



D4.2: Showcase exhibiting Reinforcement Learning for dialogue strategies in the in-car domain

Oliver Lemon, Kallirroi Georgila, Matthew Stuttle

Distribution: Public

TALK

Talk and Look: Tools for Ambient Linguistic Knowledge
IST-507802 Deliverable 4.2

August 29, 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	M18 = June 2005
Actual date of delivery	August 29, 2005
Deliverable number	4.2
Deliverable title	D4.2: Showcase exhibiting Reinforcement Learning for dialogue strategies in the in-car domain
Type	Report + Prototype
Status & version	Final 1.0
Number of pages	40 (excluding front matter)
Contributing WP	4
WP/Task responsible	UEDIN
Other contributors	
Author(s)	Oliver Lemon, Kallirroï Georgila, Matthew Stuttle
EC Project Officer	Evangelia Markidou
Keywords	Dialogue system, Information State Update, In-car, Reinforcement Learning, Clarification, Multimodal

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
	Deutsches Forschungszentrum für 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>

©2005, The Individual Authors

No part of this document may be reproduced or transmitted in any form, or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission from the copyright owner.

Contents

Summary	1
1 Introduction	2
1.1 System overview	2
1.1.1 A system exhibiting Reinforcement Learning	3
1.1.2 Overview of system features	3
1.2 Research issues	4
1.2.1 Moving between domains: COMMUNICATOR and in-car dialogues	4
1.2.2 Fragmentary clarifications	4
1.3 Contents of the software distribution	5
2 The “in-car” scenario	6
2.1 The town environment	7
3 Component-level description	8
3.1 System components	8
3.1.1 DIPPER dialogue manager	8
3.1.2 ATK speech recogniser	9
3.1.3 Dialogue policy learner agent	13
3.1.4 Festival2 speech synthesizer	15
3.1.5 Map Agent	15
3.1.6 Database agent	17
3.1.7 Parsing	19
4 Feature level description	21
4.1 Features of the Baseline system	21
4.2 Example multimodal dialogue	22
5 Conclusion and future work	24
5.1 Summary of achievements	24
5.2 Possibilities for further system development	25
5.3 Planned evaluation and experiments	25

A The Information State definition, an example Information State in XML format, and the Database for the SACTI Town and MySQL Agent 27

Summary

This report accompanies a software deliverable and gives details of the software: a prototype multimodal dialogue system using reinforcement learning for in-car scenarios, developed at Edinburgh University and Cambridge University for the TALK project. This prototype is the first “Information State Update” (ISU) dialogue system to exhibit reinforcement learning of dialogue strategies, and also has a novel fragmentary clarification feature. A demonstration of the prototype is available on request. This report describes the main components and functionality of the system, as well as the purposes and future use of the system, and surveys the research issues involved in its construction. Evaluation of this system (i.e. comparing the baseline system with hand-coded versus learned dialogue policies) is a task planned for later in the project (late 2005/early 2006). Our initial evaluation of learned dialogue policies [LGH⁺05, HLG05] suggests that the learned policy system will perform at least as well as a reasonable hand-coded system (the TALK policy learned for COMMUNICATOR dialogue management outperforms all the individual hand-coded COMMUNICATOR systems).

Chapter 1

Introduction

The baseline in-car system described below has been constructed primarily in order to be able to collect data for Reinforcement Learning (RL) approaches to multimodal dialogue management, and also to test and further develop learnt dialogue strategies in a realistic application scenario. For these reasons we were required to build a system which:

- covers a realistic domain of useful “in-car” conversation and a wide range of dialogue phenomena (e.g. confirmation, initiative, clarification, information presentation)
- can be used to complete measureable tasks (i.e. there is a measure of successful and unsuccessful dialogues useable as a reward signal for Reinforcement Learning)
- logs all interactions in a standard conforming to the TALK data collection format (see TALK status report T6.5s1 [YSW⁺05])
- contains an interface to a dialogue strategy learner module
- allows multimodal interactions.

In this report we will describe the software system that we have developed to meet these requirements. First we describe the domain in which the dialogue system operates (an “in-car” information system). Then we will describe the major components of the system and give examples of their use. We then describe the important features of the system in respect of the dialogue phenomena that they support. Finally, we will describe the future directions for this work.

1.1 System overview

The baseline dialogue system is built around the DIPPER dialogue manager [BKLO03]. This system is initially 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 have then modified the DIPPER dialogue manager so that it can consult learnt strategies (for example strategies learnt from the 2000 and 2001 COMMUNICATOR data [LGH⁺05]), based on its current information state, and then execute dialogue

actions from those strategies. This allows us to compare hand-coded against learnt strategies within the same system (i.e. the other components such as the speech-synthesizer, recogniser, GUI, etc. all remain fixed).

1.1.1 A system exhibiting Reinforcement Learning

The central motivation for building this dialogue system is as a platform for Reinforcement Learning (RL) experiments. The system exhibits RL in 2 ways:

- It can be run in online learning mode with real users. Here the RL agent is able to learn from successful and unsuccessful dialogues with real users. Learning will be much slower than with simulated users, but can start from an already learnt policy, and slowly improve upon that.
- It can be run using an already learnt policy (e.g. the one reported in D4.1 [LGH⁺05], learnt from COMMUNICATOR data). This mode can be used to test the learnt policies in interactions with real users.

Please see TALK deliverable D4.1 [LGH⁺05] for an explanation of the techniques developed for Reinforcement Learning with ISU dialogue systems.

1.1.2 Overview of system features

The following features are currently implemented in the baseline system:

- Interfaced to learnt dialogue policies
- Multiple tasks: information seeking for hotels, bars, and restaurants
- Basic slot-filling dialogues
- Overanswering/ user-initiative
- Open speech recognition using n-grams
- Use of dialogue plans
- Open-initiative initial system question
- Basic user-goal/task recognition
- Confirmations - explicit and implicit based on ASR confidence
- Fragmentary clarifications based on word confidence scores
- Template-based descriptions of database entities
- Multimodal output - highlighting and naming entities on GUI
- Startover, quit, and help commands
- Simple user commands (e.g. "Show me all the indian restaurants")
- Logging in TALK ISU format (see example in the appendix)

1.2 Research issues

As well as the general research involved in building a rich multimodal dialogue system, the work presented here explores a number of research themes, in particular the issues of using learnt dialogue policies, learning dialogue policies in online interaction with users, fragmentary clarification, and reconfigurability.

1.2.1 Moving between domains: COMMUNICATOR and in-car dialogues

The learnt policies in deliverable D4.1 [LGH⁺05] focussed mostly on the COMMUNICATOR systems, which were used for flight-booking dialogues. There we reported learning a promising initial policy for COMMUNICATOR dialogues, but the issue arises of how we could transfer this policy to new domains – for example the in-car domain.

In the in-car scenarios the genre of “information seeking” is 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), see [YSW⁺05].

One question we address in this 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 learnt from our experiments with COMMUNICATOR will also be good policies for other slot-filling tasks – that is, that we are learning “generic” slot-filling or information seeking dialogue policies. In chapter 3 we describe how the dialogue policies learnt for slot filling on the COMMUNICATOR data set can be abstracted and used for slot filling in the in-car scenarios.

1.2.2 Fragmentary clarifications

Another research issue we have been able to explore in constructing this system is the issue of generating fragmentary clarifications. Instead of a system simply saying “Sorry, please repeat that” or some such similar simple clarification request when there is a speech recognition failure, we were able to use the word confidence scores output by the ATK speech recognizer to generate more intelligent fragmentary clarification requests such as “Did you say a cheap chinese restaurant?”. This works by obtaining an ASR confidence score for each recognized word. We are then able to try various techniques for clarifying only the user utterance. Many possibilities arise, for example:

- explicitly clarify only the highest scoring content word below the rejection threshold
- explicitly clarify only the lowest scoring content word
- explicitly clarify all content words under the rejection threshold
- implicitly clarify all content words, explicitly clarify the lowest scoring content word
- implicitly clarify the highest scoring content word below the rejection threshold, and explicitly clarify the lowest scoring content word
- implicitly clarify the highest scoring content word below a threshold specific to implicit confirmation, or explicitly clarify the highest scoring content word below a threshold specific to explicit confirmation (always lower than the implicit confirmation threshold).

We have implemented the last of these strategies to date. The current platform enables us to test alternative strategies, and develop more complex ones. One future possibility is to combine n-best processing with word-confidence scores.

