eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

University Master's Degree
Computational **E**ngineering and **I**ntelligent **S**ystems

Konputazio Zientziak eta Adimen Artifiziala Saila –
Departamento de Ciencias de la Computación e Inteligencia Artificial

K I S A

I C S I

Master's Thesis

Dialog Systems Based on Markov Decision Processes Over two Real Tasks

Casanueva Perez, Iñigo

Director

Torres Barañano, María Inés
Departamento de Electricidad y Electrónica
Facultad de Ciencia y Tecnología
Leioa

informatika fakultatea
facultad de informática

KZAA /CCIA

September 2012

Abstract

In this work the state of the art of the automatic dialogue strategy management using Markov decision processes (MDP) with reinforcement learning (RL) is described. Partially observable Markov decision processes (POMDP) are also described. To test the validity of these methods, two spoken dialogue systems have been developed. The first one is a spoken dialogue system for weather forecast providing, and the second one is a more complex system for train information. With the first system, comparisons between a rule-based system and an automatically trained system have been done, using a real corpus to train the automatic strategy. In the second system, the scalability of these methods when used in larger systems has been tested.

1 Introduction

Since the mid 1990, researchers realized about the importance of automatic Spoken Dialogue Systems (SDS) for fields such as telephone assistance, to allow the management of simple calls. These first SDS were rule-based, which made them have very low adaptability [1]. In the next SDS generation the main goal was to make these SDS more adaptable to the user behavior as well as the possible distortions due to the automatic speech recognizer (ASR). The structure of a typical SDS is shown in 1 obtained from [3]. We can distinguish two parts, corresponding to the 2 agents that take part in the dialogue. On one side we have the user, and on the other side we have the SDS. The SDS can be decomposed in three levels. First of all, we have the acoustic level ASR and voice synthesizer, which transforms acoustic signals (Y) in words (W). In the second level we have the semantic level, which interprets the words (W) obtained from the ASR and transforms them in concepts (C) that the machine is able to understand. In the last level, we have the dialogue manager, which interprets the concepts (C) and decides which action the system should take. The system works as a turn taking cycle where the data flows in the direction of the arrow along the three levels, switching interactions between the user and the system until one of them decides to finish the dialogue. Anyways, more complex turn taking models can be implemented as explained in [5]

Work objectives are to do a review of the bibliography about MDP based SDS and to develop two real systems which can validate these methods. In the first section the state of the art of actual spoken dialogue systems is explained, as well of the different methods to plan the strategy with a wide explanation of MDP based dialogue managers. In the next section the development of a weather forecast SDS in described as well as the results obtained from its testing. In the third, the same description is done for a more complex train information SDS. And in the last the conclusions obtained from working with these techniques are presented. The bibliography reviewed is about automatic dialogue management , so when developing the whole systems we will focus on the dialogue management module and the rest of the modules (ASR and semantic understanding

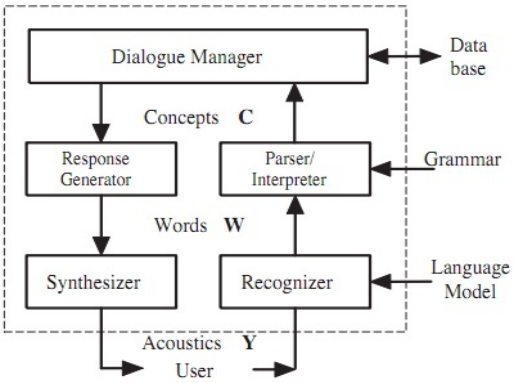

Figure 1: Arquitecture of a SDS.

module) will be developed in the simplest way possible or previously designed modules will be used.

2 State of the art

In this section the bibliography reviewed to work on this project, and therefore to develop the two real systems shown later, is described.

