state variables. In such cases, the machine learning technique ceases to be an out-of-box solution. Unfortunately, for the reinforcement learning approach, adding new variables is not a minor fix. The entire policy has to be re-learned. As noted above, modeling the state space is fundamental, and affects everything.

3.2.4. Policy

The ultimate product of utilizing reinforcement learning methods for dialogue management is a policy that is optimal with respect to the data. Suppose that somehow tractability ceased to be a limiting factor, and that an optimal policy could be learned for arbitrarily large state and action spaces. Even in this ideal situation, the question of how beneficial an optimal policy is for application developers still remains.

Consider the issue of representation. As mentioned before, the policy can be represented in various ways, but always prescribes an optimal action that the SDS should take. Although it might seem as if this is what developers want—namely, a black box which tells the system what to do—it fundamentally wrests control of the dialogue flow from their hands, something that developers generally resist, for good reason. Of all the black boxes in a SDS (and there could be several, such as the speech recognizer), the one that affects the users the most is the dialogue manager because that is where all system actions are decided. Because the business of application developers revolves around satisfying the needs of their customers, if their customers tell them, for example, that they tried the SDS and were puzzled about how the system took a particular action after having gone through a series of exchanges, the developer better know how to fix that one particular action. This kind of fix, which would be relatively straightforward with dialogue strategies explicitly written out in code or represented in a call-flow graph such as in Fig. 1, is much harder to execute within the reinforcement learning framework because everything is interwoven. To get an action to change, and moreover, to change something several turns into the dialogue, may entail modifying R, S, T, Eq. (1), and/or γ. In short, a considerable amount of expertise in reinforcement learning is required to do the job, which might have been as easy as changing a few lines of code in a more conventional approach.

To get a better idea of how resistant application developers may be to reinforcement learning, and machine learning techniques for dialogue management in general, consider the case of the statistical language model (SLM). If prevalence is any indication of preference, then "phrase-based" context-free grammars are much more prevalent and preferred in the commercial world than SLMs, despite the fact that SLMs have been around longer historically and have demonstrated superior performance. Although application developers may be aware of the benefits of statistical modeling, such as greater robustness to out-of-grammar utterances (Rosenfeld, 2000), they often prefer the deterministic character of context-free grammars because they know how to control and modify them. And when they modify them, they can predict exactly what the results will be.

Fundamentally, what is at issue is the need to ensure VUI completeness. When application developers settle on a VUI specification document with their customers, this is a contract. The system has to perform exactly the way its behavior was specified and will be tested as such. If application developers feel that SLMs add too much uncertainty, imagine how they will feel about statistical dialogue management. In fact, the need to ensure VUI completeness renders most "inference-based" dialogue managers (Pieraccini and Huerta, 2005) problematic to industry. Inference-based dialogue managers rely on engines that select actions based on a formal description of the domain (e.g. in terms of goals or plans) and drive the application to a solution by attempting to satisfy well-defined criteria. With dialogue managers based on finite-state control, such as the call-flow graph, how the SDS will behave in any situation is explicit in the representation. Because inference-based dialogue managers often process many variables and may sometimes even involve a series of inference steps, it is not immediately clear how the system will behave.

Compared to other inference-based dialogue managers that utilize, for example, forms (e.g., Papineni et al., 1999) or goal hierarchies (e.g., Wei and Rudnicky, 2000), reinforcement learning has an even more complicated set of parameters (R, S, T, Eq. (1), γ) to adjust in order to obtain a desired system behavior in a specific context. The application developer could adjust the policy, but as soon as that is done, it immediately ceases to be optimal with respect to the data. However, this may not be so bad if all that an application developer really wants is a first stab at a reasonable policy, to reduce design time. Furthermore, if the data was limited in some fashion, then it really does not matter if the policy is no longer optimal because it was only optimal with respect to the data on which it was learned anyway.