1.3 Contents of the software distribution

The software in this deliverable consists of the:

- DIPPER dialogue manager (information state, update rules, resources)
- ATK speech recognizer and Language Models
- Dialogue policy learner agent
- Festival speech synthesizer agent
- Map agent and map for the SACTI in-car domain
- Database agent and MySQL database for the SACTI in-car domain.

Additional software required to run the system, but not produced by the TALK project (e.g. Java, OAA, Prolog) can be downloaded and installed via the internet.

Chapter 2

The “in-car” scenario

The scenario we have designed the system to cover is that of information seeking about a town, for example its hotels, restaurants, and bars. We imagine a driver who is travelling towards this town, or is already there, who wishes to accomplish relatively complex tasks, such as finding an Italian restaurant near their hotel, or finding all the wine bars in town, and so on. The driver/user should be able to specify queries using natural dialogue, and will receive system output that is a mixture of spoken and graphical information (e.g. a description of an item and a map showing its location). In this baseline system we do not accommodate driver graphical input, since this will not be possible in the hands-busy condition of driving. (In future versions we may accommodate gestural input via the ErgoCommander interface.)

The example town is taken from the UCAM SACTI corpus collection materials, and contains a number of hotels, bars, restaurants, and tourist attractions. The database describing these entities is given in the appendix.

The user should be able to get information on a range of locations in the town, and the dialogue system will be used to specify and refine the user queries, as well as to present information to the user.

The following dialogue illustrates these two basic processes:

- System: How can I help you?
- User: I'm looking for a cheap Italian restaurant.
- System: Would you like something central?
- User: I don't mind.
- System: There are 2 restaurants matching your requirements. La Casa on Main street is an affordable Italian restaurant with cheerful decor [Display on MAP]. Goodfellas on High street serves a range of pizzas [Display on MAP]. Which restaurant option would you like?
- User: The first.
- System: The phone number of La Casa [Display on MAP] is 237-6753.

2.1 The town environment

Figure 2.1 shows the map of the town used for the system. The system has the ability to highlight and name entities on this map. This functionality is described in section 3.1.5.

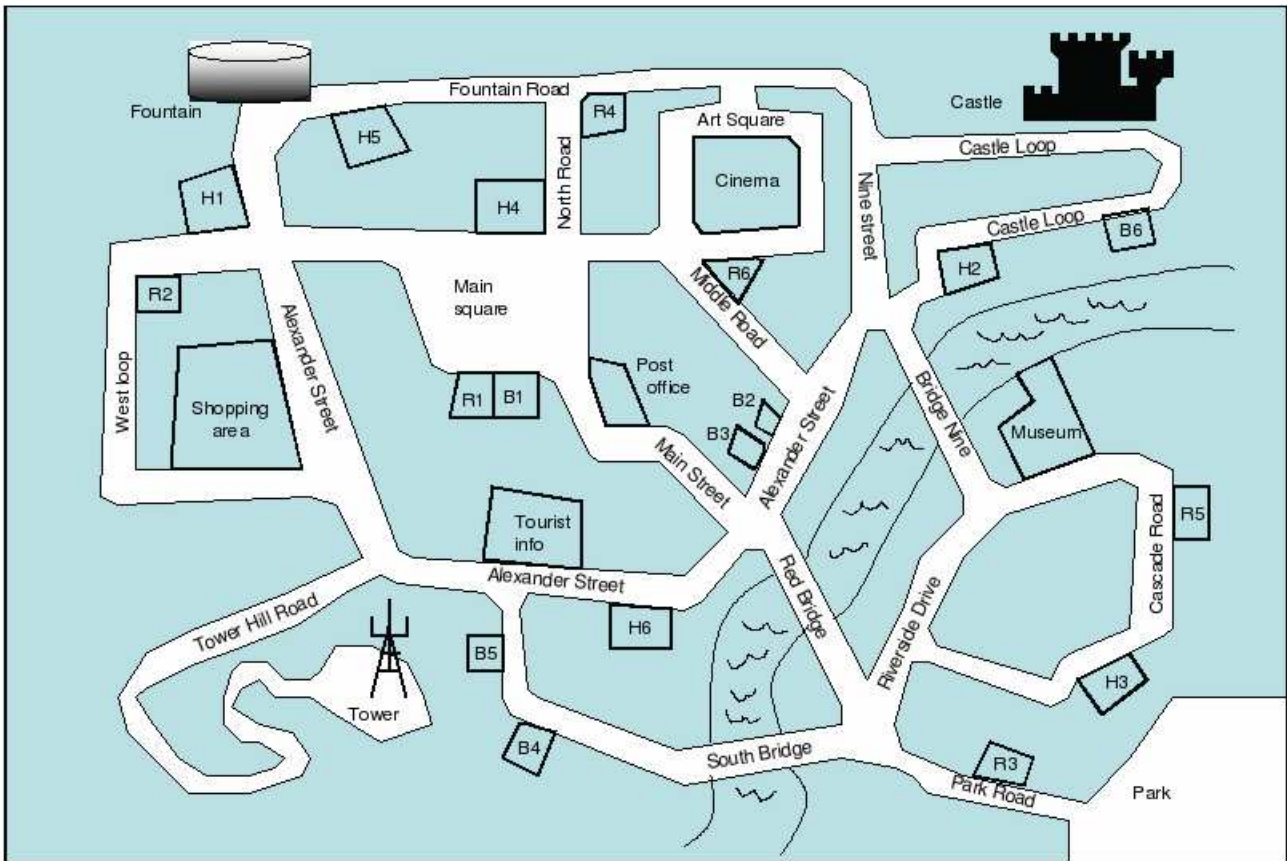


Figure 2.1: The SACTI map for the in-car scenario. R1-5 are restaurants. H1-6 are hotels, and B1-5 are bars.

The entities in this environment are described in a MySQL database (see section 3.1.6) and are listed in the appendix.

We now describe the dialogue system components.

Chapter 3

Component-level description

This chapter describes the components of the baseline in-car dialogue system. Communication between components is handled by OAA's asynchronous hub architecture [CM01], and all components currently run under Linux.

3.1 System components

The major components are:

- DIPPER [BKLO03] for ISU-based dialogue management¹
- ATK for speech recognition²
- Dialogue Policy Learner Agent³
- Festival2 [TBC98] for speech synthesis⁴
- Multimodal Map interface (an OAA agent written in java)
- Database agent (an OAA wrapper to MySQL, written in java)

We now describe these components.

3.1.1 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

¹Both Java and Prolog version are available, with OAA wrappers.

²See <http://mi.eng.cam.ac.uk/~sjy/software.htm>. OAA wrapper developed in C++.

³This is written in Python and has an OAA wrapper in C.

⁴OAA wrapper developed in C.

variety of supporting Open Agent Architecture (OAA, [CM01]) “agents” for dialogue system tasks such as speech recognition.

DIPPER follows Trindi [LT00] closely, taking its record structure and datatypes to define information states (see an example information state in figure 3.1.1). 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 “solvable” 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 figures 3.2 and 3.3) 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 $\langle \text{name}, \text{conditions}, \text{effects} \rangle$. 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 `is^lastspeaker` refers to the the field `lastspeaker` in the record `is`.

In the update rules shown in figures 3.2 and 3.3 `solve2` is used to request OAA services. For example

```
solve2(text-to-speech(is^output))
```

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, but they could be handled by any other agent which registers the text-to-speech solvable with the OAA facilitator (the hub).

Figure 3.1 shows part of the Information State (IS) used in the in-car system. The full IS definition is given in the appendix.

Figure 3.2 shows an update rule which initialises the ATK speech recogniser and the file where the logging agent will store all the information states of the dialogue.

Figure 3.3 shows an update rule which triggers explicit confirmation requests from the system. It is triggered when the ASR confidence is lower than a threshold, and there have been fewer than 3 explicit confirmation attempts on the slot value. The main effects are to generate an explicit confirmation request and maintain the speech act history.

Figure 3.4 shows an example of DIPPER’s graphical user interface for designing and debugging the dialogue manager.

