D4.1: Integration of Learning and Adaptivity with the ISU approach

Oliver Lemon, Kallirroi Georgila, James Henderson,
Malte Gabsdil, Ivan Meza-Ruiz, Steve Young

Distribution: Public

TALK
Talk and Look: Tools for Ambient Linguistic Knowledge
IST-507802 Deliverable 4.1

February 14, 2005

Project funded by the European Community
under the Sixth Framework Programme for
Research and Technological Development

The deliverable identification sheet is to be found on the reverse of this page.

Project ref. no.	IST-507802
Project acronym	TALK
Project full title	Talk and Look: Tools for Ambient Linguistic Knowledge
Instrument	STREP
Thematic Priority	Information Society Technologies
Start date / duration	01 January 2004 / 36 Months

Security	Public
Contractual date of delivery	M12 = december 2004
Actual date of delivery	February 14, 2005
Deliverable number	4.1
Deliverable title	D4.1: Integration of Learning and Adaptivity with the ISU approach
Type	Report
Status & version	Final 1.0
Number of pages	74 (excluding front matter)
Contributing WP	6
WP/Task responsible	UEDIN
Other contributors	
Author(s)	Oliver Lemon, Kallirroi Georgila, James Henderson, Malte Gabsdil, Ivan Meza-Ruiz, Steve Young
EC Project Officer	Kimmo Rossi
Keywords	Reinforcement Learning, Information State Update, Annotation, User Modelling, In-car, dialogue system

The partners in TALK are:

Saarland University	USAAR
University of Edinburgh HCRC	UEDIN
University of Gothenburg	UGOT
University of Cambridge	UCAM
University of Seville	USE
Deutches Forschungszentrum fur Künstliche Intelligenz	DFKI
Linguamatics	LING
BMW Forschung und Technik GmbH	BMW
Robert Bosch GmbH	BOSCH

For copies of reports, updates on project activities and other TALK-related information, contact:

The TALK Project Co-ordinator
Prof. Manfred Pinkal
Computerlinguistik
Fachrichtung 4.7 Allgemeine Linguistik
Postfach 15 11 50
66041 Saarbrücken, Germany
pinkal@coli.uni-sb.de
Phone +49 (681) 302-4343 - Fax +49 (681) 302-4351

Copies of reports and other material can also be accessed via the project's administration homepage, `http://www.talk-project.org`

Contents

Summary

This report details work done in the early phases of the TALK project to integrate "Information State Update" (ISU)-based dialogue systems with techniques from machine learning (e.g. memory-based learning, reinforcement learning). We explain how to use data gathered from information states of ISU dialogue managers in a variety of different learning techniques. We present initial results showing that various classifier learning techniques over ISU representations can reduce error rates by over 50%. Ultimately we focus on Reinforcement Learning (RL) techniques for automatic optimisation of dialogue strategies. This has involved constructing an automatic annotation system which builds ISU state representations for the COMMUNICATOR corpora of human-machine dialogues, and then developing novel techniques for learning from this data. We describe a learning method based on linear function approximation (in order to deal with the very large state spaces generated by the ISU approach), and a hybrid method which combines supervised learning and reinforcement learning. We also describe 2 different types of user simulation that we have built from the COMMUNICATOR data for testing purposes. Our initial results are encouraging. We show that over a run of 1000 simulated dialogues our learned policy has an average reward (computed as a function of the task completion and dialogue length) of over twice the average score of the COMMUNICATOR systems, and more than a third greater than the best of the COMMUNICATOR systems. We also report on progress in a baseline system for in-car dialogues and describe plans for future research.

Chapter 1

Introduction

This report details work done in the early phases of the TALK project to integrate "Information State Update" (ISU)-based dialogue systems with techniques from machine learning (e.g. memory-based learning, reinforcement learning). We explain how to use data gathered from information states of ISU dialogue managers in a variety of different learning techniques (chapter 2). We present initial results showing that various classifier learning techniques over ISU representations can reduce error rates by over 50%. Ultimately we focus on Reinforcement Learning (RL) techniques for automatic optimisation of dialogue strategies (chapter 3). This has involved constructing an automatic annotation system which builds ISU state representations for the COMMUNICATOR corpora of human-machine dialogues, and then developing novel techniques for learning from this data. We describe a learning method based on linear function approximation (in order to deal with the very large state spaces generated by the ISU approach), and a hybrid method which combines supervised learning and reinforcement learning (chapter 5). We also describe 2 different types of user simulation that we have built from the COMMUNICATOR data for testing purposes. Our initial results are encouraging. We show that over a run of 1000 simulated dialogues our learned policy has an average reward (computed as a function of the task completion and dialogue length) of over twice the average score of the COMMUNICATOR systems, and more than a third greater than the best of the COMMUNICATOR systems. We also report on progress in a baseline system for in-car dialogues and describe plans for future research (chapters 6 and 7).

1.1 Dialogue Management, Learning, and Adaptivity

The dialogue manager is the core control component of a multimodal dialogue system. Based on the current dialogue context (or "Information State") and each user input, a new output action must be determined and the information state updated. The focus of WP4 of TALK is on using machine learning techniques to optimise dialogue managers during development and to further adapt them on-line as they are exposed to real user interactions. These techniques work by modelling the dialogue as a Markov Decision Process (MDP), assigning rewards (both positive and negative) to various dialogue states and then finding the dialogue policy which leads to the maximum expected reward starting from any dialogue state [You00, SLKW02].

The Information State Update (ISU) approach as developed in the TRINDI and SIRIDUS projects [LT00, LRH+00], successfully provided a backbone for naturally interactive, yet practical, spoken dialogue sys-

tems. In this deliverable we focus on enhancing this perspective using machine learning techniques. Some recent work on adaptive dialogue systems concerns Reinforcement Learning of dialogue "policies" – pairings of dialogue actions with contextual factors (e.g. [SLKW02]). Although this approach to dialogue is still relatively immature it promises a powerful framework for automatic learning and optimisation. Here we explore how it can be integrated within a structured, formal account of the dynamics of dialogue context such as that developed under the ISU framework. If successful, the potential for improved robustness and on-line adaptation of dialogue systems should be considerable.

We now review several recent approaches in this area. Seminal work in this area was published in [LP97, LPE00]. We will later point out the differences in our approach, and the opportunities that it presents.

1.2 Current State of the art

Some recent work on dialogue systems concerns Reinforcement Learning of dialogue "policies" – pairings of dialogue actions with contextual factors (e.g. [SLKW02, LKSW00, You00, LPE00, SY02]). Here we survey these approaches.

1.2.1 The NJFUN system

NJFun [LKSW00] is representative of recent efforts to apply Reinforcement Learning techniques to dialogue management. It is a telephone-based tourist information system used to access a database via 3 attribute values or "slots": activity type, location, and time of day. The system is constrained to only ask for each attribute a maximum of two times (thus reducing the size of the possible policy space). In [LKSW00], Litman et al. concentrate on learning 2 aspects of dialogue management strategy, *initiative* and *confirmation*, leading to a search space of 2^{42} potential strategies (i.e. state-action pairs). These states are explored through experiments with human users.

Since human dialogues are costly to collect, researchers typically limit the size of the state space, or use simulated users. The authors of [LKSW00] choose to work in a mimimal and carefully designed state space.

NJFUN Methodology

The following research procedure was employed:

1. Represent a dialogue strategy as a mapping from states S to a set of dialogue actions A.
2. Deploy an initial training system which generates exploratory data w.r.t. S
3. Construct an MDP model from the training data
4. Redeploy the system using the learned policy/strategy
5. Evaluate the system w.r.t. a hand-coded dialogue strategy.

Dialogue strategies in NJFUN

In order to use Reinforcement Learning techniques all potential actions for each state must be specified. Sometimes it is easy for a human to decide on the correct action in a state (e.g. initially, greet the user) and so these sorts of decisions were made in advance (again, reducing the size of the state space for learning). Action choices were restricted to: *initiative* when asking or re-asking for an attribute, whether to *confirm* an attribute once it is obtained.
3 types of initiative were explored: *user, system*, and *mixed*:

- *user*: the system asks open questions with an unrestricted grammar for recognition
- *system*: the system uses directive prompts with restricted grammars
- *mixed*: the system uses directive prompts with a non-restrictive grammar.

There were 2 also confirmation options in NJFUN: *explicit*, and *no confirmation*:

- *explicit*: the system asks the user to verify an attribute value
- *no confirmation*: the system does not generate a confirmation prompt.

System control: the "operations vector"

The "operations vector" in [LKSW00] is not used for learning, but for system control. It has 14 variables: 2 are for whether the system has greeted the user and for which attribute type the system is trying to obtain. Then for each of the 3 attributes, 4 variables track whether the value has been obtained, the confidence in the value (if obtained), the number of times the system has asked that attribute, and the type of ASR grammar most recently used in asking for it.

The state space *S*

For learning, the state space contains 7 variables (see figure 1.1), and is computed from the history of the operations vector.

Greet	Attr	Conf	Val	Times	Gram	Hist
0,1	1,2,3,4	0,1,2,3,4	0,1	0,1,2	0,1	0,1

Figure 1.1: NJFun state features and values

The details of the state vector are as follows:

- Greet: whether the system has greeted the user or not (0=no, 1=yes)
- Attr: which attribute type the system is trying to obtain or verify (1=activity, 2=location, 3=time, 4=done)

- Conf: system ASR confidence after obtaining value for an attribute (0,1,2 = low,med,high ASR confidence, 3= “yes” after confirmation question, 4= “no” after a confirmation question)
- Val: whether system has obtained a value for the attribute (0=no, 1=yes)
- Times: number of times the system has asked for the attribute (max = 2)
- Gram: type of grammar most recently used to obtain the attribute (0= non-restrictive, 1= restrictive)
- Hist: trouble-in-past: whether the system had trouble understanding the user earlier in the conversation (0=bad, 1=good)

The number of states that can actually occur is 62 (for example greet=0 can only occur in the first state, all other states have greet=1).

NJFUN’s initial strategy: EIC

EIC, “Exploratory for Initiative and Confirmation”, is a state-action mapping given by a large table (see figure 2 of [LKSW00]). The table shows only those states for which the system actually has an action choice, i.e. the *exploratory* part of the strategy.
For example see figure 1.2.

Greet	Attr	Conf	Val	Times	Gram	Hist	Action1	Action2
0	1	0	0	0	0	0	GreetS	GreetU
1	1	0	0	1	0	0	ReAsk1S	ReAsk1M

Figure 1.2: Part of the NJFUN system’s EIC strategy

The EIC strategy chooses *randomly* between Action1 and Action2 in each state, to maximize exploration and minimize data sparseness. Each ActionN is a system utterance or prompt (see examples below). There are 42 choice states, with 2 choices each, leading to 2^{42} potential global dialogue strategies. RL is then used to identify an “optimal” strategy from this large space.
Note that prompts must be carefully designed to ensure coherence when exploring all possible action sequences in EIC. For example:

- GreetS: “Welcome to NJFun. Please say an activity name or say “list activities” for a list of activities I know about”.
- GreetU: “Welcome to NJFun. How may I help you?”
- ReAsk1S: “I know about amusement parks, aquariums, cruises, historic sites, museums, parks, theatres, wineries, and zoos. Please say an activity name from this list.”
- ReAsk1M: “ Please tell me the activity type. You can also tell me the location and time.”

The action “Tell” is the final system action, where the user is presented with search results. The reward for actions is always zero, apart from for “Tell” in the final state (vector 1 4 0 0 0 0 0).

Optimising the NJFUN strategy

NJFun was implemented with the EIC strategy, and was used to collect training dialogues. These dialogues were used to build an empirical MDP. In total 54 subjects were used for training and 21 subjects for testing, and each user tried to complete 6 application tasks over the telephone. The tasks were first presented on a web page and the system asked for feedback at end of each task. After their phone call, subjects filled out a web survey form. A total of 311 training dialogues were collected, and only 8 (out of 42) states were visited fewer than 10 times.

A binary reward function was chosen, based on the strongest possible measure of task completion: the value is 1 if the system queries the database with exactly the attributes specified in the task description, and 0 otherwise.

After learning, there were still some states for which no action was preferred - these states still use a random choice in the final dialogue policy.

Evaluation of NJFUN

The system was reimplemented using the learned strategy (which [LKSW00] argued to be intuitively explicable). 21 subjects were used on the same 6 tasks resulting in 124 complete test dialogues. EIC was the baseline strategy for comparison and task completion rose from 52% to 64%, with $p < .06$.

Data was divided into the first 2 tasks ("novice users") and tasks 3-6 ("expert users") to control for learning effects. The learned strategy led to a large and significant improvement in task completion for experts and a non-significant degradation for novices.

A companion paper [SKLW00] shows that the learned strategy also performs better than several "fixed" strategies (e.g. such as to always use system initiative and no-confirmation). Evaluation was also conducted on ASR metrics, weak-task-completion, and user satisfaction. The ASR and weak-completion scores improved significantly, but user satisfaction did not.

1.2.2 Policy optimisation using simulated users

[SY01] note that in NJFun a consequence of using real data for training is that the state space must be fixed in advance. Furthermore, since real data is costly to collect, the amount that can be obtained in practice is rather small. Scheffler and Young [SY01, SY02] avoided this by using a two-stage approach:

- Train a user simulation model using data obtained from a prototype dialogue system.
- Perform policy optimisation using synthetic data generated by the user simulation model.

The assumption underlying this approach is that a model of the user trained using data collected with one specific dialogue strategy will then generalise to other strategies. Note that the model includes both aspects of user behaviour and the effect of errors in the speech recognition and understanding components.

The user simulation model operates at the "intention level". Early attempts to use this approach were based on simple n-gram models trained directly on dialogues transcribed at the intention level. However, since an n-gram can only retain the immediate history, the resulting user behaviour was unfocused and aimless. The user changed goals frequently and the resulting learnt dialogue strategies were not useful.

In contrast, Scheffler and Young's model was based on the assumption that the user acts in a consistent and goal directed fashion, while varying his/her actions in a probabilistic way. The internal user state is modelled around a goal structure comprised of the following variables that are updated throughout the course of the dialogue:

- The current contents of the user goal.
- Status variables associated with the goal fields.
- Secondary goal structures representing the user's beliefs on the current system goal and the newly completed goal (if any).
- Application specific variables describing items such as the type (eg. direct question) of the previous prompt.

The user and error models also use parameters estimated from the training data which quantify the frequencies of different user goal initialisations, how often particular types of details will be specified in mixed initiative situations, how likely each possible insertion, deletion and substitution error is to occur during recognition, and what the distribution of confidence levels is for both correct and incorrect recognitions. The models were generated automatically from a dialogue task specification and the tagged syntaxes used by the recognition / understanding subsystem. They were then used to generate intention level versions of responses to system utterances, governed by the model probabilities as well as the current user state. By interfacing them with an implementation of the system, complete dialogues were simulated.

By focussing on a specific application, and by using a prototype dialogue system to bootstrap from, Scheffler and Young developed a quantitative, data driven model of the user behaviour and system recognition performance that allowed them to use relatively sophisticated techniques for dialogue policy optimisation. In particular, they used Q-learning with elligibility traces. These provide a means of combining Q-learning, where state-action values are updated based on the current estimate of the value for the next state, with Monte Carlo learning where state-action values are updated based on the reward at the end of the episode.

The learned policies generated by using the user simulation model outperformed handcrafted policies using the same state space, and gave performance similar to that of a system which had been through multiple iterations of refinement by a team of applications engineers. Overall, this work showed that user simulation models are essential if time-domain Monte Carlo-like learning methods are to be used, but in order to be useful, such models must exhibit goal-directed behaviour. However, the user model developed by Scheffler and Young was application dependent and it required an existing spoken dialogue system in order to derive its structure. What is clearly needed is a more general goal-directed model which can be estimated from data without the need for an existing dialogue system or extensive hand-crafting.

1.2.3 Summary and challenges

This brief survey of recent machine learning approaches to dialogue management illustrates that the approach is promising, with significant task completion improvements in the case of NJFUN, and the work of [SY01] also outperforming hand-crafted dialogue policies.

However, note that none of these efforts has been integrated with a structured, formal account of the dynamics of dialogue context, such as that developed under "Information State Update" (ISU) projects such as TRINDI, SIRIDUS, and WITAS. In the next two chapters we explore different learning techniques and how they can be applied to ISU-based dialogue systems.

One of the principal challenges in this area is how to handle the very large state spaces which the ISU approach introduces. The TALK objectives in this area are to answer the following questions:

- What kind of abstract representations best serve this approach to adaptivity and learning?
- Which part of the information state are most amenable to optimisation and what reward functions are needed to achieve it?
- How can the large ISU state space be mapped effectively into a tractable subset for reinforcement learning?

In chapter 3 we describe an early example implementation of our chosen approach, using linear function approximation to deal with the very large state spaces generated by ISU systems. We then present results for learning dialogue strategies over the COMMUNICATOR corpus of flight-booking human-computer dialogues.

Chapter 2

Classifier Machine Learning methods for ISU dialogue systems

This chapter explains how we have explored the use of several different machine learning techniques with Information State Update dialogue management. In the papers [GL04a] and [MRL05] we reported results using Memory-based learning and Maximum Entropy learning respectively – this chapter summarizes that work. In [LH04] and [GL04b] we outlined the Reinforcement Learning approach (see chapter 3) which we will explore further in the remainder of the project.

2.1 Memory-based learning

In research reported in [GL04a] we explored a memory-based learning (MBL) approach in conjunction with ISU representations. This work was also shared with TALK task 2.1 (d) "Filtering speech recognition hypotheses". Here we report on the task from the point of view of our novel combination of ISU dialogue management with machine learning techniques. In status report T2.1s1 and deliverable 2.1 we will present this work with respect to the problem of re-ranking recognition hypotheses according to plausibility in the domain.

2.1.1 The problem: context-sensitive speech recognition

A crucial problem in the design of spoken dialogue systems is to decide for an incoming recognition hypothesis whether a system should *accept* it (consider correctly recognised), *reject* it (assume misrecognition), or *ignore* it (classify as noise or speech not directed to the system). In addition, a more sophisticated dialogue system might decide whether to *clarify* or *confirm* certain hypotheses.

Obviously, incorrect decisions at this point can have serious negative effects on system usability and user satisfaction. On the one hand, accepting misrecognised hypotheses leads to misunderstandings and unintended system behaviours which are usually difficult to recover from. On the other hand, users might get frustrated with a system that behaves too cautiously and rejects or ignores too many utterances. Thus an important feature in dialogue system engineering is the tradeoff between avoiding task failure (due to misrecognitions) and promoting overall dialogue efficiency, flow, and naturalness.

In this work, we investigated the use of machine learners trained on a combination of acoustic confidence

and pragmatic plausibility features to predict the quality of incoming n-best recognition hypotheses to a spoken dialogue system. These predictions were then used to select a "best" hypothesis and to decide on appropriate system reactions. We evaluated this approach in comparison with a baseline system that combines a fixed recognition confidence threshold with dialogue-state dependent recognition grammars [LG04a, Lem04].

Classifying the n-best recognition hypotheses instead of the single-best (e.g. [WWL00]) allows us to avoid grammar switching (which simplifies dialogue system design and implementation) and lets us circumvent the potential pitfall of activating wrong recognition grammars that are bound to fail if the user says something unexpected. The main difference between our approach and previous work on hypothesis reordering (e.g. [CR01]) is that we make a decision as to whether the dialogue system should accept, clarify, reject, or ignore an user utterance. We applied our methodology to a system that implements the "Information State Update" (ISU) approach to dialogue management and therefore expect it to be applicable to a variety of related spoken dialogue systems.

The chapter is organised as follows. Section 2.1.2 introduces WITAS, the spoken dialogue system we are working with, and which we use to collect data (Section 2.1.3) and to derive baseline results (Section 2.1.4). Section 2.1.5 describes our learning experiments for classifying and selecting from n-best recognition hypotheses and Section 2.1.6 reports our results.

2.1.2 The WITAS System

WITAS [LG04b] is a multimodal command and control spoken dialogue system that allows a human operator to interact with a simulated "unmanned aerial vehicle" (UAV): a small robotic helicopter with on-board planning capabilities. The human operator is provided with an interactive (i.e. mouse clickable) map and specifies mission goals for the UAV through natural language commands spoken into a headset. The UAV can carry out different activities such as flying to a location, following a vehicle, delivering objects, or landing. WITAS uses the Nuance 8.0 speech recogniser with language models compiled from a Gemini grammar [DGA+93a], which is also used for parsing and generation.

