The Hidden Agenda User Simulation Model

Jost Schatzmann and Steve Young, *Fellow, IEEE*

***Abstract*—A key advantage of taking a statistical approach to spoken dialogue systems is the ability to formalise dialogue policy design as a stochastic optimization problem. However, since dialogue policies are learnt by interactively exploring alternative dialogue paths, conventional static dialogue corpora cannot be used directly for training and instead, a user simulator is commonly used. This paper describes a novel statistical user model based on a compact stack-like state representation called a *user agenda* which allows state transitions to be modeled as sequences of push- and pop-operations and elegantly encodes the dialogue history from a user's point of view. An expectation-maximisation based algorithm is presented which models the observable user output in terms of a sequence of hidden states and thereby allows the model to be trained on a corpus of minimally annotated data. Experimental results with a real-world dialogue system demonstrate that the trained user model can be successfully used to optimise a dialogue policy which outperforms a hand-crafted baseline in terms of task completion rates and user satisfaction scores.**

***Index Terms*—Dialogue management, Markov decision process, planning under uncertainty, spoken dialogue system (SDS), user simulation.**

I. Introduction and Overview

A. Statistical Spoken Dialogue Systems

THE general architecture of a conventional spoken dialogue system (SDS) is shown in Fig. 1. The speech recognizer receives the acoustic signal x_u emitted by the user, translates it into a feature-based representation and outputs the most likely sequence of words $\tilde{w}_u$. The text-based output of the speech recognizer is then semantically decoded by a speech understanding component and associated with a meaning representation, typically in the form of a *dialogue act* $\tilde{a}_u$ [1]. On the output side, the reverse process is followed. The machine generated dialog act a_m is converted into a sequence of words w_m which is synthesized using a text-to-speech component to produce the acoustic output signal x_m.

The task of the dialogue manager (DM) at the core of a SDS is to control the flow of the dialogue, handle the uncertainty arising from speech recognition and understanding errors, and perform forward planning. The DM interprets the observed (and potentially corrupted) user act $\tilde{a}_u$, resolves contextual references, and updates the machine's internal state s_m. Based on s_m, an appropriate system response a_m is then selected according to the system's *dialogue policy* which defines the machine's conversational behavior.

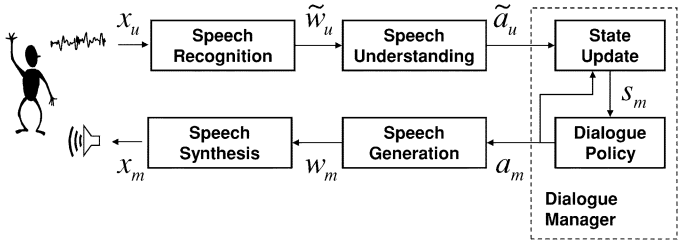

Fig. 1. Main components of an SDS.

In recent years, statistical approaches to SDS have gained increasing popularity [2], [3] since they allow design criteria to be expressed as objective reward functions and dialogue policy learning to be cast as a stochastic optimization problem. Using the Markov-Decision-Process model as a formal representation of human–computer dialogue, the DM is cast as a learning agent operating in a discrete state space $\mathcal{S}_m$ and using an action set $\mathcal{A}_m$. At each time step, the DM is in some state $s_m \in \mathcal{S}_m$, takes action $a_m \in \mathcal{A}_m$, receives a real-valued numerical reward $r(s_m, a_m)$ and transitions to the next state s'_m. A dialogue policy $\pi(s_m) \rightarrow a_m$ can thus be viewed as a deterministic mapping from states to actions. The optimal policy π^* is defined as the one that maximizes the expected total reward per dialogue and it can be learnt using reinforcement learning [4], [5].

B. Simulation-Based Reinforcement Learning

Since dialogue policies are learned by interactively exploring alternative dialogue paths, conventional static dialogue corpora cannot be used directly to train a statistical DM. Instead, a two-stage approach is used (see Fig. 2). In the first stage, a statistical model of user behavior is trained on a limited amount of dialogue data collected with real users using a system prototype or a Wizard-of-Oz setup. In the second stage, the dialogue manager is optimized using reinforcement-learning through interaction with the simulated user. User behavior is typically modeled at the abstract level of dialogue acts since this avoids the unnecessary complexity of generating acoustic speech signals or word sequences. As shown in Fig. 2, an error model can also be added to simulate the noisy communication channel between the user and the system [6]. The simulation-based approach allows any number of training episodes to be generated so that the learning DM can exhaustively explore the space of possible strategies.

Manuscript received June 06, 2008; revised November 23, 2008. Current version published April 01, 2009. This work was supported in part by the U.K. EPSRC under Grant Agreement EP/F013930/1 and in part by the EU FP7 Programme under Grant Agreement 216594 (CLASSIC Project: www.classic-project.org). The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Ruhi Sarikaya.

J. Schatzmann is with the Engineering Department, Cambridge University, Cambridge CB21PZ, U.K. (e-mail: schatzmann@gmail.com).

S. Young is with the Information Engineering Division, Engineering Department, Cambridge University, Cambridge CB21PZ, U.K. (e-mail: sjy@eng.cam.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASL.2008.2012071

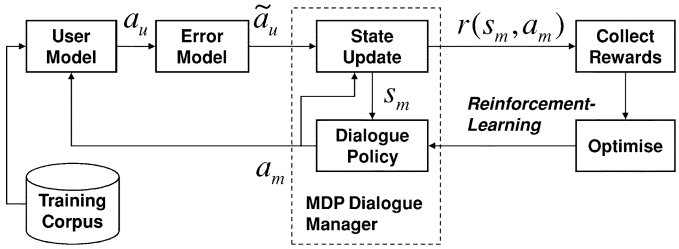

Fig. 2. Dialogue policy optimization with a simulated user.

Given that the simulated user generalizes well to unseen dialogue situations, it also enables the DM to deviate from the dialogue policies present in the training corpus, hence making it possible to explore new and potentially better policies.

C. Statistical User Modeling

A survey of user simulation techniques for dialogue optimization is given in [7]. Much of the difficulty in building a good user simulator lies in combining the conflicting objectives of reproducing the complex characteristics of user behavior in a realistic fashion while simultaneously maintaining a small and readily trainable model parameter set. Early work on semantic level user simulation by Levin and Pieraccini [8] investigated the use of a simple bigram model $P(a_u \mid a_m)$ for predicting user responses to machine acts. While the parameters of this model could be easily trained on data, the generated dialogues often lacked realism, with the simulated user continuously changing its goal, repeating information or violating logical constraints. Later work hence examined the use of explicit goal representations [9], [10] and longer dialogue histories [11], [12] to ensure greater coherence in user behavior over the course of a dialogue. Generally, some representation of the user state $s_u \in \mathcal{S}_u$ is required to capture the relevant dialogue history and achieve consistency in user behavior. A variety of different state space definitions and techniques for modeling $P(a_u \mid s_u)$ can be found in the literature, including feature-rich information state-based approaches [11], [13], clustering techniques [14], [15], and hidden Markov models [16].

A practical problem arising when training state-based user models, is that the true user state s_u is not observable in human–computer dialogue data. In the existing state-based approaches, this problem is typically circumvented by labeling training data with dialogue state information and conditioning user output on the annotated dialogue state $s_d \in \mathcal{S}_d$ rather than the unobservable user state s_u. While this simplifies the training process, it requires the state space $\mathcal{S}_d$ to be defined in advance and providing the necessary annotation involves a considerable effort. If done manually, labeling is expensive and it can be difficult to ensure inter-annotator agreement. Using an automatic tool for dialogue state annotation [11] can improve efficiency, but the development of the tool itself is a time-consuming process and there is no way of evaluating if the automatic annotation is correct without manually inspecting large amounts of data. A new model parameter estimation approach that requires only the derivation of dialog acts from the directly observable user utterances as proposed in this paper is thus highly desirable.

D. Paper Outline

This paper introduces a novel statistical method for user simulation based on the concept of a stack-like representation of the user state. Referred to as the *user agenda*, this structure of pending dialogue acts serves as a convenient mechanism for encoding the dialogue history and the user's "state of mind" (Section II). To allow model parameter estimation to be performed on minimally labeled training data without state-specific annotation, an expectation-maximization (EM)-based algorithm is presented which models the observable user output in terms of a sequence of hidden agenda states (Section III). While the space of possible agenda states and state transitions is vast, it is shown that tractability can be achieved by reducing action selection and state updates to a sequence of atomic push- and pop-operations. Using dynamically growing tree structures to represent state sequences and a summary-space mapping for state transitions, parameter estimation can then be successfully carried out on minimally annotated data.

Considerable attention in this paper is devoted to evaluation methods and results (Sections IV to VI). Following an overview of the experimental setup in Section IV-A, dialogue policy training experiments are described in Section IV-B and a rule-based baseline dialogue manager is described in Section IV-C. The simulation-based results reported in Section V show that policies trained with the agenda-based user model outperform those trained with a competing baseline simulator and indicate the benefit of training under noisy conditions. A user study conducted with 36 subjects demonstrates that the robust performance of the learned policies also carries over to a real-world human interaction task (Section VI). Finally, Section VII concludes the paper with a summary and an outline of future work.

II. Hidden Agenda User Model

A. User Behavior at a Semantic Level

Human–machine dialogue at a semantic level can be formalized as a sequence of states and *dialogue acts*. Dialogue acts generally "represent the meaning of an utterance at the level of illocutionary force" [17]. They enable the user model and dialogue manager interface to be standardized and they serve as an annotation standard for labeling dialogue data [18]. The definition of dialogue act taxonomies is an ongoing research area and a variety of different proposals can be found in the literature [19]–[21].