Online policy learning for dialogue management holds great promise in this regard. As mentioned previously, dialogue researchers have mostly focused on model-based reinforcement learning approaches. Although online policy learning algorithms exist for model-based approaches (Barto et al., 1995), model-free approaches are more commonly utilized (Watkins and Dayan, 1992). Online policy learning is a promising area because the SDS would not be limited by the data on which it was trained. Without having explored all actions in all states, the system could engage in the type of exploration versus exploitation dilemma that characterizes classical reinforcement learning (Sutton and Barto, 1998). To date, very few dialogue researchers using reinforcement learning have investigated online policy learning (Chickering and Paek, 2007). Outside of reinforcement learning, promising statistical approaches for performing exploration versus exploitation of dialogue strategies have been shown to improve error recovery over time (Bohus et al., 2006).

3.2.5. Evaluation

The evaluation, and often the training, of reinforcement learning techniques for spoken dialogue systems has mostly centered on user simulation. Ever since researchers began examining reinforcement learning for dialogue management, they have realized that obtaining data to learn a policy would be problematic (Eckert et al., 1997, 1998). Because it is impractical, time-consuming and burdensome to have a SDS explore all different types of actions with real users, the idea was to learn a generative model of the user so that user actions could be simulated in response to system actions (see Schatzmann et al., 2006 for a survey). With good user modeling, a SDS could be rapidly prototyped and evaluated. Although this line of research is promising and might greatly benefit practical deployment (Pietquin and Dutoit, 2006), the challenge of making sure that the user model truly reflects what real users are likely to do, which oftentimes is dependent on very subtle aspects of the dialogue design and task domain, is a daunting task.

From a practical standpoint, coming up with a user simulation that provides realistic training data takes a lot of work, especially compared to what application developers currently do now, which is to create an initial prototype to collect real user data, and then iteratively improve the system through the normal development cycle. If user simulations ever become fully domain-independent and reusable, it is more likely that application developers will use them for requirements testing than for training reinforcement learning policies.

As noted in Section 3.2.1, having an explicit objective function can be advantageous in that it can serve as an evaluation metric. Oftentimes, that metric is difficult to realize without the aid of user simulation. User simulation provides a testing environment for conducting controlled experiments which might be too burdensome for real users. Unfortunately, just because a SDS does well with respect to average or total reward in simulations does not guarantee that real users will “reward” the system accordingly. In fact, although the objective function is supposed to globally optimize the dialogue, it has never really been empirically evaluated against systems that optimize local (myopic) decisions. Local optimization may provide a better user experience in cases where users unexpectedly change their behavior in responding to the system; that is, when local rewards change. For example, when users become suddenly frustrated, a SDS that is focused on local decisions may be better prepared to take actions that mollify and keep them engaged. Of course, R can be a function of user frustration as well, but, as we discussed in Section 3.2.2, that may or may not be feasible.

The question is how well do the simple reward functions that are commonly used within the reinforcement learning framework reflect real users’ reaction to a SDS? After all, depending on the objective function, which itself can be suspect, the cost of some types of errors, such as misunderstandings, can be worse than others, such as false rejections (Bohus and Rudnicky, 2005a). The best practice of course is to conduct user studies in addition to simulation experiments, which, because of lack of time and resources, is not often pursued by researchers. An important notable exception is dialogue research (Walker, 2000; Singh et al., 2002) where usability ratings are gathered at the end of dialogue interactions, and used to fit a performance function that also serves as a reward function. Although this approach may not necessarily make the reward function more accessible to application developers, as discussed in Section 3.2.2, at least user feedback is leveraged to improve the system.

Current practices for evaluating reinforcement learning-based systems need to be scrutinized more carefully. Originally (Eckert et al., 1997, 1998), user simulations were utilized to take an existing SDS with a large, representative corpus, and tune it according to a desired performance measure and user model profile. This approach can be regarded as a very reasonable post hoc optimization procedure for gearing a SDS towards a particular user population once a corpus is collected. Because a large, representative corpus is a luxury for most researchers, user simulations began to be utilized to generate data for learning policies. A big concern is the common practice of testing policies that have been trained on a simulated user using the same simulated user. This is essentially cheating. As pointed out in (Schatzmann et al., 2005), policies trained with a poor user simulation model may appear to perform well when tested on the same model, but fail when tested on a better user simulation model. Fortunately, the converse was not true: policies learned with a good model will still perform well when tested on a poor model.