WITAS Information States

[LG04b] describes in detail the components of WITAS' Information State (IS) and the update procedures for processing user input and generating system responses. Here, we briefly introduce parts of the IS which are needed to understand the system's basic workings, and from which we extracted dialogue-level and task-level information for our learning experiments:

- *Dialogue Move Tree* (DMT): a tree-structure, in which each subtree of the root node represents a "thread" in the conversation, and where each node in a subtree represents an utterance made either by the system or the user.
- *Active Node List* (ANL): a list that records all "active" nodes in the DMT; active nodes indicate conversational contributions that are still open in some sense, and to which new utterances can attach.
- *Activity Tree* (AT): a tree-structure representing the current, past, and planned activities that the UAV performs.

- *Salience List* (SL): a list of NPs introduced in the current dialogue ordered by recency.
- *Modality Buffer* (MB): a temporary store that registers click events on WITAS' map.

The DMT and AT are the core components of WITAS' Information State. The SL and MB are subsidiary data-structures needed for interpreting and generating anaphoric expressions and definite NPs. Finally, the ANL plays a crucial role in integrating new user utterances into the DMT.

2.1.3 Data Collection

For our experiments, we used data collected in a small user study with the grammar-switching version of the WITAS system [Lem04]. In this study, six subjects from our University (4 male, 2 female) had to solve five simple tasks with the system, resulting in 30 complete dialogues.

The subjects' utterances were recorded as 8kHz 16bit waveform files and all aspects of the Information State transitions during the interactions were logged as html files. Altogether, 303 utterances were recorded in the user study ($\approx$ 10 user utterances/dialogue).

Labeling

We transcribed all user utterances and parsed the transcriptions offline using WITAS' natural language understanding component in order to get a "gold-standard" labeling of the data. Each utterance was labeled as either *in-grammar* or *out-of-grammar*, depending on whether its transcription could be parsed or not, or as *crosstalk*: a special marker that indicated that the input was not directed to the system (e.g. noise, laughter, self-talk, the system accidentally recording itself). For all *in-grammar* utterances we stored their interpretations (quasi-logical forms) as computed by WITAS' parser.

Simplifying Assumptions

The evaluations in the following sections make two simplifying assumptions. First, we consider user utterances correctly recognised only if the logical form of the transcription is the same as the logical form of the recognition hypothesis. This assumption can be too strong because the system might react appropriately even if the logical forms are not literally the same. Second, if a transcribed utterance is out-of-grammar, we assume that the system cannot react appropriately. Again, this assumption might be too strong because the recogniser can sometimes accidentally map an utterance to a logical form that is equivalent to the one intended by the user.

2.1.4 The Baseline System

The baseline for our experiments is the behaviour of the WITAS system that was used to collect the experimental data. We chose this baseline because it already showed a significant improvement over first-best recognition, by using dialogue context as a predictor of language models for speech recognition (see [Lem04]). We evaluated the performance of the baseline by analyzing the dialogue logs from the user study. With this information, it was possible to decide how the system reacted to each user utterance. We distinguish between the following three cases:

1. *accept*: the system accepted the recognition hypothesis of a user utterance as correct.
2. *reject*: the system rejected the recognition hypothesis of a user utterance given a fixed confidence rejection threshold.
3. *ignore*: the system did not react to a user utterance at all.

These three classes map naturally to the gold-standard labels of the transcribed user utterances: the system should *accept* in-grammar utterances, *reject* out-of-grammar input, and *ignore* crosstalk.

Context Sensitive Speech Recognition

WITAS uses the ANL to implement a "grammar-switching" approach to speech recognition [Lem04]. At any point in the dialogue, there is a "most active node" at the top of the ANL. The dialogue move type of this node defines the name of a language model that is used for recognising the next user utterance. For instance, if the most active node is a *yes-no-question* then the appropriate language model is defined by a small context-free grammar covering phrases such as "yes", "that's right", "okay", "negative", "maybe", and so on.

This technique showed significantly better recognition rates than a previous version of the system that used the full grammar for recognition at all times (11.5% reduction in overall utterance recognition error rate). Note however that an inherent danger with grammar-switching is that the system may have wrong expectations and thus might activate a language model which is not appropriate for the user's next utterance, leading to misrecognitions or incorrect rejections.

Results

Table 2.1 summarizes the evaluation of the baseline system.

	System behaviour		
User utterance	accept	reject	ignore
in-grammar	154/22	8	4
out-of-grammar	45	43	4
crosstalk	12	9	2

Accuracy: 65.68%
Weighted f-score: 61.81%

Table 2.1: WITAS system baseline results

Table 2.1 should be read as follows: looking at the first row, in 154 cases the system understood and accepted the correct logical form of an in-grammar utterance by the user. In 22 cases, the system accepted a logical form that differed from the one for the transcribed utterance.[1] In 8 cases, the system rejected

[1]For the computation of accuracy and weighted f-scores, these were counted as wrongly accepted out-of-grammar utterances.

an in-grammar utterance and in 4 cases it did not react to an in-grammar utterance at all. The second row of Table 2.1 shows that the system accepted 45, rejected 43, and ignored 4 user utterances whose transcriptions were out-of-grammar and could not be parsed. Finally, the third row of the table shows that the baseline system accepted 12 utterances that were not addressed to it, rejected 9, and ignored 2.

Table 2.1 shows that the major problem with the baseline system is that it accepts too many user utterances. In particular, the baseline system accepts the wrong interpretation for 22 in-grammar utterances, 45 utterances which it should have rejected as out-of-grammar, and 12 utterances which it should have ignored. All of these cases will generally lead to unintended actions by the system.

2.1.5 Classifying and Selecting N-best Recognition Hypotheses

We aimed at improving over the baseline results by considering the n-best recognition hypotheses for each user utterance. Our methodology consists of two steps: i) we automatically classify the n-best recognition hypotheses for an utterance as either correctly or incorrectly recognised and ii) we use a simple selection procedure to choose the "best" hypothesis based on this classification. In order to get multiple recognition hypotheses for all utterances in the experimental data, we re-ran WITAS' speech recogniser with the full recognition grammar and 10-best output and processed the results offline with WITAS' parser, obtaining a logical form for each recognition hypothesis.

Hypothesis Labeling

We labeled all hypotheses with one of the following four classes: *in-grammar*, *oog (WER* $\leq$ *50)*, *oog (WER* $>$ *50)*, or *crosstalk*. The *in-grammar* and *crosstalk* classes correspond to those described for the baseline. However, we decided to divide up the *out-of-grammar* class into the two classes *oog (WER* $\leq$ *50)* and *oog (WER* $>$ *50)* to get a more fine-grained classification. In order to assign hypotheses to the two *oog* classes, we computed the word error rate (WER) between recognition hypotheses and the transcription of corresponding user utterances. If the WER was $\leq$ 50%, we labeled the hypothesis as *oog (WER* $\leq$ *50)*, otherwise it was labeled as *oog (WER* $>$ *50)*.

The motivation behind splitting the *out-of-grammar* class into two subclasses is that we want to distinguish between different "degrees" of misrecognition that can be used by the dialogue system to decide whether it should initiate clarification. The basic idea is that if an utterance contains at least as many correctly as incorrectly recognised words (i.e. if its WER $\leq$ 50), the dialogue system might be able to clarify or confirm the utterance with the user, instead of rejecting it outright.[2] Using the WER as an indicator for whether an utterance should be clarified or rejected is adopted from [Gab03], based on the fact that WER correlates with concept accuracy [BEG$^+$96]. Note that by collapsing the two *oog* classes, we arrive at the same classes used for the baseline and can compare results.

We also introduced the distinction between *oog (WER* $\leq$ *50)* and *oog (WER* $>$ *50)* in the gold standard for the classification of (whole) user utterances. We split the *out-of-grammar* class into two sub-classes depending on whether the 10-best recognition results included at least one hypothesis with a WER $\leq$ 50 compared to the corresponding transcription. Thus, if there is a recognition hypothesis which is close to the transcription, the utterance will be labeled as *oog (WER* $\leq$ *50)*. In order to relate these classes to

[2] WITAS currently does not support this type of clarification dialogue; the *oog (WER* $\leq$ *50)* class is therefore only of theoretical interest. However, an extended system could easily use this information to decide when clarification should be initiated.

different system behaviours, we define that utterances labeled as *oog (WER $\leq$ 50)* should be *clarified* and utterances labeled as *oog (WER $>$ 50)* should be *rejected*.

Classification: Feature Groups

We represent recognition hypotheses as 20 dimensional feature vectors for automatic classification. The feature vectors combine acoustic confidences, low-level acoustic information, information from the WITAS Information State, and domain knowledge about the different tasks in the scenario. The following list gives an overview of all features (described in more detail below).

1. **Recognition (6):** *nnbest, hypothesisLength, confidence, confidenceZScore, confidence-StandardDeviation, minWordConfidence*
2. **Utterance (3):** *minAmp, meanAmp, RMS-amp*
3. **Dialogue (9):** *currentDM, currentCommand, mostActiveNode, DMBigramFrequency, qaMatch, aqMatch, #unresolvedNPs, #unresolvedPronouns, #uniqueIndefinites*
4. **Task (2):** *taskConflict, taskConstraintConflict*

All features were extracted automatically from utterance waveforms, recognition output, and WITAS' IS logs. The recognition (REC) and utterance (UTT) feature groups reflect aspects of the recogniser's confidence assessment for an hypothesis, the position of the hypothesis in the n-best list, the length of the hypothesis (in words), and information about the amplitude in the speech signal (extracted with the UNIX `sox` utility). The motivation for including the amplitude features is that they might help to detect crosstalk utterances (e.g. the system accidentally recognising itself).

The dialogue features (DIAL) represent information derived from WITAS' Information State and can be coarsely divided into two sub-groups. The first group includes features representing general coherence constraints on the dialogue: the dialogue move types of the current utterance and of the most active node in the ANL, the command type of the current utterance (if it is a command, *null* otherwise), statistics on which move types typically follow each other, and two features (*qaMatch* and *aqMatch*) that explicitly encode whether the current and the previous utterance form a valid question answer pair (e.g. *yn-question* followed by *yn-answer*). The second group includes features that indicate how many definite NPs and pronouns cannot be resolved in the current Information State (e.g. "the car" if no car was mentioned before) and a feature indicating the number of indefinite NPs that can be uniquely resolved in the Information State (e.g. "a tower" where there is only one tower in the domain).

Finally, the task features (TASK) reflect conflicting instructions to the UAV. *taskConflict* indicates a conflict if the current dialogue move type is a command and that command already appears as an active task in the AT. *taskConstraintConflict* counts the number of conflicts that arise between the currently active tasks in the AT and the hypothesis. For example, if the UAV is already flying somewhere the preconditions of the action operator for `take_off` ($altitude = 0$) conflict with those for `fly` ($altitude \neq 0$), so that "take off" would be an unlikely command in this context.

Learners and Selection Procedure

We use the memory based learner TiMBL [DZvdSvdB02] and the rule induction learner RIPPER [Coh95] to predict the class of each of the 10-best recognition hypotheses for a given utterance. In a second step,

we decide which (if any) of the classified hypotheses we actually want to pick as the best result and how the user utterance should be classified as a whole. This task is decided by the following selection procedure which implements a preference ordering *accept* > *clarify* > *reject* > *ignore*.

1. Scan the list of classified n-best recognition hypotheses top-down. Return the first result that is classified as *accept* and classify the utterance as *accept*.
2. If 1. fails, scan the list of classified n-best recognition hypotheses top-down. Return the first result that is classified as *clarify* and classify the utterance as *clarify*.
3. If 2. fails, count the number of rejects and ignores in the classified recognition hypotheses. If the number of rejects is larger or equal than the number of ignores classify the utterance as *reject*.
4. Else classify the utterance as *ignore*.

This procedure is applied to choose from the classified n-best hypotheses for an utterance, independent of the particular machine learner, in all of the following experiments.

Since we only had a limited amount experimental data in this study (10 hypotheses for each of the 303 user utterances), we used a "leave-one-out" crossvalidation setup for classification. This means that we classified the 10-best hypotheses for a particular utterance based on the 10-best hypotheses of all 302 other utterances and repeated this 303 times.

2.1.6 Results and Evaluation

The middle part of Table 2.2 shows the classification results for TiMBL and RIPPER when run with default parameter settings (the other results are included for comparison). The individual rows show the performance when different combinations of feature groups are used for training. The results for the three-way classification are included for comparison with the baseline system and are obtained by combining the two classes *clarify* and *reject*. Note that we do not evaluate the performance of the learners for classifying the individual recognition hypotheses but the classification of (whole) user utterances (i.e. including the selection procedure to choose from the classified hypotheses).

The results show that both learners profit from the addition of more features concerning dialogue context and task context for classifying user speech input appropriately. The only exception from this trend is a slight performance decrease when task features are added in the four-way classification for RIPPER. Note that both learners already outperform the baseline results even when only recognition features are considered. The most striking result is the performance gain for TiMBL (almost 10%) when we included the dialogue features. As soon as dialogue features are included, TiMBL also performs slightly better than RIPPER.

Note that the introduction of (limited) task features, in addition to the DIAL and UTT features, did not have dramatic impact in this study. We expect that using a greater number and variety of task features would have further benefits.

System or features used for classification	Acc/wf-score (3 classes)	Acc/wf-score (4 classes)	Acc/wf-score (3 classes)	Acc/wf-score (4 classes)
Baseline	65.68/61.81%			
	TiMBL		RIPPER	
REC	67.66/67.51%	63.04/63.03%	69.31/69.03%	66.67/65.14%
REC+UTT	68.98/68.32%	64.03/63.08%	72.61/72.33%	70.30/68.61%
REC+UTT+DIAL	77.56/77.59%	72.94/73.70%	74.92/75.34%	71.29/71.62%
REC+UTT+DIAL+TASK	77.89/77.91%	73.27/74.12%	75.25/75.61%	70.63/71.54%
TiMBL (optimised params.)	86.14/86.39%	82.51/83.29%		
Oracle	94.06/94.17%	94.06/94.18%		

Table 2.2: Classification Results

Optimising TiMBL Parameters

In all of the above experiments we ran the machine learners with their default parameter settings. However, recent research [DH02, MRvdB+03] has shown that machine learners often profit from parameter optimisation (i.e. finding the best performing parameters on some development data). We therefore selected 40 possible parameter combinations for TiMBL (varying the number of nearest neighbors, feature weighting, and class voting weights) and nested a parameter optimisation step into the "leave-one-out" evaluation paradigm in the following way.[3]

1. Set aside the recognition hypotheses for one of the user utterances.
2. Randomly split the remaining data into an 80% training and 20% test set.
3. Run TiMBL with all possible parameter settings on the generated training and test sets and store the best performing settings.
4. Classify the left-out hypotheses with the recorded parameter settings.
5. Iterate.

Note that our optimisation method is not as sophisticated as the "Iterative Deepening" approach described by [MRvdB+03] but is similar in the sense that it computes a best-performing parameter setting for each data fold.

Table 2.3 shows the classification results when we run TiMBL with optimised parameter settings and using all feature groups for training.

[3] We only optimised parameters for TiMBL because it performed better with default settings than RIPPER and because the findings in [DH02] indicate that TiMBL profits more from parameter optimisation.

	System Behaviour			
User Utterance	accept	clarify	reject	ignore
in-grammar	159/2	11	16	0
out-of-grammar (WER $\leq$ 50)	0	25	5	0
out-of-grammar (WER $>$ 50)	6	6	50	0
crosstalk	2	5	0	16

Acc/wf-score (3 classes): 86.14/86.39%
Acc/wf-score (4 classes): 82.51/83.29%

Table 2.3: TiMBL classification results with optimised parameters

Table 2.3 shows a remarkable 9% improvement for the 3-way and 4-way classification in both accuracy and weighted f-score, compared to using TiMBL with default parameter settings. In terms of WER, the baseline system (cf. Table 2.1) accepted 233 user utterances with a WER of 21.51%, and in contrast, Ti_OP only accepted 169 user utterances with a WER of 4.05%. This low WER reflects the fact that if the machine learning system accepts an user utterance, it is almost certainly the correct one. Note that although the machine learning system in total accepted far fewer utterances (169 vs. 233) it accepted more correct utterances than the baseline (154 vs. 159).

Evaluation

The baseline accuracy for the 3-class problem is 65.68% (61.81% weighted f-score). Our best results, obtained by using TiMBL with parameter optimisation, showed a 25% weighted f-score improvement over the baseline system.

We can compare these results to a hypothetical "oracle" system in order to obtain an upper bound on classification performance. This is an imaginary system which performs perfectly on the experimental data given the 10-best recognition output. The oracle results reveal that for 18 of the in-grammar utterances the 10-best recognition hypotheses did not include the correct logical form at all and therefore have to be classified as *clarify* or *reject* (i.e. it is not possible to achieve 100% accuracy on the experimental data). Table 2.2 shows that our best results are only 8%/12% (absolute) away from the optimal performance.

Costs and χ^2 Levels of Significance

We used the χ^2 test of independence to statistically compare the different classification results. However, since χ^2 only tells us whether two classifications are different from each other, we introduce a cost measure (Table 2.4) for the 3-way classification problem.[4]

Table 2.4 captures the intuition that the correct behaviour of a dialogue system is to accept correctly recognised utterances and ignore crosstalk (cost 0). The worst a system can do is to accept misrecognised

[4]We only evaluate the 3-way classification problem because there are no baseline results for the 4-way classification available.

	System behaviour		
User utterance	accept	reject	ignore
in-grammar	0	2	2
out-of-grammar	4	2	2
crosstalk	4	2	0

Table 2.4: Cost measure

utterances or utterances that were not addressed to the system. The remaining classes are assigned a value in-between these two extremes. Note that the cost assignment is not validated against user judgments and should therefore only be taken as an indicator for the relative quality of a system for the purpose of comparison.

Table 2.5 shows the differences in cost and χ^2 levels of significance when we compare the classification results. Here, Ti_OP stands for TiMBL with optimised parameters and the stars indicate the level of statistical significance as computed by the χ^2 statistics (*** indicates significance at $p = .001$, ** at $p = .01$, and * at $p = .05$).[5]

	Baseline	RIPPER	TiMBL	Ti_OP
Oracle	-232^{***}	-116^{***}	-100^{***}	-56
Ti_OP	-176^{***}	-60^{*}	-44	
TiMBL	-132^{***}	-16		
RIPPER	-116^{***}			

Table 2.5: Cost comparisons and χ^2 levels of signifi cance for 3-way classifi cation

The cost measure shows the strict ordering: Oracle < Ti_OP < TiMBL < RIPPER < Baseline. Note however that according to the χ^2 test there is no significant difference between the oracle system and TiMBL with optimised parameters. Table 2.5 also shows that all of our experiments significantly outperform the baseline system.

Conclusion

We used a combination of acoustic confidence and pragmatic plausibility features to predict the quality of incoming recognition hypotheses to a multi-modal dialogue system. We classified hypotheses as *accept*, (*clarify*), *reject*, or *ignore*: functional categories that can be used by a dialogue manager to decide appropriate system reactions. The approach is novel in combining machine learning with n-best processing for spoken dialogue systems using the Information State Update approach.

Our best results, obtained by using TiMBL with optimised parameters, show a 25% weighted f-score improvement over a baseline system that uses a "grammar-switching" approach to context-sensitive speech recognition, and are only 8% away from the optimal performance that can be achieved on the data. Clearly,

[5]Following [Hin95], we leave out categories with expected frequencies < 5 in the χ^2 computation and reduce the degrees of freedom accordingly.

this improvement would result in better dialogue system performance overall. Parameter optimisation improved the classification results by 9% as compared to using the learner with default settings, which shows the importance of this tuning step.

Clearly, these results show that using dialogue context features from an ISU approach, in combination with learning methods, we can significantly improve dialogue system performance.

2.2 Maximum Entropy

We have also explored how to use ISU features with Maximum entropy learning methods for parse selection in dialogue systems [MRL05].

Maximum Entropy (*ME*) is a machine learning technique which is popular in Natural Language Processing since it has been shown to perform well in different classification tasks (e.g. CONLL03 [TKSDM03]). Most of this success derives from the fact that the generated model is based on as few assumptions as possible, which allows a lot of flexibility during classification. The created model, which represents the learnt behaviour, relies on a set of features f_i and a set of weights λ_i which constrain these features. During experimentation several sets of features are proposed to generate different models until a good one is found. A complete list of the features used to generate the different models in our work can be found in section 2.2.4. In particular we used the implementation of *ME* introduced by [Lee04] given previous experience with the tool.