3.1.2 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

```

infostate(record([is:record([
    flglearn:atomic,
    numinformationstate:atomic,
    turn:atomic,
    speaker:atomic,
    lastspeaker:atomic,
    input:stack(atomic),
    lastinput:stack(atomic),
    lastinputconf:stack(atomic),
    lastinputconfuttered:stack(atomic),
    output:stack(atomic),
    nextmoves:stack(Acts),
    cursysmove:stack(Acts),
    lastmoves:stack(Acts),
    message:stack(atomic),
    confirmnumber:atomic,
    confidenceword:atomic,
    confidence:atomic,
    implicitconf:atomic,
    ngram:atomic,
    speechact:stack(atomic),
    speechactshist:stack(atomic),
    subtask:stack(atomic),
    subtaskshist:stack(atomic),
    task:stack(atomic),
    taskstep:atomic,
    taskstepname:atomic,
    prompttype:atomic,
    filledslotsvaluesshow:stack(atomic),
    filledslotsshow:stack(atomic),
    filledslotsvalues:stack(atomic),
    filledslots:stack(atomic),
    groundedslots:stack(atomic),
    filledslotsvalueshist:stack(atomic),
    filledslotshist:stack(atomic),
    groundedslotshist:stack(atomic),
    status:atomic,
    int:stack(Acts)]])) :-
Acts = record([pred:atomic,
               dp:atomic,
               prop:record([pred:atomic,
                           args:stack(atomic)])])).

```

Figure 3.1: Part of the in-car system Information State definition

```

urule(initialisation,
  [
    is^lastspeaker=''      %%% CONDITIONS (=start of dialogue)
  ],
  [
    %%% EFFECTS: OAA calls and IS updates
    assign(is^flglearn, 0), % 0 for fixed dialogue policy, 1 for learnt dialogue policy

% initialise GUI with SACTI map
    solve(displayAgent_change_background_image('/talk/incar/guiagent/sactiMap.gif',
      ResDisplay)),

% ATK speech recogniser initialisation
    solve2(atkinitrecogniser('ATKrecognisercfg','general_diction_ucase',
      'yesno_lattice_ucase_o,wordloopnew',Res)), % Result 0 is ok, 1 problem
    assign(is^recognflag, Res),

    prolog(write('ATK initialized')),
    prolog(nl),

% open file for logging
    solve2(openissequencefile('/group/project/talk/incar/incar_log.txt',ResTxt)),
    assign(is^txtflag, ResTxt),

% update Information State
    assign(is^lastspeaker, user),
    assign(is^turn, system),
    assign(is^status, initialisation),

    assign(is^dbretrievalstep, 3), % the system will present 3 options at a time

    assign(is^flgtaskchange, 0),
    push(is^task, 'task'),
    assign(is^taskstepname, 'greet'),
    assign(is^taskstep, 0),

    push(is^lastinputconf,[null]),
    push(is^lastinputconfuttered,[null]),
    push(is^cursysmove,null)
  ]
).

```

Figure 3.2: Initialisation Update rule from the in-car system


```

urule(generation_confirmation_explicit,
  [
    %%% CONDITIONS
    is^confirmnumber\=3,
    is^confidence=1, % i.e. ASR conf lower than confirmation threshold
    is^lastspeaker=user,
    prolog(check_slot('([quit_request],u)',top(is^filledslots),ResSlot)),
    ResSlot=0,
    prolog(check_slot('([restart_request],u)',top(is^filledslots),ResSlot1)),
    ResSlot1=0,
    prolog(check_slot('([ask_happy],s)',top(is^filledslots),ResSlot2)),
    ResSlot2=0,
    is^flgsave=0
  ],
  [
    %%% EFFECTS
    % increment the counter
    prolog(increase_num(is^confirmnumber,ConfNumber)),
    assign(is^confirmnumber,ConfNumber),
    % generate explicit_confirm prompt (TextOut)
    prolog(utter_prompt('([explicit_confirm],s)',top(is^lastinputconfuttered),TextOut,
      Type,ValueYes,ValueNo,SpeechAct,top(is^task),is^taskstepname)),
    % say TextOut via Festival TTS
    solve2(callfestival('/festival/bin',TextOut,_Y)),
    % update Information State
    assign(is^confidence,0),
    push(is^lastmoves,['([explicit_confirm],s)']),
    clear(is^subtask),
    push(is^subtask, is^taskstepname),
    clear(is^speechact),
    push(is^speechact, SpeechAct),
    push(is^subtaskshist, top(is^subtask)),
    push(is^speechactshist, top(is^speechact)),
    clear(is^output),
    push(is^output,TextOut),
    assign(is^prompttype,Type),
    assign(is^lastspeaker,system),
    clear(is^input),
    clear(is^recogninput),
    clear(is^incrementinput),
    assign(is^turn,user),
    assign(is^flgsave, 1) % now the system will be ready to save the information state
  ]
).

```

Figure 3.3: Explicit confirmation update rule from the in-car system

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 acoustic models trained on the SACTI corpora. Lattices are the language models used during recognition. They can be either grammar-based or n-grams. 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 (see figure 3.1.1).

- `atkcallrecogniser(LatticeName, is \hat{n} gram, RecString, ConfWord, Conf)`: this solvable is called 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 and the recognition confidence score both for the whole utterance and the individual word confidence scores. The second argument specifies whether we are using n-grams or grammars.
- `atkmfreerecogniser(Res)`: this solveable is called at the end of the dialogue to free memory and produces 0 for success and 1 if an error occurs.

Language modelling

Currently for language modelling we use bigram language models built from the UCAM SACTI corpora. However, the agent can also support trigrams and grammar-based language models.

3.1.3 Dialogue policy learner agent

This agent is based on the “RLsimulation” agent described in [LGH⁺05] section 5.2.1. As described there it acts as an interface between the DIPPER dialogue manager and the system simulation based on RL. In particular it has the following solvable:

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

The argument `system` shows that the agent is used for driving the system (based on RL). The second argument is the name of the file that contains all information about the current information state, which is required by the RL algorithm to produce an action. The action returned by the RL agent is a combination of conversational domain, speech act, and task. The `result` is 0 in success and 1 in failure. When run in online learning mode the agent not only produces an action when supplied with a state, but at the end of every dialogue it uses the reward signal to update its learnt policy. The reward signal is defined in the RL agent, and is currently a linear combination of task success metrics combined with a fixed penalty for dialogue length (see TALK D4.1 [LGH⁺05]).

This agent can be called whenever the system has to decide on the next dialogue move to make. In the original hand-coded system this decision is made by way of a dialogue plan (using the “deliberate” solveable). The RL agent can be used to drive the entire dialogue policy, or can be called only in certain circumstances. This makes it useable for whole dialogue strategies, but also, if desired, it can be targetted only on specific dialogue management decisions (e.g. implicit versus explicit confirmation, as was done by [LKS00]).

One important research issue is that of transferring learnt strategies between domains. We learnt a strategy for the COMMUNICATOR flight booking dialogues [LGH⁺05, HLG05], but this is generated by rather different scenarios than the in-car dialogues. However, both are “slot-filling” or information-seeking applications. We defined a mapping (described below) between the states and actions of both systems, in order to construct an interface between the learnt policies for COMMUNICATOR and the in-car baseline system.

Mapping between COMMUNICATOR and the in-car system

There are 2 main problems to be dealt with here:

- mapping between in-car system information states and COMMUNICATOR information states
- mapping between learnt COMMUNICATOR system actions and in-car system actions.

The learnt COMMUNICATOR policy tells us, based on a current IS, what the optimal system action is (for example `request_info(dest_city)` or `acknowledgement`). Obviously, in the in-car scenario we have no use for task types such as “destination city” and “departure date”. Our method therefore is to abstract away from the particular details of the task type, but to maintain the information about dialogue moves and the *slot numbers* that are under discussion. That is, we construe the learnt COMMUNICATOR policy as a policy concerning how to fill up to 4 (ordered) informational slots, and then access a database and present results to the user. We also note that some slots are more essential than others. For example, in COMMUNICATOR it is essential to have a destination city, otherwise no results can be found for the user. Likewise, for the in-car tasks, we consider the food-type, bar-type, and hotel-location to be more important to fill than the other slots. This suggests a partial ordering on slots via their importance for an application.

In order to do this we define the following mappings between COMMUNICATOR dialogue actions and in-car dialogue actions, for each sub-task type of the in-car system.

COMMUNICATOR action	in-car action
dest-city	food-type
depart-date	food-price
depart-time	food-location
dest-city	hotel-location
depart-date	room-type
depart-time	hotel-price
dest-city	bar-type
depart-date	bar-price
depart-time	bar-location

Table 3.1: Action mappings

Note that we treat each of the 3 in-car subtasks (hotels, restaurants, bars) as a separate slot-filling dialogue thread, governed by COMMUNICATOR actions. This means that the very top level of the dialogue (“How may I help you”) is not governed by the learnt policy. Only when we are in a recognized task do we ask

the COMMUNICATOR policy for the next action. Since the COMMUNICATOR policy is learnt for 4 slots, we “pre-fill” a slot in the IS when we send it to the Dialogue Policy Learner Agent in order to retrieve an action.