2.1 Statistical framework for SDS

A good description of a statistical framework for SDS can be found on [3]. The typical SDS block diagram is represented in fig 2 taken from [3]. The system is controlled by a dialogue manager, which generates a series of actions a_t, dependant of the actual state s_t. These actions (Usually acoustic data) are interpreted by the user, who will generate a response y_t (Acoustic data) which will be distorted by the environment noise n_t. This data has to be converted in binary data by the ASR [2] so the semantic understanding module can transform it into a sequence of concepts c_t. In function of these concepts the system will change its state from s_t to s_{t+1} and a reward signal r_{t+1} will be generated.
The state, action, acoustic signal and concept sequence joint distribution can be decomposed as:

$$P(s_{t+1}, a_t, y_t, c_t|s_t, n_t) = P(s_{t+1}|c_t, s_t)P(a_t|s_t)P(y_t|a_t, s_t)P(c_t|y_t, a_t, s_t, n_t)$$

Where the first term $P(s_{t+1}|c_t, s_t)$ represents the model and gives the probability of a transition to the state s_{t+1} given the previous state and the incoming concept sequence. The second term $P(a_t|s_t)$ represents the dialogue manager, the action taken given the actual state. The third $P(y_t|a_t, s_t)$ represents the user model, described in a following section, and the last one $P(c_t|y_t, a_t, s_t, n_t)$ represents the ASR and semantic understanding process.

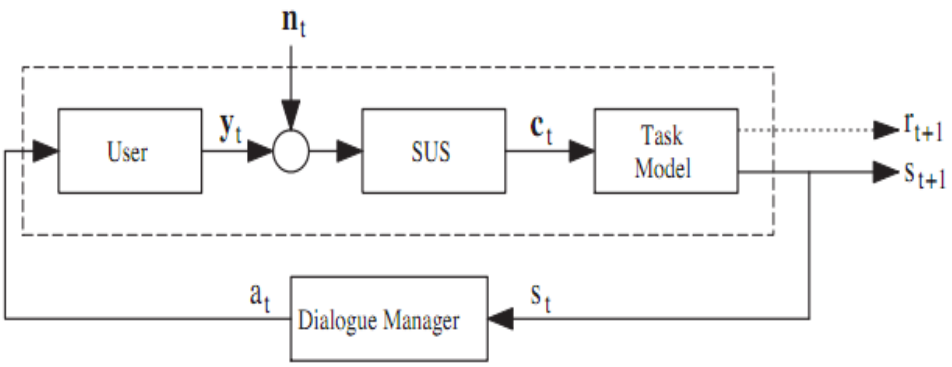

Figure 2: Block diagram of a SDS.

2.2 Markov decision processes

One of the most used strategies to develop these models are the MDP [3] [4]. In a MDP based SDS, the systems optimizes the dialogue strategy using reinforcement learning, giving a reward value to each decision taken by the manager, and maximizing the accumulated value of the reward. Usually the reward values are set to make the dialogue the shortest, most concrete and easiest possible. Taking into account that it has to provide the user the information required by him. But these reward values depend on the system designer and modifying them result into different optimal strategies. The MDP are characterized by a system that meets the Markov assumption (Each time step variables depend only of the previous time step) the tuple (S, A, T, R), where:

•S is a finite set of states; It describes the full set of "information states" the system may be in.
•A is a finite set of actions; It describes the full set of actions the system may take.
•T : is a state transition function such as $T(s', a, s) = p(s_{t+1} = s'|s_t = s, a_t = a)$; It models the a priori transition probability from a state s to a state s' if action a has been taken.
•$R : S \times S \times A$ is the reward function such as $T(s', a, s) = \varepsilon(r_{t+1}|s_{t+1} = s', a_t = a, s_t = s)$; It models the immediate reward obtained when action a is taken in state s

The function to maximize is the full session reward, which may have a discount factor γ':

$$\sum_{t=0}^{\infty}(\gamma' R(s_t, a_t)) \tag{1}$$

To choose in each state the action which maximizes the expected reward the *value function* (VF) is used:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} T(s',a,s)[R(s',a',s) + V^{\pi}(s')] \tag{2}$$

Which recursively computes the expected reward for each state. The optimal dialogue strategy is the deterministic strategy which gives the max VF ($V^*(s)$) such as:

$$V^*(s) = max_a \sum_{s'} T(s',a,s)[R(s',a',s) + V^*(s')] \tag{3}$$

then the optimal dialogue policy is given by:

$$\pi^*(s) = argmax_a \sum_{s'} T(s',a,s)[R(s',a',s) + V^*(s')] \tag{4}$$

Which is the policy that chooses the action which has the highest expected reward in function of the state transition matrix and the reward matrix.