2.2.1 The problem: parse selection in dialogue

For thsi study we focussed on the problem parse-selection in an ISU-based dialogue system. For each user utterance, our Maximum Entropy classifier takes as input the features of the current dialogue context (see section 2.2.4) and of the logical forms of the hypotheses to be classified, and returns a decision whether to *accept* or *reject* each logical form.

For example see figure 2.1, an extract from our corpus [6]. This shows *n* different logical forms produced by the parsing component (Gemini [DGA+93b]) of a dialogue system (WITAS, [LG04b]), for a single user utterance. In this work we aimed to choose one logical form from the available hypotheses (or to reject the interaction) using information from the hypothesis and the dialogue context. In this example, the baseline system behaviour is to choose the first hypothesis, but the desired behaviour is to choose the second hypothesis, since it corresponds to the transcription of the user utterance. We also have the goal of identifying the informative elements of the dialogue context for this task. In order to accomplish this we use a machine learning approach to build a classifier which is able to identify correct hypotheses from the parser.

2.2.2 The corpus

In our experiments we used the corpus developed in [LG04b, Lem04, GL04a]. The corpus corresponds to the interactions of six subjects from Edinburgh University (4 male, 2 female) in which they each perform

[6] This example corresponds to the user utterance "go to the tower". This has a manual transcription, but the utterance was passed through the speech recogniser and generates *n* hypotheses. Each hypothesis has a confi dence score and a corresponding logical form.

```
Transcription  : go to the tower
  Logical form :
  [command([go],[param_list([[pp_loc(to,arg([np([det([def],the),
                                          [n(tower,sg)]])])])])])]
Hypothesis 1   : go to the towers
  Confidence   : 70
  Logical form :
    [command([go],[param_list([[pp_loc(to,arg([np([det([def],the),
                                            [n(tower,pl)]])])])])])]
Hypothesis 2   : go to the tower
  Confidence   : 70
  Logical form :
    [command([go],[param_list([[pp_loc(to,arg([np([det([def],the),
                                            [n(tower,sg)]])])])])])]
...
Hypothesis 5   : to the tower
  Confidence   : 65
  Logical form :
    wh_answer([param_list([[pp_loc(to,arg([np([det([def],the),
                                            [n(tower,sg)]])])])])])
...
```

Figure 2.1: Example corpus excerpt

five simple tasks with the WITAS system, resulting in 30 complete dialogues. The corpus is made up of the manual transcription of each utterance, the 10-best hypotheses from the speech recogniser (Nuance 8.0), the logical form of the transcriptions and hypotheses from the parser (Gemini [DGA+93b]), and the Information States of the system [LG04b]. The corpus consists of 339 utterances. Only 303 utterances are intelligible to the system, which means that 36 correspond to utterances which were identified as noise. The corpus has a total of 188 types of utterances, because many utterances were used several times (e.g. “yes”, “go here”).

2.2.3 The baseline system

Our baseline is the behaviour of the WITAS Dialogue System described in [LG04b, Lem04, GL04a]. Note that this system had good task completion rates (see [HLC+03]) and thus constitutes a sound baseline.

The baseline performance was evaluated by the analysing of the logs from the user study. Each of the hypotheses was labelled with *accept* or *reject* labels depending on the reaction of the system.

At this point, there are four cases:

1. One hypothesis is accepted, and it corresponds with the manual transcription. This is a *true positive* (tp) event.
2. One hypothesis is accepted, but it does not correspond with the manual transcription. This is a *false positive* (fp) event (e.g. the user said “Now” but the system accepted a logical form corresponding to “no”).
3. None of the hypotheses were accepted, but there was a manual transcription for this interaction. This means that the system failed to recognise an input that should have been accepted. This is a *false negative* (fn) event.

4. Finally, none of the hypothesis was accepted, but there was not a logical form for manual transcription. This means that the interaction should have been rejected by the system (e.g. user self-talk) This is a *true negative* (tn) event.

For the baseline case the precision, recall and FB1 figures were calculated, and we obtained the results presented in table 2.7. These results allow us to evaluate the performance of the baseline system. The strategy used by the system is good at classifying *false negatives* (only 2 cases), this situation implies an impressive *Recall* figure, however the *precision* is not good (there are 97 *fps*). From 2.7 we can observe that the strategy used by the baseline to choose the logical form is not adequate, since there are many false positives. An improvement in the behaviour of the parser would result in a reduction in the number of false positives.

2.2.4 The ISU dialogue context features

We divided the features into three groups (speech processing, logical form, and information state), depending on the type of information they represent:

Speech processing features

In this group there are just two features: the *confidence* of the speech recogniser in the hypothesis and the *position* of this related to the other hypotheses. The confidence is a score from 0 to 100%, in this case we created 10 groups each representing 10 units of confidence.

Logical Form features

We extract information about the dialogue move class, the main verb, and the complement in order to collect features of the logical form.
The following features are extracted in this group:

- Dialogue move class (*dm*): this is codified on the head of the logical form. (e.g., `command` and `wh_query`).
- Main verb (*verb*): the main verb involved in the logical form.
- Complements (*comps*): in this case the information corresponds to the complements of the main verb. We extract the noun (*noun*), the number (*num*), and the the preposition (*prep*) (if the noun phrase is headed by a preposition).

Information State Features

We extract IS features from: *the hypothesis, current and past moves*, and the *information state* data structures. In particular, we extracted dialogue context features from the following data structures of the Information State:

- Dialogue move tree (*DMT*): In this case we extracted from the *n*-most active nodes the *dm* and *verb* features (for $2 \leq n \leq 6$)

Feature	Source
Confidence (*conf*)	speech recogniser
Position (*pos*)	speech recogniser
Turn (*turn*)	Information State (IS)
Dialogue Move (*dm*)	IS: hypothesis, current move, previous moves, *DMT*
Main verb (*verb*)	IS: hypothesis, current move, previous moves, *DMT*
Complements (*comps*)	IS: hypothesis, current move
Noun phrase (*noun*), number(*num*)	IS: Salience List *(SL)*

Table 2.6: Context Features for the ME Classifier

- Salience list: In this case we extracted from the n most recent noun phases the *noun* and *num* features (for $2 \leq n \leq 4$)
- Last n-moves: This corresponds to the *dm* and *verb* from the last moves (for $1 \leq n \leq 2$). This case can be considered as uni-gram or bi-gram of the dialogue moves.

We also extracted the *turn* which corresponds to an identifier of the speaker who had the previous turn (*user* or *system*). No more information was used in this case. Table 2.6 summarises the context features.

2.2.5 Results and Analysis

We used the standard metrics "precision", "recall" and "$FB1$" to measure the performance of the system. Of course, one important limitation of the current study is the small size of the corpus. For this reason a "leave-one-out" cross validation method was chosen to evaluate performance.

Our current results, after experimenting with several combinations of feature groups, are shown in 2.7. Note that the raw figure for parse correctness of the system has risen by 17.8%. This is a 54.5% reduction

	Baseline	ME Classifier
True positives	146	141
False positives	97	21
True negatives	58	117
False negatives	2	23
Precision	60.08%	87.04%
Recall	98.65%	85.98%
FB1	74.78%	86.50%
Parse correctness	67.33%	85.15%

Table 2.7: Results: Baseline versus Maximum Entropy Classifier

in parse error rate. We expect that better results could be achieved with further experimentation and parameter optimisation.

In qualitative terms, the strategy used by the classifier is also better than the baseline. First, there is not such a large difference between the *precision* and *recall* figures. Second, it was possible for the classifier to identify cases where the first parse hypothesis did not correspond to the correct one.

During experimentation with different feature groups, we found the following contextual features to be most informative for this task,

1. Speech processing (both confidence and position features)
2. Dialogue move, main verb, and complement of the hypothesis
3. Dialogue move, main verb, and complement of the previous move
4. The turn feature
5. The dialogue move of the four most active nodes on the *DMT*

This shows that dialogue context features in the information state are also important in the parse selection task.

2.3 Summary

The two experiments presented in this chapter show that using ISU-based representations of dialogue context, together with learning methods, can significantly improve both speech recognition and parsing performance in dialogue systems. This is evidence for the hypothesis that ISU context features are useful in dialogue systems engineering, especially in interpreting user inputs. A conjecture arising from this is that similar context features might also be important when it comes to the task of deciding what actions a system should take next in a dialogue – i.e. in the problems of action choice.

Note that the classifier-based techniques described in this chapter do not directly allow us to address the issue of learning optimal dialogue *policies* – the optimal actions that a dialogue system should take in different contexts (e.g. ask-question(destination-city), supply-info(flight-option(1))). This is why we also explore the application of Reinforcement Learning techniques.

Having surveyed these results on classifier-learning based on ISU features, in the next chapter we turn to our main theme – the learning and automatic optimisation of dialogue policies.

Chapter 3

Reinforcement Learning methods for ISU dialogue systems

This chapter explains the main theoretical framework which we will use in the project: Reinforcement Learning (RL) over ISU representations using linear function approximation. Chapter 5 later shows a concrete example of the application of the methods we explain here.

3.1 Introduction to Reinforcement Learning

Sutton and Barto [SB98a] write that,

> "Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics – trial-and-error search and delayed reward – are the two most important distinguishing features of reinforcement learning."

These properties - delayed reward and trial-and-error search - appear to make RL techniques a good fit with the problem of automatically optimising dialogue strategies, because in task-oriented dialogue often the "reward" of the dialogue (e.g. successfully ordering a pizza) is not obtainable immediately, and the large space of possible dialogues for any task makes some degree of trial-and-error exploration necessary.

The central idea is to define reward functions for combinations of system actions A and states S at a particular time, with the goal of finding the policy $\pi : S \rightarrow A$ (selections of actions with respect to states) which maximizes total expected reward. Learning the "best" action strategy is considered to be equivalent to computing an optimal policy for action choice in a Markov decision process (MDP). An optimal policy, given an MDP, can be computed using reinforcement learning (RL). The "Q-value", $Q(S,A)$ defines the expected cumulative reward of action A in state S. Q-values can be estimated using the RL *value-iteration* algorithm [SB98a].

3.2 ISU dialogue management and RL

In the ISU framework, each new input from the user must be interpreted, and the information state at time (IS_t) updated accordingly. Grounding and planning processes may then update the IS further. Finally, based on the updated IS, system dialogue acts will be generated, for example to ask the user a question or provide the user with some needed information. The type of system dialogue act generated depends on the dialogue policy.

Figure 3.1 illustrates this basic update process.

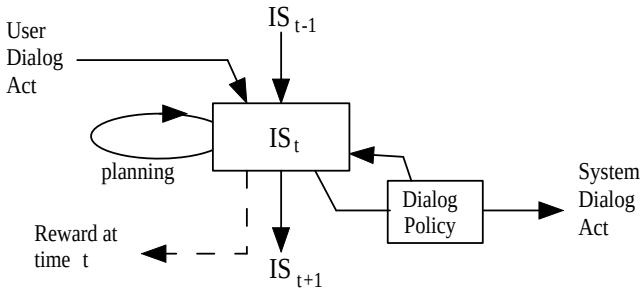

Figure 3.1: Basic Information State Update

A key requirement in the design of systems providing robust dialogue management is the ability to evolve dialogue policies which are in some sense optimal. This can be done by treating the dialogue as a Markov decision process (MDP) and using reinforcement learning to find the optimal policy [LP97, You00]. This can be done fairly straightforwardly for ISU-based dialogue systems by defining a reward function for each IS value.

3.3 Linear function approximation

The ISU framework is significantly different from the frameworks used in previous work on Reinforcement Learning (RL) for dialogue management in that the number of possible states is extremely large. This makes RL intractable, unless we can exploit commonalities between different states. The feature-based nature of ISU state representations expresses exactly these commonalities between states through the features that the states share. There are a number of techniques that could be used for RL with feature-based representations of states, but the simplest and most efficient is linear function approximation.

The core component of any RL system is the estimation of the expected future reward (the Q-value). The question arises: given a state and an action that could be taken in that state,[1] what total reward can we expect between now and the end of the dialogue? Once we have this function, the dialogue management policy reduces to simply choosing the action which maximizes the expected future reward for the current state.

We use linear function approximation to estimate the expected future reward. The input to the linear function is a vector of real values $f(s)$ for the state s. The trained parameters of the linear function are a

[1]The expected future reward also depends on the dialogue management policy which the system will use in the future. This self-referential nature of RL is the topic of much RL research, and will be discussed more below.

vector of weights w_a for each action a. An estimate $Q(s,a)$ of the expected future reward given a state s and an action a is the inner product of these two vectors[2] .

$$Q(s,a) = f(s)^T w_a$$

The weights w_a are learned from data, but the mapping $f(s)$ from states to vectors must be specified beforehand. Each value in these vectors represents a possible commonality between states, so it is through the definition of $f(s)$ that we control the notion of commonality which will be used by the linear function approximation. The definition of $f(s)$ we are currently using is a straightforward mapping from feature-value pairs in the information state s to values in the vector $f(s)$, as will be discussed in section 5.3, and as is shown in the appendix. One area of future research is to investigate more complicated mappings $f(s)$. This would bring us into the current research topic of kernel-based methods, which we hope to explore in future work. Kernels are used to compensate for the over-simplicity of linear functions, and can be used to express more complicated notions of commonality between states [STC04].

3.4 User Simulations

In order to generate enough training data for Reinforcement Learning, and also to evaluate different dialogue strategies against a reasonable baseline, an important idea is to build simulated users of dialogue systems. In chapter 5 we explain how we built 2 types of simulated user from annotated COMMUNICATOR data, and used them to evaluate different dialogue management policies.

3.5 Future work: Using Bayes Networks and POMDPs

3.5.1 Background

Modelling a spoken dialogue system as a Markov Decision Process (MDP) provides a convenient framework for dialogue policy optimisation. However, it depends rather crucially on making decisions based on full knowledge of the true information state. In practice, however, there are two major sources of uncertainty in a spoken dialogue system. Firstly, the user's goals are unknown and indeed, much of the dialogue strategy might be devoted to finding out what these goals are. Secondly, the user's speech acts are subject to a very noisy channel and the output of the speech understanding component will incur a high error rate. The net result of this is that in a conventional dialogue system, the information state records the system's "best guess" at the true underlying state. Each system dialogue action then depends entirely on this "best guess" without consideration as to what other possible states the system might be in. Crucial variables can be augmented with confidence values in order make decisions more sensitive to uncertainty, but this has limited effectiveness. As dialogues become more complex, the opportunity to misunderstand the user's intentions and set off down an unproductive track becomes ever more frequent.

One rather elegant solution to these problems is to maintain not just the single "best guess" of the information state but instead a distribution over dialogue states. This allows system decisions to be taken in the full knowledge of the current uncertainty. If there are many possible information states, the dialogue

[2]We will use the notation $x^T y$ to denote the inner product between vectors x and y (i.e. "x transpose times y").

will proceed cautiously focussing on resolving the uncertainty. If however there is a single state with probability close to one, then the dialogue can proceed rapidly to conclusion.

The extension of MDPs to deal with uncertainty in the state space leads to so-called Partially Observable MDPs (POMDPs). In the remainder of this project we will investigate methods of explicitly modeling uncertainty in dialogue system design. This is a very new area of research and there are many problems to solve. In particular, the use of distributions over complex state spaces results in severe tractability issues. These can be dealt with in a number of ways, but the main approaches are (a) factoring the state space such that only a small part is "uncertain"; and (b) limiting the distribution to a few "most likely" states. Solution methods can be either local n-step approximations or limited searches over the full span of the dialogue. In the former case, simple Bayes Net probability estimation may provide the most effective tool. In the latter, a full POMDP solution is required. The project plan separates these out as two distinct deliverables (D 4.3 and D 4.4) but in practice there will be considerable overlap.

3.5.2 Using the POMDP Framework with the ISU Approach

The key idea is to partition the Information State into three distinct sets of information:

s_u - the user's goal, this will be a choice from a finite set of possibilities.

a_u - the last user action. The uncertainty in this arises from the fact that the speech understanding components make errors. Typically, there is an N-best list of semantically distinct hypotheses each of which leads to a distinct information state.

s_d - details of the dialogue state, e.g. what is grounded, questions under discussion, history, etc. The dialogue manager computes this deterministically but since it depends on uncertain variables (in s_u and a_u, it is itself uncertain.

The overall information state s is then $[s_u, a_u, s_d]$.

As a dialogue progresses, a trellis of IS values is generated. Initial uncertainty derives from lack of information regarding the user's goal. At each dialogue turn, uncertainty in the decoding of the user's dialogue act (a_u) adds further uncertainty. Based on each possible combination of previous state and new values for s_u and a_u, the information state update rules will compute a new value for s_d. In some, cases this process will result in a conflict which reduces the probability of that state to zero.

The POMDP itself views the dialogue as a system with unobservable state s. As in a regular MDP, a reward $r(s,a)$ is assigned to every state action pair. The goal is then to find a policy Π which maximises the total expected reward

$$R = \sum_{t=0}^{\infty} \gamma^t \sum_{s} b_t(s) r(s, a_t)$$

where $b_t(s)$ is the state distribution at time t, and $a_t = \Pi(b_t)$ is the action taken when the distribution of states is b_t.

Some preliminary work has been completed using a very small scale dialogue problem which allows a complete and exact analysis. This work has verified the superiority of the POMDP framework compared to the MDP framework. The research questions that are now being addressed in TALK tasks 4.3 and 4.4 are

- how to represent b when $|s|$ is large?
- how to implement belief state estimation when $|s|$ is large?
- how to represent and optimise the policy $\Pi(b)$ when $|s|$ is large?

We now move on to discuss how to extract state features suitable for RL approaches from ISU dialogue systems.

Chapter 4

Extracting context features for learning

This chapter explains the general techniques we have developed for integrating ISU-based dialogue systems with machine learning methods. In particular we explain how to extract suitable features for learning from ISU-based dialogue managers. In general we require data that has either been generated and logged by ISU systems (e.g. WITAS [GL04a, MRL05]), or that has been subsequently annotated (or a mixture of both). In all these cases we specify certain types of information that should be logged or annotated (section 4.2). See status report T6.5s1 for details of the annotated data archive being developed for TALK , and in particular the initial project standards including data model and DTD for ISU-based dialogue annotations.

In this chapter we survey basic principles for annotating dialogue data with feature values for learning approaches. Chapter 5 later presents an example annotation of the COMMUNICATOR dataset and our results on applying learning techniques to this data.

4.1 Example: the DATE annotation scheme

We present the DATE scheme as an example of a starting point for dialogue act annotations. Other schemes could be adopted and extended, so long as they provide good coverage of the dialogue acts observed in a corpus. For example, when considering a corpus of tutorial dialogues, we could start with DATE and add appropriate dialogue act types such as "give-negative-feedback" and so on. The critical point is to adopt a coherent ontology of dialogue acts which covers the prominent phenomena in the data.

The DATE (Dialogue Act Tagging for Evaluation) scheme [WPB01] was developed for providing quantitative metrics for comparing and evaluating the 9 different DARPA COMMUNICATOR spoken dialogue systems, and for this reason is a good starting point for our project. The scheme employs three orthogonal dimensions of utterance classification:

- *conversational domain*: about-task, about-communication, situation-frame
- *task-subtask*: top-level-trip (origin, destination, date, time, airline, trip-type, retrieval, itinerary), ground (hotel, car)
- *speech act*: request-info, present-info, offer, ack, status-report, explicit-confirm, implicit-confirm, instruction, apology, opening/closing

The conversational domain dimension categorizes each utterance as belonging to a particular "arena of conversational action". About-task refers to the domain task (in COMMUNICATOR this is air travel booking), and about-communication refers to conversational actions managing the communication channel (e.g. "are you still there?"). Situation-frame utterances manage the "culturally relevant framing expectations" in the dialogue (e.g. that the conversation will be in english, or that the system cannot issue airline tickets).

The task-subtask dimension relates to a model of the domain tasks that the dialogue system is designed to support. In COMMUNICATOR there were 2 main tasks - booking a flight ("top-level-trip"), and "ground" which was to determine whether the user also wanted to book a car rental and/or a hotel. The subtasks were elements such as finding the date and time of the flight.

The speech-act dimension relates to the utterance's communicative goal. These are relatively standard, and are described in detail in [WPB01].

4.2 Annotation principles for ISU systems

The question arises of what types of information should ideally be logged or annotated for the purposes of optimising ISU dialogue systems via RL. After discussions with the TALK partners, we have developed the following initial specification, which has been used as a basis for the data model described in T6.5s1.