For the experiments presented in this paper the Cambridge University Engineering Department (CUED) dialogue act set [22] is used. The CUED set is compact and designed to cover the communicative functions needed to model simple database retrieval tasks. Its main distinction in comparison to other dialogue action sets is that it allows utterances to be decoded or labeled as distributions over alternative dialogue acts in a way that avoids the computational problems arising when allowing multiple dialogue acts per utterance. For this purpose, the scheme uses action type definitions that allow each utterance to be represented as a single act rather than a combination of acts. By allowing dialogue acts to be associated with a probability, each utterance can be labeled with a set of dialogue act hypotheses, with their

probabilities summing to one. The syntax of the CUED dialogue action set requires each act to be of the form

$$\texttt{acttype}(\underbrace{\texttt{a}=\texttt{x},\texttt{b}=\texttt{y},\ldots}_{\text{act items}})\{\text{prob}\}$$

where `prob` denotes the probability of the hypothesis. The `acttype` specifies the type of dialogue act such as `inform(...)`, `request(...)`, or `hello(...)`. The following (possibly empty) list of arguments `a = x, b = y` are referred to as dialogue act items. These items are usually slot-value pairs such as `food = French` or `drinks = wine`, but can also be individual slot names such as `addr`. For example, in the domain of tourist information described later in Section IV, the utterance *"I am looking for a cheap Chinese restaurant near the Cinema."* would be encoded as `inform(type = restaurant, pricerange = cheap, food = Chinese, near = Cinema)`, whereas *"Can you give me the address and phone number of Pizza Palace?"* would be encoded as `request(addr, phone, name = ''Pizza Palace'')`. The shorthand notation $a_u[i]$ will be used to denote the ith item of the act a_u and $\mathbb{T}(a_u)$ denotes the `acttype` of a_u.

B. State Decomposition Into Goal and Agenda

At any time t, the user is in a state $s_u \in \mathcal{S}_u$, takes action $a_u \in \mathcal{A}_u$, transitions into the intermediate state s'_u, receives machine action a_m, and transitions into the next state s''_u where the cycle restarts. Note that throughout this paper the double dash notation will be used to denote the state at time $t+1$, thus

$$s_u \rightarrow a_u \rightarrow s'_u \rightarrow a_m \rightarrow s''_u \rightarrow \cdots. \tag{1}$$

Assuming a Markovian state representation, user behavior can be decomposed into three models: $P(a_u \,|\, s_u)$ for user action selection, $P(s'_u \,|\, a_u, s_u)$ for the state transition into s'_u, and $P(s''_u \,|\, a_m, s'_u)$ for the transition into s''_u. Inspired by agenda-based approaches to dialogue management [23], [24] the user state s_u is factored into an agenda A and a goal G such that $s_u = (A, G)$ where $G = (C, R)$ consists of constraints C and requests R. During the course of the dialogue, the goal G ensures that the user behaves in a consistent, goal-directed manner. The constraints C specify the required venue, e.g., "a centrally located bar serving beer," and the requests R specify the desired pieces of information, e.g., "the name, address and phone number of the venue." Both C and R can be conveniently represented as lists of slot-value pairs, as shown in the following example:

$$C = \begin{bmatrix} \text{type} = \text{bar} \\ \text{drinks} = \text{beer} \\ \text{area} = \text{central} \end{bmatrix} \quad R = \begin{bmatrix} \text{name} = \\ \text{addr} = \\ \text{phone} = \end{bmatrix}.$$

The user agenda A is a stack-like structure containing the pending user dialogue acts that are needed to elicit the information specified in the goal. At the start of the dialogue a new goal is randomly generated using the system database and the agenda is populated by converting all goal constraints into `inform()` acts and all goal requests into `request()` acts. In addition, a `bye()` act is added at the bottom of the agenda to close the dialogue. The initial agenda A for the example introduced above would therefore be as shown below. As the dialogue progresses, the agenda is dynamically updated and acts are selected from the top of the agenda to form user acts. In the example, the user response a_u = `inform(type = bar, drinks = beer)` is generated by popping $n = 2$ items off A to give A' as shown as follows:

$$A = \begin{bmatrix} \text{inform}(\text{type} = \text{bar}) \\ \text{inform}(\text{drinks} = \text{beer}) \\ \text{inform}(\text{area} = \text{central}) \\ \text{request}(\text{name}) \\ \text{request}(\text{addr}) \\ \text{request}(\text{phone}) \\ \text{bye}() \end{bmatrix}$$

$$\longrightarrow$$

$$\text{Pop} \quad a_u = \text{inform}(\text{type} = \text{bar}, \text{drinks} = \text{beer})$$

$$A' = \begin{bmatrix} \text{inform}(\text{area} = \text{central}) \\ \text{request}(\text{name}) \\ \text{request}(\text{addr}) \\ \text{request}(\text{phone}) \\ \text{bye}() \end{bmatrix}.$$

In response to incoming machine acts a_m, new user acts are pushed onto the agenda and no longer relevant ones are removed. The agenda thus serves as a convenient way of tracking the progress of the dialogue as well as encoding the relevant dialogue history. Dialogue acts can also be temporarily stored when actions of higher priority need to be issued first, hence providing the simulator with a simple model of user memory (see Fig. 3 for a detailed illustration). When using an n-gram based approach, by comparison, such long-distance dependencies between dialogue turns are neglected unless n is set to a large value, which in turn often leads to poor generalization and unreliable model parameter estimates.

Another, perhaps less obvious, advantage of the agenda-based approach is that it enables the simulated user to take the initiative when the dialogue is corrupted by recognition errors or when the incoming system action is not relevant to the current task. The latter point is critical for training statistical dialogue managers because policies are typically learned from a random start. The "dialogue history" during the early training phase is often a sequence of random dialogue acts or dialogue states that has never been seen in the training data. In such cases, the stack of dialogue acts on the agenda enables the user model to take the initiative and behave in a goal-directed manner even when the system appears to be aimless.

C. Action Selection and State Transition Models

The decomposition of the user state s_u into a goal G and an agenda A simplifies the models for action selection and state transition. The agenda (of length N) is assumed to be ordered according to priority, with $A[N]$ denoting the top and $A[1]$ denoting the bottom item. Forming a user response is thus equivalent to popping n items off the top of the stack. Letting $a_u[i]$

Initialisation *(Generate goal constraints and requests and populate the agenda)*

$$C_0 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \end{bmatrix} \quad R_0 = \begin{bmatrix} name = \\ addr = \\ phone = \end{bmatrix} \quad A_0 = \begin{bmatrix} inform(type = bar) \\ inform(drinks = beer) \\ inform(area = central) \\ request(name) \\ request(addr) \\ request(phone) \\ bye() \end{bmatrix}$$

Sys 0 Hello, how may I help you? *(Push 0 items onto the agenda)*
Usr 1 I'm looking for a nice bar serving beer. *(Pop 2 items off the agenda)*

$$C_1' = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \end{bmatrix} \quad R_1' = \begin{bmatrix} name = \\ addr = \\ phone = \end{bmatrix} \quad A_1' = \begin{bmatrix} inform(area = central) \\ request(name) \\ request(addr) \\ request(phone) \\ bye() \end{bmatrix}$$

Sys 1 Ok, a wine bar. What price range? *(Add 1 constraint, push 2 items onto the agenda)*

$$C_2 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \\ prange = cheap \end{bmatrix} \quad R_2 = \begin{bmatrix} name = \\ addr = \\ phone = \end{bmatrix} \quad A_2 = \begin{bmatrix} negate(drinks = beer) \\ inform(prange = cheap) \\ inform(area = central) \\ request(name) \\ request(addr) \\ request(phone) \\ bye() \end{bmatrix}$$

Usr 2 No, beer please! *(Pop 1 item off the agenda)*
Sys 2 A bar serving beer, correct? *(Push 1 item onto the agenda)*

$$C_3 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \\ prange = cheap \end{bmatrix} \quad R_3 = \begin{bmatrix} name = \\ addr = \\ phone = \end{bmatrix} \quad A_3 = \begin{bmatrix} affirm() \\ inform(prange = cheap) \\ inform(area = central) \\ request(name) \\ request(addr) \\ request(phone) \\ bye() \end{bmatrix}$$

Usr 3 Yeah something cheap in the town centre. *(Pop 3 items off the agenda)*

$$C_3' = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \\ prange = cheap \end{bmatrix} \quad R_3' = \begin{bmatrix} name = \\ addr = \\ phone = \end{bmatrix} \quad A_3' = \begin{bmatrix} request(name) \\ request(addr) \\ request(phone) \\ bye() \end{bmatrix}$$

Sys 3 Murphy's on Main Sq. serves cheap beer. *(Fill 2 requests, delete req. acts for filled slots)*

$$C_4 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \\ prange = cheap \end{bmatrix} \quad R_4 = \begin{bmatrix} name = Murphy's \\ addr = Main\ Sq \\ phone = \end{bmatrix} \quad A_4 = \begin{bmatrix} request(phone) \\ bye() \end{bmatrix}$$

Usr 4 And what's the phone number? *(Pop 1 item off the agenda)*
Sys 4 The number is 796 69 94. *(Fill 1 request)*

$$C_5 = \begin{bmatrix} type = bar \\ drinks = beer \\ area = central \\ prange = cheap \end{bmatrix} \quad R_5 = \begin{bmatrix} name = Murphy's \\ addr = Main\ Sq \\ phone = 7966994 \end{bmatrix} \quad A_5 = \begin{bmatrix} bye() \end{bmatrix}$$

Usr 5 Thanks, goodbye! *(Pop 1 item off the agenda)*

Fig. 3. Sample dialogue showing the state of the user goal and agenda.

denote the ith dialogue act item in a_u, the resulting user act is formed as follows:

$$a_u[i] := A[N - n + i] \quad \forall i \in [1..n], \quad 1 \leq n \leq N. \tag{2}$$

Using $A[N - n + 1..N]$ as a shorthand for the top n items on A, the action selection model becomes

$$P(a_u \mid s_u) = \delta(a_u, A[N - n + 1..N])P(n \mid A, G) \tag{3}$$

where the Kronecker delta function $\delta(p, q)$ is 1 iff $p = q$ and zero otherwise. This implies that the user response a_u can only be generated in state $s_u = (A, G)$ if a_u can be popped off A. The probability of the corresponding pop operation depends on the number n of popped items and is conditioned on A and G.

The state transition models $P(s_u' \mid a_u, s_u)$ and $P(s_u'' \mid a_m, s_u')$ are rewritten as follows. Letting A' denote the agenda after popping off a_u and using $N' = N - n$ to denote the size of A', we have

$$A'[i] := A[i] \quad \forall i \in [1..N']. \tag{4}$$

Using this definition of A' and assuming that the goal remains constant when the user executes a_u, the first state transition depending on a_u is entirely deterministic as it only involves popping a given number of items off the agenda

$$\begin{aligned} P(s_u' \mid a_u, s_u) &= P(A', G' \mid a_u, A, G) \\ &= \delta(A', A[1..N'])\delta(G', G). \end{aligned} \tag{5}$$

The second state transition based on a_m is decomposed into *goal update* and *agenda update* steps

$$P(s''_u \,|\, a_m, s'_u) = \underbrace{P(A'' \,|\, a_m, A', G'')}_{\text{agenda update}} \underbrace{P(G'' \,|\, a_m, G')}_{\text{goal update}} \quad (6)$$

and the model parameter set θ can now be summarized as

$$\theta = \{P(n \,|\, A, G), P(A'' \,|\, a_m, A', G''), P(G'' \,|\, a_m, G')\}. \quad (7)$$

Representations for the update steps shown on the RHS of (6) are discussed in the following sections.