As for the state mappings, these follow the same principles. That is, we abstract from the in-car states to form states that are usable by COMMUNICATOR . This means that, for example, an in-car state where food-type and food-price are filled with high confidence is mapped to a COMMUNICATOR state where dest-city and depart-date are filled with high confidence, and all other state information is identical (modulo the task names). Note that in a future version of the in-car system where task switching is allowed we will have to maintain a separate view of the state for each task.

In terms of the integration of the learnt policies with the DIPPER system update rules, we have a system flag which states whether or not to use a learnt policy. If this flag is present, a different update rule fires when the system determines what action to take next. For example, instead of using the `deliberate` predicate to access a dialogue plan, we instead call the Dialogue Policy Learner Agent via OAA, using the current Information State of the system. This will return a dialogue action to the DIPPER update rule. In future work we will evaluate how well the learnt policies work for real users of the in-car system.

3.1.4 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. Currently we use the default festival voice, though it is possible to use limited domain voices.

3.1.5 Map Agent

A multimodal map agent was developed (in java) at DFKI and integrated into the DIPPER dialogue manager by UEDIN. It uses the map shown in chapter 2 to display information to the user. It has the following OAA solveables:

```
+-----+
| displayAgent_change_Background_image(PathToNewImage, Result) |
+-----+
change the background image
- example:
  oaa_Solve(displayAgent_change_background_image('../data/userMap.gif', 'Result'), [])

+-----+
| displayAgent_display_circle(PositionX, PositionY, Radius, RGB_Red,RGB_Green,
RGB_Blue, Label, Result ) |
+-----+
draw a circle around a point (x,y) with the given radius and the
```

```

defined RGB-color
  RED   : (255,0,0)
  GREEN : (0,255,0)
  BLUE  : (0,0,255)

```

- example:

```
oaa_Solve(displayAgent_display_circle(150,100,50,255,0,0,'myCircle','Result'), [])
```

```

+-----+
| displayAgent_remove_circle(Label, Result) |
+-----+

```

remove the circle

- example:

```
oaa_Solve(displayAgent_remove_circle('myCircle','Result'), [])
```

```

+-----+
| displayAgent_add_Button(PositionX, PositionY, Width, Height, Label, Result) |
+-----+

```

add a button to the frame.

The button sends out a solvable request("NameOfTheButton_pressed"), which must be solved by another agent.

- example:

```
oaa_Solve(displayAgent_add_button(100,100,50,10,'MyButton','Result'),[])
```

```

+-----+
| displayAgent_remove_Button(Label, Result) |
+-----+

```

remove the buttons which were added by the addButton request

- example: oaa_Solve(displayAgent_remove_button('MyButton','Result'), [])

```

+-----+
| displayAgent_display_text(PositionX, PositionY, Text, Label, RGB_Red, RGB_Green,
RGB_Blue, Result) |
+-----+

```

display a given text at point (x,y) with a label and a

```

defined RGB-color
  RED   : (255,0,0)
  GREEN : (0,255,0)

```

BLUE : (0,0,255)

- example:

```
oaa_Solve(displayAgent_display_text(100,100,'MyText','MyLabel',255,0,0,
'Result'), [])
```

```
+-----+
| displayAgent_remove_text(Label, Result) |
+-----+
```

remove the texts with the given label which added by the display_text request

- example: oaa_Solve(displayAgent_remove_text('MyLabel', 'Result'), [])

```
+-----+
| displayAgent_remove_all(ObjectType, Result) |
+-----+
```

remove all Objects of type "ObjectType":

```
possibilities: 1. Circles
                2. Textlabels
                3. Buttons
```

- example: oaa_Solve(displayAgent_remove_all('Circles', 'Result'), [])

- example: oaa_Solve(displayAgent_remove_all('Textlabels', 'Result'), [])

- example: oaa_Solve(displayAgent_remove_all('Buttons', 'Result'), [])

3.1.6 Database agent

The database agent, written in java at DFKI, is a generic interface to the popular open source database server MySQL (<http://www.mysql.com/>). In this application we use the specific database shown in the appendix. Many application-specific solveables are implemented, (e.g. `getHotelByStreet(Street)`) to find a hotel in town by entering a street). However, the most generic OAA solveables are as follows:

```
+-----+
| getXYForEntity( Id, Name, Result ) |
+-----+
```

get the X and Y co-ordinates of an arbitrary object by its id or name.

Note: As we can get the co-ordinates by either id *or* name we use the empty list (e.g. []), to indicate that one of the arguments is empty.

If both arguemnts are given, id is used to determine the location.

examples:

```
oaa_Solve(getXYForEntity('H3', 'YOUTH HOSTEL', Result), [])
```

```

oaa_Solve(getXYForEntity('H3', [], Result), [])
oaa_Solve(getXYForEntity('H3', 'YOUTH HOSTEL', Result), [])

```

```

+-----+
| sendGenericDBQuery([column0, ... , columnN-1], [condition0, ... , conditionM-1], Result) |
+-----+
get N columns from those rows where M conditions hold.

```

Note that here the arguments are two lists and a variable in which we place the result. The first list indicates the columns we want to retrieve, e.g. description, cuisine. Note that all columns can be retrieved by using the wildcard “*”.

The N columns of row i are only retrieved when M conditions hold. A condition is “type=hotel” or “price ≤ 30” for example. Note that the relation between them is a logical AND. It is also possible to give zero conditions, and we indicate that with the empty list, e.g. “[]”.

Here are some examples of the use of the generic solveable:

1. `oaa_Solve(sendGenericDBQuery([id, name], ['type=hotel', 'price≤ 30'], Result), [])` will retrieve all hotels which are cheaper than 30 and print their ids and names, i.e. “Result” will be bound to:

```
[[H1, HOTEL PRIMUS],[H5, ART HOUSE HOTEL],[H6, ALEXANDER HOTEL]].
```

2. `oaa_Solve(sendGenericDBQuery([id, name, 'X', 'Y'], ['type=bar'], Result), [])` would retrieve all bars and print their id, name and location.
3. `oaa_Solve(sendGenericDBQuery([*], ['type=restaurant'], Result), [])` would retrieve all columns of all restaurants.
4. `oaa_Solve(sendGenericDBQuery([id, description], [], Result), [])` would retrieve the id and description of every entity in town.

Integration with DIPPER dialogue management

Integration with the DIPPER dialogue manager was performed at Edinburgh. When the user fills in slots regarding their search (e.g. all cheap italian restaurants), these are then sent by DIPPER to the Database agent, and solutions are returned. Based on the number of solutions returned by the database DIPPER describes the solutions to the user, and uses the Map Agent to display their positions on the map.

The baseline system output to the user is as follows:

- A sentence indicating the number of database results, e.g. “There are 4 items matching your requirements”
- A description of each of the items, pausing every 3 items, for example “The Bochka is a lively russian restaurant on Main Street.”
- A question asking the user if they wish to select one of these items, e.g. “Are you interested in any of these options or would you like me to continue?”

If the user then selects an option (e.g. “the first one please”) then that item only will be displayed on the map and some additional information will be presented to the user. (e.g. for restaurants, the address and the phone number).

3.1.7 Parsing

Parsing is performed by multiple prolog predicates in the DIPPER resources.pl file. The parser is robust and is based on patterns associated with the dialogue tasks and slots that have to be filled. For example, if the user says “I would like a single room in a hotel which is not central.” the following pattern matchings will take place: ‘single’ will match subtask ‘room-type’, ‘hotel’ will match subtask ‘general’ and ‘not central’ will match subtask ‘location’. Therefore the system will understand that the user needs information about hotels, will switch to hotel-task and fill slots ‘room-type’ with the value ‘single’ and ‘hotel-location’ with the value ‘not central’.

The result would have been the same even if the user’s utterance was ungrammatical and disfluent, e.g. “I think I want a, let me think, uh a single room in some hotel not central.”.



Figure 3.4: A DIPPER dialogue manager screenshot from the baseline system

Chapter 4

Feature level description

We first list the major features of the system and then give an example dialogue which illustrates the most important of them.