Another common practice is to evaluate reinforcement learning policies against hand-crafted solutions (e.g., from an existing SDS) using average or total reward as a metric. The problem is that the hand-crafted solutions are not typically optimized according to same objective function, so it is not a fair comparison. If, for example, a learned policy is evaluated against a hand-crafted confidence threshold, then that threshold should be tuned to maximize the expected cumulative reward.

4. Influencing industry dialogue management

In exploring whether current reinforcement learning techniques can be of practical benefit to application developers, we discussed how inference-based dialogue managers pose difficulties for application developers who need to ensure VUI completeness. On the other hand, we also discussed how the need to ensure VUI completeness, which involves exhaustively anticipating and handling all possible user input, is itself an argument for automating at least parts of dialogue management. In applying machine learning techniques to dialogue management, it is important to strike a balance between automation and control. In this section, we discuss where to find that balance in light of the issues raised in the previous two sections so that research techniques can more effectively influence the design and practice of dialogue management in industry.

4.1. Balancing control and automation

As described in Section 2.1, VUI completeness demands that all possible outcomes of interacting with a SDS be anticipated so that no unpredicted input can result in an unforeseen behavior. This, of course, is a tall order, especially for any non-trivial commercial application. Although it may seem as if any technology (e.g., machine learning techniques) that can lighten this burden would find an immediate audience with application developers, that is not the case. Because application developers treat VUI completeness as not just an architectural requirement but a guarantee to their customers, it is critical that the adopted technology respects and facilitates the desire of application developers to maintain control of this guarantee. Hence, a fundamental principle for engaging industry practitioners is this: *Give application developers enough control to allow them to ensure VUI completeness*.

In the future, it may be that the technologies for automating dialogue management will become so successful that application developers will no longer be needed to ensure and maintain VUI completeness. Likewise, customers will just trust the technologies. But until that day comes (if ever), application developers need to have enough control so that they can convince their customers that their systems are VUI complete. And if problems should ever arise after deployment, application developers need to convince them that these problems can be immediately fixed. Note that application developers need *enough* control, not necessarily *full* control. In fact, there are many aspects of ensuring VUI completeness that application developers would be better off automating, as we delineate in the next section. However, maintaining control at the macro-level is vital because that is where application developers demonstrate VUI completeness to their customers. Hence, finite-state control dialogue managers, such as the call-flow graph, which utilize the same abstractions as the VUI specification, have great appeal for application developers and customers alike (Pieraccini and Huerta, 2005).

Any method for automating dialogue management at the macro-level should be transparent to application developers as well as their customers. Furthermore, it should allow application developers to have fine-grain control of system behavior, while at the same time, being expressive enough to allow the implementation of complex behavior in a simple and cost-effective way (Pieraccini and Huerta, 2005). With any automation technique, the maxim attributed to Alan Kay should always hold: *Simple things should be simple and complex things should be possible*.

One objection that might be raised against giving application developers *any* control at the macro-level is that whatever they do will be “hand-crafted” and as such, prone to error. This is certainly the perspective of die-hard machine learning advocates who argue that optimal behavior can only be obtained by learning from data. The assumption, that anything hand-crafted must be more prone to error than anything learned from data, needs to be carefully evaluated. Although hand-crafting will probably not fit data as well as any machine learning technique, this does not in-and-of-itself guarantee that a hand-crafted rule that has been devised to optimize the same criterion as the machine learning technique will perform any worse in the long run. In fact, the advantage of hand-crafted rules is they can be easily modified as changes occur and components are ported from one application to another. As VUI designers observe the effectiveness of hand-crafted rules, they often codify them into best practices. This does not mean, of course, that machine learning techniques cannot be part of best practices. Indeed, best practices themselves may involve any number of machine learning techniques, as for example, tuning grammar weights and confidence thresholds. This just means that human *inspectability*, maintainability and control are equally important objective criteria.