III. Model Parameter Estimation

A. User State as a Hidden Variable

Estimating the parameters of the action selection and state transition models is nontrivial, since the goal and agenda states are not observable in the training data. As explained in the introduction to the paper, previous work on the state-based approach to statistical user simulation [11], [13], [14] has circumvented the problem of the unobservable user state by labeling training data with dialogue state information and conditioning user output on the observable dialogue state. While this simplifies the training process, providing the necessary annotation is prone to error and requires considerable effort.

The parameter estimation approach presented here avoids the need for dialogue state annotation by modeling the observable user and machine dialogue acts in terms of a *hidden* sequence of agendas and user goal states. More formally, the dialogue data $\mathcal{D}$ containing dialogue turns 1 to T

$$\mathcal{D} = \{\mathbf{a_u}, \mathbf{a_m}\} = \{a_{m,1}, a_{u,1} \ldots, a_{m,T}, a_{u,T}\} \quad (8)$$

is modeled in terms of latent variables $X = \{\mathbf{A}, \mathbf{G}\}$ where $\mathbf{A} = \{A_1, A'_1, \ldots, A_T, A'_T\}$ and $\mathbf{G} = \{G_1, G'_1, \ldots, G_T, G'_T\}$. Collecting the results from Section II, noting that from (5) the choice of n deterministically fixes A', and using A''_t to denote A_{t+1}, the joint probability can be expressed as

$$P(X, \mathcal{D}) = \prod_{t=1}^{T-1} P(n_t \,|\, A_t, G_t) \times P(A''_t \,|\, a_{m,t}, A'_t, G''_t) P(G''_t \,|\, a_{m,t}, G'_t) P(a_{m,t}). \quad (9)$$

The goal is to learn maximum likelihood (ML) values for the model parameter set θ such that the log likelihood $\mathcal{L}(\theta) = \log P(\mathcal{D} \,|\, \theta)$ is maximized

$$\theta_{\mathrm{ML}} = \arg\max_\theta \mathcal{L}(\theta) = \arg\max_\theta \log P(\mathcal{D} \,|\, \theta) = \arg\max_\theta \log \sum_X P(X, \mathcal{D} \,|\, \theta). \quad (10)$$

Due to the presence of the sum over the latent variables in (10), the direct optimization of $\mathcal{L}(\theta)$ is not possible, however, an iterative expectation-maximization (EM)-based approach can be used to find a (local) maximum of the latent variable model likelihood. As shown in [25], this involves maximizing the auxiliary function

$$Q\left(\theta, \theta^{(k-1)}\right) = \sum_X P\left(X, \mathcal{D} \,\middle|\, \theta^{(k-1)}\right) \log P(X, \mathcal{D} \,|\, \theta). \quad (11)$$

and leads to the parameter re-estimation formulas given in (12)–(14), shown at the bottom of the page.

B. Agenda Updates as a Sequence of Push Actions

Implementing the latent variable approach described above in a tractable fashion is not straightforward. The sizes of the user and machine dialogue act sets $\mathcal{A}_u$ and $\mathcal{A}_m$ used for the experiments presented in this paper are $|\mathcal{A}_u| \approx 10^3$ and $|\mathcal{A}_m| \approx 10^3$. Using typical values for the goal specifications used in previous SDS user trials [26], the size of the goal state space can be estimated as $|\mathcal{G}| \approx 10^7$. The size of the agenda state space $\mathcal{A}$ depends on the number of unique user dialogue acts $|\mathcal{A}_u|$ as defined above and the maximum number of user dialogue acts on the agenda. The maximum length of the agenda is a design choice, but it is difficult to simulate realistic dialogues unless it is set to at least 8. As shown in [25], $\mathcal{A}$ thus comprises the vast

$$\hat{P}(n \,|\, A, G) = \frac{\sum_t P\left(A_t = A, G_t = G \,\middle|\, \mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)}\right) \delta(n_t, n)}{\sum_t P\left(A_t = A, G_t = G \,\middle|\, \mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)}\right)} \quad (12)$$

$$\hat{P}(A'' \,|\, a_m, A', G'') = \frac{\sum_t P\left(A''_t = A'', A'_t = A', G''_t = G'' \,\middle|\, \mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)}\right) \delta(a_{m,t}, a_m)}{\sum_t P\left(A'_t = A', G''_t = G'' \,\middle|\, \mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)}\right) \delta(a_{m,t}, a_m)} \quad (13)$$

$$\hat{P}(G'' \,|\, a_m, G') = \frac{\sum_t P\left(G''_t = G'', G'_t = G' \,\middle|\, \mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)}\right) \delta(a_{m,t}, a_m)}{\sum_t P\left(G'_t = G' \,\middle|\, \mathbf{a_u}, \mathbf{a_m}, \theta^{(k-1)}\right) \delta(a_{m,t}, a_m)} \quad (14)$$

number of $|\mathcal{A}| \approx 10^{20}$ potential agenda states and the number of parameters needed to model $P(A'' \,|\, a_m, A', G'')$ is of the order

$$|\mathcal{A} \times \mathcal{A}_m \times \mathcal{A} \times \mathcal{G}| \approx 10^{50}. \tag{15}$$

These estimates show that when no restrictions are placed on A'', the space of possible state transitions from A' to A'' is vast. While the figures given above are estimates specific to the agenda model and dialogue domain presented here, similar tractability problems are likely to arise with any user model training algorithm where the true state is assumed to be hidden—regardless of the state representation. The agenda-based state representation has the significant advantage that each state can be assumed to be derived from the previous state, and that each transition entails only a limited number of well-defined atomic operations. As will be shown in the remainder of this section, this allows tractability to be achieved without unduly limiting the expressive power of the model.

More specifically, the agenda transition from A' to A'' can be viewed as a sequence of push-operations in which dialogue acts are added to the top of the agenda. In a second "clean-up" step, duplicate dialogue acts, "empty" acts, and unnecessary `request()` acts for already filled goal request slots must be removed but this is a deterministic procedure so that it can be excluded in the following derivation for simplicity. Considering only the push-operations, the items 1 to N' at the bottom of the agenda remain fixed and the update model can be rewritten as follows:

$$\begin{aligned} &P(A'' \,|\, a_m, A', G'') \\ &\quad = P(A''[1..N'], A''[N'+1..N''] \,|\, a_m, \\ &\qquad A'[1..N'], G'') \\ &\quad = \delta(A''[1..N'], A'[1..N']) \\ &\qquad \times P(A''[N'+1..N''] \,|\, a_m, G''). \end{aligned} \tag{16}$$

The second term on the RHS of (16) can now be further simplified by assuming that every dialogue act item (slot-value pair) in a_m triggers one push-operation. This assumption can be made without loss of generality, because it is possible to push an "empty" act (which is later removed) or to push an act with more than one item. The advantage of this assumption is that the known number M of items in a_m now determines the number of push-operations. Hence, $N'' = N' + M$ and

$$\begin{aligned} &P(A''[N'+1..N''] \,|\, a_m, G'') \\ &\quad = P(A''[N'+1..N'+M] \,|\, a_m[1..M], G'') \qquad (17) \\ &\quad = \prod_{i=1}^{M} P(\underbrace{A''[N'+i]}_{a_{\text{push}}} \,|\, \underbrace{a_m[i]}_{a_{\text{cond}}}, G''). \qquad (18) \end{aligned}$$

The expression in (18) shows that each item $a_m[i]$ in the system act triggers one push operation, and that this operation is conditioned on the goal. For example, given that the item $\mathtt{x} = \mathtt{y}$ in $a_m[i]$ violates the constraints in G'', one of the following might be pushed onto A'': `negate()`, `inform(x = z)`, `deny(x = y, x = z)`, etc.

Let $a_{\text{push}} \in \mathcal{A}_u$ denote the pushed act $A''[N'+i]$ and $a_{\text{cond}} \in \mathcal{A}_m$ denote the conditioning dialogue act containing the single dialogue act item $a_m[i]$. Omitting the Kronecker delta function in (16), the agenda update step then reduces to the repeated application of a *push transition model* $P(a_{\text{push}} \,|\, a_{\text{cond}}, G'')$. The number of parameters needed to model $P(a_{\text{push}} \,|\, a_{\text{cond}}, G'')$ is of the order

$$|\mathcal{A}_u \times \mathcal{A}_m \times \mathcal{G}| \approx 10^{13}. \tag{19}$$

While still large, this number is significantly smaller then the number of parameters needed to model unrestricted transitions from A' to A'' [cf. (15)].

C. Summary Space Model for Push Transitions

To further reduce the size of the model parameter set and achieve a tractable estimation of $P(a_{\text{push}} \,|\, a_{\text{cond}}, G'')$, it is useful to introduce the concept of a "summary space," as has been previously done in the context of dialogue management [27]. First, a function ϕ is defined for mapping the machine dialogue act $a_{\text{cond}} \in \mathcal{A}_m$ and the goal state $G'' \in \mathcal{G}$ from the space of machine acts $\mathcal{A}_m$ and goal states $\mathcal{G}$ to a smaller summary space $\mathcal{Z}_{\text{cond}}$ of "summary conditions" as follows:

$$\phi : \mathcal{A}_m \times \mathcal{G} \mapsto \mathcal{Z}_{\text{cond}} \quad \text{with} \quad |\mathcal{A}_m \times \mathcal{G}| \gg |\mathcal{Z}_{\text{cond}}|. \tag{20}$$

For example, all system acts a_m which attempt to `confirm()` a slot value pair $\mathtt{a} = \mathtt{x}$ that violates an existing user goal constraint $\mathtt{a} = \mathtt{y}$ are mapped to the summary condition $\mathrm{ReceiveConfirmAXnotOk}[a = x]$.

Second, a "summary push action" space $\mathcal{Z}_{\text{push}}$ is defined, which groups real user dialogue acts into a smaller set of equivalence classes. Using a function ω, summary push actions are mapped back to "real" dialogue acts

$$\omega : \mathcal{Z}_{\text{push}} \mapsto \mathcal{A}_u \quad \text{with} \quad |\mathcal{Z}_{\text{push}}| \ll |\mathcal{A}_u|. \tag{21}$$

The summary push action *PushNegateAY*, for example, maps to the real dialogue act $\mathtt{negate}(\mathtt{a} = \mathtt{y})$ (``No, I want $a = y$!''). Note that both mappings ϕ and ω are deterministic and need to be handcrafted[1] by the system designer as will be discussed in more detail below.