4.1 Features of the Baseline system

The following features are implemented in the baseline system:

- Interface to learnt dialogue policies
- Multiple tasks: information seeking for hotels, bars, and restaurants
- Basic slot-filling dialogues
- Overanswering/ user-initiative
- Open speech recognition using n-grams
- Use of dialogue plans
- Open-initiative initial system question
- Basic user-goal/task recognition
- Confirmations - explicit and implicit based on ASR confidence
- Fragmentary clarifications based on word confidence scores
- Template-based descriptions of database entities
- Multimodal output - highlighting and naming entities on GUI
- Startover, quit, and help commands
- Simple user commands (e.g. “Show me all the indian restaurants”)
- Logging in the TALK ISU format (see example in the appendix)

This leads to rich and novel dialogue capabilities, such as fragmentary clarification strategies, and the ability to log all interactions in a format that will be useful for future learning experiments.

The following example dialogue shows many of these capabilities:

4.2 Example multimodal dialogue

Speaker	Utterance	Feature
S	How can I help you?	Open initiative initial question
U	I'm looking for a cheap chinese restaurant.	User initiative
S	Would you like something central?	Intention recognition, Dialogue plan
U	Yes.	
S	The Dragon Way is an affordable cantonese restuarant located on Main Street [displayed on map].	Multimodal presentation
S	Are you happy with those options?	
U	No. I want a double room.	
S	Would you like something expensive cheap or reasonably priced?	Intention recognition, Dialogue plan
U	A luxury hotel, in the center of town.	
S	There are 2 hotel options meeting your requirements. The Royal Hotel [displayed on map] is a renovated nineteenth century palace, featuring private balconies. It has spacious well appointed rooms, many of which offer stunning views. All rooms are en suite, and breakfast is included. The Hotel Primus [displayed on map] has big, bright, clean rooms, some en suite, but with some noise from the busy street below. Which hotel option would you like?	Multimodal presentation (see figure 4.1)
U	The second one please.	
S	The phone number of the Hotel Primus is 2094-227.	
S	Would you like any further information?	
U	I want to find a jazz bar.	
S	I'm sorry, what kind of bar are you looking for?	Intention recognition, Fragmentary clarification
U	A jazz bar.	
S	A jazz bar. Would you like something central?	Implicit confirmation, Dialogue Plan
U	Yes please.	

Table 4.1: Example dialogue, showing system features

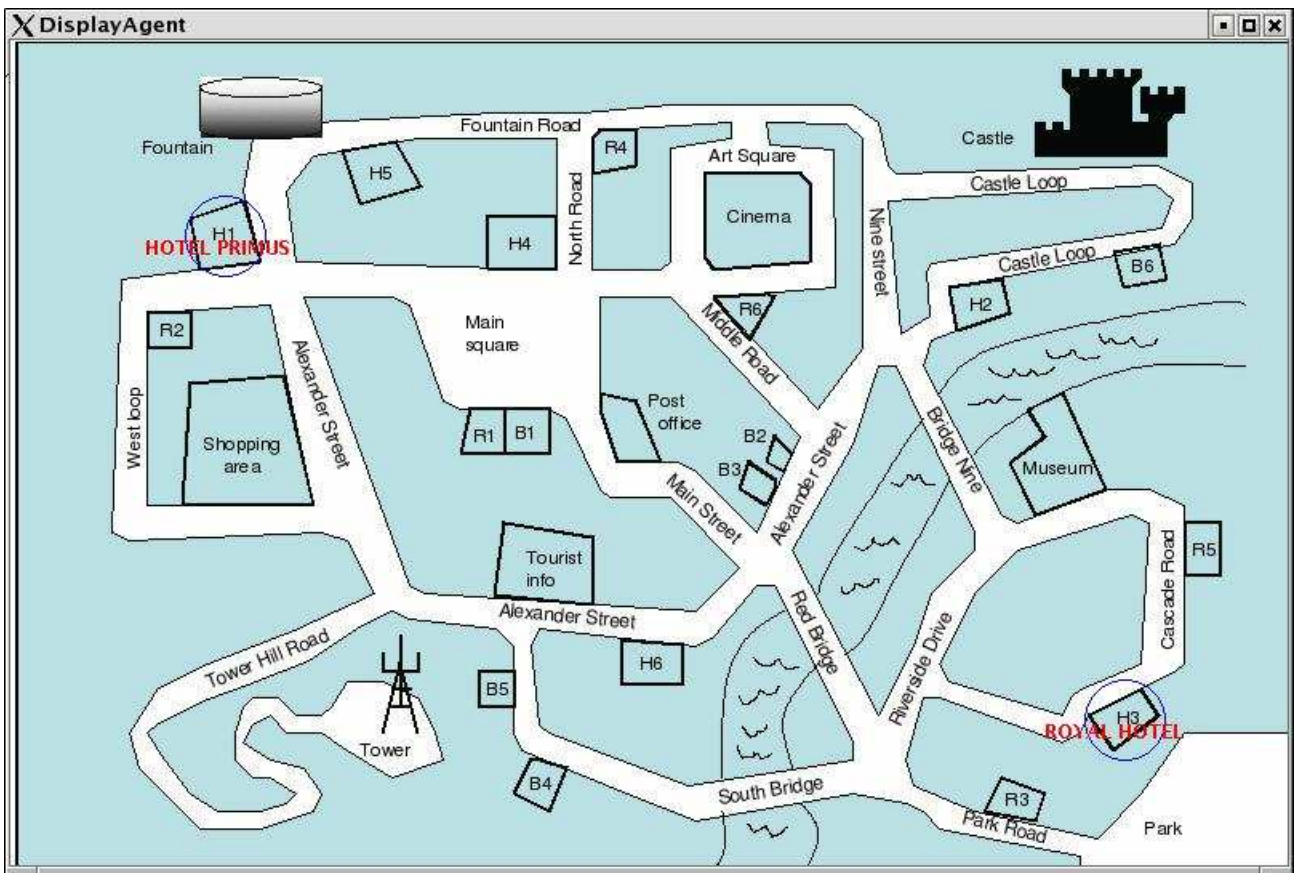


Figure 4.1: The multimodal map display for the example dialogue

Chapter 5

Conclusion and future work

This report has described work done in the TALK project in building a software prototype baseline “Information State Update” (ISU)-based dialogue system in the in-car domain, with the ability to use dialogue policies derived from machine learning and also to perform online learning through interaction. We described the scenarios, gave a component level description of the software, and a feature level description and example dialogue.

Evaluation of this system (i.e. comparing the baseline system with handcoded versus learned dialogue policies) is a task planned for later in the project (late 2005/early 2006). Our initial evaluation of learned dialogue policies [LGH⁺05, HLG05] suggests that the learned policy system will perform at least as well as a reasonable hand-coded system (the TALK policy learned for COMMUNICATOR dialogue management outperforms all the individual hand-coded COMMUNICATOR systems).

The primary evaluation of this system will compare learnt dialogue policies with the hand-crafted baseline system. This will produce data on the usability of the baseline system, as well as results on the effectiveness of the learnt dialogue policies. Another evaluation will test the novel fragmentary clarification strategies implemented in this baseline system.

5.1 Summary of achievements

The main achievements made in designing and constructing this baseline system have been:

- Combining learned dialogue policies with an ISU dialogue manager. This has been done for online learning, as well as for strategies learned offline.
- Mapping learned policies between domains. i.e. mapping Information States and system actions between DARPA COMMUNICATOR and in-car information seeking tasks.
- Fragmentary clarification strategies: the combination of ATK word confidence scoring with DIPPER ISU-based dialogue management rules allows us to explore novel clarification techniques.
- Software integration: building a functioning multimodal dialogue system with the ATK, DIPPER, MySQL, Festival, java DisplayAgent, and learned dialogue strategy components.

5.2 Possibilities for further system development

To further develop the system the following extensions are possible:

- Extend to the sightseeing/ tourist information tasks.
- Develop and test GF grammar for in-car scenarios.
- Interface GF with DIPPER.
- Combine word confidence scores with n-best lists for novel fragmentary clarification strategies.

5.3 Planned evaluation and experiments

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

- how good are the learnt strategies compared to the baseline system strategies, for real users?
- which state variables are particularly relevant to determining dialogue confirmation and information presentation strategies?
- what is the effect of different reward signals on the resulting dialogue policies?