Another more general objection is that it is doubtful that current hand-crafted approaches will be able to continue to scale to meet the demands of ever more sophisticated applications. This is a legitimate concern. Academic research that does not have immediate applicability to industry problems but address issues that may be decades ahead into the future is valuable for exactly this reason. However, unless a new technology suddenly emerges that allows a SDS to autonomously learn and interact as well as its human counterpart, and as such inspires the trust of its employers and users, it is likely that employers will always want to preserve human inspectability, maintainability and control. So far, the abstractions that developers have created to scale applications so that they can meet the demands of a sophisticated domain, as for example, the call-flow graph shown in Fig. 1, have been able to do the job and ensure VUI completeness.

It is also important to note that in certain situations, application developers may be willing to trade off lack of inspectability and control for better performance. For example, if a SDS is primarily used in acoustically challenging environments, users may be better off with a fully automated system that continually learns than a hand-crafted solution that requires periodic updating. So far, however, industry has not witnessed the levels of performance that would warrant complete trust.

The value of human oversight and control in dialogue management should never be underestimated. Interacting with other human beings is incredibly complex, and it is not clear that automated systems will be able to effectively process all the nuances of interaction that human beings monitor. For example, in cultures that utilize honorifics, grammatically and semantically correct expressions can still be "socially inappropriate" (as defined by a native). Little nuances that hinge on social and cultural expectations can derail an entire interaction. This is why the process of software internationalization or localization (Esselink, 2000), which involves adapting products such as publications, hardware or software for non-native environments, especially other nations and cultures, is a respected part of software development. Cultural values, social context, and even governmental issues such as censorship and privacy, are all factors that can hinder or promote the successful deployment of a commercial SDS to a wide and diverse user population.

4.2. Potential areas for automation

Guided by the principle that we should give application developers enough control of spoken dialogue management to allow them to ensure VUI completeness, in this section we highlight a few research areas that are likely to be embraced by industry. These are meant as examples of how techniques for automating different aspects of spoken dialogue management can relieve application developers of difficult challenges while at the same time giving them enough control for VUI completeness. As will be evident in the examples, the research areas listed involve fine-grain automation, as opposed to macro-level automation, of dialogue management. Indeed, one way of achieving a balance between automation and control is to focus automation only on fine-grain decisions, such as the exact wording of a prompt. This is not intended to argue that macro-level automation of dialogue management is not possible, but rather that it is currently unclear what form that kind of automation would take. Furthermore, several of the examples purposely involve some degree of reinforcement learning in order to show that, despite the criticisms raised in Section 3, reinforcement learning techniques can be made attractive to application developers. Before delving into the examples, we first provide a short background on performance metrics in industry.

4.2.1. Industry performance metrics

It is common practice in industry to routinely upgrade a SDS in order to improve its performance. Typically, the SDS is deployed for a certain amount of time until sufficient statistics facilitate a reliable estimation of its performance, generally characterized by a number of measurable parameters. Depending on the task the SDS is designed to automate, several key performance indicators (KPD) are used by customers (even appearing in customer–vendor contracts) to measure and assess the quality of a deployed solution. One KPD that is widely used in the customer-care industry is the so-called "automation rate" (a.k.a., self-service, deflection, or containment rate). The automation rate is the ratio between the calls that are completely handled by the SDS and the number of calls it receives. The calls which are not automated typically fall into different categories, such as "escalated" or "partially automated" calls (e.g., calls where the system provides some level of automation, but then escalates it to a human agent), "abandoned" calls (e.g., calls where the user hangs up at an early stage of the interaction), and "out-of-scope" calls (e.g., calls that fall outside the domain of competence of the SDS).

Automation rate is a KPD that businesses care about since customer savings, return-on-investment (ROI), and often the vendors' revenue are all directly related to it. Although some amount of arbitrariness is involved (e.g., the definition of a "successful" automated call is somewhat arbitrary for borderline calls), the automation rate is an objective metric that can be performed automatically on system logs. On the other hand, another measure of success which businesses also care about, namely, user experience, seems to reach beyond the realm of objectively measurable quantities. However, improving the user experience, in general, can have the effect of increasing automation rate, as evidenced for example in the strong correlation found between certain features of the interaction (e.g. the number of speech recognition errors), the user experience, and the overall automation rate in a PARADISE analysis (Walker et al., 2000, 2001). Improvements in automation rate cannot be obtained at the expense of a reduction of the user experience.