Letting $z_{\text{push}} \in \mathcal{Z}_{\text{push}}$ and $z_{\text{cond}} \in \mathcal{Z}_{\text{cond}}$, agenda state transitions can now be modeled in summary space using

$$P(a_{\text{push}} \,|\, a_{\text{cond}}, G'') \approx P(z_{\text{push}} \,|\, z_{\text{cond}}) \tag{22}$$

where $z_{\text{cond}} = \phi(a_{\text{cond}}, G'')$ and $a_{\text{push}} = \omega(z_{\text{push}})$. For the experiments presented in this paper, roughly 30 summary conditions and 30 summary push actions were defined (see examples and discussion below). The total number of parameters $P(z_{\text{push}} \,|\, z_{\text{cond}})$ needed to model agenda state transitions is therefore $|\mathcal{Z}_{\text{cond}} \times \mathcal{Z}_{\text{push}}| \approx 900$, i.e., small enough to be estimated on real dialogue data.

Fig. 4 shows a simplified example illustrating the summary space technique for agenda updates. The incoming machine act $a_m = \mathtt{confreq}(\mathtt{p} = \mathtt{q}, \mathtt{r})$ in this example is an implicit confirmation of the slot-value pair $\mathtt{p} = \mathtt{q}$ and a request for the slot $\mathtt{r}$. The update step proceeds as follows.

[1]Linear function approximation may provide a more principled approach for mapping to and from summary space (see [28])

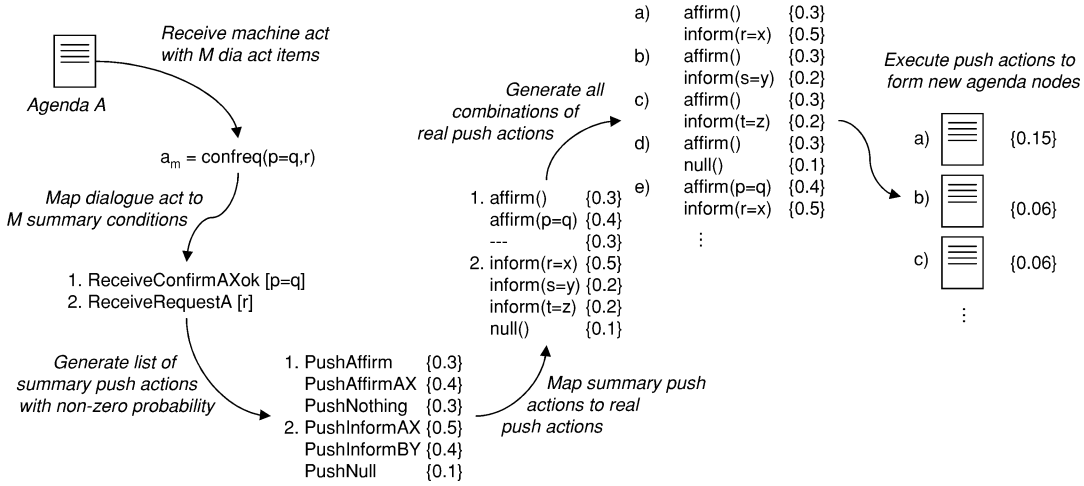

Fig. 4. Simplified example illustrating the summary space technique for agenda updates.

1) Based on the current state of the goal (not shown here), the first step is to map each dialogue act item (slot-value pair) to a summary condition z_{cond}. Given that the confirmation $\texttt{p} = \texttt{q}$ in the example does not violate any of the constraints in the user goal, it is mapped to $\text{ReceiveConfirmAXok}[p = q]$. The request for `r` is mapped to $\text{ReceiveRequestA}[r]$.
2) A list of summary push actions z_{push}, each with probability $P(z_{\text{push}} \mid z_{\text{cond}})$, is now generated for each summary condition z_{cond}. A (shortened) list of examples is shown in the figure. The summary push action *PushInformAX*, for instance, implies that an `inform()` act with the requested slot (in this case `r`) is pushed onto the agenda. Note that summary push actions with zero probability can be discarded at this point.
3) The summary push actions are now mapped to real push actions. This is a 1-to-1 mapping for most summary push actions, but some summary push actions can map to several real push actions. This is illustrated in the figure by the summary push action *PushInformBY*, which implies that the corresponding real push action is an `inform()` dialogue act containing some slot-value pair $B = Y$ other than the requested slot, in this case $\texttt{s} = \texttt{y}$ or $\texttt{t} = \texttt{z}$. In such cases, the probability mass is split evenly between the real push actions for a summary push action, as shown in the figure.
4) Using one real push action from each summary condition, a list of all possible combinations of push actions is now generated. Each combination represents a series of dialogue acts to be pushed onto the agenda. As shown in the figure, each combination is used to create a new agenda. The transition probability is computed as the product of the real push actions that were used to make the transition.

The set of summary conditions $\mathcal{Z}_{\text{cond}}$ and summary push actions $\mathcal{Z}_{\text{push}}$ is domain-independent and independent of the number of slots and database entries, hence allowing the method to scale to more complex problem domains and larger databases. The definition of $\mathcal{Z}_{\text{cond}}$ and $\mathcal{Z}_{\text{push}}$, and the implementation of the handcrafted mappings $\phi : \mathcal{A}_m \times \mathcal{G} \mapsto \mathcal{Z}_{\text{cond}}$ and $\omega : \mathcal{Z}_{\text{push}} \mapsto \mathcal{A}_u$ requires detailed knowledge of the dialogue act set and basic familiarity with user behavior in slot-filling dialogue scenarios. The handcrafted mappings, however, are not dependent on the specific domain and application—only the general class of application.

A systematic approach to the design process can be taken by defining a single summary condition for each machine act type, e.g., *ReceiveHello* for `hello()`, *ReceiveInformAX* for $\texttt{inform}(\texttt{a} = \texttt{x})$, *ReceiveConfirmAX* for $\texttt{confirm}(\texttt{a} = \texttt{x})$. Similarly, a single summary push action is defined for each user act type, e.g., *PushHello* for `hello()`, *PushRequestA* for $\texttt{request}(\texttt{a})$, *PushBye* for `bye()`. This ensures that the set of summary conditions covers the space of possible machine acts and goal states, and that the set of summary push actions covers the space of possible user acts. In the case of the CUED dialogue act set described in Section II-A and [29], this results in a set of approximately 15 summary conditions and 15 summary push actions. $\mathcal{Z}_{\text{cond}}$ can then be further refined by "splitting" summary conditions: *ReceiveConfirmAX* for example, can be split into *ReceiveConfirmAXok* and *ReceiveConfirmAXnotOk* to distinguish the two cases where the given slot-value pair $\texttt{a} = \texttt{x}$ matches/violates the existing user goal constraint for the slot `a`. Similarly, $\mathcal{Z}_{\text{push}}$ can be refined by choosing a more fine grained set of summary push actions: *PushRequestA* for example, can be split into *PushRequestForUnknownSlotA* and *PushRequestForFilledSlotA*. This process typically requires some trial-and-error, but the time needed to make these iterative refinements[2] is still insignificant compared to the effort involved in annotating a dialogue corpus with state specific information. Moreover, the result is reusable across a class of slot-filling applications with the same dialogue act set.

D. Representing Agenda State Sequences

Given the vast size of the agenda state space, the direct enumeration of all states in advance is clearly intractable. The actual number of states needed to model a particular dialogue act sequence, however, is much smaller, since agenda transitions are restricted to push/pop operations and conditioned on dialogue

[2]For the system described here, these refinements required approximately 1 day.

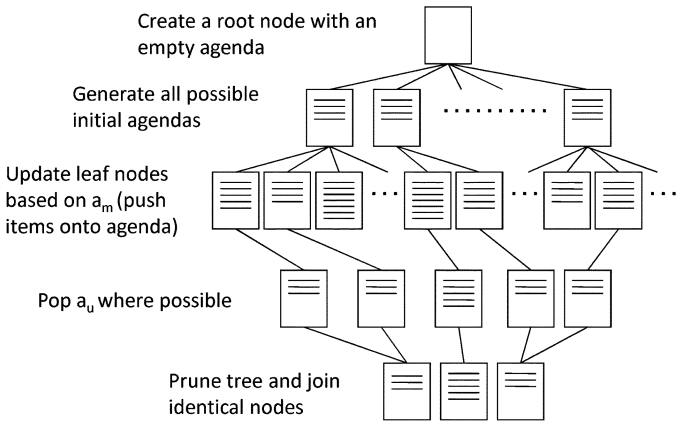

Fig. 5. Tree-based method for representing state sequences.

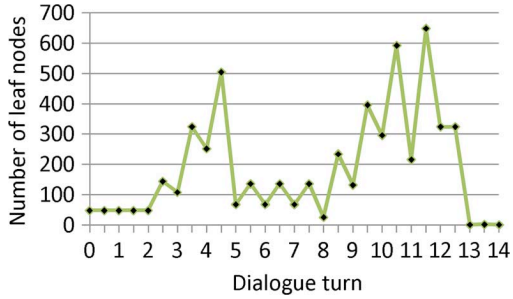

Fig. 6. Number of agenda tree leaf nodes after each observation during a training run.

context. The training algorithm can exploit this by generating state-sequences on-the-fly, and discarding any state sequence X for which $P(X, \mathcal{D} \,|\, \theta) = 0$.

A suitable implementation for this is found in the form of a dynamically growing tree, which allows agenda states to be represented as tree-nodes and state transitions as branches. The tree is initialized by creating a root node containing an empty agenda and then populating the agenda according to the goal specification as explained in Section II-B. Since the initial ordering of dialogue acts on the agenda is unknown, all permutations of constraints and requests must be created as shown in Fig. 5.

Following initialization, the dialogue is "parsed" by growing the tree and creating branches for all possible state sequences. Updates based on a machine dialogue act a_m involve mapping each item in a_m to its corresponding summary condition z_{cond} using the function ϕ. For each z_{cond} a list of summary push actions z_{push} is generated, discarding cases where $P(z_{\text{push}} \,|\, z_{\text{cond}}) = 0$. The summary push actions are then mapped back to real push actions using ω and used to create new agendas which are attached to the tree as new branches. The probability of the transition/branch is computed as the product of the probabilities of the real push actions (cf. Fig. 4 in Section III-C). The leaf nodes are then cleaned up in a deterministic procedure to remove empty and duplicate dialogue acts, to delete all dialogue acts below a `bye()` act, and to remove all requests for items that have already been filled in the user goal.