Bibliography

- [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.
- [HLG05] James Henderson, Oliver Lemon, and Kallirroi Georgila. Hybrid Reinforcement/Supervised Learning for Dialogue Policies from COMMUNICATOR data. In *IJCAI workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2005.
- [LGH⁺05] Oliver Lemon, Kallirroi Georgila, James Henderson, Malte Gabsdil, Ivan Meza-Ruiz, and Steve Young. D4.1: Integration of Learning and Adaptivity with the ISU approach. Technical report, TALK Project, 2005.
- [LKSW00] Diane Litman, Micheal Kearns, Satinder Singh, and Marilyn Walker. Automatic optimization of dialogue management. In *Proc. COLING*, 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.
- [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.
- [YSW⁺05] Steve Young, Matt Stuttle, Karl Weilhammer, Oliver Lemon, and Nate Blaylock. T6.5s1: Annotated Data Archive (status report). Technical report, TALK Project, 2005.

Appendix A

The Information State definition, an example Information State in XML format, and the Database for the SACTI Town and MySQL Agent

The in-car system DIPPER Information State definition:

```
infostate(record([is:record([
    flglearn:atomic,
    flgsystemsimulation:atomic,
    flgfirsttime:atomic,
    flgopenfile:atomic,
    processing:atomic,
    rescontinuesystem:atomic,
    rescontinueuser:atomic,
    numinformationstate:atomic,
    numturn:atomic,
    turn:atomic,
    turnstarttime:atomic,
    turnendtime:atomic,
    turnnumber:atomic,
    turnsystem:atomic,
    turnuser:atomic,
    turnnumberlearn:atomic,
    turnsystemlearn:atomic,
    turnuserlearn:atomic,
    speaker:atomic,
    lastspeaker:atomic,
    utterancestarttime:atomic,
```


utteranceendtime:atomic,
utterancenumber:atomic,
dialogueacttype:atomic,
convdomain:atomic,
convdomainlearn:atomic,
input:stack(atomic),
transinput:stack(atomic),
recogninput:stack(atomic),
incrementinput:stack(atomic),
lastinput:stack(atomic),
lastinputconf:stack(atomic),
lastinputconfuttered:stack(atomic),
output:stack(atomic),
nextmoves:stack(Acts),
cursysmove:stack(Acts),
lastmoves:stack(Acts),
message:stack(atomic),
confirmnumber:atomic,
confidenceword:atomic,
confidence:atomic,
implicitconf:atomic,
ngram:atomic,
speechact:stack(atomic),
speechactshist:stack(atomic),
subtask:stack(atomic),
subtaskshist:stack(atomic),
speechactlearn:stack(atomic),
speechactshistlearn:stack(atomic),
subtasklearn:stack(atomic),
subtaskshistlearn:stack(atomic),
task:stack(atomic),
taskstep:atomic,
taskstepname:atomic,
tempvalueyes:atomic,
tempvalueno:atomic,
prompttype:atomic,
filledslotsvaluesshow:stack(atomic),
filledslotsshow:stack(atomic),
filledslotsvalues:stack(atomic),
filledslots:stack(atomic),
groundedslots:stack(atomic),
filledslotsvalueslearn:stack(atomic),
filledslotslearn:stack(atomic),
groundedslotslearn:stack(atomic),
qud:stack(atomic),

obligation:stack(atomic),
systemintention:stack(atomic),
usergoal:stack(atomic),
worderrorratenoins:atomic,
worderrorraterate:atomic,
sentenceerrorraterate:atomic,
keyworderrorraterate:atomic,
computeerrorratesreturnvalue:atomic,
comment:atomic,
filledslotsvalueshist:stack(atomic),
filledslotshist:stack(atomic),
groundedslotshist:stack(atomic),
filledslotsvalueshistlearn:stack(atomic),
filledslotshistlearn:stack(atomic),
groundedslotshistlearn:stack(atomic),
qudhist:stack(atomic),
obligationshist:stack(atomic),
systemintentionshist:stack(atomic),
usergoalshist:stack(atomic),
previouslyfilledslotsvalues:stack(atomic),
previouslyfilledslots:stack(atomic),
previouslygroundedslots:stack(atomic),
previouslyfilledslotsvalueslearn:stack(atomic),
previouslyfilledslotslearn:stack(atomic),
previouslygroundedslotslearn:stack(atomic),
status:atomic,
flgsave:atomic,
txtflag:atomic,
recognflag:atomic,
findinfoflag:atomic,
flgtaskchange:atomic,
helpflg:atomic,
returnstart:atomic,
reask:atomic,
deliberation:atomic,
realiserflag:atomic,
oplansteps:queue(atomic),
oplansteps1:queue(atomic),
optionindx:atomic,
dbidlist:atomic,
dbnamelist:atomic,
dbaddrlist:atomic,
dbphonestlist:atomic,
dbdesclist:atomic,
dbxlist:atomic,

```
dbylist:atomic,  
dbretrievalindex:atomic,  
dbretrievalstepindex:atomic,  
dbretrievalstep:atomic,  
dbquerylength:atomic,  
int:stack(Acts))])) :-  
Acts = record([pred:atomic,  
               dp:atomic,  
               prop:record([pred:atomic,  
                            args:stack(atomic)])])).
```

Example Information State in the TALK XML format:

```

<state state_number= "11">
  <dialogue_level>
    <turn> user </turn>
    <turn_number> 4 </turn_number>
    <speaker> user </speaker>
    <conversational_domain> about_task </conversational_domain>
    <speech_act> [yes_answer] </speech_act>
    <asr_input> yes </asr_input>
    <transcription_input> </transcription_input>
    <semantic_info_transcription> </semantic_info_transcription>
    <output> </output>
    <semantic_info_output> </semantic_info_output>
  </dialogue_level>
  <task_level>
    <task_subtask> [hotel_location] </task_subtask>
    <filled_slot> [hotel_location] </filled_slot>
    <filled_slot_value> [central] </filled_slot_value>
    <confirmed_slot> [hotel_location,hotel_price] </confirmed_slot>
    <obligation> </obligation>
    <system_intention> </system_intention>
  </task_level>
  <low_level>
    <word_error_rate_no_insertions> </word_error_rate_no_insertions>
    <word_confidence> 0.95 </word_confidence>
    <sentence_error_rate> </sentence_error_rate>
    <keyword_error_rate> </keyword_error_rate>
    <comment> </comment>
  </low_level>
  <history_level>
    <filled_slot_status> [top_level_trip],[room_type],[hotel_price],
      [hotel_location] </filled_slot_status>
    <filled_slot_value_status> [hotel],[single],[inexpensive]
      [central] </filled_slot_value_status>
    <confirmed_slot_status> [top_level_trip],[room_type],[hotel_price],
      [hotel_location] </confirmed_slot_status>
    <speech_act_history> opening_closing,request_info,[provide_info],
      request_info,[provide_info],implicit_confirm,request_info,
      [provide_info],implicit_confirm,request_info,[yes_answer]
    </speech_act_history>
    <task_subtask_history> meta_greeting_goodbye,top_level_trip,
      [top_level_trip],room_type,[room_type],room_type,hotel_price,
      [hotel_price],hotel_price,hotel_location,[hotel_location]
  </history_level>
</state>

```

```
</task_subtask_history>
<filled_slot_history> [top_level_trip],[room_type],[hotel_price],
    [hotel_location] </filled_slot_history>
<filled_slot_value_history> [hotel],[single],[inexpensive],
    [central] </filled_slot_value_history>
<confirmed_slot_history> [top_level_trip],[],[room_type],
    [hotel_location,hotel_price] </confirmed_slot_history>
</history_level>
</state>
```