We can divide the types of information required into 5 main levels:

- dialogue-level
- task-level
- low-level
- history-level
- reward level

We should also divide the logging and annotations required into information about utterances, and information about states. Utterances (by humans or systems) will have dialogue-level, task-level, and low-level features, while dialogue states will additionally contain some history-level information. Entire (completed) dialogues will be assigned reward features.

Dialogue-level features

For each utterance we require the following features at the dialogue level :

- dialogue act (perhaps along the lines of the DATE scheme, see above)
- ASR hypothesis
- Transcription of user input
- System intention

- System output string
- changes to issues/questions under discussion
- changes to common ground
- changes to obligations
- speaker
- turn number
- modality
- salient NPs
- salient verbs
- syntactic parse
- semantic parse
- Concept error rate

Task-level features

For each utterance we require the following information about the task level:

- task/subtask-type
- number of database query results (for slot-filling/information-seeking dialogues)
- changes to filled slot values
- confidence in each slot (e.g. low/medium/high for restaurant-type)
- user goal(s)

Low-level features

For each utterance (including multimodal inputs) we require the following logging of low-level features:

- start/end time/duration
- minAmp, meanAmp, RMSamp of speech signal
- Nbest ASR hypotheses
- Word error rate
- confidence zScore, confidence SD, minimum confidence
- per hypothesis: ASR confidence score

- per hypothesis: hypothesis length (words)
- ASR confidence rejection threshold
- ASR timeout setting
- signal amplitude
- word-based confidence scores
- input modality
- available output modalities
- XY co-ordinates of mouse gestures
- n-best list of objects referred to graphically
- environmental features (e.g. performance on primary driving task)

History-level features

The history-level annotations/logs of states should be computable from the above information over the history of utterances. Therefore these features can be expanded after the annotation/logging is completed. With some dialogue managers this can be achieved by logging whole Information States. As an initial set of history-level features, for each state of the dialogue we suggest the following:

- cumulative filled slot values
- cumulative issues/questions under discussion
- cumulative common ground
- cumulative obligations
- system intentions/agenda
- user goals history
- turn assignment
- last N user dialogue acts
- last M system dialogue acts
- number of clarifications
- number of implicit/explicit confirmations
- number of abandoned (sub)tasks
- dialogue duration

- number of turns
- average/min/max ASR confidence scores
- average/min/max Word Error Rates
- average/min/max Concept Error Rates
- last X system modalities
- last Y user modalities
- update rule(s) fired by the dialogue manager

Reward level features

The reward features for each dialogue may be gathered either by online questionnaires, paper questionnaires, or may even be computed from individual user actions (e.g. hang up). We suggest the following:

- task completion (actual and perceived)
- dialogue duration
- number of turns
- user satisfaction
- PARADISE evaluation metrics (see [WKL00] and chapter 5 for examples)

4.3 Using logs/annotations for Reinforcement Learning

A constraint on the information to be useful for machine learning is that all captured features should in principle be available to a dialogue system at runtime – so that a dialogue system can compute what to do next in any state. This excludes, for example, word-error rate from the state information usable for RL, since it can only be computed after transcription of user speech. (In this case, for example, ASR confidence scores should be used instead.)

A key requirement is also that systems should be able to explore different actions in the same states. Without exploration of the policy space, the reward signal will not vary as required. For example we should consider building systems that can try different multimodal output strategies. These system actions will be logged at the dialogue level.

We now present an application of these ideas, which proves the feasibility of our approach.

Chapter 5

Learning about COMMUNICATOR strategies

This chapter presents a concrete application of the general techniques described in chapters 3 and 4. In particular we describe how we automatically annotated the COMMUNICATOR data using an extended version of the DATE annotation scheme to produce sequences of dialogue information states that feed into reinforcement learning algorithms as well as supervised learning methods.

Here we also present our user simulation techniques, and our initial results – a learnt dialogue strategy which shows significant performance gains over comparable COMMUNICATOR system dialogues.

5.1 The UEDIN COMMUNICATOR annotation

The process of annotating the COMMUNICATOR data with information states has been implemented using DIPPER [BKLO03] and OAA [CM01]. Several OAA agents have been developed to accomplish this task (see Figure 5.2). The flow chart of the automatic annotation task is depicted in Figure 5.1.

Figure 5.2 illustrates the components of the automatic annotation system.

The first OAA agent (`readXMLfile`) is used for reading the COMMUNICATOR XML file, which contains all the information about dialogues, turns, utterances, transcriptions, and so on. Each time the agent reads some information from the XML file, a corresponding DIPPER update rule is triggered and the current information state is updated accordingly.

Each information state corresponds to an utterance in the COMMUNICATOR data. A turn may contain one or more utterances. Most of the time one turn includes utterances of the same speaker. There are cases though in which both the user and the system speak within the same turn. This does not cause problems in annotation since everything is computed on the utterance level. Only the COMMUNICATOR 2001 data contains timing information.

A second OAA agent (`saveISsequence`) appends the current information state values to the file that will finally contain the whole sequence of information states.

The `readXMLfile` agent reads the XML file line by line and updates the current information state. Considering that some lines of the XML file indicate the beginning or end of a turn or utterance and therefore do not contain system or user actions, some kind of merging is required. This is applied at the end,

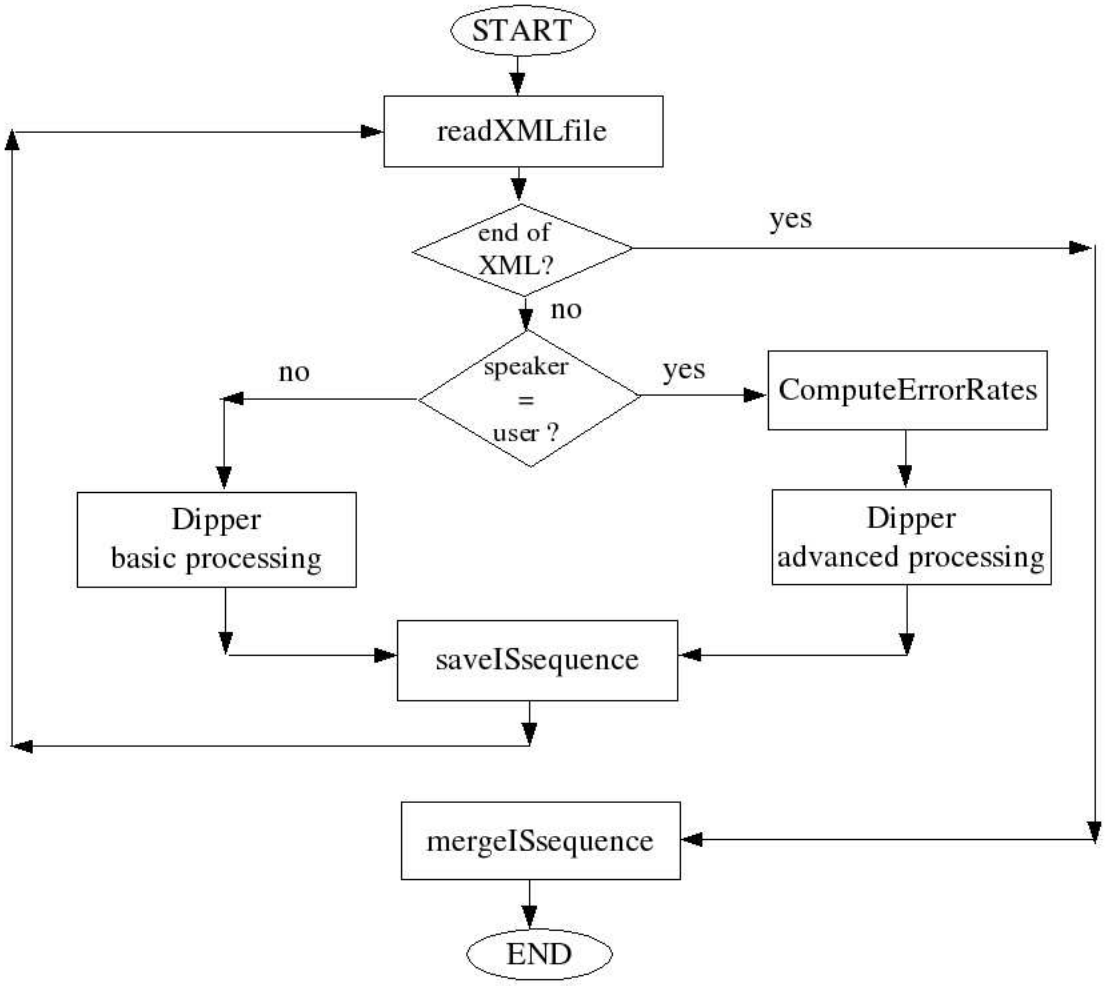

Figure 5.1: The flow chart of the automatic annotation task

after every line of the XML file has been read and processed, and thus the resulting file contains only information states that correspond to system or user actions. The merging process is performed by the `mergeISsequence` function at a post-processing level, separately from the OAA architecture.

The COMMUNICATOR data contains both the output of the speech recognition engine for a user utterance and a manual transcription of the same utterance carried out by a human annotator. However, it does not provide speech recognition confidence scores. The Word Error Rate (WER) is considered to be related to confidence scores and thus each time a user utterance is read from the XML file a third agent is called to estimate error rates (the `ComputeErrorRates` agent). Four different error rates are estimated. The classical WER is based on the following formula:

$$WER = 100\left(\frac{Nins + Ndel + Nsub}{N}\right)\%$$

where N is the number of words in the transcribed utterance, and Nins, Ndel, Nsub are the number of insertions, deletions and substitutions respectively in the speech recognition output. WER-noins, which is

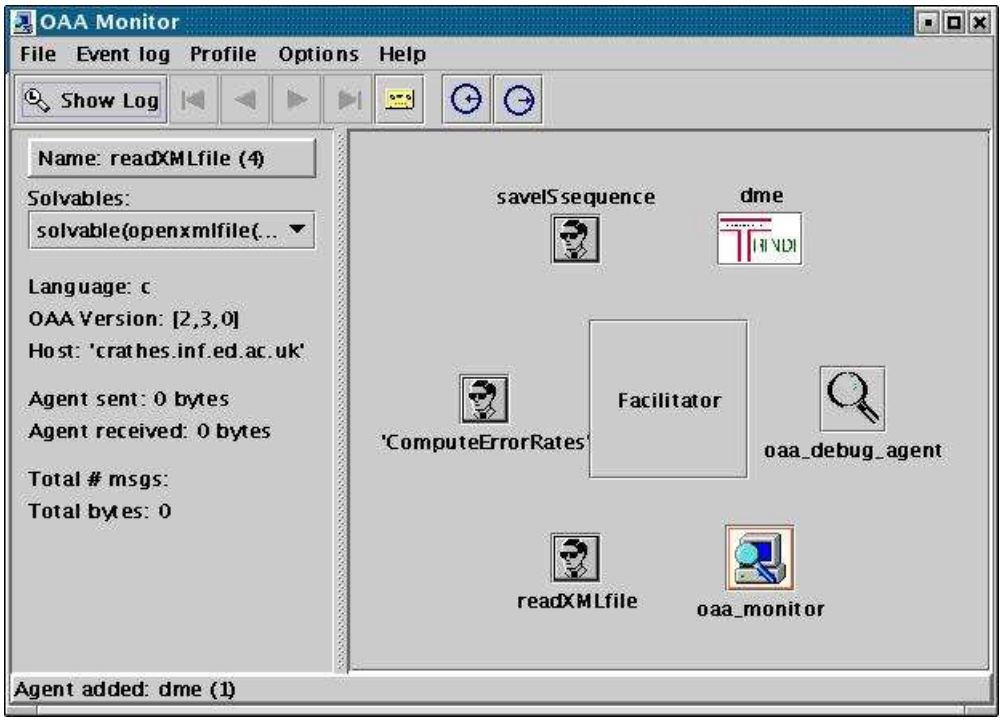

Figure 5.2: The automatic annotation system viewed using the OAA monitor

WER without taking into account insertions is also computed. The distinction between WER and WER-noins is made on the following basis. WER shows the overall recognition accuracy whereas WER-noins the percentage of words correctly recognised. The sentence error rate (SER) is computed on the whole sentence. That entails that the speech recognition output is considered to be correct only if it is exactly the same as the manually transcribed utterance. All the above error estimations have been performed using the HResults tool of HTK [YEK+02], which is called by the `ComputeErrorRates` OAA agent. Finally the keyword error rate (KER) is also computed by `ComputeErrorRates` and shows the percentage of the correctly recognised keywords. This is also a very important metric regarding the efficiency of the dialogue.

It should be noted that speech phenomena such as pauses, fillings, noises, etc. that are transcribed by human annotators are not taken into account when error rates are estimated because most speech recognisers do not include them in their outputs, even though they are considered by their acoustic models. Therefore if such phenomena were included while estimating errors there wouldn't be a fair comparison between the speech recognition output and the human transcription of the utterance.

Even though all the above agents that are all implemented in C++ play a crucial rule in the annotation task, the heart of the system is part of the DIPPER resources file and is written in Prolog. The most important subtask of the annotation procedure is to translate the user's input and find its effect on the dialogue, or in other words to associate the user's utterance with the correct speech acts and tasks. This is not trivial at all, therefore multiple levels of parsing are required and are performed using Prolog clauses.

In the COMMUNICATOR data the system's side of dialogue is already annotated using the DATE scheme. The role of the Prolog clauses is to do the same for the user's side. The user utterances (both speech recognition output and human transcription) are enriched with tags indicating the semantics of some words, e.g.

cities, airlines, etc. This, in addition to the fact that user utterances tend not to be very complex in this type of dialogue, facilitates parsing and thus a keyword-based parser seems to be adequate for the first level of parsing in our task.

The different tasks and speech acts used for annotating system utterances are described in detail in [WPB01]. We use the following tasks and speech acts for annotating user utterances in addition to the ones used for the system prompts' annotation:

Tasks which get values:

```
orig_city, continue_orig_city, dest_city, continue_dest_city, depart_date,
continue_depart_date, return_date, depart_time, continue_depart_time, return_time,
arrive_time, continue_arrive_time, return_arrive_time.
```

Note that orig_city and dest_city are also used in the annotation of the system utterances.

Tasks which do not get values, and are either present or absent:

```
continue_trip, no_continue_trip, return_trip, no_return_trip,
accept_hotel_offer, reject_hotel_offer, accept_flight_summary,
reject_flight_summary, accept_car_offer, reject_car_offer, start_over.
```

Speech acts:

```
provide_info, reprovide_info, correct_info, reject_info, yes_answer, no_answer.
```

Note that each utterance may involve multiple tasks and each task could include multiple speech acts. In the COMMUNICATOR XML file there is no distinction between the origin city and destination city in multiple-leg trips. That is, the tag "dest_city" could be used for any type of destination, regardless of whether the trip is single, return or multiple-leg. However, we believe that it is important to annotate these distinctions so that there is no overwriting of the values in filled slots such as "dest_city", "depart_date", etc. Moreover, the COMMUNICATOR data does not distinguish between departure and arrival dates or times, and sometimes it has a date labelled as time.

During processing the user's utterance the automatic annotation system will take into account the history of the dialogue and the previous system utterances' labelling and thus attempt to decide whether one or more slots should be filled, grounded or even emptied if the slot is not confirmed by the user. We define a piece of information as "grounded" only if it has been positively confirmed. Thus grounding processing takes place only after system utterances labelled as explicit or implicit confirmation. Here one could argue that computing grounding information is based on the assumption that the COMMUNICATOR systems had the notion of grounding incorporated into their structure, but this kind of information is not included in the COMMUNICATOR corpus. Nevertheless, the fact that we attempt to ground a slot only when the system asks for confirmation makes our assumption "safe". We use the terms "grounding" and "positively confirmed" as synonymous. Moreover, only the speech recognition output is used for processing and deciding on the slots that will be filled or grounded. The human transcription of the user's input is only considered for computing error rates as explained above. This also ensures that we do not base our annotation on information that the systems did not have in runtime.

Let's examine an extract taken from the COMMUNICATOR 2001 data. System utterances are marked with "S(n)" and user utterances as "U(n)" where n is the number of the utterance. For the system utterances the speech act and task pairs are given, for the user utterances only the speech recognition output is provided:

```
(S1) what city are you leaving from? (request_info, orig_city)
(U1) <CITY>hartford connecticut</CITY>
(S2) a flight from <CITY>hartford</CITY>. (implicit_confirm, orig_city)
(S3) where would you like to go? (request_info, dest_city)
(U2) <CITY>orlando florida</CITY>
(S4) traveling to <CITY>orlando</CITY>. (implicit_confirm, dest_city)
(S5) on what date would you like to travel? (request_info, depart_arrive_date)
(U3) <DATE_TIME>october three first late morning</DATE_TIME>
(S6) traveling <DATE_TIME>late morning</DATE_TIME>. (implicit_confirm, depart_arrive_date)
(S7) on what date would you like to travel? (request_info, depart_arrive_date)
(U4) <DATE_TIME>october thirty one</DATE_TIME>
...
```

In utterance (U3) the user gives the departure date and time. However, the speech recognition output "october three" is not a valid date so the system understands only the time "late morning" and tries to confirm it in (S6). As we see in (S6) the speech act is implicit confirmation and the task is tagged as "depart_arrive_date" instead of "depart_arrive_time". Similar phenomena are bound to cause problems to correctly annotating user utterances. In the above example, in (U3) the automatic annotation system will fill slot "depart_time" with the value "late morning". Then it will read the next system utterance, in our case (S6) and if it is based only on the task label (depart_arrive_date) it will attempt to ground the wrong slot "depart_date" or in other words it will try to ground a slot that has not been filled yet. Therefore routines have been implemented so that the automatic annotation system can distinguish between valid dates or times. Note also in (U3) that both the date and time are included in the same tag, which is also another reason the annotation system should not entirely rely on system utterances' tagging.

Suppose that we have another example now. The user in (U3) does not give the departure date but instead only replies to the confirmation prompt about the destination city. There are 6 general ways the user could reply:

1. yes-class, e.g. "yes"
2. no-class, e.g. "no"
3. yes-class, city, e.g. "yes, orlando"
4. no-class, city, e.g. "no, boston"
5. no-class, city, city, e.g. "not orlando, boston"
6. city, e.g. "orlando" or "boston"

where "yes-class" corresponds to words or expressions like "yes", "okay", "right", "correct", etc. In the same way "no-class" stands for "no", "wrong", etc.

In the first 5 cases it is easy for the annotation system to infer that there is positive or negative confirmation and thus ground the slot or not accordingly because of the appearance of “yes-class” or “no-class”. However, in case (6) the annotation system should compare the user’s utterance with the previous system’s prompt for confirmation in order to decide whether the slot should be grounded or not. If the user says “orlando” he reprovides information and the slot “dest_city” is grounded whereas if he/she utters “boston” he/she corrects the system, which means that the slot “dest_city” is not grounded and therefore its current value will be removed. In case (5) the user rejects the value of the slot and corrects it at the same time. The above case is another example of the difficulty and complexity of the task of automatically annotating the COMMUNICATOR corpus. The annotation system does not allow a slot to be grounded unless it has already been filled and updates the history of filled slots and grounded slots accordingly.

In general all the COMMUNICATOR systems follow the confirmation strategies depicted in Figure 5.3.

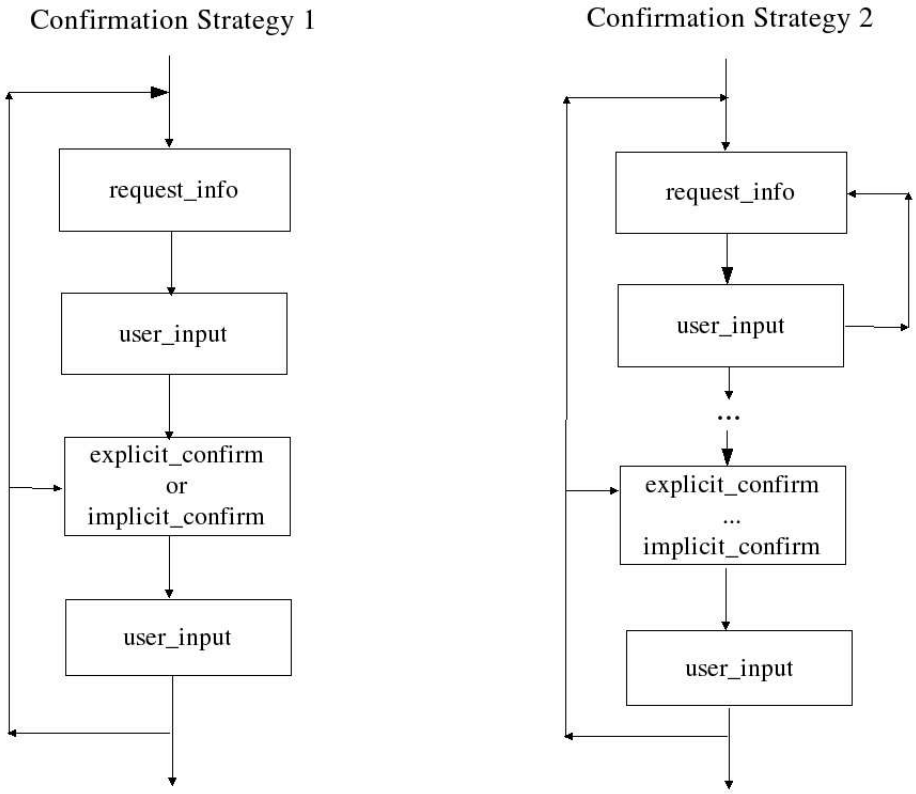

Figure 5.3: Confi rmation strategies in COMMUNICATOR systems

In the first strategy the system asks the user to fill a slot, then it asks for confirmation (explicit or implicit) and moves to the next slot if the user confirms or may keep asking for confirmation if the user does not cooperate. In the second strategy the system asks the user to fill several slots and then attempts to confirm them in one single step. That means that the system’s turn could consist of several utterances labelled as “explicit_confirm” or “implicit_confirm”. Before confirmation the slots could be filled either in a single turn or in multiple turns.