In the next step, the tree is updated based on the observed user act a_u. This part simplifies to popping a_u from the top of the agenda wherever this is possible. Agendas which do not allow a_u to be popped off represent states with zero probability and can be discarded. In all other cases, a new node with the updated agenda is attached to the tree. The branch is marked as a pop-transition and its probability is computed based on the number of items popped.

Once the update based on a_u is completed, the tree is pruned to reduce the number of nodes and branches. First, all branches which were not extended during the dialogue turn, i.e., branches where a_u could not be popped off the leaf node agenda, are removed. All remaining branches represent possible sequences of agenda states with nonzero probability for the dialogue acts seen so far. In a second step, a more aggressive type of pruning can be carried out by removing all branches which do not have a given minimum leaf node probability. After pruning, the size of the tree is further reduced by joining nodes with identical agendas.

Fig. 6 shows the number of agenda tree leaf nodes during a typical training episode on a sample dialogue. As explained above, the tree is extended for each machine dialogue act and 1 or more new nodes are attached to each tree branch, so that the number of leaf nodes stays constant or increases. Pop operations are then performed where possible, the tree is pruned and identical nodes are joined so that the number stays constant or decreases. At the end of the dialogue, only a single leaf node with an empty agenda remains.

E. Action Selection and Goal Update Model

The action selection and goal update models experience similar tractability problems as the agenda update model, but in both cases a straightforward solution was found to produce satisfactory results. To simplify the action selection model $P(n \,|\, A, G)$, the random variable n can be conditioned on the type of dialogue act of the top item on the agenda stack. Letting $\mathbb{T}(\cdot)$ denote the type of a dialogue act, this may be expressed as

$$P(n \,|\, A, G) = P(n \,|\, \mathbb{T}(A[N])). \tag{23}$$

The probability distribution $P(n \,|\, \mathbb{T}(A[N]))$ over small integer values for n (typically in the range from 0 to 6) can then be estimated directly from dialogue data by obtaining frequency counts of the number of dialogue act items in every user act.

The goal update model $P(G'' \,|\, a_m, G')$ is decomposed into separate update steps for the constraints and requests. Assuming that R'' is conditionally independent of C' given C'' it is easy to show that

$$P(G'' \,|\, a_m, G') = P(R'' \,|\, a_m, R', C'')P(C'' \,|\, a_m, R', C'). \tag{24}$$

The two update steps can be treated separately and implemented deterministically using two rules. 1) If R' contains an empty slot `u` and a_m is a dialogue act of the form $\texttt{inform}(\texttt{u} = \texttt{v}, \texttt{r} = \texttt{s}, \ldots)$, then R'' is derived from R' by setting $\texttt{u} = \texttt{v}$ given that no other information in a_m violates any constraints in C''. 2) If a_m contains a request for the slot `x`, a new constraint $\texttt{x} = \texttt{y}$ or $\texttt{x} = \texttt{dontcare}$ is added to C' to form C''. Note that this does not imply that the user necessarily responds to a system request

for any slot x, since the agenda update model does not enforce a corresponding user dialogue act to be issued.

The goal update model implementation described here allows the user goal to change over the course of a dialogue, but restricts the space of possible goal state transitions to deterministic updates. This simplifies model parameter estimation because the sequence of goal states can be directly inferred from the observable sequence of dialogue acts. However, it limits the user simulator to dialogue tasks where the user does not relax or modify its constraints. The behavior of real users in situations where the desired venue does not exist, for example, cannot always be covered by the current implementation. The deterministic update model is also not trainable, meaning that the probability of different goal state transitions cannot be learned from data.

F. Applying the Forward/Backward Algorithm

Using the summary space mapping for agenda transitions and simplifying assumptions for the goal update and action selection model described above, the parameter update equation set defined by (12)–(14) reduces to a single equation

$$\hat{P}(z_{\text{push}} \mid z_{\text{cond}}) = \frac{\sum_k P(z_{\text{push},k} = z_{\text{push}}, z_{\text{cond},k} = z_{\text{cond}} \mid \mathbf{a_u}, \mathbf{a_m}, \theta)}{\sum_k P(z_{\text{cond},k} = z_{\text{cond}} \mid \mathbf{a_u}, \mathbf{a_m}, \theta)}. \tag{25}$$

Note that k is used here rather than t, since every dialogue turn t involves two state transitions, and there are hence $K = 2T$ observations and update steps. The parameter update equation can now be efficiently implemented by applying the forward/backward algorithm, as shown in [25].

IV. Evaluation Method

A. Domain Specification and Experimental Setup

All experiments presented in this paper are set in the tourist information domain. The user is assumed to be visiting a fictitious town called "Jasonville," and requests the help of the dialogue system to find a particular hotel, bar or restaurant according to a given set of constraints. For example, the user might be looking for a "cheap Chinese restaurant near the Main Square," "a 3-star hotel on the riverside," or a "wine bar playing Jazz music." Once a suitable venue has been identified, the user may request further information about this venue such as the address or phone number. All of the venues have a unique name and are described using slot-value pairs such as type = hotel, pricerange = moderate, stars = 3, area = riverside, etc. In total, the database contains 31 entries, each of which is described by up to 13 slots, with each slot taking one out of approximately five to ten possible values.

The evaluation setup used in this paper is illustrated in Fig. 7 and the remainder of this paper roughly follows the pictured sequence of experiments. First, user model parameter estimation was carried out on a training dataset consisting of 160 dialogues from the Jasonville domain. These were recorded with a prototype dialogue manager [30] and 40 human subjects, each of whom completed four dialogues, as reported in [26]. The utterances in the corpus were transcribed and annotated according to the CUED dialogue act taxonomy (as described in Section II-A). No dialogue state specific annotation was added. In total, the corpus contains 1804 user utterances and 5765 words. A Word Accuracy (WAcc) of 75.15% and a Semantic Accuracy (SAcc) of 61.35% was computed based on the number of substitutions, insertions, and deletions, as defined by [31].

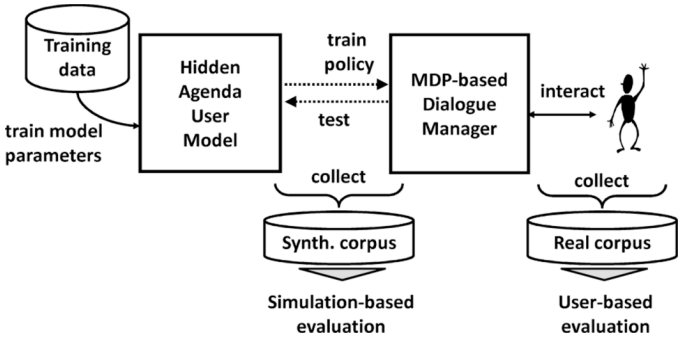

Fig. 7. Experimental setup.

As shown in Fig. 7, an MDP-based dialogue manager was implemented and trained via interactions with the simulated user model to learn a dialogue policy (Section IV-B). A rule-based dialogue manager was also implemented to provide a competitive baseline (Section IV-C). System evaluation was then carried out through simulation experiments (Section V) and a study with real users (Section VI). In related work not presented here, experiments were also carried out with a prototype POMDP-based dialogue manager [26]. Additional evaluation results comparing the statistical properties of simulated and real data can further be found in [32].

B. MDP Policy Training

To evaluate the agenda-based user model in the context of a working dialogue system, a state-of-the-art MDP-based dialogue manager was implemented. Its state space representation covers the status (UNKNOWN, FILLED, or CONFIRMED) and value of each slot (type, food, pricerange, etc.). In addition to the slot status variables, a number of other flags and variables are maintained. This includes a list of the slots requested by the user in the most recent user dialogue act (e.g., phone, addr, etc.), as well as a list of all currently pending system and user confirmation requests. The state representation also tracks the number of database entities matched by the current set of slot-values and classifies the confidence of the last user act as either very low, low, medium, high, or very high. In total, the state space definition allows for 57 600 unique states.

For the action set, nine high-level "summary" acts are defined: GREET, REQUEST, IMP-CONFIRM, EXP-CONFIRM-ONE, EXP-CONFIRM-ALL, OFFER, INFORM, REQMORE, and GOODBYE.[3] These specify the broad action class selected by the dialogue manager and reduce the number of state-action combinations that need to be explored during policy training. A handcrafted mapping is defined to map from DM summary acts to real dialogue acts according to the current dialogue state. For example, a GREET act is mapped to a hello() act, a REQUEST act is

[3]Note that this set of dialogue manager actions is very similar to the summary action set previously used by [29].

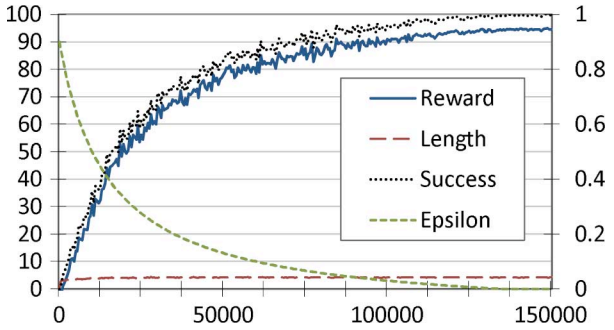

Fig. 8. MDP policy training graph. Each point indicates the average value for the last 1000 training dialogues. As shown, ϵ decreases over time from 0.9 to 0.0. The success rate converges to 99.8%, the length of the dialogues converges to approximately 4.3 turns and the average reward converges to 94.5 points.

mapped to a `request()` act for the next UNKNOWN slot, and a EXP-CONFIRM-ALL act is mapped to a `confirm()` act including all FILLED slot-value pairs.

The reward function used for all experiments awards 100 points for successful dialogue completion. This includes recommending a correct venue and satisfying all additional requests for extra information (e.g., the address and phone number), zero points are awarded otherwise. A correct venue is defined as a venue that matches all of the constraints on the user's goal. A 1 point penalty is deducted for each dialogue turn to encourage efficient dialogue completion. Letting n denote the number of dialogue turns, the reward function may be expressed as

$$\text{Reward} = \begin{cases} 100 - n, & \text{if completed successfully} \\ 0 - n, & \text{otherwise.} \end{cases} \tag{26}$$

While more sophisticated reward functions are easily conceivable, the simple choice made here illustrates that it is not necessary to guide the learning algorithm by providing rewards for partial completion.