These considerations are reflected in the way commercial applications are typically improved and updated. Improving speech recognition performance is an activity that per-se enhances user experience and the automation rate. As such, vendors employ routine processes for data acquisition and grammar tuning, as discussed earlier. But there are other elements of the spoken dialogue interaction that require a more global approach, where the use of massive amounts of data can potentially improve automation rate, and ultimately, the penetration and acceptance of speech interfaces in the wider consumer market. These elements fall into at least three categories: (1) *task-independent behaviors* (e.g., error correction and confirmation behavior), (2) *task-specific behavior*s (e.g., logic associated with certain customer-care practices), and (3) *task-interface behaviors* (e.g., prompt selection).

What the three elements have in common today is the lack of robust guiding principles which are validated by empirical evidence. In general, good VUI designers can come up with good designs based on their experience, but more often than not, they come up with several *competing* designs or choice-points, which can only be validated in a statistical manner. Hence, it is possible to leverage machine learning techniques while still allowing designers to have enough control to ensure VUI completeness. And we will see, even reinforcement learning can be gainfully employed to improve dialogue management and provide enormous value to industry.

4.2.2. Task-independent behaviors

In examining reinforcement learning techniques, we discussed the difficulties of selecting the proper state space variables. Some parameters, such as the confidence of the recognition result, seem to be variables that are likely to play an important role in making any decision about what action to take next regardless of the task. This suggests that perhaps task-independent strategies can be taken as universal actions regardless of the content of the utterance. This might be true for dialogue management of clarification and error recovery (Bohus and Rudnicky, 2005b). Although various development platforms do support the inclusion of a “hidden sub-dialogue” component for managing clarification and error recovery, these components still need to be tuned, which is for the most part a labor-intensive, manual process. Any research that could automate the tuning so that it optimizes performance would provide great value. For example, Bohus et al. (2006) describe an online learning approach that significantly improved non-understanding recovery rate where confidence intervals for the likelihood of success for each recovery strategy were computed in real-time and then used for exploration versus exploitation in a deployed research SDS.

Another aspect of task-independent behavior concerns the handling of operator requests in customer-care applications. The automation rate, and hence the business advantage of deploying a SDS, is greatly reduced by users who request an operator early on in the course of the interaction. Specific solutions that have been hand-crafted to address this problem include: denying escalation in certain parts of the application task; providing educational dialogues which advise users on what the system can and cannot do; updating users constantly of their progress towards their goals; and predicting whether the dialogue has gone awry and invoking actions to remedy that. Again, these hand-crafted solutions can be augmented by data-driven policies which can be learned and optimized within a machine learning framework.

As an example, consider the difficult task of designing an escalation strategy. There comes a point in an interaction where a SDS should be able to decide whether to continue handling the task or to fall back to a strategy such as transferring the user to a human agent. Typically, this is done by relying on heuristic considerations, such as the number of speech rejections, or after having exhausted all possible alternative resolution steps. Machine learning can be used at an abstract level to predict whether the outcome of the dialogue will be successful or not (Walker et al., 2002). If the outcome also has a direct impact on a computable utility, such as revenue, savings, or call-center costs, as researchers have shown, it can be included in the optimization criterion (Paek and Horvitz, 2004; Levin and Pieraccini, 2006). In fact, it is possible to find an optimal point where an escalation decision has to be taken in order to maximize a reward, or reduce the cost of the transaction.