2.2.1 Discount factor and equation convergence

When implementing the strategy selection algorithm, if done recursively, after a short number of states the problem becomes unviable. Although the equation converges, it may do it in a very far temporal horizon son the number of recursive iterations must be bounded. To do this a value called Discount Factor $0 \leq \gamma < 1$ (DF) which multiplies the recursivity in the equation 3 so it becomes:

$$V^*(s) = max_a \sum_{s'} T(s',a,s)[R(s',a',s) + \gamma V^*(s')] \tag{5}$$

This way, the contribution of the following rewards goes decaying in the time and in some number of iterations it will become almost zero, allowing the computation of a limited number of iterations (e.g. with a discount factor of 0.95 with 20 iterations is enough.)

2.2.2 Dynamic programming approach

Even limiting the number of recursive iterations, it is not feasible computing the optimal strategy with the equation 3 (The computational cost would be $O((|A! * |S|)^{rec_{lim}})$ where $|A|$ number of actions, $|S|$ the number of states and rec_{lim} the iteration limit). Due to the meeting of the Markov condition by the model (Each state transition depends only of the previous state), the problem can be solved as a dynamic programming problem. We start computing the VF for the iteration rec_{lim} and the results of each iteration are stored in an array of size $[|S|, rec_{lim}]$ where each value $e_{s,i}$ represents the max VF in the state s and the recursivity i. Each value of the last recursivity ($i = 0$) maps with an action of the action set. This way we can compute the optimal strategy with cost of $O(|A| * |S| * rec_{lim})$. In the figure 3 the way to compute the VF with dynamic programming is shown. The rows represent the recursivity iteration and the columns the states.

Iteration						
	$V_1^*(1)$	$V_1^*(2)$	...	$V_1^*(s)$	...	$V_1^*(s_{max})$
	...	...	...	...	...	...
	$V_i^*(1)$	$V_i^*(2)$	...	$V_i^*(s)$	...	$V_i^*(s_{max})$
	...	...	...	...	...	...
	$V_{n-1}^*(1)$	$V_{n-1}^*(2)$	...	$V_{n-1}^*(s)$	...	$V_{n-1}^*(s_{max})$
	$V_n^*(1)$	$V_n^*(2)$	...	$V_n^*(s)$	...	$V_n^*(s_{max})$
						State

Figure 3: Table of the dynamic programming computing of the Value Function

2.3 User models

In order to develop a system based on MDP, it is necessary to estimate the state transition function $T(s`, a, s) = p(s'|s, a)$. To this end, a corpus of dialogues where the state and action history of the SDS is recorded is needed. But very large corpuses are needed to have a good estimation of the T matrix, and in order to have all the parameters of the matrix estimated, is necessary to try every action at every state to check the state transitions. This leads to dialogues with a system which chooses actions almost at random to get a training corpus, which obviously is very disgusting for human users. To avoid training the system with humans, user models are created [6] [7]. This way, the first training step is performed with a stochastic user simulator that interacts with the SDS, generating responses which simulate what a human user would do (As shown in figure 4). This allows getting a dialogue corpus much faster and easier than interacting with real users, and the transition matrix can be inferred from it.

User modeling for dialogue strategy training is an open topic with researchers working with different techniques, as it is surveyed in [12]. Here, some statistical predictive user models are explained, such as N-grams, Graph-based models, Bayesian networks, Machine-Learning techniques and Hidden Markov models.

N-gram based user models [6] where the first proposed models and they choose as action $a_{u,t}$ the one with highest probability given the last n system actions. This method has the advantage of being domain independent, but they do not take in account de dialogue state information so they usually lead to nonsense dialogues. The user model used in this work is a simplification of this kind of model.

The importance of the user model in the SDS design is huge since the final performance of the system with real users has a big dependence on how well does the user model recreate real user responses, and a system trained with a bad user model may perform well when tested with that model, but it will have a bad performance when tested with a better model or a real user.