For policy training, a standard Monte Carlo control algorithm for reinforcement learning was implemented [5]. The algorithm estimates the value of each summary act in each state. An ϵ-greedy approach is used to balance exploration and exploitation: At each dialogue turn, a random action is selected with probability ϵ (which decreases over time), while the best action has the probability $1 - \epsilon$. During each training dialogue, the algorithm records all state-action pairs visited by the DM. Once the dialogue has completed, the reward signal is computed and used to update all visited state-action pairs. A plot of the training progress over 150 000 dialogues is shown in Fig. 8.

C. Rule-Based DM Baseline

In order to benchmark the trained policies against a standard handcrafted dialogue manager, a rule-based dialogue policy was designed by hand. To provide a fair comparison, this baseline policy uses the same state representation and summary action set as the MDP dialogue manager and thus has access to the same state information as the trained policy. Great care was taken to fine tune the rule-based DM to optimize the reward function given in (26).

The handcrafted policy starts the dialogue with a GREET summary action to welcome the user and then issues a REQUEST action to narrow down the number of database entries matching the users query. If the constraints provided by the user do not match any of the database entries, then the system immediately issues an OFFER act which is mapped to a system utterance of the form "There is no 5-star hotel in the moderate pricerange." If there is exactly one matching item, the system will make an OFFER matching the given constraints, e.g., "The Ville Hotel is a nice 5-star hotel in the moderate price range."

If the information provided by the user matches more than one item in the database, the DM keeps asking for further information as long as there are four or more matching items in the database before proceeding to make an OFFER. This was found to produce optimal results with the given reward function. By using IMP-CONFIRM actions the FILLED slots can be implicitly confirmed while simultaneously asking for an UNKNOWN slot. If however, there has been no "progress" for 2 turns (i.e., if the user repeatedly says "I don't care" in response to a query, or if the user repeats information he/she has provided before) then the system makes an OFFER instead of requesting further information. If sufficient information is available for making a recommendation and `null()` or `silence()` user acts are received in two consecutive turns (possibly due to poor recognition performance) the system also makes an OFFER.

Once an offer has been made, the task specification typically requires users to ask for further information about the recommended venue (e.g., "What is the address of that place?"). The system then uses INFORM acts to provide the requested information. Users may also attempt to confirm pieces of information ("Is that in the moderate price range?"). When the user has completed the task and does not request or provide further information, then the system issues a REQMORE act to say "Can I help you with anything else?". If the user response to this is "No," then the system closes the dialogue by saying GOODBYE.

V. Simulation-Based Evaluation Results

A. Cross-Model Evaluation With a Handcrafted Baseline Simulator

This paper adopts a cross-model approach to evaluation whereby policies are trained and tested on different user models, as previously suggested by [33]. To provide a competitive baseline for comparisons with the trained agenda model, a handcrafted simulator [34] was designed to reproduce user behavior for the given tourist information domain as naturally as possible. The handcrafted baseline simulator was developed and refined over several months and used extensively for training a variety of MDP- and POMDP-based statistical dialogue managers. It uses the same goal and agenda-based state representation as the trained agenda simulation model and therefore has access to the same state information. In contrast to the trained model, however, the state transition and action selection models are implemented using a manually defined set of rules for each type of system dialogue act a_m. These rules govern the behavior of the simulated user and are largely deterministic. Where possible, some degree of randomness is introduced to allow for a greater variety in user behavior. For example, if the type of a_m is `inform()` and an act item `x = y` violates the constraints in the user goal G'', then one of the

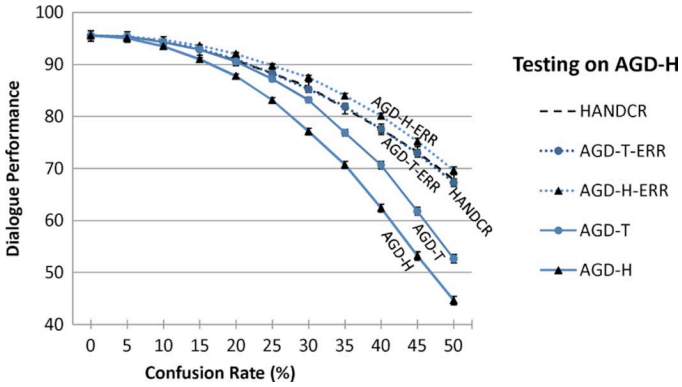

Fig. 9. Simulation-based evaluation on the handcrafted simulator AGD-H. Each data point is based on 15 000 simulated dialogues. Error bars indicate 99% confidence intervals.

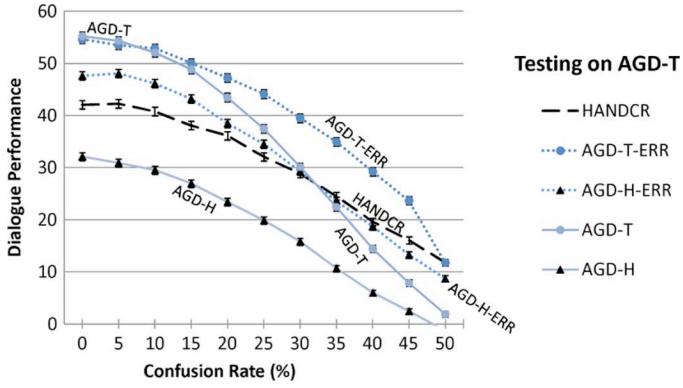

Fig. 10. Simulation-based evaluation on the trained simulator AGD-T. Each data point is based on 15 000 simulated dialogues. Error bars indicate 99% confidence intervals.

following is pushed onto A'': `negate()`, `inform(x = z)`, or `deny(x = y, x = z)`. Previously published results [34] show that the generated user responses are sufficiently realistic to successfully train a dialogue policy which works competitively when tested on real users.

B. Overview of Trained and Tested Policies

MDP policies were trained using both the handcrafted simulator and the trained agenda model. In each case, one policy was trained under noise-free conditions and one policy was trained under simulated noise conditions with a 25% semantic error rate [6]. Policy evaluation experiments were then run with the following five policies:

- the handcrafted rule-based policy (HANDCR);
- the policy learned with the handcrafted agenda simulator (AGD-H);
- the policy learned with the handcrafted agenda simulator under noisy conditions (AGD-H-ERR);
- the policy learned with the trained agenda simulator (AGD-T);
- the policy learned with the trained agenda simulator under noisy conditions (AGD-T-ERR).

Each of the five policies was tested on the handcrafted AGD-H simulator (Fig. 9) and the trained AGD-T simulator (Fig. 10). In common with previous work by Lemon and Liu [35] which evaluates policies under "low" and "high" noise settings, the results presented here show dialogue performance over a range of simulated error conditions. These were achieved by setting a *confusion rate* in the simulator ranging from 0 to 50%. Subsequent evaluation showed that these simulator confusion rates resulted in an actual semantic error rate which varied linearly from 0 to 30%. At each step, 15 000 dialogues were generated and dialogue performance was computed using the reward function defined by (26) in Section IV-B.

C. Evaluation on the Handcrafted Simulator (AGD-H)

Fig. 9 shows the performance of the five policies when tested on the handcrafted agenda-based simulator (AGD-H). As can be seen, the handcrafted policy (labeled HANDCR) performs very well. The four trained policies, all match the performance of the handcrafted policy at 0% error rate, but cannot outperform it. This demonstrates the competitiveness of the handcrafted policy and illustrates the large amount of manual effort invested to optimize this policy for the AGD-H simulator.

As the error rate increases, both of the policies trained without noise drop off rapidly. It is interesting to note that the policy learned with the trained AGD-T user simulator decreases more slowly than the policy learned with the handcrafted simulator AGD-H. This shows that the AGD-T simulator produces more varied behavior during training and thus leads to a slightly more robust policy.

By training under simulated noise conditions, the performance of the trained policies can be improved dramatically. As shown in Fig. 9, the policies trained with noise (AGD-H-ERR, AGD-T-ERR) outperform those trained without noise (AGD-H, AGD-T) by more than ten points at high error rates. One may also note that the policies trained under noisy conditions match or slightly outperform the handcrafted policy HANDCR, even at high error rates. This means that the learning process automatically discovers the same optimal policy settings which were found when manually designing the handcrafted policy. A final interesting detail to note here is that the policy trained with the AGD-H-ERR simulator performs slightly better than the AGD-T-ERR simulator when testing on the HDC simulator at high error rates. This shows that the best results are achieved when training and testing conditions match perfectly and confirms that the learning algorithm works.

D. Evaluation on the Trained Simulator (AGD-T)

Fig. 10 shows the performance of the five policies when tested on the AGD-T simulator. The first thing to note is that the AGD-T simulator is a much harder test case: an absolute decrease of approximately 40 points in dialogue performance is observed. Again, the benefits of training in noisy conditions with a statistical user model are evident: The AGD-T-ERR policy outperforms the AGD-T policy, and the AGD-H-ERR policy outperforms the AGD-H policy. Also, the AGD-T policy outperforms the AGD-H policy, and the AGD-T-ERR policy outperforms the AGD-H-ERR policy.

An interesting detail can be spotted when comparing the dialogue performance of the AGD-T and AGD-T-ERR policy at 0% error rate. Here the policy trained without noise performs

slightly better than the policy trained with noise. This is plausible, because performance should be highest when the training and testing conditions match. It shows that training under noisy conditions may require a tradeoff in that while the policy will work better when noise levels are high, it may be slightly worse when noise levels are low.

Another interesting finding shown in the figure is that when testing on the AGD-T model the handcrafted policy (HANDCR) works significantly better than the policies trained with the AGD-H and the AGD-H-ERR simulator. This indicates that the handcrafted policy is more robust to unseen dialogue situations than the policy trained on a handcrafted user model. The result illustrates the potential pitfalls of training and testing on the same user model discussed in [33] and underlines the importance of a cross-model evaluation.

The experiment shows that the policy trained with the AGD-T-ERR model outperforms the handcrafted policy when testing on the AGD-T model. This result must be interpreted with care: it may indicate that the handcrafted policy works less well on real users than anticipated when testing on the AGD-H simulator. The performance gain achieved here with the trained policy, however, may be simply due to the fact that training and testing use the same model. The more significant result is that the AGD-T-ERR policy approximately matches the handcrafted policy and the AGD-H-ERR policy when testing on the AGD-H model. This shows that the policy learned with the trained agenda model transfers well to the handcrafted user model, but not vice versa.