For confirmation strategy 1 it is proved adequate to look only 1 or 2 steps backwards in the history of system utterances. We consider only the following speech acts: request-info, explicit-confirm, implicit-

confirm and offer. Other utterances that are labelled e.g. as instructions are not taken into account because they do not affect whether a slot will be filled or grounded. Thus in order to decide which slots should be filled and/or grounded each time the following pattern is used:

```
patterns(continue, return, previous speech acts, previous tasks,
parsing result of user's input, list of resulting filled and/or
grounded slots, new continue, new return, list of resulting speech
acts).
```

Here is an example pattern:

```
patterns(0,1,[implicit_confirm,_],[dest_city,_],[date,time],[grounded_orig_city,
return_date,return_time],1,0,[provide_info]).
```

The variables "continue" and "return" show whether we are processing a multiple-leg or return trip so that the corresponding slots are updated. In the above example we have a return trip. The parsing result of the user input is actually a list of tags that carry the semantic representation of the input, in this case the user gave a date and a time. The above rule "says" that if the system asked for an implicit confirmation of the destination city and then performed any other action, and the user gave a date and a time it means that he/she accepted the value of the destination city which is therefore grounded and also filled slots "return_date" and "return_time". The reason we pick "return_date" and "return_time" and not "depart_date" or "depart_time" is the fact that we are processing a return trip. The variables "new continue" and "new return" will remain the same since nothing indicates any change in the type of trip. Finally the resulting speech act is "provide_info". In practice the pattern contains more information which is omitted here to make the idea easier to understand. At a later stage the slot that will eventually be filled is associated with the corresponding value.

The second dialogue strategy however cannot be dealt with patterns like the one described above because there could be too many combinations and thus it would be impossible to keep track of patterns. Therefore a more sophisticated approach is required. The basic idea of this approach is that the annotation system will parse the user's input and check whether it contains information on tasks included in the previous series of confirmation requests. If it does then this is an indication that the user objects to the value given by the system to a filled slot. It could also be the case however that the user simply reprovides some information. It is obvious that multiple levels of processing are required again to make sure that the correct slots are filled and/or grounded.

Here are the information state fields. This uses an extension of the DATE annotation scheme.

The Information State fields are as follows:

- Information State Number

DIALOGUE LEVEL

- Turn
- Turn Start Time
- Turn End Time

- Turn Number
- Speaker
- Utterance Start Time
- Utterance End Time
- Utterance Number
- Conversational Domain
- Speech Act
- ASR Input
- Transcription Input
- Output

TASK LEVEL

- Task
- Filled Slot Value
- Filled Slot
- QUD
- Common Ground
- Obligation
- System Intention
- User Goal

LOW LEVEL

- Word Error Rate-noins
- Word Error Rate
- Sentence Error Rate
- Keyword Error Rate
- Comment

HISTORY LEVEL

- Speech Acts Hist

- Tasks Hist
- Filled Slots Values Hist
- Filled Slots Hist
- QUD Hist
- Common Ground Hist
- Obligations Hist
- System Intentions Hist
- User Goals Hist

Here is an example annotated Information State from the UEDIN COMMUNICATOR data (the data in the TALK annotated data archive[1] is in fact marked up with XML tags conforming to the project DTD).

```
STATE 16

DIALOGUE LEVEL
Turn: user
TurnStartTime: 988306684.960
TurnEndTime: 988306687.810
TurnNumber: 6
Speaker: user
UtteranceStartTime: 988306684.960
UtteranceEndTime: 988306687.810
UtteranceNumber: 6
ConvDomain:
SpeechAct: [provide_info]
AsrInput: <date_time>october thirty one</date_time>
TransInput: <date_time>october thirty one</date_time>
Output:

TASK LEVEL
Task: [depart_date]
FilledSlotValue: [october thirty one]
FilledSlot: [depart_date]
QUD:
CommonGround: [depart_time]
Obligation:
SystemIntention:
UserGoal:

LOW LEVEL
```

[1] Available via the project CVS repository.

```
WordErrorRatenoins:  0.00
WordErrorRate:  0.00
SentenceErrorRate:  0.00
KeyWordErrorRate:  0.00
Comment:

HISTORY LEVEL
SpeechActsHist: [],opening_closing,[],opening_closing,instruction,request_info,
[provide_info],implicit_confirm,request_info,[provide_info],
implicit_confirm,request_info,[provide_info],implicit_confirm,
request_info,[provide_info]
TasksHist: [],meta_greeting_goodbye,[],meta_greeting_goodbye,
meta_instruct,orig_city,[orig_city],orig_city,dest_city,[dest_city],
dest_city,depart_arrive_date,[depart_time],depart_arrive_date,
depart_arrive_date,[depart_date]
FilledSlotsValuesHist: [null],[],[],[hartford connecticut],
[orlando florida],[october three first late morning],
[october thirty one]
FilledSlotsHist: [null],[],[],[orig_city],[dest_city],
[depart_time],[depart_date]
QUDHist:
CommonGroundHist: [],[],[],[orig_city],[dest_city],[depart_time]
ObligationsHist:
SystemIntentionsHist:
UserGoalsHist:
```

REWARD LEVEL

Here is an example reward annotation (at the end of the sequence of dialogue states):

```
pin="10081" system="att" date="01/05/2001" time="153100"
Actual Task Completion="0"
Perceived Task Completion ="0"
Task Ease="1"
Comprehension Ease="2"
User Expertise="4"
System behaved as Expected="2"
Future Use="1"
```

5.1.1 Evaluating the automatic annotations

Our next step is to evaluate our automatic annotation system by comparing its output with the actual (ATC) and perceived (PTC) task completion metrics as they are given in the COMMUNICATOR corpus. If the final state of a dialogue – that is, the information about the filled and grounded slots – agrees with the ATC amd PTC for the same dialogue, this indicates that the annotation is consistent with the task completion metrics. If however, there is no consistency between the results of annotation and ATC and PTC in

the corpus then we need to further improve our annotation system. Apart from this planned automatic evaluation, manual evaluation will also be performed by humans. Thus humans will check the correctness of the sequence of information states produced by the annotation system. We have performed such an analysis for a small number of dialogues only informally at the moment, during system development, with relatively good results. As will be described in the sequel, the first results of our supervised and reinforcement learning techniques trained with the automatically annotated data are promising, which also indicates that a significant number of dialogues have been annotated correctly.

5.2 The COMMUNICATOR user and system simulations

5.2.1 The simulation environment

The basic idea behind user simulation is that it is feasible to test the performance of automatically learned strategies against simulated users in an efficient and inexpensive way. Otherwise we would need to have real users which would not be cost-effective and would require much more time and effort. In addition, every time we modified our strategies we would have to repeat all experiments with human users from scratch. Another reason that makes simulation very interesting and useful is that it can provide us with a huge amount of data for retraining and improving our automatic learning strategies.

The environment used for simulating both the system and the user is based on OAA architecture and DIPPER and can support various agents for system or user simulation. For system simulation we use either Reinforcement Learning (RL) or n-grams. For user simulation supervised learning or n-grams are used. Currently the environment supports the following combinations:

1. system based on RL vs. user based on supervised learning
2. system based on RL vs. user based on n-grams
3. system based on n-grams vs. user based on supervised learning
4. system based on n-grams vs. user based on n-grams

Of course n-grams are also a type of supervised learning, however, their implementation is different from the one we call “supervised learning” and that is why we distinguish between them. More details are given in the subsequent sections.

The system architecture is modular and therefore it is easy to incorporate other agents based on different techniques for simulating either the system or the user. It can be done by slightly modifying some DIPPER rules. Figure 5.4 shows the OAA agents of the simulation environment.

`RLsimulation agent`: this acts as an interface between the simulation environment and the system simulation based on RL or the user simulation based on supervised learning and is used in cases (1), (2), and (3).

`saveIS agent`: this passes the current information state to the RL simulation agent. The reason is that both the system simulation based on RL or the user simulation based on supervised learning require as input the current information state so that they decide on their next action. Again this agent is used only in cases (1), (2), and (3).

`saveISfinal agent`: this saves the final state of every simulated dialogue so that it can be used for evaluation.

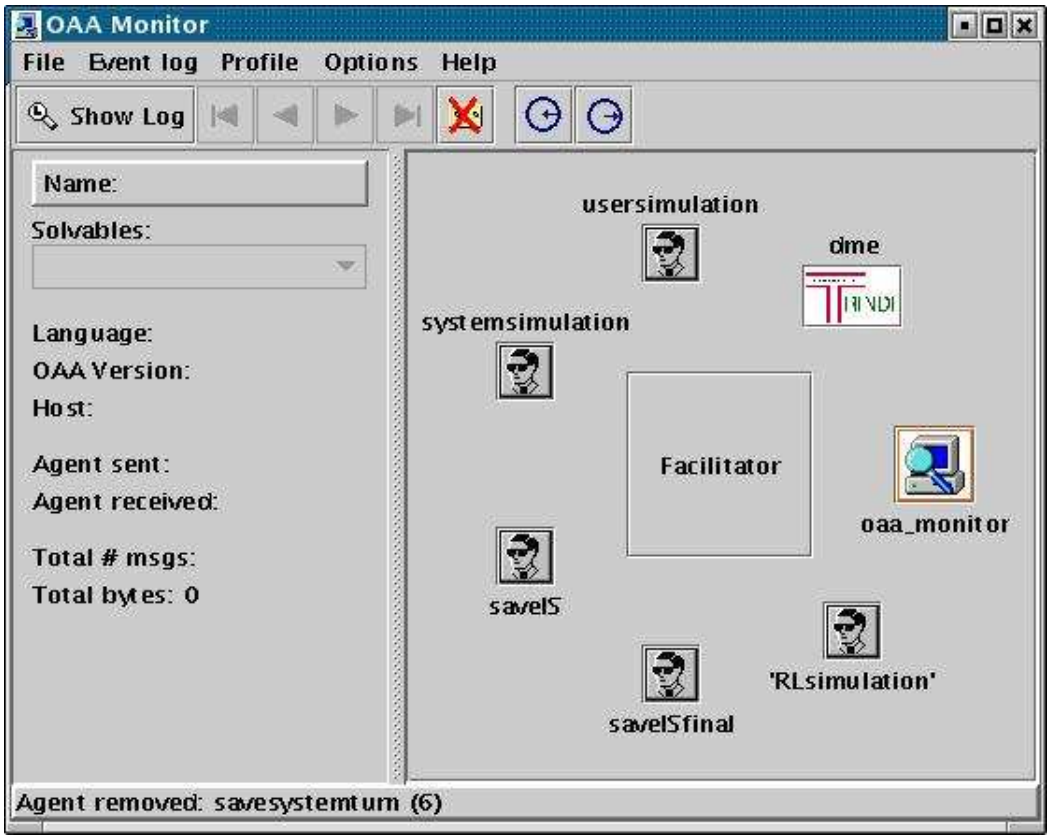

Figure 5.4: The simulation system

`systemsimulation` and `usersimulation` agents: they act as an interface to the system simulation based on n-grams and the user simulation based on n-grams respectively.
`DIPPER`: this component is responsible for controlling interaction between agents, keeping track of dialogue history and calculating grounding. The agents for system or user simulation that are based on n-grams require as input the history of speech act-task pairs in order to decide on their next action. However, the agents that use RL or supervised learning require a much more detailed representation of the information state. They have been trained with sequences of information states that include information on grounding and taht have been produced by the automatic annotation system. Therefore in order to ensure their high performance they should be provided with grounding information in simulation too. Thus DIPPER calculates grounding in a similar way as in the automatic annotation system. The difference is that it does not need to parse the user input to infer the speech act and task but instead gets them from the simulation agents.

Simulating ASR errors

Moreover, during training the RL or supervised learning methods used information about the word error rate. Therefore in simulation mode DIPPER generates word error rates for each user action based on the distribution of word error rates in the COMMUNICATOR data. For about 70% of the user utterances the speech recognition performance was perfect (WER=0%), for 10% of the user utterances the speech

recogniser produced a WER of 100% and for the rest of the user utterances WER values varied between 0 and 100%. In the future we intend to use sentence error rates and keyword error rates as well, both during training our automatic learning techniques and in simulation, since they are already included in the information states produced by the automatic annotation system.

5.2.2 The n-gram simulation

The simulation based on n-grams treats pairs of speech acts and tasks as n-grams. Thus once it gets as input a history of speech act-task pairs it will decide on the next action (speech act-task pair).
For both the "systemsimulation" and "usersimulation" OAA agents there are 3 solvables: `initsimulationsystem`, `callsimulation`, and `freesimulationsystem`:

- `initsimulationsystem(Speaker, Nvalue, VocabFile, IdngramFile, Result)`:

  this function initialises n-grams. It takes as input the type of simulation (system or user), the value of n in the n-grams, the name of the vocabulary file, the names of the files containing the n-grams and returns 0 when everything is correctly initialised or 1 if an error appears.

- `callsimulation(FirstTimeFlg, History, ResSpeechact, ResTask, ResContinue)`:
  this function gets as input the history of speech act-task pairs and produces as output a new speech act and task, that is, a new action that will be passed to DIPPER. If there is not an n-gram for the current history the function will back-off to lower n-grams. Nevertheless in system simulation mode backing-off is avoided because it leads to completely random dialogues. Actually it is permitted only in the first action of a turn (`FirstTimeFlg=1`) to ensure that the system will always produce an action. The `ResContinue` variable shows whether the system will continue to produce a new action in the same turn or it will release the turn. The user always releases the turn after one action.

- `freesimulationsystem(Result)`:
  this frees memory and again returns 0 in success and 1 in failure.

Both the system simulation and the user simulation agents use the same functions as described above as well as the same n-grams. The variable "Speaker" distinguishes between them and picks during initialisation only the appropriate n-grams, that is, the n-grams that lead to system actions for system simulation and to user actions for user simulation. The system can perform multiple actions in the same turn, while the user can only perform one action.
Based on the annotation of the COMMUNICATOR data as a sequence of information states, the CMU-Cambridge Statistical Language Modelling Toolkit v2 [CR97] is used for generating n-grams. If the length of history is equal to n then we generate n-grams starting from unigrams to (n+1)-grams.

5.2.3 The linear function approximation user simulation

For a second user simulation, we use supervised learning with linear function approximation. The user is modelled as a stochastic process which selects user actions based only on the current state of the dialogue. The probability that the model will choose an action a given a state s is computed using the normalized exponential function applied to a linear function of the state vector.

$$P_{user}(a_i|s) \approx \frac{\exp(f_{user}(s)^T w_{user,a_i})}{\sum_j \exp(f_{user}(s)^T w_{user,a_j})}$$

The normalized exponential is used to ensure that the result will be a probability distribution over actions a.

Because the information relevant to choosing a user action may be different from that relevant to choosing a system action, the mapping from states to vectors $f_{user}(s)$ is in general different from that used for system actions. This mapping is computed in the same way as discussed for system actions in section 5.3, but using a different set of feature-value pairs, as specified in the appendix A.

The weights w_{user,a_i} are trained from the same data used to train the reinforcement learning model. For every user action in the data, we compute the information state which preceeds that action, and train the user model to try to predict that action given the state.

5.3 Using the data for Reinforcement Learning

We use the annotated COMMUNICATOR data to train a Reinforcement Learning system. This involves defining a mapping $f(s)$ from an information state s to a vector of real values, and defining a mapping $r(d,i)$ from a dialogue d and a position in that dialogue i to a reward value. Given these two definitions, we can train a linear function to estimate the expected future reward given a state and an action.

$$E[\sum_{i>j} r(d,i)] \approx Q_{data}(s_j,a) = f(s_j)^T w_a$$

The reward function $r(d,i)$ is computed using the reward level of annotation. For all states other than the final state, we provide a reward of -1. This encodes the idea that, all other things being equal, short dialogues are better than long ones. For the final state we provide a reward which is the sum of the rewards for each feature in the reward annotation. "Actual Task Completion" and "Perceived Task Completion" are both worth a reward of 100 if they are non-zero, and 0 otherwise. The remaining reward features have values ranging from 0 to 5. Their reward is their value times the weight shown in table 5.1. The relative values of these later weights was determined by the empirical analysis reported in [WAB+01].

The state vector mapping $f(s)$ is computed using the first four levels of annotation. We went through these annotations and identified the features which, initially, we consider relevant for dialogue management. These features were of three types. For features which take numbers as values, we used a simple function to map these numbers to a real number between 0 and 1, with the absence of any value being mapped to 0. For features which can have arbitrary text as their values, we used 1 to represent the presence of text and 0 to represent no value. The remaining features all have either a finite set of possible values, or a list of such values. Features with a list value are first converted to a list of pairs consisting of the feature and each value. For every possible feature-value pair, we define an element of the vector $f(s)$ which is 1 if that feature-value pair is present in the state and 0 if it is not. These form the vast majority of our features. The complete list of feature-value pairs which we used are given in appendix A.

Actual Task Completion	100
Perceived Task Completion	100
Task Ease	9
Comprehension Ease	7
System behaved as Expected	8
Future Use	9

Table 5.1: The weights used to compute a dialogue's final reward value. The first two features' weights are multiplied by 0 or 1, and the rest are multiplied by up to 5.

5.3.1 A Hybrid approach

We initially tried using the estimate of expected future reward $Q_{data}(s,a)$ discussed in the previous section to define our dialogue policy. The dialogue policy simply selected the action a with the highest $Q_{data}(s,a)$ given the state s. However, we found that this policy was very different from the policy observed in the COMMUNICATOR data, almost never choosing the same action as was in the data. This simply means that the actions which have been learnt to have the best future rewards are not the ones that were typically chosen by the COMMUNICATOR systems in those states. Such actions would then lead to states unlike anything observed in the data, making the estimates for these states highly unreliable. In addition, the future reward depends on the policy the system uses in the future, so if the policy is different from that observed in the data, then the estimate $Q_{data}(s,a)$ is not even relevant.

The solution to these problems which is typically used in RL research is to generate new data as learning progresses and the policy changes. The RL system can thus explore the space of possible states and policies, generating new data which is relevant to each explored policy and its states. In future research we intend to use our user simulations to allow such exploration, but initially we want to have a solution to the problem of applying RL to a fixed set of data. Having only a fixed set of data is unusual in RL research, where exploration is often considered an integral part of RL. However, this situation is the normal case in other areas of machine learning, because data is generally expensive to produce, and is usually produced by someone other than the machine learning researcher. After all, a user simulation is not the same as a human user, and generating real data with human users is very expensive.

There have been some proposals for learning a policy which is different from that used to generate the data (called off-policy learning), but these methods have been found not to work well with linear function approximation [SB98b]. They also do not solve the problem of straying from the region of state space which has been observed.

We have investigated a novel hybrid approach which combines RL with supervised learning. The supervised learning is used to model the policy which the systems in the data actually use, which we model as a probabilistic policy.

$$P(a|s) \approx S_{data}(s,a)$$

The function $S_{data}(s,a)$ is computed with linear function approximation, just like $Q_{data}(s,a)$, except that a normalized exponential function is used so that the result is a probability distribution over actions a, just like the user simulation in section 5.2.3.