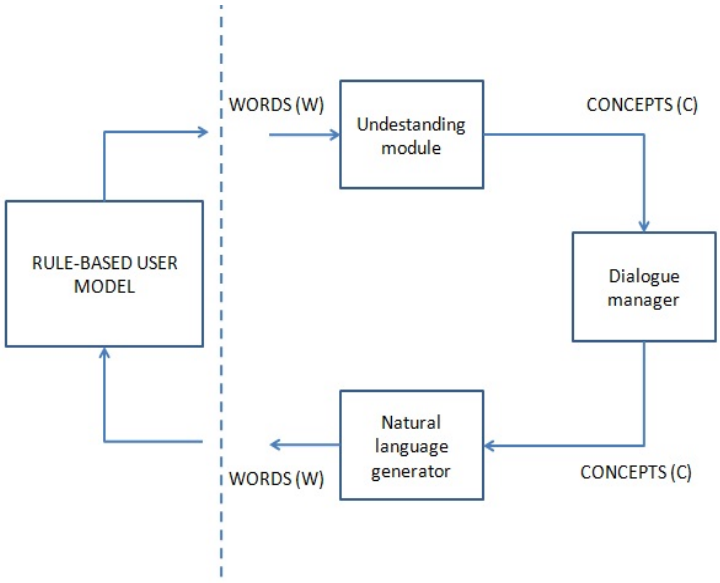

Figure 4: Diagram with user model

2.4 Partially observable Markov decision processes

Unfortunately, MDP require full knowledge of the state, which is not always true because of problems such as the understanding error of the ASR and the errors of understanding system, which does not always transform the sequence of words into the correct sequence of concepts. As these problems do happen in real systems, some solutions such as parallel dialog state hypotheses and local use of confidence scores have been proposed. The best framework to work with uncertainty is to cast the SDS as a partially observable MDP (POMDP) [11]. In a POMDP instead of knowing the real state of the system at each time step, we work with a probability distribution over states called belief state $b(s)$, inferred from the observations we get from the user. A POMDP is characterized by the tuple (S, A, T, R, O, Z) where:

•S, A, T, R form a MDP (2.2);
•O is a finite set of observations got from the environment; and
•Z is the observation function such as $Z(o, s, a) = p(o|s, a)$

The belief state transition function is defined as:

$$b'(s') = p(s'|o', a, b) = k * p(o'|s', a) \sum_{s \in S} p(s'|a, s) b(s) \tag{6}$$

Where k is a normalization constant and the numerator of this constant consists of the observation function Z, transition matrix T, and current belief state b. b' is the belief state in time step $t + 1$, such as $b = b_t$ and

$b' = b_{t+1}$(full explanation in [11]). The reward function becomes:

$$p(b, a) = \sum_{s \in S} R(s, a) \cdot b(s) \tag{7}$$

Basically, the POMDP works as a MDP where all the equations explained before are calculated over a distribution of states $b(s)$ instead of just one. Its working cycle is explained in figure 5 where the circles represent random variables, squares represent decision nodes and diamonds utility nodes. Shaded variables represent unobservable variables. Solid directed arcs indicate causal effect and dashed ones indicate that a distribution is used (image taken from [11]). This approach provides a unified way to solve some of the main problems of the MDP. Unfortunately, computing each parameter over a state distribution makes this approach computationally very complex, and it scales poorly with the size of the state space. To solve this researchers are working on approaches that reduce the computational cost of the models such as Bayesian update of dialogue state [13] and hierarchical reinforcement learning [7].

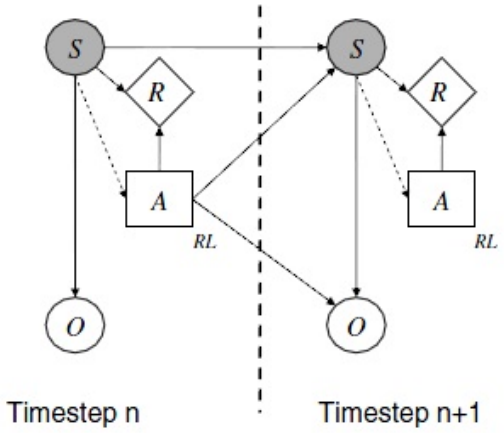

Figure 5: Influence diagram representation of a POMDP

3 MDP based SDS in real world

When taking these theoretical frameworks into the real world some problems arise. Some publications try to describe these problems [8] [9]. To test these problems and obtain conclusions about the validity of these systems when implemented, two MDP based SDS have been developed. First of all a not very complex system, designed to provide weather forecast, and then a SDS for train information with a considerably greater complexity. Some modifications of the value function explained in 5 have also been used. The steps followed in the design and test of both systems has been the following:

a)Design the architecture of the system; Although the objective is to test the automatic dialogue strategy learning methods, to test the system all the architecture of a system explained in figure 1 must be implemented. This step includes the design of the interface, the understanding module and the natural language generator, among other support modules. The dialogue manager is developed in the next step.

b)Design the MDP based dialogue manager; In this step, the algorithm of optimal action selection has been implemented, with the modules needed to train it, such as a user model to obtain a corpus of dialogues. In addition, two more additional action selection algorithms have been designed as slight modifications of the equation 5.

c)Test the model; In the last step, the system is tested with the three action selection algorithms (equation 5 and two more action selection algorithms explained later) and the obtained results are explained.