4.2.3. Task-specific behaviors

Industry practitioners have developed effective best practices for building simple transactional applications of the first and second generation dialogue systems (Acomb et al., 2007). As such, there are well-known strategies for interacting with users to book flights or to order pizzas. Trying to learn these strategies may end up being a futile academic exercise. This is not true for third generation, or problem solving, dialogue systems. Customer-care applications that help user solve technical support, service provisioning (e.g. mobile telephony), and billing issues are rather complex. Often the optimal process and steps necessary to approach and solve these issues are not known in advance, or only superficially known by subject matter experts (SME). In fact, most of these applications fall under the general rubric of *troubleshooting*, which, by its very nature and complexity constitutes the perfect venue for machine learning that involves knowledge representation and reasoning (e.g., using influence diagrams as in Heckerman et al., 1995). For example, although industry-wide best practices exist for troubleshooting internet connectivity, software applications, and complex devices, these practices derive from the domain knowledge and experience of SMEs. Because human knowledge and experience is limited, the possibility of training an automated system that encodes their knowledge using vast amounts of data to provide customer care is an exciting endeavor. In fact, automated solutions for complex customer care have already started to be successfully adopted by industry to handle millions of calls (Acomb et al., 2007), and some researchers have even started trying to apply a POMDP approach to troubleshooting (Williams, 2007).

4.2.4. Task-interface behaviors

Crafting prompts for the interface of a SDS is a difficult task, and human–computer interaction researchers have shown that the exact wording of a prompt can significantly affect user performance and user perception of usability (Nass and Brave, 2005). Even though industry best practices do exist for crafting prompts (Balentine and Morgan, 2001), after deployment, application developers sometimes find that they need to adjust the prompts to achieve higher performance. Here is where machine learning techniques can be of practical benefit. Application developers typically have a set of possible prompts they would like to explore for any given state of the dialogue. As such, they could leave the prompts (i.e., the competing prompt designs) to a reinforcement learning algorithm that could then explore the different outcomes in terms of any specified KPDs, and settle on those solutions that seem to result in the highest performance. This kind of use of reinforcement learning was investigated with respect to system and mixed-initiative prompts (Lewis and Fabbrizio, 2006) and also with respect to open prompts in a natural language understanding scenario (Acomb et al., 2007). Another early example of using reinforcement learning to pick between a constrained set of choice-points was the ELVIS system (Walker, 2000), which explored choices in initiative, summarization and read strategies for email and used fixed strategies elsewhere. The results of these works suggest that the use of experimentation and exploitation in VUI design—which has historically been considered more of an art rather than a science—can not only improve commercial applications, but also give application developers a way to maintain control while allowing machine learning to optimize what they find difficult.

Choosing the right prompt from among a set of competing prompts based on a specified objective function is just a simple use of the reinforcement learning paradigm. The natural evolution of this practice includes the selection of competing VUI designs, styles, and even personas. The difficult and expensive part, however, is to come up with a number of competing designs and strategies that individually make sense and can bring substantial improvement to the overall performance. This undoubtedly would still require a human in the loop, which is desired anyway to assure customers of VUI completeness.

5. Conclusion

As spoken dialogue systems continue to expand into all different kinds of consumer services, research in spoken dialogue management has the potential to have a direct impact on the lives of countless consumers who will be interacting with these systems. The speech industry is burgeoning, and a ripe opportunity presents itself for dialogue researchers who are interested in bringing the fruits of their labor to the public. But they must understand how commercial development and deployment works in order to effectively influence the way applications developers design and practice dialogue management.

In this paper, we discussed the kinds of requirements, specification and tuning that is typically used for dialogue management in industry. We highlighted the important role of VUI completeness, and how application developers need to ensure VUI completeness as a guarantee to their customers. In considering whether the load of VUI completeness can be lightened by machine learning techniques, we examined to what extent current reinforcement learning techniques, which have been gaining momentum in research, can be of practical benefit to application developers. Finally, we discussed how, in applying machine learning techniques to dialogue management, a balance must be found between automation and control if research is to effectively influence the design and practice of dialogue management in industry. In particular, we emphasized how any technology for automating aspects of dialogue management should give application developers enough control of spoken dialogue management to allow them to ensure VUI completeness. We hope that the discussion and ideas raised by this paper will allow more research in dialogue management to find its way into commercial applications.

References

Acomb, K., Bloom, J., Dayanidhi, K., Hunter, P., Krogh, P., Levin, E., Pieraccini, R., 2007. Technical support dialog systems issues problems and solutions. In: Proc. HLT Workshop on Bridging the Gap Academic and Industrial Research in Dialog Technology.

Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L., Stent, A., 1998. An architecture for a generic dialogue shell. Nat. Lang. Eng. 6 (3-4), 213–228.