MySQL Database for the SACTI Town:

```
-- MySQL dump 9.11
--
-- Host: localhost      Database: town_info
-- -----
-- Server version      4.0.24_Debian-5-log
--
-- Current Database: town_info
--

CREATE DATABASE /*!32312 IF NOT EXISTS*/ 'town_info2';

USE town_info2;

--
-- Table structure for table 'all_entities'
--

CREATE TABLE 'all_entities' (
  'key' int(11) NOT NULL default '0',
  'type' varchar(255) NOT NULL default 'none',
  'id' char(2) NOT NULL default 'no',
  'name' varchar(255) NOT NULL default 'none',
  'address' varchar(255) NOT NULL default 'none',
  'phone' varchar(255) NOT NULL default 'none',
  'price' varchar(255) NOT NULL default 'none',
  'per' varchar(255) NOT NULL default 'none',
  'currency' varchar(4) NOT NULL default '-1',
  'room_type' varchar(255) NOT NULL default 'none',
  'cuisine' varchar(255) NOT NULL default 'none',
  'location' varchar(255) NOT NULL default 'none',
  'description' text NOT NULL,
  'X' int(4) NOT NULL default '0',
  'Y' int(4) NOT NULL default '0',
  PRIMARY KEY ('key')
) TYPE=MyISAM;

--
-- Dumping data for table 'all_entities'
--

INSERT INTO 'all_entities' VALUES (0,'bar','B1','BAR METROPOL','Main square','
```

```
(7)812-310','moderate','none','none','none','none','central','has an upscale,
modern decor with live, soft jazz',324,245);
INSERT INTO 'all_entities' VALUES (1,'bar','B2','CAFE BLU','Alexander Street','
(7)812-314','inexpensive','none','none','none','none','central','has loud music,
cheap beer, dancing and is open until five in the morning',480,260);
INSERT INTO 'all_entities' VALUES (2,'bar','B3','BUFFALO BILLS','Alexander Street','
(7) 812-329','inexpensive','none','none','none','none','central','is known for
margaritas, tequilas, and Mexican beers',466,278);
INSERT INTO 'all_entities' VALUES (3,'bar','B4','EUROPA','South Bridge','
(2) 512 3283','expensive','none','none','none','none','not central','is a
tiny bar with exclusive clientele. Being on the guest list is a must.',331,464);
INSERT INTO 'all_entities' VALUES (4,'bar','B5','PLANTER\\\'S','South Bridge','
(4) 135 227','moderate','none','none','none','none','not central','is a wine bar
with an extensive selection of unusual and familiar wines',305,405);
INSERT INTO 'all_entities' VALUES (5,'bar','B6','MURPHY\\\'S','Castle Loop','
(6) 131 352','moderate','none','none','none','none','not central','is an authentic
Irish pub, serving an array of stouts',708,141);
INSERT INTO 'all_entities' VALUES (6,'hotel','H1','HOTEL PRIMUS','Alexander Street','
(2)094-227','expensive','none','none','single','none','central','has big, bright,
clean rooms, some en suite, but with some noise from the busy street below',134,122);
INSERT INTO 'all_entities' VALUES (7,'hotel','H1','HOTEL PRIMUS','Alexander Street','
(2)094-227','expensive','none','none','double','none','central','has big, bright,
clean rooms, some en suite, but with some noise from the busy street below',134,122);
INSERT INTO 'all_entities' VALUES (8,'hotel','H2','HOTEL PRESIDENT','Castle Loop','
(7)192-277','moderate','none','none','single','none','central','is a two night minimum
stay Business-style hotel and is efficiently staffed. All rooms are en suite and reakfast incl
INSERT INTO 'all_entities' VALUES (9,'hotel','H3','ROYAL HOTEL','Cascade Road','
(7)027-003','expensive','none','none','single','none','central','is a renovated
nineteenth century palace, featuring private balconies. It has spacious well
appointed rooms, many of which offer stunning views. All rooms are en suite, and
breakfast is included.',696,424);
INSERT INTO 'all_entities' VALUES (10,'hotel','H3','ROYAL HOTEL','Cascade Road','
(7)027-003','expensive','none','none','double','none','central','is a renovated
nineteenth century palace, featuring private balconies. It has spacious well appointed
rooms, many of which offer stunning views. All rooms are en suite, and breakfast is
included.',696,424);
INSERT INTO 'all_entities' VALUES (11,'hotel','H4','YOUTH HOSTEL','Main Square','
(7)281-446','cheap','none','none','single','none','central','is a hostel serving
students and budget travellers. It has bunk beds in shared rooms with shared toilet facilities
INSERT INTO 'all_entities' VALUES (12,'hotel','H5','ART HOUSE HOTEL','Fountain Road','
(7)252-996','cheap','none','none','single','none','central','is a basic hotel,
frequented by artists and musicians. It has rooms decorated in bright colours, but
few offer views, and all toilet facilities are shared. Breakfast is not included.',232,83);
INSERT INTO 'all_entities' VALUES (13,'hotel','H5','ART HOUSE HOTEL','Fountain Road','
(7)252-996','cheap','none','none','double','none','central','is a basic hotel,
```

```

frequented by artists and musicians. It has rooms decorated in bright colours, but
few offer views, and all toilet facilities are shared. Breakfast is not included.',232,83);
INSERT INTO 'all_entities' VALUES (14,'hotel','H6','ALEXANDER HOTEL','Alexander Street',
'(7)874-874','moderate','none','none','single','none','central','is very comfortable, and
pleasantly decorated but has small rooms. It has a very friendly staff. Toilet
facilities are shared and breakfast is included.',401,388);
INSERT INTO 'all_entities' VALUES (15,'hotel','H6','ALEXANDER HOTEL','Alexander Street',
'(7)874-874','moderate','none','none','double','none','central','is very comfortable, and
pleasantly decorated but has small rooms. It has a very friendly staff. Toilet
facilities are shared and breakfast is included.',401,388);
INSERT INTO 'all_entities' VALUES (16,'restaurant','R1','BOCHKA','Main square',
(2)095-252','cheap','none','none','none','tavern meals','central','is a tavern open
around the clock, serving excellent sandwiches, soups, tavern meals, and salads.',300,245);
INSERT INTO 'all_entities' VALUES (17,'restaurant','R2','SIBERIAN TIGER','West loop',
(7)095-926','expensive','none','none','none','russian','central','has savoury Russian
cooking, accompanied by first class music by Bolchoi musicians.',100,182);
INSERT INTO 'all_entities' VALUES (18,'restaurant','R3','SAINT PETERSBURG','Park Road',
(7)812-311','moderate','none','none','none','indian','central','is the most
interesting restaurant in town serving indian fusion food with chic decoration.',630,476);
INSERT INTO 'all_entities' VALUES (19,'restaurant','R4','NOBLE NEST','North Road',
(7)812-312','moderate','none','none','none','chinese','central','is a relaxed, family
style restaurant serving Chinese food',378,68);
INSERT INTO 'all_entities' VALUES (20,'restaurant','R5','THE GRAND','Cascade Road',
(7)812-329','expensive','none','none','none','classy','not central','is one of the
only two Leading Restaurants in the country. It offers impeccable service and is known for its
caviar.',746,318);
INSERT INTO 'all_entities' VALUES (21,'restaurant','R6','CHEZ SERGU','Art Square',
(7)812-465','expensive','none','none','none','french','central','is a classic French
restaurant, with an extensive wine list.',462,168);

```

```

--
-- Table structure for table 'bars'
--

```

```

CREATE TABLE 'bars' (
  'id' varchar(255) NOT NULL default '',
  'name' varchar(255) NOT NULL default '',
  'address' varchar(255) NOT NULL default '',
  'phone' varchar(255) NOT NULL default '',
  'price' set('inexpensive','moderate','expensive') NOT NULL default '',
  'location' varchar(255) NOT NULL default '',
  'description' text NOT NULL,
  KEY 'name' ('name')
) TYPE=MyISAM;