VI. Policy Evaluation With Real Users

A. User Study Setup

To validate the results obtained in simulations, a user study was carried out with 36 human subjects recruited from outside the research group. The participants included 13 male and 23 female native British speakers. None of the participants had previously participated in a user study involving spoken dialogue systems. The evaluation was run under controlled conditions in three rooms with very quiet noise conditions. Each room was equipped with a state-of-the-art desktop machine with 2 GB of RAM and a Quadcore CPU running at 2.4 GHz. All recordings were done in an "open-mic" setting with KOSS CS-100 close-talking headsets/microphones.

Three different systems were tested during the user study to investigate 1) the effect of the statistical user simulation model and 2) the effect of training under noisy conditions. All three systems were based on the same MDP dialogue manager described in the previous section and differed only in their dialogue policy. One policy was trained using the handcrafted (AGD-H) simulator, one was trained using the trained statistical agenda model (AGD-T), and one was trained using the statistical agenda model under noisy conditions with a simulated 25% semantic error rate (AGD-T-ERR).

The computing hardware and all other dialogue system components were identical. For the speech recognizer the ATK/HTK toolkit was used, with a standard trigram language model and vocabulary of approximately 2000 words. The n-best output of the recognizer was limited to the single most likely hypothesis. Semantic representations in the form of CUED dialogue acts (cf. Section II-A) were extracted using a rule-based Phoenix decoder [36], [37] on the output of the recognizer. The language generation component was implemented using simple templates for mapping system dialogue acts to word-level utterances. These were then synthesized using the publicly available FLITE text-to-speech engine.

Example Task

It is your partner's birthday and you would like to go to a luxurious restaurant to celebrate it. Italian is your partner's favourite cuisine. It would be very romantic if the restaurant had a view over the river. You want to get the phone number so that you can make a reservation.

Q 1: Did you find all the information you were looking for? (Yes/No)

Q 2: On a scale from 1 to 5, how easy and intuitive did you find the system? (1=not at all, 5=very easy & intuitive)

Fig. 11. Sample task instruction and questionnaire.

Each of the 36 subjects completed two dialogues with each of the three systems. Dialogue tasks were randomly selected from a set of 13 task specifications and the order of the systems and tasks was also chosen randomly. All task specifications were presented in written form, as shown by the example task specification in Fig. 11. It was ensured that the database always contained exactly one matching venue for each task specification. To evaluate the perceived task completion (PTC) rate and the level of user satisfaction, subjects were also asked to answer two questions after completing each task, as shown in Fig. 11.

B. User Study Results

The full evaluation corpus consists of 216 dialogues, containing a total of 1524 user turns and 8336 words. When averaging over all utterances, the WAcc based on the number of word substitution, insertion, and deletion errors is 66.1% and the SAcc is 81.2%. The Semantic Error Rate (SER), i.e., the percentage of user turns where the top recognition hypothesis does not exactly match the true user act is 28.3%. The results of the user study are shown in Tables I and II.

1) Statistical Versus Handcrafted User Simulation: The objective actual task completion results show that the policy trained with the AGD-T simulator clearly outperforms the policy trained with the handcrafted AGD-H simulator, both in terms of partial and full completion rates (Table I). The objective dialogue performance score of 83.15 achieved with the AGD-T model exceeds the score of 72.21 obtained with the AGD-H simulator, a statistically significant relative improvement of 15.15% ($p < 0.001$).

The subjective user scores agree with the objective metrics: The perceived task completion (PTC) rate of 97.2% is a statistically significant 16.7% relative improvement over the score achieved by the AGD-H simulator (83.3%), as shown in Table II. This is statistically significant according to Fisher's exact test, with a two-tailed P value of 0.0091. The perceived task completion score is higher than the actual task completion score because users did not always ask for all pieces requested in the task

TABLE I
ACTUAL TASK COMPLETION STATISTICS

Completion	Policy	%success	#turns	perf.±std. err.
Partial	AGD-H	84.72	2.83	81.89±4.55
	AGD-T	90.28	2.33	87.94±3.68
	AGD-T-ERR	91.67	2.51	**89.15±3.55**
Full	AGD-H	77.78	5.57	72.21±5.13
	AGD-T	87.50	4.35	**83.15±4.04**
	AGD-T-ERR	83.33	4.74	78.60±4.59

Partial completion is awarded when a correct venue is recommended; full completion is awarded when all extra pieces of information, eg. address and telephone number are also supplied correctly. The table shows the percentage of dialogues partially/fully completed and the average number of turns to partial/full completion. When the dialogue was not completed, the full dialogue length was counted. Performance is computed using (26). The sample size for each row is 72.

TABLE II
PERCEIVED TASK COMPLETION

Policy	Total	no	yes	%yes
AGD-H	72	12	60	83.3
AGD-T	72	2	70	**97.2**
AGD-T-ERR	72	6	66	91.7

Perceived Task Completion was measured using the following question: "Did you find all the information you were looking for?" (Yes/No). For each of the three policies, the two-tailed P value is < 0.0001 using a standard sign test.

TABLE III
USER SATISFACTION

Policy	Total	1	2	3	4	5	mean
AGD-H	72	6	3	16	19	28	3.83
AGD-T	72	0	3	11	27	31	**4.19**
AGD-T-ERR	72	0	6	10	30	26	4.06

User Satisfaction was measured using the following question: "On a scale from 1 to 5, how easy and intuitive did you find the system?" (1=not at all, 5=very easy and intuitive).

specification and did not always notice when the system recommended an incorrect venue (i.e., a venue that did not match all of the user's constraints).

The user satisfaction scores obtained from questionnaires (Table III) also show that the policy learned with the statistical user model is rated significantly higher (i.e., easier to use) by real users than the policy learned with the handcrafted simulator ($4.19 > 3.83$, statistically significant according to a Chi-Square test with $\mathrm{p} < 0.02$). This confirms the result obtained in simulation runs and shows the benefit of training with a statistical model of user behavior. The results also compare very favorably with the task completion rates of 64% [38] and 81.8% [13] reported in previous user studies on reinforcement-learning of dialogue policies.

2) Training With and Without Noise: As predicted in simulation runs, the policy learned with the trained simulator under noisy conditions (AGD-T-ERR) outperforms the policy learned with the handcrafted simulator (AGD-H) (statistically significant with $p < 0.001$). The policy trained in noisy conditions (AGD-T-ERR) also outperforms the policy trained on the agenda model without noise (AGD-T) on the objective partial completion metrics (statistically significant with $p < 0.025$). However, all other ATC and PTC scores achieved when training with noise are generally lower than when training without noise. The subjective user satisfaction scores draw the same picture.

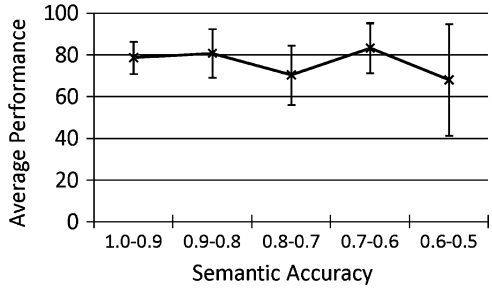

Fig. 12. Bins group dialogues according to their average semantic accuracy (SAcc). Dialogue performance is averaged over all dialogues in each SAcc bin. Error bars indicate 99% confidence intervals.

Again, the simulator without noise (AGD-T) outperforms the simulator with noise (AGD-T-ERR), which in turn outperforms the handcrafted simulator (AGD-H). This disagrees with the result predicted during simulation runs and shows that the simulation-based policy evaluation results do not always carry over to real users.

Further analysis of the recorded dialogues shows that the SER of 25% assumed during training is approximately in line with the actual SER encountered during testing with real users (28.3% when averaging over all 216 dialogues). The negative impact of noise on dialogue performance in our simulation experiments, however, was clearly overestimated. In Fig. 12, dialogues are grouped into bins according to their semantic accuracy, and the average performance is shown for each bin from 100% (1.0) down to 50% (0.5). As indicated by the 99% confidence intervals, dialogue performance varies more strongly as the error rate increases. The mean, however, is shifted only very slightly showing that real users are much less affected by recognition errors than the simulated users.

3) Testing on Real vs. Simulated Users: A comparison of the results obtained with real and simulated users reveals a number of interesting findings. Most importantly, the ranking of the learned AGD-H and AGD-T policies obtained with real users is predicted correctly in simulation runs. The predicted performance scores, however, are significantly different from those observed with real users. At very low error rates, the tests on real users show a dialogue performance score (across all three tested policies) of approximately 80 points (see Fig. 12). The simulation runs with the AGD-H model significantly overestimate the performance with scores of approximately 95 points (Fig. 10). The AGD-T model on the other hand significantly underestimates the performance with scores under 60 points (Fig. 9).

This demonstrates that both simulators despite their usefulness for training policies are not accurate predictors of dialogue performance. It also illustrates a second interesting point: Although the AGD-T model is a worse predictor of dialogue performance than the AGD-H model, the dialogue policy learned with the AGD-T model performs better on real users than the policy learned with the AGD-H model. This relates to an interesting discussion in the literature on statistical user simulation: Should the user model mimic the user population in the given training data as truthfully as possible or should it vary from the observed behavior to expose the learning dialogue manager to unseen user behavior? As in Rieser and Lemon [14], one may

argue that the training model should be (partly) evaluated based on whether it allows for a desired amount of deviation from the seen user behavior. The testing model, on the other hand, should above all produce a reliable rank ordering of dialogue policies, as argued by Williams [39].

VII. Conclusion

This paper has presented a novel statistical method for user simulation which models the observable semantic-level user output in terms of a sequence of *hidden* user goals and agendas. The goal consists of slot-value pairs describing the user's requests and constraints, and ensures consistency in user behavior over the course of a dialogue. The agenda stores a list of pending user dialogue acts and serves as a convenient way of encoding dialogue context from a user's point of view. Its stack-like format allows state transitions and user act generation to be modeled using simple push- and pop-operations without unduly limiting the expressive power of the model.

Since the sequence of user state transitions is treated as unobservable, model parameter estimation cannot be performed using simple maximum likelihood frequency counting. To solve this problem, an EM-based training algorithm is presented which uses dynamically growing tree structures and a "summary space" mapping to achieve tractability. This approach is more complex then previously presented user model parameter estimation techniques, but it has the significant advantage of not requiring any state-specific dialogue annotation.