The hybrid approach we have investigated is based on the assumption that we can't model the expected future reward for states in the unobserved portion of the state space. Thus we simply specify a fixed reward

for these unobserved states. By setting this fixed reward at a low level, it amounts to a penalty for straying from the observed portion of the state space. We then estimate the expected future reward as a mixture of the fixed reward U for unobserved states with the $Q_{data}(s,a)$ estimate for observed states, where the mixture coefficient is the probability that performing a in s will lead to an observed state.

$$E[\sum_{i>j} r(d,i)] \approx Q_{data}(s,a)P_{observed}(s,a) + U(1 - P_{observed}(s,a))$$

We do not actually try to compute this estimate, but instead try to compute estimates which have the same relative value for different a and a given s, since this is all that is needed to choose a. After some approximation, we get the following future reward function.

$$Q_{hybrid}(s,a) = Q_{data}(s,a)S_{data}(s,a) + U(1 - S_{data}(s,a)) \approx Q_{data}(s,a)P(a|s) + U(1 - P(a|s))$$

We use this $Q_{hybrid}(s,a)$ function to choose the actions for our hybrid policy. By adjusting the value of the unobserved state penalty U, we can adjust the extent to which this model follows the supervised policy defined by $S_{data}(s,a)$ or the reinforcement learning policy defined by $Q_{data}(s,a)$.

5.4 Experimental Results

We evaluate our trained dialogue management policies by running them against our trained user simulations. All these models were trained using the annotated COMMUNICATOR data for the ATT, BBN, CMU, and SRI systems. We compare our results against the performance of these same four systems, using an evaluation metric discussed below.

5.4.1 The testing setup

For these initial experiments, we restrict our attention to users who only want single flight-leg bookings. This means there are only 4 essential slots to be filled: origin city, destination city, departure date, and departure time. To achieve this restriction, we first selected all those COMMUNICATOR dialogues which did not contain trip continuations. This subset was used for evaluating the systems and for training the user model. The system model was trained on the full set of dialogues, since it should not know the user's goals in advance.

When we first tested the hybrid policy, we found that it never closed the dialogue. We think that this is due to the system action (annotated in DATE) "meta_greeting_goodbye", which is used both as the first action and as the last action of a dialogue. The hybrid policy expects this action to be chosen before it will close the dialogue, but the system never chooses this action at the end of a dialogue because it is so strongly associated with the beginning of the dialogue. This is an example of the limitations of linear function approximation, which we plan to address by splitting this action into two actions, one for "greeting" and one for "goodbye". In the meantime, we have augmented the hybrid policy with a rule which closes the dialogue after the system chooses the action "offer", to offer the user a flight. We have also added rules which close the dialogue after 100 states (i.e. total of user and system actions), and which release the turn if the system has done 10 actions in a row without releasing the turn.

5.4.2 The evaluation metric

To evaluate the success of a dialogue, we take the final state of the dialogue and use it to compute a scoring function. We want the scoring function to be similar to the reward we compute from the quality measures provided with the COMMUNICATOR data, but because we do not have these quality measures for the simulated dialogues, we cannot use the exact same reward function. When we compare the hybrid policy against the COMMUNICATOR systems, we apply the same scoring function to both types of dialogues so that we have a comparable evaluation metric for both.

Because currently we are only considering users who only want single flight-leg bookings, the scoring function only looks at the four slots relevant to these bookings: origin city, destination city, departure date, and departure time. We give 25 points for each slot which is filled, plus another 25 points for each slot which is also confirmed (i.e. grounded), making the maximum score 200 points (for a length zero dialogue). We also deduct 1 point for each action performed by the system, to penalize longer dialogues, so the maximum possible score is 198 (i.e. optimally there are only 2 system actions: ask for all the user info, and then offer a flight).

The motivation behind this evaluation metric is that confirmed slots are more likely to be correct than slots which are just filled. If we view the score as proportional to the probability that a slot is filled correctly, then this scoring assumes that confirmed slots are twice as likely to be correct. When combining the scores for different slots, we do not try to model the all-or-nothing nature of the COMMUNICATOR task-completion quality measures, but instead sum the scores for the individual slots. This sum makes our scoring system value partial completions more highly, but inspection of the distributions of scores indicates that this difference does not favour either the hybrid policy or the COMMUNICATOR systems.

Although this evaluation metric could reflect the relative quality of individual dialogues more accurately, we believe it provides a good measure of the relative quality of different systems. First, the exact same metric is applied to every system. Additional information which we have for some systems but not all, such as the COMMUNICATOR user questionnaires, are not used. Second, the systems are being run against approximately equivalent users. The user simulation is trained on exactly the same user actions which are used to evaluate the COMMUNICATOR systems, so the user simulations mimic exactly these users. In particular, the simulation is able to mimic the effects of a speech recognition error, since it is just as likely as the real users to disagree with a confirmation or provide a new value for a previously filled slot. The nature of the simulation model may make it systematically different from real users in some way, but we know of no argument for why this would bias our results in favour of one system or another.

Later in the project, we expect to also run the learned policies against human users. This will give us a more realistic evaluation of the learned policies, and allow more accurate quality measures to be collected through a user questionnaire. This should allow a more direct comparison against the COMMUNICATOR systems.

5.4.3 Comparisons between systems

So far we have run experiments to answer two questions. First, in our hybrid policy, what is the best balance between the supervised policy and the reinforcement learning policy? Second, how well does the hybrid policy perform compared to the COMMUNICATOR systems that it was trained on?

We trained models of both $S_{data}(s,a)$ and $Q_{data}(s,a)$, and then used them in hybrid policies with various values for the unobserved state penalty U. For both functions, we trained them for 100 iterations through the training data, at which point there was little change in the training error. Not much effort was put into

optimising the training procedure, since these are only initial tests. During testing, each hybrid policy was run for 1000 dialogues against the linear function approximation user model. The final states for each one of these dialogues was then fed through the scoring function and averaged across dialogues. The results are shown in table 5.2. The values for U were chosen based on the average number of decisions per dialogue which were different from that which the purely supervised policy would pick, which were 0 (U=-1000), 1 (U=0), 2 (U=40), and 5 (U=80), respectively.

Table 5.2 also shows some results for running the hybrid system against the Ngram user simulation. This user model seems to be easier to interact with than the linear user model. In particular, the resulting dialogues are better in terms of grounding.

U	total score	filled slots score	grounded slots score	length penalty
-1000	114.2	89.1	47.4	-22.2
0	101.3	69.5	51.6	-19.7
40	100.6	69.9	51.9	-21.1
80	96.0	67.0	49.4	-20.4
vs Ngram:				
80	105.7	68.8	57.2	-20.3

Table 5.2: The average scores for different values of the unobservable state reward U, and the three components of these scores.

These initial results show a clear trend whereby the more the system sticks with the supervised policy, the better it does. In other words, the best the hybrid policy can do is simply mimic the typical behaviour it observes in the systems in the data. This is a disappointing result, in that we were expecting reinforcement learning to provide some improvement over the supervised policy, provided the hybrid policy wasn't too different from the supervised policy. We intend to investigate this result further through a careful analysis of what decisions the reinforcement learning component is influencing and under what circumstances this influence might be beneficial, so that we can better target the way we use reinforcement learning in the hybrid policy. For example, the results in table 5.2 suggest that reinforcement learning improves grounding, but perhaps ends the dialogue before all the slots are filled.

To evaluate how well the hybrid policy performs compared to the COMMUNICATOR systems, we extracted the final states from all the non-continuation dialogues and fed them through the scoring function. The average scores are shown in tables 5.3 and 5.4 , along with the best performing hybrid policy and the scores averaged over all systems' dialogues. These scores were computed from 79 (BBN), 132 (CMU), 258 (ATT), and 174 (SRI) dialogues, respectively.

Table 5.3 shows the results computed from the complete dialogues. These results show a clear advantage for the hybrid policy over the COMMUNICATOR systems. The hybrid policy fills more slots and does it in fewer steps. Because the number of steps is doubtless effected by the hybrid policy's built-in strategy of stopping the dialogue after the first flight offer, we also evaluated the performance of the COMMUNICATOR systems if we also stopped these dialogues after the first flight offer, shown in table 5.4. The COMMUNICATOR systems do do better when stopped at the first flight offer, but the ordering of the systems is the same. They do better on length, but worse on grounding, and about the same on filled slots.[2]

[2]Some of the decrease in grounding performance may be because the dialogue is stopped before the user action

System	total score	filled slots score	grounded slots score	length penalty
hybrid (TALK)	114.2	89.1	47.4	-22.2
BBN	67.5	77.2	59.2	-68.9
CMU	49.7	60.2	47.4	-57.9
ATT	39.5	55.6	33.1	-49.3
SRI	21.0	52.4	0.0	-31.4
combined COMM	40.0	58.4	30.3	-48.6

Table 5.3: The average scores for the different systems, and the three components of these scores.

system	total reward	filled slots reward	grounded slots reward	length penalty
hybrid (TALK)	114.2	89.1	47.4	-22.2
BBN	83.2	74.1	24.1	-15.0
CMU	63.9	55.1	26.9	-18.1
ATT	55.3	55.8	24.4	-25.0
SRI	27.8	52.2	0.0	-24.4
combined COMM	53.0	56.9	18.3	-22.2

Table 5.4: The average reward scores after the first flight offer for the different systems, and the three components of these scores.

It is perhaps surprising that the hybrid policy does better than any of the COMMUNICATOR systems, since the hybrid policy shown is the purely supervised version, which simply mimics the typical behaviour of these exact systems. (Note that all the other hybrid policies in table 5.2 also perform better than all the systems.) One possible explanation is that the hybrid policy represents a kind of multi-version system, where decisions are made based on what the majority of systems would do. Multi-version systems are well known to perform better than any one system alone.

Of course, there may be something about the way we have designed these tests which makes our system come out particularly well. One possibility arises from the fact that the computation of grounding used to produce the states for the four systems is not as accurate as that used for the simulations. This might explain the particularly bad score received by the SRI system, but for the BBN and CMU systems their grounding scores are not the source of the hybrid system's advantage. We will soon remedy this discrepancy, which will also allow us to include the other four COMMUNICATOR systems in the dataset. The other four systems were not included because their strategies were not handled as well by the previous computation of grounding annotations.

Another possibility is that the way we have selected non-continuation dialogues means that bad dialogues are disproportionately selected. However, there are actually not many continuation dialogues (54), and their average score is lower than that for the non-continuation dialogues (20.9). Future tests will consider the full range of dialogues for both the systems and the hybrid policy.

following the flight offer, so any confirmation attempted by the system during the same turn cannot succeed. We have not yet evaluated the extent of this problem because a fair comparison would also require a change to the hybrid system's policy, but it is clear that this problem could not account for all the difference in total performance.

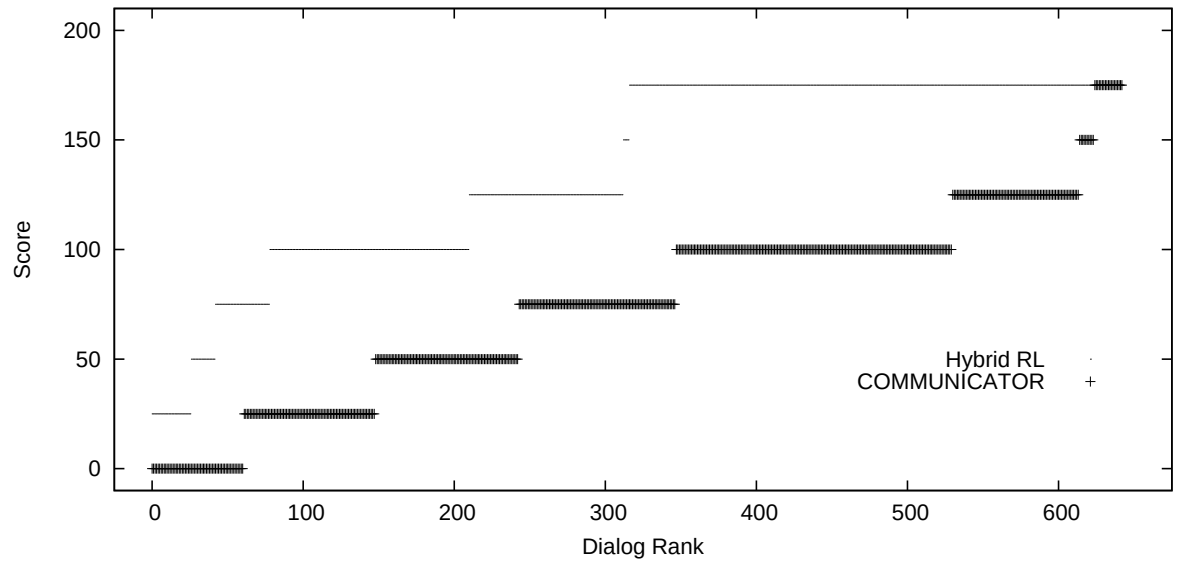

Figure 5.5: Comparing slot scores for the TALK learnt policy ("Hybrid RL") versus the COMMUNICATOR systems (disregarding penalties for dialogue length)

For a more detailed comparison of the systems, figure 5.5 and figure 5.6 plot the scores for the average COMMUNICATOR dialogues versus the learned policy's simulated dialogues, as a function of their rank. Because the COMMUNICATOR systems do better when stopped after the first flight offer, we use these results for these plots. In figure 5.5, the length of each bar reflects how many dialogues achieved the associated score for the number of filled and/or grounded slots. None of the dialogues actually get the maximum score of 200, but very many of the learned policy dialogues score 175, compared to very few of the COMMUNICATOR dialogues reaching that score. Figure 5.6 confirms this pattern when the length of the dialogue is also taken into account.

5.5 Conclusion

These initial results with a learned policy are very promising. They indicate that linear function approximation is a viable approach to the very large state spaces produced by the ISU framework. They also suggest that this approach is able to combine the advantages of different systems, even if only supervised learning is used. Further improvement is possible by tailoring the representation of states and actions based on our experience so far.

The next step is to better exploit the advantages of reinforcement learning. We will take a more standard RL approach by running the learned policy against different simulated users and exploring parts of the state space which are not included in the COMMUNICATOR data. The supervised policy is a good starting point for this exploration, and could be used to help constrain the range of policies which needs to be explored.

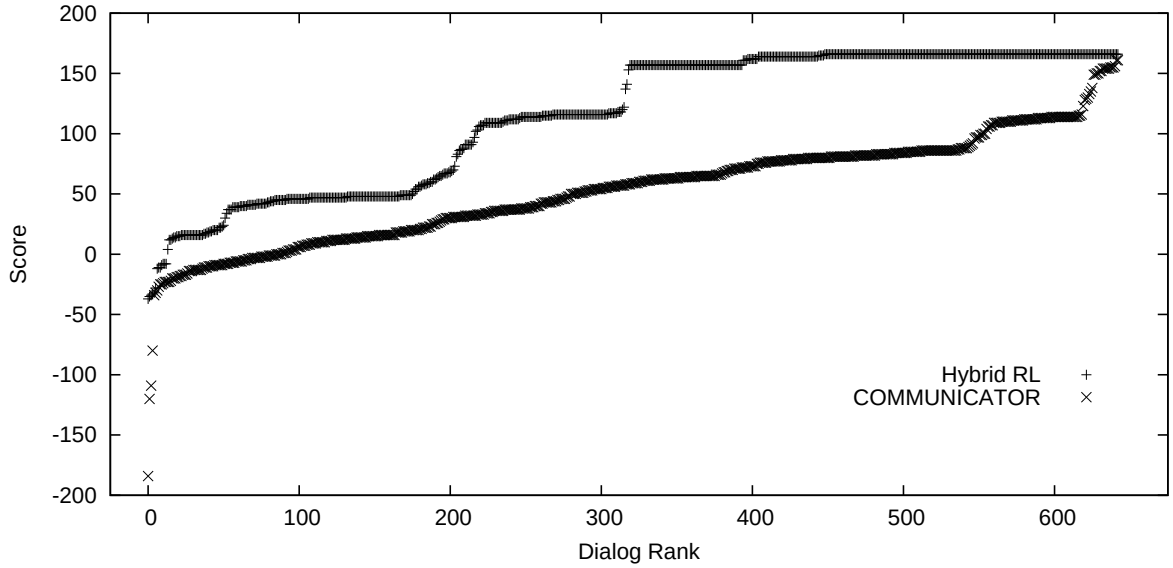

Figure 5.6: Comparing scores for the TALK learnt policy ("Hybrid RL") versus the COMMUNICATOR systems (taking into account penalties for dialogue length)

Chapter 6

An In-car baseline system for RL

This chapter describes a baseline dialogue system for reinforcement learning experiments, which is being developed at UEDIN and UCAM.

6.1 Moving between domains: COMMUNICATOR, WITAS, and in-car dialogues

This deliverable has focussed mostly on the COMMUNICATOR systems, which were used for flight-booking dialogues, and on the WITAS system, used for co-operative command and control dialogues. We have learned a promising initial policy for COMMUNICATOR dialogues, and some useful classifiers for handling user input more robustly in WITAS. What impact can we expect these results to have on the genre of in-car dialogue and the in-car systems being developed in TALK ?

Exploring reconfigurability

In the in-car scenarios we expect both the genres of "information seeking" (COMMUNICATOR) and "command and control" (WITAS) dialogue to be central. The UCAM SACTI corpora have driver information requests (e.g. searching for suitable hotels) as a major component, and the USAAR in-car MP3 dialogues show both information seeking and device command-and-control behaviour (e.g. search for a song, add it to a playlist, play).

One question we will address in future work is to what extent dialogue policies learnt from data gathered for one system, or family of systems, can be re-used or adapted for use in another system. We conjecture that the slot-filling policies learned from our experiments with COMMUNICATOR will also be good policies for other slot-filling tasks – that we may be learning "generic" slot-filling or information seeking dialogue policies. Similarly, we could explore whether the classifiers learnt for user input in WITAS could also operate effectively over similar features of other ISU systems. In this way the results of WP4 and WP2 (on reconfigurable dialogue systems) can inform each other.

6.2 System description

The baseline dialogue system is built around the DIPPER dialogue manager [BKLO03] and will be used to address issues in the evaluation of hand-coded versus learned dialogue strategies (see section 1.1).

With reference to the in-car application scenario, this system will initially be used to conduct information-seeking dialogues with a user (e.g. find a particular hotel or restaurant), using hand coded dialogue strategies (e.g. always use implicit confirmation, except when ASR confidence is below 50%, then use explicit confirmation). We will then modify the DIPPER dialogue manager so that it can consult learned strategies (for example strategies learned from the 2000 and 2001 COMMUNICATOR data), based on its current information state, and then execute dialogue actions from those strategies. This will allow us to compare hand-coded against learned strategies within the same system (i.e. the speech-synthesizer, recogniser, GUI, etc. will all remain fixed).

6.3 System components

Communication between components is handled by OAA's asynchronous hub architecture [CM01], and all components currently run under linux. The major components are:

- ATK for speech recognition[1]
- DIPPER [BKLO03] for ISU-based dialogue management[2]
- Learned dialogue policy "advisor" agent[3]
- Multimodal Map interface (to be adapted from the UGOT Tram system), for multimodal input and output[4]
- Grammatical Framework (GF) parser agent [5]
- Festival2 [TBC98] for speech synthesis[6]

We now describe these components.

6.3.1 DIPPER dialogue manager

In [BKLO03] an ISU dialogue management system called DIPPER is presented, along with the formal syntax and semantics of its information state update language. This is the system we currently use in TALK research concerning learning and adaptivity. As well as a dialogue manager, DIPPER includes a variety of supporting Open Agent Architecture (OAA, [CM01]) "agents" for dialogue system tasks such as speech recognition.

[1] See `http://mi.eng.cam.ac.uk/~sjy/software.htm` . OAA wrapper developed in C++
[2] Both Java and Prolog version are available, with OAA wrappers.
[3] This is written in Python and has an OAA wrapper in C
[4] This component is being developed by DFKI.
[5] Wrapper developed by UGOT.
[6] Wrapper developed in C.

```
infs(record([is:record([
       lastspeaker:atomic,
       turn:atomic,
       gnd:record([dh:stack(atomic),
                   obl:stack(atomic)]),
       input:stack(atomic),
       lastinput:stack(atomic),
       output:stack(atomic),
       nextmoves:stack(Acts),
       lastmoves:stack(Acts),
       int:stack(Acts)])])) :-
       Acts = record([pred:atomic,
                      dp:atomic,
                      prop:record([pred:atomic,
                                   args:stack(atomic)])]).
```

Figure 6.1: An example DIPPER Information State definition

DIPPER follows Trindi [LT00] closely, taking its record structure and datatypes to define information states (see an example information state in figure 6.3.1). However, in contrast to TrindiKit, in DIPPER all control is conveyed by update rules, there are no additional update and control algorithms, and there is no separation between update and selection rules. Furthermore, the update rules in DIPPER do not rely on Prolog unification and backtracking, and allow developers to use OAA "solvables" in their effects. An OAA solvable is a request for a service provided by some other software component in a community of agents, for example "start-recognise-speech" for a speech recogniser or "text-to-speech(Text)" for a speech synthesizer.