```



```
--
-- Dumping data for table `bars`
--

INSERT INTO `bars` VALUES ('B1','BAR METROPOL','Main square',
'(7)812-310','moderate','central','has an upscale, modern dcor with live, soft jazz');
INSERT INTO `bars` VALUES ('B2','CAF BLU','Alexander Street',
'(7)812-314','inexpensive','central','has loud music, cheap beer, dancing and is
open until five in the morning');
INSERT INTO `bars` VALUES ('B3','BUFFALO BILLS','Alexander Street',
'(7) 812-329','inexpensive','central','is known for margaritas, tequilas, and
Mexican beers');
INSERT INTO `bars` VALUES ('B4','EUROPA','South Bridge',
'(2) 512 3283','expensive','not central','is a tiny bar with exclusive clientele.
Being on the guest list is a must.');
```

```
INSERT INTO `bars` VALUES ('B5','PLANTER\\\'S','South Bridge',
'(4) 135 227','moderate','not central','is a wine bar with an extensive selection
of unusual and familiar wines');
INSERT INTO `bars` VALUES ('B6','MURPHY\\\'S','Castle Loop',
'(6) 131 352','moderate','not central','is an authentic Irish pub, serving an
array of stouts');
```

```
--
-- Table structure for table `cinema0`
--

CREATE TABLE `cinema0` (
  `film` varchar(255) NOT NULL default '',
  `rating` varchar(255) NOT NULL default '',
  `schedule` varchar(255) NOT NULL default '',
  `time` varchar(255) NOT NULL default '',
  `price` smallint(6) NOT NULL default '0',
  `currency` char(3) NOT NULL default '',
  `per` set('person') NOT NULL default ''
) TYPE=MyISAM;
```

```
--
-- Dumping data for table `cinema0`
--

INSERT INTO `cinema0` VALUES ('Get Johnson','18','8 PM','everyday',8,'EUR','person');
INSERT INTO `cinema0` VALUES ('Get Johnson','18','10 PM','everyday',8,'EUR','person');
INSERT INTO `cinema0` VALUES ('The Hospital','PG','5 PM','','8','EUR','person');
INSERT INTO `cinema0` VALUES ('The Hospital','PG','9 PM','','8','EUR','person');
```

```
INSERT INTO `cinema0` VALUES ('Giraffe','U','1 PM','','5','EUR','person');
INSERT INTO `cinema0` VALUES ('Giraffe','U','3 PM','','5','EUR','person');
```

```
--
-- Table structure for table `hotels`
--
```

```
CREATE TABLE `hotels` (
  `id` varchar(255) NOT NULL default '',
  `name` varchar(255) NOT NULL default '',
  `address` varchar(255) NOT NULL default '',
  `phone` varchar(255) NOT NULL default '',
  `price` varchar(255) NOT NULL default 'none',
  `per` varchar(255) NOT NULL default 'none',
  `currency` varchar(4) NOT NULL default '-1',
  `room_type` set('single','double') NOT NULL default '',
-- `price` mediumint(255) unsigned NOT NULL default '0',
-- `per` set('person','night','bed') NOT NULL default '',
-- `currency` char(3) NOT NULL default '',
  `location` varchar(255) NOT NULL default 'none',
  `description` text NOT NULL
) TYPE=MyISAM;
```

```
--
-- Dumping data for table `hotels`
--
```

```
INSERT INTO `hotels` VALUES ('H1','HOTEL PRIMUS','Alexander Street',
'(2)094-227','single','expensive','none','none','central','has big, bright, clean rooms,
some en suite, but with some noise from the busy street below');
INSERT INTO `hotels` VALUES ('H1','HOTEL PRIMUS','Alexander Street',
'(2)094-227','double','expensive','none','none','central','has big, bright, clean rooms,
some en suite, but with some noise from the busy street below');
INSERT INTO `hotels` VALUES ('H2','HOTEL PRESIDENT','Castle Loop',
'(7)192-277','single','moderate','none','none','central','is a two night minimum stay
Business-style hotel and is efficiently staffed. All rooms are en suite and breakfast is inclu
INSERT INTO `hotels` VALUES ('H3','ROYAL HOTEL','Cascade Road',
'(7)027-003','single','expensive','none','none','central','is a renovated
nineteenth-century palace, featuring private balconies. It has spacious well-appointed
rooms, many of which offer stunning views. All rooms are en suite, and breakfast is
included.');
```

```
INSERT INTO `hotels` VALUES ('H3','ROYAL HOTEL','Cascade Road',
```

```
'(7)027-003','double','expensive','none','none','central','is a renovated
nineteenth-century palace, featuring private balconies. It has spacious well-appointed
rooms, many of which offer stunning views. All rooms are en suite, and breakfast is
included.');
```

```
INSERT INTO `hotels` VALUES ('H4','YOUTH HOSTEL','Main Square',
'(7)281-446','single','cheap','none','none','central','is a hostel serving students
and budget travellers. It has bunk-beds in shared rooms with shared toilet facilities.');
```

```
INSERT INTO `hotels` VALUES ('H5','ART HOUSE HOTEL','Fountain Road',
'(7)252-996','single','cheap','none','none','central','is a basic hotel, frequented by
artists and musicians. Rooms are decorated in bright colours, but few offer views, and
all toilet facilities are shared. Breakfast is not included.');
```

```
INSERT INTO `hotels` VALUES ('H5','ART HOUSE HOTEL','Fountain Road',
'(7)252-996','double','cheap','none','none','central','is a basic hotel, frequented by
artists and musicians. Rooms are decorated in bright colours, but few offer views, and
all toilet facilities are shared. Breakfast is not included.');
```

```
INSERT INTO `hotels` VALUES ('H6','ALEXANDER HOTEL','Alexander Street',
'(7)874-874','single','moderate','none','none','central','is very comfortable, and
pleasantly decorated but has small rooms. It has a very friendly staff. Toilet
facilities are shared and breakfast is included.');
```

```
INSERT INTO `hotels` VALUES ('H6','ALEXANDER HOTEL','Alexander Street',
'(7)874-874','double','moderate','none','none','central','is very comfortable, and
pleasantly decorated but has small rooms. It has a very friendly staff. Toilet
facilities are shared and breakfast is included.');
```

```
--
-- Table structure for table `map-objects`
--

CREATE TABLE `map-objects` (
  `ID` varchar(255) NOT NULL default '',
  `name` varchar(255) NOT NULL default '',
  `X` int(11) NOT NULL default '0',
  `Y` int(11) NOT NULL default '0'
) TYPE=MyISAM;

--
-- Dumping data for table `map-objects`
--

INSERT INTO `map-objects` VALUES ('B1','BAR METROPOL',324,245);
INSERT INTO `map-objects` VALUES ('B2','CAF BLU',480,260);
INSERT INTO `map-objects` VALUES ('B3','BUFFALO BILLS',466,278);
INSERT INTO `map-objects` VALUES ('B4','EUROPA',331,464);
INSERT INTO `map-objects` VALUES ('B5','PLANTER\S',305,405);
INSERT INTO `map-objects` VALUES ('B6','MURPHY\S',708,141);
```

```
INSERT INTO `map-objects` VALUES ('H1','HOTEL PRIMUS',134,122);
INSERT INTO `map-objects` VALUES ('H2','HOTEL PRESIDENT',606,164);
INSERT INTO `map-objects` VALUES ('H3','ROYAL HOTEL',696,424);
INSERT INTO `map-objects` VALUES ('H4','YOUTH HOSTEL',319,126);
INSERT INTO `map-objects` VALUES ('H5','ART HOUSE HOTEL',232,83);
INSERT INTO `map-objects` VALUES ('H6','ALEXANDER HOTEL',401,388);
INSERT INTO `map-objects` VALUES ('R1','BOCHKA',300,245);
INSERT INTO `map-objects` VALUES ('R2','SIBERIAN TIGER',100,182);
INSERT INTO `map-objects` VALUES ('R3','SAINT PETERSBURG',630,476);
INSERT INTO `map-objects` VALUES ('R4','NOBLE NEST',378,68);
INSERT INTO `map-objects` VALUES ('R5','THE GRAND',746,318);
INSERT INTO `map-objects` VALUES ('R6','CHEZ SERGU',462,168);

--
-- Table structure for table `restaurants`
--

CREATE TABLE `restaurants` (
  `id` varchar(255) NOT NULL default '',
  `name` varchar(255) NOT NULL default '',
  `address` varchar(255) NOT NULL default '',
  `phone` varchar(255) NOT NULL default '',
  `price` varchar(255) NOT NULL default 'none',
  `per` varchar(255) NOT NULL default 'none',
  `currency` varchar(4) NOT NULL default '-1',
-- `price` mediumint(255) NOT NULL default '0',
-- `per` set('person') NOT NULL default '',
-- `currency` char(3) NOT NULL default '',
  `cuisine` varchar(255) NOT NULL default '',
  `location` varchar(255) NOT NULL default 'none',
  `description` text NOT NULL,
  KEY `name` (`name`)
) TYPE=MyISAM;

--
-- Dumping data for table `restaurants`
--

INSERT INTO `restaurants` VALUES ('R1','BOCHKA','Main square',
'(2)095-252','cheap','none','none','tavern meals','central','is a tavern open around
the clock, serving excellent sandwiches, soups, tavern meals, and salads.');
```

```
cooking, accompanied by first class music by Bolchoi musicians.');
```

```
INSERT INTO `restaurants` VALUES ('R3','SAINT PETERSBURG','Park Road',  
'(7)812-311','moderate','none','none','indian','central','is the most interesting  
restaurant in town serving indian fusion food chic decoration.');
```

```
INSERT INTO `restaurants` VALUES ('R4','NOBLE NEST','North Road',  
'(7)812-312','moderate','none','none','chinese','central','is a relaxed, family  
style restaurant serving Chinese food');
```

```
INSERT INTO `restaurants` VALUES ('R5','THE GRAND','Cascade Road',  
'(7)812-329','expensive','none','none','classy','not central','is one of the only  
two Leading Restaurants in the country. It has impeccable service and is known for its caviar.
```

```
INSERT INTO `restaurants` VALUES ('R6','CHEZ SERGU','Art Square',  
'(7)812-465','expensive','none','none','french','central','is a classic  
French restaurant, with an extensive wine list.');
```