The experimental results presented in this paper demonstrate that the model can be successfully trained on a limited amount of minimally labeled dialogue data. The trained model may then be used to learn a competitive MDP dialogue policy using a standard Reinforcement-Learning algorithm. Using a simulation-based cross-model evaluation, the learned policy outperforms a handcrafted policy and a policy learned with a competing baseline simulator. The simulation runs also demonstrate the advantage of training policies in noisy conditions. An extensive user study involving 216 dialogues with 36 different participants confirms the competitive performance of the policy learned with the trained agenda user model. The results show an actual task completion rate of 87.5% and a perceived task completion rate of 97.2%, outperforming the policy learned with a handcrafted simulator by more than 10%. User satisfaction with the agenda-based system is also significantly higher than with the baseline.

To extend the user model to more complex dialogue scenarios, future work will address the shortcomings of the current goal update model implementation. As pointed out in Section III-E of this paper, the deterministic implementation presented here is not trainable and places restrictions on the user goal transitions which limit the behavior of the simulated user. Covering a broader range of user behavior may also require an extension to the set of agenda model summary conditions and summary push actions. This unfortunately requires detailed knowledge of the dialogue act format and familiarity with the agenda model. An interesting research question is thus whether the process of selecting summary conditions and actions can be automated. A possible starting point for research in this direction may be to cluster user states and user outputs based on their similarity. Finding a suitable distance metric for measuring the similarity between different agendas, goals, and dialogue acts, however, is unlikely to be trivial.

While the simulation-based experiments presented in this paper demonstrate the potential benefits of training under more realistic noise conditions, it has not been possible to confirm this result with real users. Future work should thus revisit the error model to improve the accuracy of its predictions. At a later stage, it would also be interesting to conduct policy training and testing experiments with real users in a noisier setting or with a mix of native and non-native speakers.

Acknowledgment

The authors would like to thank B. Thomson, M. Gašić, S. Keizer, F. Mairesse, and K. Yu.

References

[1] D. R. Traum, "Speech acts for dialogue agents," in *Foundations of Rational Agency*, M. Wooldridge and A. Rao, Eds. Dordrecht, Germany: Kluwer, 1999, pp. 169–201.
[2] S. Young, "Talking to machines (statistically speaking)," in *Proc. ICSLP*, Denver, CO, 2002, pp. 9–16.
[3] O. Lemon and O. Pietquin, "Machine learning for spoken dialogue systems," in *Proc. Eurospeech*, Antwerp, Belgium, 2007.
[4] E. Levin and R. Pieraccini, "A stochastic model of computer-human interaction for learning dialogue strategies," in *Proc. Eurospeech*, Rhodes, Greece, 1997.
[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. Cambridge, MA: MIT Press, 1998.
[6] J. Schatzmann, B. Thomson, and S. Young, "Error simulation for training statistical dialogue systems," in *Proc. ASRU*, Kyoto, Japan, 2007.
[7] J. Schatzmann, K. Weilhammer, M. Stuttle, and S. Young, "A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies," *Knowledge Eng. Rev.*, vol. 21, no. 2, pp. 97–126, 2006.
[8] E. Levin, R. Pieraccini, and W. Eckert, "A stochastic model of human-machine interaction for learning dialog strategies," *IEEE Trans. Speech Audio Process.*, vol. 8, no. 1, pp. 11–23, Jan. 2000.
[9] K. Scheffler, "Automatic Design of Spoken Dialogue Systems," Ph.D. dissertation, Cambridge Univ., Cambridge, U.K., 2002.
[10] O. Pietquin, "A Framework for Unsupervised Learning of Dialogue Strategies," Ph.D. dissertation, Polytech de Mons, Mons, Belgium, 2004.
[11] K. Georgila, J. Henderson, and O. Lemon, "Learning user simulations for information state update dialog systems," in *Proc. Eurospeech*, Lisbon, Portugal, 2005.
[12] M. Frampton and O. Lemon, "Learning more effective dialogue strategies using limited dialogue move features," in *Proc. ACL*, Sydney, Australia, 2006.
[13] O. Lemon, K. Georgila, and J. Henderson, "Evaluating effectiveness and portability of reinforcement learned dialogue strategies with real users: The TALK TownInfo Eval," in *Proc. SLT*, Palm Beach, Aruba, 2006.
[14] V. Rieser and O. Lemon, "Cluster-based user simulations for learning dialogue strategies," in *Proc. ICSLP*, Pittsburgh, PA, 2006.
[15] H. Ai and D. Litman, "Knowledge consistent user simulations for dialog systems," in *Proc. ICSLP*, Antwerp, Belgium, 2007.
[16] H. Cuayahuitl, S. Renals, O. Lemon, and H. Shimodaira, "Human-computer dialogue simulation using hidden Markov models," in *Proc. ASRU*, San Juan, Puerto Rico, 2005.
[17] J. L. Austin, *How to do Things With Words*. Oxford, U.K.: Clarendon, 1962.
[18] H. C. Bunt, Rules for the interpretation, evaluation and generation of dialogue acts," Technische Universiteit Eindhoven, IPO annual progress report 16, 1981, pp. 99–107.
[19] M. G. Core and J. F. Allen, "Coding dialogs with the DAMSL annotation scheme," in *Working Notes of AAAI Symp. Communicative Action in Humans and Machines*, Boston, MA, 1997.
[20] M. Walker and R. Passonneau, "DATE: A dialogue act tagging scheme for evaluation of spoken dialogue systems," in *Proc. HLT2001*, San Diego, CA, 2001.

[21] A. Stolcke, "Dialogue act modeling for automatic tagging and recognition of conversational speech," *Comput. Linguist.*, vol. 26, no. 3, pp. 339–373, 2000.
[22] S. Young, "CUED standard dialogue acts," Cambridge Univ. Eng. Dept., Cambridge, U.K., Tech. Rep., 2007 [Online]. Available: http://mi.eng.cam.ac.uk/research/dialogue/LocalDocs/dastd.pdf.
[23] X. Wei and A. Rudnicky, "An agenda-based dialog management architecture for spoken language systems," in *Proc. ASRU*, Keystone, CO, 1999.
[24] D. Bohus and A. Rudnicky, "Ravenclaw: Dialog management using hierarchical task decomposition and an expectation agenda," in *Proc. Eurospeech*, Geneva, Switzerland, 2003.
[25] J. Schatzmann, B. Thomson, and S. Young, "Statistical user simulation with a hidden agenda," in *Proc. SIGDial*, Antwerp, Belgium, 2007.
[26] B. Thomson, J. Schatzmann, K. Weilhammer, H. Ye, and S. Young, "Training a real-world POMDP dialogue system," in *Proc. NAACL-HLT 2007*, Rochester, NY, 2007, Bridging the Gap Workshop.
[27] J. D. Williams and S. Young, "Scaling up POMDPs for dialog management: The "Summary POMDP" method," in *Proc. ASRU*, San Juan, Puerto Rico, 2005.
[28] J. Henderson, O. Lemon, and K. Georgila, "Hybrid reinforcement/supervised learning for dialogue policies from communicator data," in *Proc. Workshop Knowl. Reasoning Practical Dialog Syst., IJCAI*, Edinburgh, 2005.
[29] S. Young, J. Williams, J. Schatzmann, M. Stuttle, and K. Weilhammer, "The hidden information state approach to dialogue management," Cambridge Univ. Eng. Dept., , Tech. Rep. CUED/F-INFENG/TR.544, 2005.
[30] S. Young, J. Schatzmann, K. Weilhammer, and H. Ye, "The hidden information state approach to dialog management," in *Proc. ICASSP*, Honululu, HI, 2007.
[31] M. Boros, W. Eckert, F. Gallwitz, G. Gorz, G. Hanrieder, and H. Niemann, "Towards understanding spontaneous speech: Word accuracy vs. Concept accuracy," in *Proc. ICSLP*, Philadelphia, PA, 1996.
[32] J. Schatzmann, "Statistical user and error modeling for spoken dialogue systems," Ph.D. dissertation, Univ. of Cambridge, Cambridge, U.K., 2008.
[33] J. Schatzmann, M. Stuttle, K. Weilhammer, and S. Young, "Effects of the user model on simulation-based learning of dialogue strategies," in *Proc. ASRU*, San Juan, Puerto Rico, 2005.
[34] J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young, "Agenda-based user simulation for bootstrapping a POMDP dialogue system," in *Proc. NAACL-HLT 2007*, Rochester, NY, 2007.
[35] O. Lemon and X. Liu, "Dialogue policy learning for combinations of noise and user simulation: Transfer results," in *Proc. SIGDial*, Antwerp, Belgium, 2007.
[36] W. Ward and S. Issar, "Recent improvements in the CMU spoken language understanding system," in *Proc. ARPA Workshop Human Lang. Technol.*, 1994.
[37] S. Issar and W. Ward, "Flexible parsing: CMU's approach to spoken language understanding," in *Proc. ARPA Workshop Spoken Lang. Technol.*, 1994.
[38] D. J. Litman, M. S. Kearns, S. Singh, and M. A. Walker, "Automatic optimization of dialogue management," in *Proc. COLING*, Saarbruecken, Germany, 2000.
[39] J. D. Williams, "A method for evaluating and comparing user simulations: The Cramer-von Mises divergence," in *Proc. ASRU*, Kyoto, Japan, 2007.

Jost Schatzmann received the B.Eng. degree from Imperial College London, London, U.K., in 2003 and the M.Phil. degree on a Gates Scholarship to study computer speech, text, and Internet technology from the University of Cambridge, Cambridge, U.K., in 2004, and the Ph.D. degree in user simulation modeling from the University of Cambridge in 2008.

His research interests lie in applications of machine learning to speech and language processing. His thesis work focused on statistical user and error modeling for simulation-based reinforcement-learning of optimal dialogue policies.

Steve Young (M'96–SM'06–F'09) received the B.A. degree in electrical science tripos and the Ph.D. degree from Cambridge University, Cambridge, U.K., in 1973 and 1977, respectively.

He was then a Lecturer at UMIST and Cambridge before being elected to a Chair in Information Engineering at Cambridge in 1995, where he is currently Head of the Information Engineering Division. Interleaved with his academic appointments, he has also held industrial positions with GEC, Entropic, and Microsoft. His main research interests lie in the area of spoken language systems including speech recognition, speech synthesis, and dialogue management. He was Editor of *Computer Speech and Language* from 1993 to 2004.

Dr. Young is a Fellow of the Royal Academy of Engineering, the Institution of Electrical Engineers, and the RSA. He is a member of the British Computer Society. In 2004, he was a recipient of an IEEE Signal Processing Society Technical Achievement Award.