Update rules (see examples in figure 6.3.1) specify the information state change potential in a declarative way: applying an update rule to an information state results in a new state. An update rule is a triple $<$ `name, conditions, effects` $>$. The conditions and effects are defined by an update language, and both are recursively defined over terms. The terms of the update language allow developers to refer to specific values within information states, either for testing a condition, or for applying an effect. For example, the term `isˆlastspeaker` refers to the the field `lastspeaker` in the record `is`.

In the update rules shown in figure 6.3.1 `solve2` is used to request OAA services. For example

```
solve2(text-to-speech(isˆoutput))
```

sends a request via OAA for a "text-to-speech" task with 1 argument (the value of the "output" field in the current IS `is`. In the current system, such requests are handled by the Festival OAA agent, buut they could be handled by any other agent which registers the text-to-speech solveable with the OAA facilitator (the hub).

Figure 6.3 shows an example of DIPPER's graphical user interface for designing and debugging the dialogue manager.

```
urule(initialisation,
     [                    ;;; CONDITIONS
     is^lastspeaker=''
      ],
     [                    ;;; EFFECTS
      assign(is^lastspeaker,user),
      solve2(deliberate('task_ask_user',
                        'talk.tf',_X)),
      solve2(execution_mode(_YY)),
      solve2(text-to-speech(
         "Welcome to the TALK system"))
     ]
    ).

urule(deliberation,
      [
       is^lastspeaker=user,
       empty(is^int)
      ],
      [
       solve2(next_step(Step)),
       push(is^int,Step),
       solve2(exec_next_step(_X))
      ]
     ).

urule(generation,
      [
      top(is^int)=[release_turn],
      is^lastspeaker=user
      ],
      [
      prolog(utter(is^nextmoves,X,Y)),
      push(is^lastmoves,(X,s)),
      clear(is^nextmoves),
      push(is^nextmoves,X),
      clear(is^output),
      push(is^output,Y),
      solve2(text-to-speech(Y)),
      assign(is^lastspeaker,system),
      clear(is^input)
      ]
     ).
```

Figure 6.2: Example DIPPER Update Rules

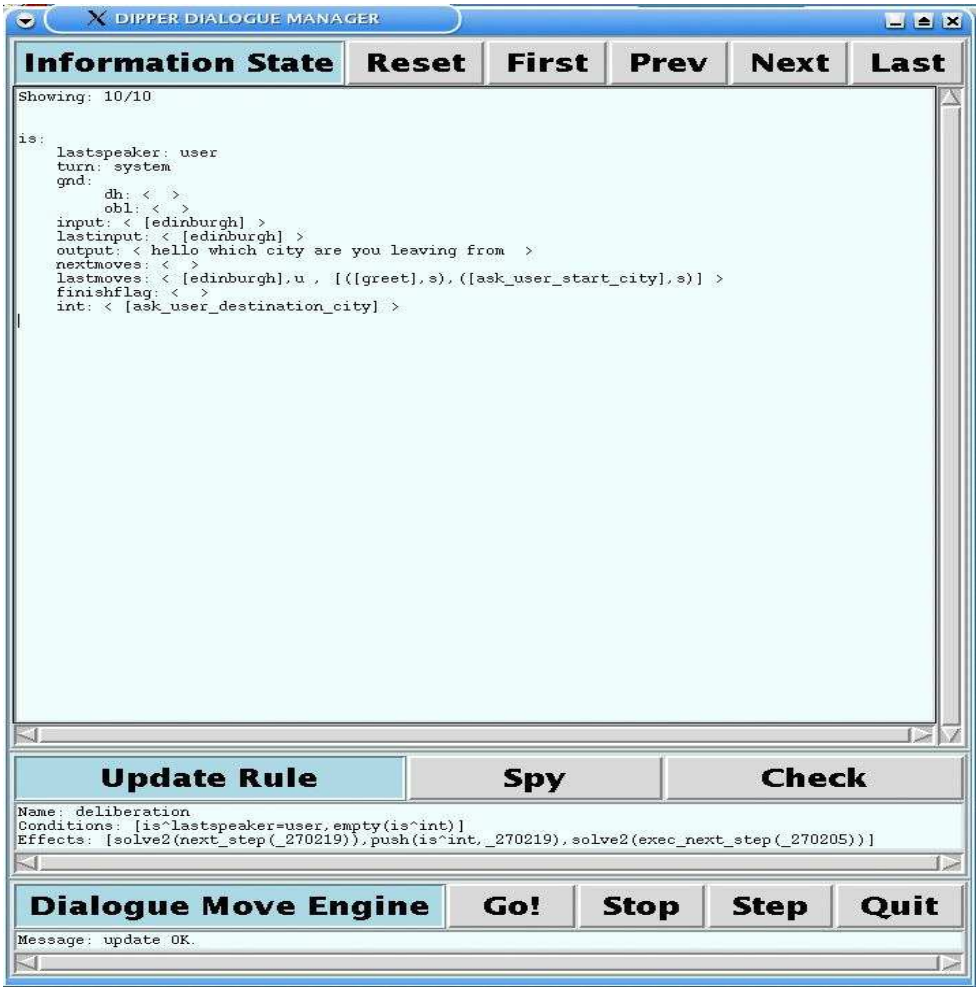

Figure 6.3: A DIPPER dialogue manager screenshot

6.3.2 ATK speech recogniser

For speech recognition we use ATK. ATK is a real time API for the HTK speech recognition toolkit. It consists of a C++ layer sitting on top of the standard HTK libraries.
We have developed an OAA wrapper for ATK in C++. It has the following solvables:

- `atkinitrecogniser(configuration file, dictionary, string of lattice names, result)`.

  The configuration file contains information that is necessary for ATK to run. For example the location of acoustic models, the type of feature extraction, whether word-internal or cross-word triphones are used, and so on. The dictionary includes all words that are supported by the speech recogniser with their phonetic transcription. The dictionary should be consistent with the used acoustic models. Currently we use the BEEP dictionary (available by anonymous ftp from `svr-ftp.eng.cam.ac.uk/pub/comp.speech/dictionaries/beep.tar.gz`) and the acoustic models provided in the ATK distribution. Lattices are the language models used during recognition. The solvable returns 0 for successful initialisation and 1 in failure.

Note that during initialisation we load all lattices and the dictionary. This is done for efficiency reasons, that is, to save memory and speed during recognition. The "atkinitrecogniser" function is called only once, at the beginning of the dialogue.

- `atkcallrecogniser(lattice name, recognised string, semantic output, confidence score)`. This solvable is loaded each time the user utters a new sentence. It takes as input the language model for the current dialogue state and it produces as output the recognised string, the semantics contained in the recognised string and the recognition confidence score. The semantic content of the user's input is incorporated into the recognition network used by ATK and therefore it can be extracted together with the recognised string. In complex cases in which additional semantic interpretation is required, parsing will be performed by using a grammar e.g. GF.
- `atkfreerecogniser(Res)`: this solveable is called at the end of the dialogue to free memory and produces 0 for success and 1 if an error appears.

Language modelling

Currently for language modelling we use a high level grammar notation based on extended Backus-Naur Form (EBNF). This grammar is then transformed to the recognition network format required by ATK using the HTK tools "HParse" and "HBuild". A simple example of an EBNF grammar follows:

```
$pause = FIL | SIL | SP ;
$want = ( I WOULD LIKE  |  I'D LIKE  |  I WANT ) TO ;
$travel = ( FLY | TRAVEL | LEAVE | DEPART) FROM ;
$city = EDINBURGH | LONDON | MANCHESTER ;
( !ENTER ( [ $want ] [ $travel ] [ $pause ] ) $city !EXIT )
```

Here is a fragment of an example dictionary from the current system:

```
!ENTER sil
!EXIT sil
SP sp
SIL sil
FIL sil
EDINBURGH eh d ih n b r ax sp
LONDON l ah n d ax n sp
LEAVE l iy v sp
TO t ax sp
TO t uw sp
```

We plan to extend and modify these resources to fit data gathered in the UCAM SACTI 1 and 2 corpora.

6.3.3 Learned policy agent

This agent is the same as the "RLsimulation" agent described in section 5.2.1. As has already been mentioned it acts as an interface between the simulation environment and the system simulation based on RL, or the user simulation based on supervised learning. In particular it has the following solvable:

`callRLsimulation(speaker, file_name_of_current_IS, conversational domain, speech act, task, result).`

The argument `speaker` shows whether the agent is used for simulating the system (based on RL) or the user (based on supervised learning). The second argument is the name of the file that contains all information about the current information state, which is required by the RL or supervised learning algorithms to produce their action, that is, conversational domain, speech act and task. The result is 0 in success and 1 in failure.

In the baseline system again we will have to provide the learned policy agent with the current information state, so it is expected that the DIPPER environment will be similar to the one used for simulation.

6.3.4 Festival2 speech synthesizer

We use Festival2 for speech synthesis in server-client mode and we have developed an OAA agent with the following solvable:

`callfestival(pathname, textinput, result).`

The 'pathname' argument includes the path where the festival client runs, 'textinput' is the text we would like to synthesize and 'result' is 0 for success and 1 in case an error arises.

6.3.5 GF parser

We plan to develop a multimodal grammar for the in-car domain. We expect to do this with guidance from UGOT. The GF system is described in detail in TALK deliverable D1.2a.

6.4 Plan for system development

To further deveop the system, we plan the following tasks:

- Obtain SACTI language models for ATK, test ASR
- Obtain Multimodal Map agent from DFKI
- Interface Multimodal Map agent with DIPPER
- Develop and test GF grammar
- Interface GF with DIPPER

Later in the project we expect to use multimodal grammars and language models.

Chapter 7

Conclusion and future work

This report has described work done in the early phases of the TALK project to integrate "Information State Update" (ISU)-based dialogue systems with techniques from machine learning (e.g. memory-based learning, reinforcement learning). We explained how to use data gathered from information states of ISU dialogue managers in a variety of different learning techniques (chapter 2). We presented some results showing that various classifier learning techniques over ISU representations can reduce error rates by over 50% in a command-and-control dialogue system (WITAS). We then focussed on Reinforcement Learning (RL) techniques for automatic optimisation of dialogue strategies (chapter 3). This involved constructing an automatic annotation system which builds ISU state representations for the COMMUNICATOR corpora of human-machine dialogues, and then developing novel techniques for learning from this data. We described a learning method based on linear function approximation (in order to deal with the very large state spaces generated by the ISU approach), and a hybrid method which combines supervised learning and reinforcement learning (chapter 5). We also described 2 different types of user simulation that we have built from the COMMUNICATOR data for testing purposes. Our initial results are encouraging. We found that over a run of 1000 simulated dialogues our learned policy has an average reward (computed as a function of the task completion and dialogue length) of over twice the average score of the COMMUNICATOR systems, and more than a third greater than the best of the COMMUNICATOR systems. We also reported on progress in a baseline system for in-car dialogues and describe plans for future research (chapter 6).

We conclude with a brief description of future work. Please see also section 3.5, which explained the POMDP and Bayesian approached to be explored further in WP4.

7.1 Planned experiments

Given the platform described here, in the remainder of the project we plan to further investigate the following research questions:

- which state variables are particularly relevant to determining low level dialogue confirmation and information presentation strategies? [WP4.2]
- what is the effect of different reward signals on resulting dialogue policy? [WP4.2]

- how can the large ISU state space be mapped effectively into a tractable subset for reinforcement learning (RL)? (e.g., using Bayes Nets, linear function approximation, etc.) [WP4.3]
- do partially observable MDPs (POMDPS) offer any advantages compared to MDPs? [WP4.4]
- how can the tractability issues inherent in POMDPs be avoided? [WP4.4]

In the context of reinforcement learning (RL), dialogues will be simulated and rewards measured for different conditions. The performance of optimised dialogues will then be contrasted with hand-coded strategies.

The dialogue manager will also be tested by embedding it in a baseline in-car system and running user trials. This will be the main evaluation approach and the primary evaluation will compare the optimised system with a hand-crafted baseline. UEDIN plans to conduct such experiments midway through the project (i.e., in mid-2005). Similar metrics to those the above can still be used, but in this case subjective measures can also be gathered from user questionnaires.

In addition to these planned experiments (as presented in D6.1) the following issues are now open for investigation:

- Reinforcement learning of multimodal presentation strategies from the USAAR MP3 data collection 2005
- Evaluating the different user simulations developed in TALK
- Reinforcement learning of dialogue policies using true exploration of the policy space against a variety of user simulations
- Experimenting with different feature sets and action sets in the linear function approaximation approach
- Examining learned feature weights to determine the relevance of ceatain ISU features for certain action choices
- Using Kernel methods
- Performing a full evaluation of the automatic annotation system.

Bibliography

[BEG+96] M. Boros, W. Eckert, F. Gallwitz, G. Görz, G. Hanrieder, and H. Niemann. Towards Understanding Spontaneous Speech: Word Accuracy vs. Concept Accuracy. In *Proceedings of ICSLP-96*, 1996.

[BKLO03] Johan Bos, Ewan Klein, Oliver Lemon, and Tetsushi Oka. DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture. In *4th SIGdial Workshop on Discourse and Dialogue*, pages 115–124, Sapporo, 2003.

[CM01] Adam Cheyer and David Martin. The Open Agent Architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 4(1/2):143–148, 2001.

[Coh95] William W. Cohen. Fast Effective Rule Induction. In *Proceedings of the 12th International Conference on Machine Learning*, 1995.

[CR97] P.R. Clarkson and R. Rosenfeld. Statistical Language Modeling Using the CMU-Cambridge Toolkit. In *Proceedings of Eurospeech*, 1997.

[CR01] Ananlada Chotimongkol and Alexander I. Rudnicky. N-best Speech Hypotheses Reordering Using Linear Regression. In *Proceedings of EuroSpeech 2001*, pages 1829–1832, 2001.

[DGA+93a] John Dowding, Jean Mark Gawron, Doug Appelt, John Bear, Lynn Cherny, Robert Moore, and Douglas Moran. GEMINI: a natural language system for spoken-language understanding. In *Proceedings of ACL-93*, 1993.

[DGA+93b] John Dowding, Jean Mark Gawron, Douglas E. Appelt, John Bear, Lynn Cherny, Robert Moore, and Douglas B. Moran. GEMINI: A natural language system for spoken-language understanding. In *Meeting of the Association for Computational Linguistics*, pages 54–61, 1993.

[DH02] Walter Daelemans and Véronique Hoste. Evaluation of Machine Learning Methods for Natural Language Processing Tasks. In *Proceedings of LREC-02*, 2002.

[DZvdSvdB02] Walter Daelemans, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch. TIMBL: Tilburg Memory Based Learner, version 4.2, Reference Guide. In *ILK Technical Report 02-01*, 2002.

[Gab03] Malte Gabsdil. Classifying Recognition Results for Spoken Dialogue Systems. In *Proceedings of the Student Research Workshop at ACL-03*, 2003.

[GL04a] Malte Gabsdil and Oliver Lemon. Combining acoustic and pragmatic features to predict recognition performance in spoken dialogue systems. In *Proceedings of ACL-04*, pages 344–351, 2004.

[GL04b] Kallirroi Georgila and Oliver Lemon. Adaptive multimodal dialogue management based on the information state update approach. In *W3C Workshop on Multimodal Interaction*, 2004.

[Hin95] Perry R. Hinton. *Statistics Explained – A Guide For Social Science Students*. Routledge, 1995.

[HLC+03] Beth-Ann Hockey, Oliver Lemon, Ellen Campana, Laura Hiatt, Gregory Aist, Jim Hieronymus, Alexander Gruenstein, and John Dowding. Targeted help for spoken dialogue systems: intelligent feedback improves naive users' performance. In *Proceedings of European Association for Computational Linguistics (EACL 03)*, pages 147 – 154, 2003.

[Lee04] Zhang Lee. Maximum Entropy Modeling Toolkit for Python and C++, 2004.

[Lem04] Oliver Lemon. Context-sensitive speech recognition in Information-State Update dialogue systems: results for the grammar switching approach. In *Proceedings of the 8th Workshop on the Semantics and Pragmatics of Dialogue, CATALOG'04*, 2004.

[LG04a] Oliver Lemon and Alexander Gruenstein. Multithreaded context for robust conversational interfaces: context-sensitive speech recognition and interpretation of corrective fragments. *ACM Transactions on Computer-Human Interaction*, 2004. (to appear).

[LG04b] Oliver Lemon and Alexander Gruenstein. Multithreaded context for robust conversational interfaces: context-sensitive speech recognition and interpretation of corrective fragments. *ACM Transactions on Computer-Human Interaction (ACM TOCHI)*, 11(3):241– 267, 2004.

[LH04] Oliver Lemon and James Henderson. Machine learning and the Information State Update approach to dialogue management. In *Proceedings of the Joint AMI-PASCAL-IM2-M4 workshop*, 2004.

[LKSW00] Diane Litman, Micheal Kearns, Satinder Singh, and Marilyn Walker. Automatic optimization of dialogue management. In *Proc. COLING*, 2000.

[LP97] E. Levin and R. Pieraccini. A stochastic model of computer-human interaction for learning dialogue strategies. In *Proceedings of Eurospeech*, pages 1883–1886, Rhodes, Greece, 1997.

[LPE00] E. Levin, R. Pieraccini, and W. Eckert. A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Transactions on Speech and Audio Processing*, 8(1):11–23, 2000.

[LRH+00] Ian Lewin, C. J. Rupp, James Hieronymous, David Milward, Steffan Larsson, and Berman. SIRIDUS system architecture and interface report. Technical Report D6.1, Siridus Project, 2000.

[LT00] Staffan Larsson and David Traum. Information state and dialogue management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*, 6(3-4):323–340, 2000.

[MRL05] Ivan Meza-Ruiz and Oliver Lemon. Using dialogue context to improve parsing performance in dialogue systems. In *International Workshop on Computational Semantics (IWCS-6)*, 2005.

[MRvdB+03] Erwin Marsi, Martin Reynaert, Antal van den Bosch, Walter Daelemans, and Véronique Hoste. Learning to predict pitch accents and prosodic boundaries in Dutch. In *Proceedings of ACL-03*, 2003.

[SB98a] Richard Sutton and Andrew Barto. *Reinforcement Learning*. MIT Press, 1998.

[SB98b] Richard Sutton and Andrew Barto. *Reinforcement Learning*. MIT Press, 1998.

[SKLW00] Satinder Singh, Michael Kearns, Diane Litman, and Marilyn Walker. Empirical evaluation of a reinforcement learning dialogue system. In *Proceedings of AAAI*, 2000.

[SLKW02] Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *Journal of Artificial Intelligence Research (JAIR)*, 2002.

[STC04] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[SY01] Konrad Scheffler and Steve Young. Corpus-based dialogue simulation for automatic strategy learning and evaluation. In *Proceedings of NAACL Workshop on Adaptation in Dialogue Systems*, 2001.

[SY02] Konrad Scheffler and Steve Young. Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. In *Proc. HLT*, 2002.

[TBC98] P. Taylor, A. Black, and R. Caley. The architecture of the the Festival speech synthesis system. In *Third International Workshop on Speech Synthesis, Sydney, Australia*, 1998.

[TKSDM03] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada, 2003.

[WAB+01] M Walker, J Aberdeen, J Boland, E Bratt, J Garofolo, L Hirschman, A Le, S Lee, S Narayanan, K Papineni, B Pellom, B Polifroni, A Potamianos, P Prabhu, A Rudnicky, G Sanders, S Seneff, D Stallard, and S Whittaker. Darpa communicator dialog travel planning systems: The june 2000 data collection. In *Eurospeech 2001*, Aalborg, Scandinavia, 2001.

[WKL00] Marilyn A. Walker, Candace A. Kamm, and Diane J. Litman. Towards Developing General Models of Usability with PARADISE. *Natural Language Engineering*, 6(3), 2000.

[WPB01] Marilyn A. Walker, Rebecca J. Passonneau, and Julie E. Boland. Quantitative and Qualitative Evaluation of Darpa Communicator Spoken Dialogue Systems. In *Meeting of the Association for Computational Linguistics*, pages 515–522, 2001.