Balentine, B., Morgan, D., 2001. How to Build a Speech Recognition Application, second ed., Style Guide for Telephony Dialogues Enterprise Integration Group.

Barto, A.G., Bradtke, S.J., Singh, S.P., 1995. Learning to act using real-time dynamic programming. Artif. Intell. 72 (1–2), 81–138.

Bohus, D., Rudnicky, A., 2005a. Sorry, I didn't catch that! An investigation of non-understanding errors and recovery strategies. In: Proc. 6th SIGDIAL Workshop on Discourse and Dialogue, pp. 128–143.

Bohus, D., Rudnicky, A., 2005b. Error handling in the raven-claw dialog management system. In: Proc. HLT-EMNLP, pp. 225–232.

Bohus, D., Langner, B., Raux, A., Black, A., Eskenazi, M., Rudnicky, A., 2006. Online supervised learning of non-understanding recovery policies. In: Proc. IEEE/ACL Workshop on Spoken Language Technologies (SLT 06), Palm Beach, Aruba.

Chickering, D.M., Paek, T., 2007. Personalizing influence diagrams: applying online learning strategies to dialogue management. User Model. User-Adap. Interact. 17 (1–2), 71–91.

Eckert, W., Levin, E., Pieraccini, R., 1997. User modeling for spoken dialogue system evaluation. In: Proc. IEEE ASRU Workshop, pp. 80–88.

Eckert, W., Levin, E., Pieraccini, R., 1998. Automatic evaluation of spoken dialogue systems. In: Proc. TWLT13: Formal Semantics and Pragmatics of Dialogue, pp. 99–110.

Esselink, B., 2000. A Practical Guide to Software Localization. John Benjamins Publishing Co.

Hansen, E.A., 1998. An improved policy iteration algorithm for partially observable MDPs. In: Jordan, Michael I., Kearns, Michael J., Solla, Sara A. (Eds.), Advances in Neural Information Processing Systems, Vol. 10. The MIT Press.

Heckerman, D., Breese, J., Rommelse, K., 1995. Decision-theoretic troubleshooting. Commun. ACM 38 (3), 49–70.

Kaelbling, L.P., Littman, M.L., Moore, A.P., 1996. Reinforcement learning: a survey. J. Artif. Intell. Res. 4, 237–285.

Kaelbling, L.P., Littman, M.L., Cassandra, A.R., 1998. Planning and acting in partially observable stochastic domains. Artif. Intell. 101 (1–2), 99–134.

Levin, E., Pieraccini, R., 2006. Value-based optimal decision for dialog systems. In: Proc. IEEE/ACL Workshop on Spoken Language Technologies (SLT 06).

Levin, E., Pieraccini, R., Eckert, W., 1998. Using Markov decision processes for learning dialogue strategies. In: Proc. IEEE Transactions on Speech and Audio Processing, Vol. 8, pp. 11–23.

Lewis, C., Fabbrizio, G., 2006. Prompt selection with reinforcement learning in an AT&T call routing application. In: Proc. Interspeech.

Litman, D., Pan, S., 2002. Designing and evaluating an adaptive spoken dialogue system. User Model. User-Adapt. Interact. 12 (2–3), 111–137.

McConnell, S., 2004. Code Complete. Microsoft Press.

Nass, C., Brave, S., 2005. Wired for Speech: How Voice Activates and Advances the Human–Computer Relationship. MIT Press, Cambridge, MA.

Ng, A.Y., Russell, S., 2000. Algorithms for inverse reinforcement learning. In: Proc. ICML, pp. 663–670.

Paek, T., Chickering, D.M., 2005. On the Markov assumption in spoken dialogue management. In: Proc. 6th SIGDIAL Workshop on Discourse and Dialogue, pp. 35–44.

Paek, T., Horvitz, E. 2004. Optimizing automated call routing by integrating spoken dialog models with queuing models. In: Proc. HLT-NAACL, pp. 41–48.

Papineni, K., Roukos, S., Ward, R., 1999. Free-flow dialogue management using forms. In: Proc. Eurospeech, pp. 1411–1414.