[WWL00] Marilyn Walker, Jerry Wright, and Irene Langkilde. Using Natural Language Processing and Discourse Features to Identify Understanding Errors in a Spoken Dialogue System. In *Proceedings of ICML-2000*, 2000.

[YEK+02] Steve Young, Gunnar Evermann, Dan Kershaw, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, and Phil Woodland. *The HTK Book*. Cambridge University Engineering Department, 2002. (for HTK version 3.2).

[You00] SJ Young. Probabilistic methods in spoken dialogue systems. *Philosophical Transactions of the Royal Society (Series A)*, 358(1769):1389–1402, 2000.

Appendix A

State Feature-Value Pairs Used with Linear Functions

Following is the list of state feature-value pairs which correspond to values in the vector input to any linear function approximation for the system. User input values are contained in square brackets.

STATE *number*
WordErrorRate: *number*
AsrInput: *text*
CommonGround: []
CommonGround: [continue_depart_time]
CommonGround: [continue_orig_city]
CommonGround: [depart_time]
CommonGround: [null]
CommonGround: [return_date]
CommonGroundHist: []
CommonGroundHist: [continue_depart_time]
CommonGroundHist: [continue_orig_city]
CommonGroundHist: [depart_time]
CommonGroundHist: [null]
CommonGroundHist: [return_date]
ConvDomain: about_communication
ConvDomain: about_task
ConvDomain: unmatched
DialogueActType: user
FilledSlot: [accept_car_offer]
FilledSlot: [accept_flight_summary]
FilledSlot: [continue_depart_date]
FilledSlot: [continue_dest_city]
FilledSlot: [continue_trip]
FilledSlot: [depart_time]
FilledSlot: [hotel_location]
FilledSlot: [no_continue_trip]
TurnNumber: *number*
ActionNumber: *number*
Output: *text*
CommonGround: [continue_depart_date]
CommonGround: [continue_dest_city]
CommonGround: [depart_date]
CommonGround: [dest_city]
CommonGround: [orig_city]
CommonGround: [return_time]
CommonGroundHist: [continue_depart_date]
CommonGroundHist: [continue_dest_city]
CommonGroundHist: [depart_date]
CommonGroundHist: [dest_city]
CommonGroundHist: [orig_city]
CommonGroundHist: [return_time]
ConvDomain: about_situation_frame
ConvDomain: tbc
DialogueActType: system
FilledSlot: []
FilledSlot: [accept_flight_offer]
FilledSlot: [accept_hotel_offer]
FilledSlot: [continue_depart_time]
FilledSlot: [continue_orig_city]
FilledSlot: [depart_date]
FilledSlot: [dest_city]
FilledSlot: [hotel_name]
FilledSlot: [no_return_trip]

FilledSlot: [not_grounded_depart_date_or_time]
FilledSlot: [orig_city]
FilledSlot: [reject_flight_offer]
FilledSlot: [reject_hotel_offer]
FilledSlot: [return_time]
FilledSlotsHist: []
FilledSlotsHist: [accept_flight_offer]
FilledSlotsHist: [accept_hotel_offer]
FilledSlotsHist: [continue_depart_time]
FilledSlotsHist: [continue_orig_city]
FilledSlotsHist: [depart_date]
FilledSlotsHist: [dest_city]
FilledSlotsHist: [hotel_name]
FilledSlotsHist: [no_return_trip]
FilledSlotsHist: [not_grounded_return_date_or_time]
FilledSlotsHist: [reject_car_offer]
FilledSlotsHist: [reject_flight_summary]
FilledSlotsHist: [rental_company]
FilledSlotsHist: [return_time]
Speaker: system
SpeechAct: []
SpeechAct: [no_answer]
SpeechAct: [reject_info]
SpeechAct: [yes_answer]
SpeechAct: apology
SpeechAct: implicit_confirm
SpeechAct: offer
SpeechAct: present_info
SpeechAct: status_report
SpeechAct: unmatched
SpeechActsHist: [correct_info]
SpeechActsHist: [provide_info]
SpeechActsHist: [reprovide_info]
SpeechActsHist: acknowledgement
SpeechActsHist: explicit_confirm
SpeechActsHist: instruction
SpeechActsHist: opening_closing
SpeechActsHist: request_info
SpeechActsHist: tbc
Task: []
Task: [accept_flight_offer]
Task: [accept_hotel_offer]
Task: [continue_depart_time]
Task: [continue_orig_city]
Task: [depart_date]

FilledSlot: [not_grounded_return_date_or_time]
FilledSlot: [reject_car_offer]
FilledSlot: [reject_flight_summary]
FilledSlot: [return_date]
FilledSlot: [return_trip]
FilledSlotsHist: [accept_car_offer]
FilledSlotsHist: [accept_flight_summary]
FilledSlotsHist: [continue_depart_date]
FilledSlotsHist: [continue_dest_city]
FilledSlotsHist: [continue_trip]
FilledSlotsHist: [depart_time]
FilledSlotsHist: [hotel_location]
FilledSlotsHist: [no_continue_trip]
FilledSlotsHist: [not_grounded_depart_date_or_time]
FilledSlotsHist: [orig_city]
FilledSlotsHist: [reject_flight_offer]
FilledSlotsHist: [reject_hotel_offer]
FilledSlotsHist: [return_date]
FilledSlotsHist: [return_trip]
Speaker: user
SpeechAct: [correct_info]
SpeechAct: [provide_info]
SpeechAct: [reprovide_info]
SpeechAct: acknowledgement
SpeechAct: explicit_confirm
SpeechAct: instruction
SpeechAct: opening_closing
SpeechAct: request_info
SpeechAct: tbc
SpeechActsHist: []
SpeechActsHist: [no_answer]
SpeechActsHist: [reject_info]
SpeechActsHist: [yes_answer]
SpeechActsHist: apology
SpeechActsHist: implicit_confirm
SpeechActsHist: offer
SpeechActsHist: present_info
SpeechActsHist: status_report
SpeechActsHist: unmatched
Task: [accept_car_offer]
Task: [accept_flight_summary]
Task: [continue_depart_date]
Task: [continue_dest_city]
Task: [continue_trip]
Task: [depart_time]

Task: [dest_city]
Task: [hotel_name]
Task: [no_return_trip]
Task: [not_grounded_return_date_or_time]
Task: [reject_car_offer]
Task: [reject_flight_summary]
Task: [rental_company]
Task: [return_time]
Task: airline
Task: depart_arrive_date
Task: dest_city
Task: flight_booking
Task: hotel
Task: hotel_location
Task: meta_ambiguity_resolution
Task: meta_error
Task: meta_hangup
Task: meta_instruct
Task: meta_request_change
Task: meta_slu_reject
Task: orig_city
Task: orig_dest_date
Task: rental
Task: rental_day_time
Task: retrieval
Task: tbc
Task: trip
Task: unknown
TasksHist: []
TasksHist: [accept_flight_offer]
TasksHist: [accept_hotel_offer]
TasksHist: [continue_depart_time]
TasksHist: [continue_orig_city]
TasksHist: [depart_date]
TasksHist: [dest_city]
TasksHist: [hotel_name]
TasksHist: [no_return_trip]
TasksHist: [not_grounded_return_date_or_time]
TasksHist: [reject_car_offer]
TasksHist: [reject_flight_summary]
TasksHist: [rental_company]
TasksHist: [return_time]
TasksHist: airline
TasksHist: depart_arrive_date
TasksHist: dest_city
Task: [hotel_location]
Task: [no_continue_trip]
Task: [not_grounded_depart_date_or_time]
Task: [orig_city]
Task: [reject_flight_offer]
Task: [reject_hotel_offer]
Task: [return_date]
Task: [return_trip]
Task: continue_trip
Task: depart_arrive_time
Task: flight
Task: ground
Task: hotel_booking
Task: hotel_name
Task: meta_correction
Task: meta_greeting_goodbye
Task: meta_inconsistent_info
Task: meta_repetition
Task: meta_situation_info
Task: meta_start_over
Task: orig_dest_city
Task: price
Task: rental_booking
Task: rental_type
Task: return_date
Task: top_level_trip
Task: trip_type
Task: unmatched
TasksHist: [accept_car_offer]
TasksHist: [accept_flight_summary]
TasksHist: [continue_depart_date]
TasksHist: [continue_dest_city]
TasksHist: [continue_trip]
TasksHist: [depart_time]
TasksHist: [hotel_location]
TasksHist: [no_continue_trip]
TasksHist: [not_grounded_depart_date_or_time]
TasksHist: [orig_city]
TasksHist: [reject_flight_offer]
TasksHist: [reject_hotel_offer]
TasksHist: [return_date]
TasksHist: [return_trip]
TasksHist: continue_trip
TasksHist: depart_arrive_time
TasksHist: flight

TasksHist: flight_booking
TasksHist: hotel
TasksHist: hotel_location
TasksHist: meta_ambiguity_resolution
TasksHist: meta_error
TasksHist: meta_hangup
TasksHist: meta_instruct
TasksHist: meta_request_change
TasksHist: meta_slu_reject
TasksHist: orig_city
TasksHist: orig_dest_date
TasksHist: rental
TasksHist: rental_day_time
TasksHist: retrieval
TasksHist: tbc
TasksHist: trip
TasksHist: unknown
Turn: system
Task: meta_release_turn
FilledSlot: [accept_car_offer,rental_company]
FilledSlot: [depart_time,depart_date]
FilledSlot: [orig_city,dest_city]
SpeechAct: [no_answer,correct_info]
SpeechAct: [reject_info,correct_info]
SpeechAct: [reprovide_info,reprovide_info]
SpeechAct: [yes_answer,reprovide_info]
Task: [accept_hotel_offer,hotel_name]
Task: [orig_city,dest_city,depart_date]
Task: [return_date,return_time]
TasksHist: ground
TasksHist: hotel_booking
TasksHist: hotel_name
TasksHist: meta_correction
TasksHist: meta_greeting_goodbye
TasksHist: meta_inconsistent_info
TasksHist: meta_repetition
TasksHist: meta_situation_info
TasksHist: meta_start_over
TasksHist: orig_dest_city
TasksHist: price
TasksHist: rental_booking
TasksHist: rental_type
TasksHist: return_date
TasksHist: top_level_trip
TasksHist: trip_type
TasksHist: unmatched
Turn: user
Task: meta_end_dialog
FilledSlot: [accept_hotel_offer,hotel_name]
FilledSlot: [orig_city,dest_city,depart_date]
FilledSlot: [return_date,return_time]
SpeechAct: [no_answer,provide_info]
SpeechAct: [reprovide_info,provide_info]
SpeechAct: [yes_answer,provide_info]
Task: [accept_car_offer,rental_company]
Task: [depart_time,depart_date]
Task: [orig_city,dest_city]

Following is the list of state feature-value pairs which correspond to values in the vector input to the linear function approximation for the user simulation. User input values are contained in square brackets.

STATE *number*
WordErrorRate: *number*
Output: *text*
CommonGround: [continue_depart_date]
CommonGround: [continue_dest_city]
CommonGround: [depart_date]
CommonGround: [dest_city]
CommonGround: [orig_city]
CommonGround: [return_time]
CommonGroundHist: [continue_depart_date]
CommonGroundHist: [continue_dest_city]
CommonGroundHist: [depart_date]
TurnNumber: *number*
AsrInput: *text*
CommonGround: []
CommonGround: [continue_depart_time]
CommonGround: [continue_orig_city]
CommonGround: [depart_time]
CommonGround: [null]
CommonGround: [return_date]
CommonGroundHist: []
CommonGroundHist: [continue_depart_time]
CommonGroundHist: [continue_orig_city]
CommonGroundHist: [depart_time]

CommonGroundHist: [dest_city]
CommonGroundHist: [orig_city]
CommonGroundHist: [return_time]
ConvDomain: about_situation_frame
ConvDomain: tbc
DialogueActType: system
FilledSlot: []
FilledSlot: [accept_flight_offer]
FilledSlot: [accept_hotel_offer]
FilledSlot: [continue_depart_time]
FilledSlot: [continue_orig_city]
FilledSlot: [depart_date]
FilledSlot: [dest_city]
FilledSlot: [hotel_name]
FilledSlot: [no_return_trip]
FilledSlot: [not_grounded_return_date_or_time]
FilledSlot: [reject_car_offer]
FilledSlot: [reject_flight_summary]
FilledSlot: [return_date]
FilledSlot: [return_trip]
FilledSlotsHist: [accept_car_offer]
FilledSlotsHist: [accept_flight_summary]
FilledSlotsHist: [continue_depart_date]
FilledSlotsHist: [continue_dest_city]
FilledSlotsHist: [continue_trip]
FilledSlotsHist: [depart_time]
FilledSlotsHist: [hotel_location]
FilledSlotsHist: [no_continue_trip]
FilledSlotsHist: [not_grounded_depart_date_or_time]
FilledSlotsHist: [orig_city]
FilledSlotsHist: [reject_flight_offer]
FilledSlotsHist: [reject_hotel_offer]
FilledSlotsHist: [return_date]
FilledSlotsHist: [return_trip]
Speaker: user
SpeechAct: [correct_info]
SpeechAct: [provide_info]
SpeechAct: [reprovide_info]
SpeechAct: acknowledgement
SpeechAct: explicit_confirm
SpeechAct: instruction
SpeechAct: opening_closing
SpeechAct: request_info
SpeechAct: tbc
SpeechActsHist: []
CommonGroundHist: [null]
CommonGroundHist: [return_date]
ConvDomain: about_communication
ConvDomain: about_task
ConvDomain: unmatched
DialogueActType: user
FilledSlot: [accept_car_offer]
FilledSlot: [accept_flight_summary]
FilledSlot: [continue_depart_date]
FilledSlot: [continue_dest_city]
FilledSlot: [continue_trip]
FilledSlot: [depart_time]
FilledSlot: [hotel_location]
FilledSlot: [no_continue_trip]
FilledSlot: [not_grounded_depart_date_or_time]
FilledSlot: [orig_city]
FilledSlot: [reject_flight_offer]
FilledSlot: [reject_hotel_offer]
FilledSlot: [return_time]
FilledSlotsHist: []
FilledSlotsHist: [accept_flight_offer]
FilledSlotsHist: [accept_hotel_offer]
FilledSlotsHist: [continue_depart_time]
FilledSlotsHist: [continue_orig_city]
FilledSlotsHist: [depart_date]
FilledSlotsHist: [dest_city]
FilledSlotsHist: [hotel_name]
FilledSlotsHist: [no_return_trip]
FilledSlotsHist: [not_grounded_return_date_or_time]
FilledSlotsHist: [reject_car_offer]
FilledSlotsHist: [reject_flight_summary]
FilledSlotsHist: [rental_company]
FilledSlotsHist: [return_time]
Speaker: system
SpeechAct: []
SpeechAct: [no_answer]
SpeechAct: [reject_info]
SpeechAct: [yes_answer]
SpeechAct: apology
SpeechAct: implicit_confirm
SpeechAct: offer
SpeechAct: present_info
SpeechAct: status_report
SpeechAct: unmatched
SpeechActsHist: [correct_info]

SpeechActsHist: [no_answer]
SpeechActsHist: [reject_info]
SpeechActsHist: [yes_answer]
SpeechActsHist: apology
SpeechActsHist: implicit_confirm
SpeechActsHist: offer
SpeechActsHist: present_info
SpeechActsHist: status_report
SpeechActsHist: unmatched
Task: [accept_car_offer]
Task: [accept_flight_summary]
Task: [continue_depart_date]
Task: [continue_dest_city]
Task: [continue_trip]
Task: [depart_time]
Task: [hotel_location]
Task: [no_continue_trip]
Task: [not_grounded_depart_date_or_time]
Task: [orig_city]
Task: [reject_flight_offer]
Task: [reject_hotel_offer]
Task: [return_date]
Task: [return_trip]
Task: continue_trip
Task: depart_arrive_time
Task: flight
Task: ground
Task: hotel_booking
Task: hotel_name
Task: meta_correction
Task: meta_greeting_goodbye
Task: meta_inconsistent_info
Task: meta_repetition
Task: meta_situation_info
Task: meta_start_over
Task: orig_dest_city
Task: price
Task: rental_booking
Task: rental_type
Task: return_date
Task: top_level_trip
Task: trip_type
Task: unmatched
TasksHist: [accept_car_offer]
TasksHist: [accept_flight_summary]
SpeechActsHist: [provide_info]
SpeechActsHist: [reprovide_info]
SpeechActsHist: acknowledgement
SpeechActsHist: explicit_confirm
SpeechActsHist: instruction
SpeechActsHist: opening_closing
SpeechActsHist: request_info
SpeechActsHist: tbc
Task: []
Task: [accept_flight_offer]
Task: [accept_hotel_offer]
Task: [continue_depart_time]
Task: [continue_orig_city]
Task: [depart_date]
Task: [dest_city]
Task: [hotel_name]
Task: [no_return_trip]
Task: [not_grounded_return_date_or_time]
Task: [reject_car_offer]
Task: [reject_flight_summary]
Task: [rental_company]
Task: [return_time]
Task: airline
Task: depart_arrive_date
Task: dest_city
Task: flight_booking
Task: hotel
Task: hotel_location
Task: meta_ambiguity_resolution
Task: meta_error
Task: meta_hangup
Task: meta_instruct
Task: meta_request_change
Task: meta_slu_reject
Task: orig_city
Task: orig_dest_date
Task: rental
Task: rental_day_time
Task: retrieval
Task: tbc
Task: trip
Task: unknown
TasksHist: []
TasksHist: [accept_flight_offer]
TasksHist: [accept_hotel_offer]

TasksHist: [continue_depart_date]
TasksHist: [continue_dest_city]
TasksHist: [continue_trip]
TasksHist: [depart_time]
TasksHist: [hotel_location]
TasksHist: [no_continue_trip]
TasksHist: [not_grounded_depart_date_or_time]
TasksHist: [orig_city]
TasksHist: [reject_flight_offer]
TasksHist: [reject_hotel_offer]
TasksHist: [return_date]
TasksHist: [return_trip]
TasksHist: continue_trip
TasksHist: depart_arrive_time
TasksHist: flight
TasksHist: ground
TasksHist: hotel_booking
TasksHist: hotel_name
TasksHist: meta_correction
TasksHist: meta_greeting_goodbye
TasksHist: meta_inconsistent_info
TasksHist: meta_repetition
TasksHist: meta_situation_info
TasksHist: meta_start_over
TasksHist: orig_dest_city
TasksHist: price
TasksHist: rental_booking
TasksHist: rental_type
TasksHist: return_date
TasksHist: top_level_trip
TasksHist: trip_type
TasksHist: unmatched
Turn: user
Task: meta_end_dialog
FilledSlot: [accept_hotel_offer,hotel_name]
FilledSlot: [orig_city,dest_city,depart_date]
FilledSlot: [return_date,return_time]
SpeechAct: [no_answer,provide_info]
SpeechAct: [reprovide_info,provide_info]
SpeechAct: [yes_answer,provide_info]
Task: [accept_car_offer,rental_company]
Task: [depart_time,depart_date]
Task: [orig_city,dest_city]
TasksHist: [continue_depart_time]
TasksHist: [continue_orig_city]
TasksHist: [depart_date]
TasksHist: [dest_city]
TasksHist: [hotel_name]
TasksHist: [no_return_trip]
TasksHist: [not_grounded_return_date_or_time]
TasksHist: [reject_car_offer]
TasksHist: [reject_flight_summary]
TasksHist: [rental_company]
TasksHist: [return_time]
TasksHist: airline
TasksHist: depart_arrive_date
TasksHist: dest_city
TasksHist: flight_booking
TasksHist: hotel
TasksHist: hotel_location
TasksHist: meta_ambiguity_resolution
TasksHist: meta_error
TasksHist: meta_hangup
TasksHist: meta_instruct
TasksHist: meta_request_change
TasksHist: meta_slu_reject
TasksHist: orig_city
TasksHist: orig_dest_date
TasksHist: rental
TasksHist: rental_day_time
TasksHist: retrieval
TasksHist: tbc
TasksHist: trip
TasksHist: unknown
Turn: system
Task: meta_release_turn
FilledSlot: [accept_car_offer,rental_company]
FilledSlot: [depart_time,depart_date]
FilledSlot: [orig_city,dest_city]
SpeechAct: [no_answer,correct_info]
SpeechAct: [reject_info,correct_info]
SpeechAct: [reprovide_info,reprovide_info]
SpeechAct: [yes_answer,reprovide_info]
Task: [accept_hotel_offer,hotel_name]
Task: [orig_city,dest_city,depart_date]
Task: [return_date,return_time]