Pieraccini, R., Huerta, J., 2005. Where do we go from here? Research and commercial spoken dialog systems. In: Proc. 5th SIGDIAL Workshop on Discourse and Dialogue, pp. 1–10.

Pieraccini, R., Lubensky, D., 2005. Spoken language communication with machines: the long and winding road from research to business. In: Proc. IEA/AIE, pp. 6–15.

Pietquin, O., Dutoit, T., 2006. A probabilistic framework for dialog simulation and optimal strategy learning. IEEE Trans. Audio, Speech Lang. Process. 14 (2), 589–599.

Polifroni, J., Seneff, S., 2000. Galaxy-II as an architecture for spoken dialogue evaluation. In: Proc. 2nd Internat. Conf. on Language Resources and Evaluation (LREC), pp. 725–730.

Rosenfeld, R., 2000. Two decades of statistical language modeling: where do we go from here? Proc. IEEE 88 (8), 1270–1278.

Roy, N., Pineau, J., Thrun, S., 2000. Spoken dialogue management using probabilistic reasoning. In: Proc. ACL.

Schatzmann, J., Stuttle, M., Weilhammer, K., Young, S., 2005. Effects of the user model on simulation-based learning of dialogue strategies. In: Proc. IEEE ASRU Workshop, pp. 412–417.

Schatzmann, J., Weilhammer, K., Stuttle, M.N., Young, S., 2006. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. Knowl. Eng. Rev. 21 (2), 97–126.

Singh, S., Litman, D., Kearns, M., Walker, M., 2002. Optimizing dialogue management with reinforcement learning: experiments with the NJ-fun system. J. Artif. Intell. Res. 16, 105–133.

Sutton, R.S., Barto, A., 1998. Reinforcement Learning: An Introduction. MIT Press.

Tetreault, J.R., Litman, D.J., 2006. Comparing the utility of state features in spoken dialogue using reinforcement learning. In: Proc. HLT-NAACL, pp. 272–279.

Walker, M.A., 2000. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. J. Artif. Intell. Res. 12, 387–416.

Walker, M.A., Kamm, C.A., Litman, D.J., 2000. Towards developing general models of usability with PARADISE. Nat. Lang. Eng.: Special Issue on Best Practice in Spoken Dialogue Systems. 6 (3–4), 363–377.

Walker, M.A., Passonneau, R.J., Boland, J.E., 2001. Quantitative and qualitative evaluation of DARPA communicator spoken dialogue systems. In: Proc. ACL, pp. 515–522.

Walker, M., Langkilde-Geary, I., Hastie, H., Wright, J., Gorin, A., 2002. Automatically training a problematic dialog predictor for the HMIHY spoken dialogue system. J. Artif. Intell. Res. 16, 293–319.

Watkins, C.J.C.H., Dayan, P., 1992. Q-learning. Mach. Learn. 8 (3), 229–256.

Wei, X., Rudnicky, A., 2000. Task-based management using an agenda. In: Proc. ANLP/NAACL, pp. 42–47.

Williams, J.D., 2007. Applying POMDPs to dialog systems in the troubleshooting domain. In: Proc. HLT Workshop on "Bridging the Gap, Academic and Industrial Research in Dialog Technology", pp. 1–8.

Williams, J.D., Young, S., 2005. Scaling up pomdps for dialog management: The summary pomdp method. In: Proc. IEEE ASRU Workshop.

Williams, J.D., Young, S., 2006. Partially observable Markov decision processes for spoken dialog systems. Comput. Speech Lang. 21 (2), 393–422.

Williams, J.D., Poupart, P., Young, S., 2005. Factored partially observable Markov decision processes for dialogue management. In: Proc. 4th Workshop on Knowledge and Reasoning in Practical Dialog Systems, pp. 76–82.

Young, S., 2002. The statistical approach to the design of spoken dialogue systems. Cambridge University Engineering Department Tech Report CUED/F-INFENG/TR.433, Cambridge, England.

Zhang, B., Cai, Q., Mao, J., Guo, B., 2001. Planning and acting under uncertainty: a new model for spoken dialogue systems. In: Proc. 16th Conf. on Uncertainty in Artificial Intelligence, pp. 572–579.