4 SDS for weather forecast

To test the validity of these frameworks a SDS that can be asked about and provide weather forecast information has been implemented. The weather data of the system is obtained from a web page (www.eltiempo.es) and the system can be asked about the following types of weather information:

•Type of weather
•Temperature
•Dawn
•Dusk
•Wind
•Rain

The user can ask for those in any city, at a specific date and hour. To simplify the development of the system, the ASR step has been omitted and the system works as a chat box with text input and output as shown in figure 6.
In the following sections, the modules developed to the design of the system are explained.

4.1 State space

The state space models the "knowledge state" of the system in each time step, mapping one state to each knowledge combination possible. The different "knowledge" the system may have are the concepts the user is asking for (Type of information) and the attributes needed to provide information about these concepts. To make the system as simple as possible the state space has been defined so the system needs to obtain four data from the user, (4 slots). These are three attributes (city, day and hour) and a concept (kind of information). This way the system has $2^4 = 16$

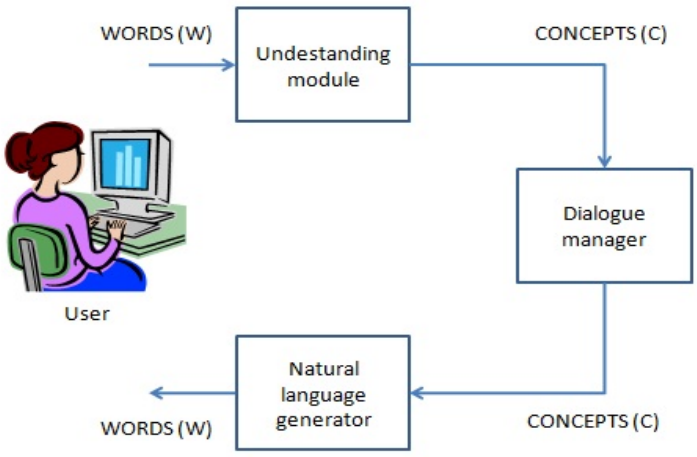

Figure 6: Arquitecture of the developed SDS

states, plus an additional terminal state to model the end of the dialogue. A very simple way to represent this, similar to the one explained in [7], is to assign to each slot a value of 1 or 0, depending if the value of the slot is known or not. (e.g. if we know the day, city and information data, but we do not know the hour, the state in binary would be 1101, the 13th state, as shown in figure 7.)

	City	Day	Hour	Information	current state
Slot value	Madrid	19-jul	??	weather	
State slot value	1	1	0	1	13

Figure 7: Example of codification of the state space for the weather forecast system.

4.2 Action space

The definition of the action space is one of the most hand-made parts of the design of the system. The actions in this space will be the actions that the system is able to take, but it would not be able to learn new ones, so the designer must be sure that all the actions needed for the dialogue goal are in the system. Including all the typical actions that a person would take in the place of the SDS has been intended. The defined action space is the following:

•A1 Greeting.
•A2 Ask for city.

•A3 Ask for day.
•A4 Ask for hour.
•A5 Ask for information.
•A6 Provide information.

There are some auxiliary actions related to specific events (e.g. if the city asked for is not in the data base, or if there still is not information for the asked day), but they are always linked to one of these actions. In some works [13], summary actions are mentioned. This concept organizes actions in a hierarchical way to make the action election level by level, thus reducing the complexity of this decision making. This could have been applied in this project in the way of clustering $A2, A3, A4$, and $A5$ in the same summary action, called ask for slot. Then after the system chose this action the slot to be asked would be chosen in a sub level decision taking. Anyways, this technique has not been applied because the system complexity in not big enough to require complexity reduction techniques.

4.3 Semantic understanding

As the concept understanding module is not the topic of this project, it has been designed with very simple rules that just map words to concepts (With the help of the Freeling toolkit, [10]). In this system concepts are city, day, hour and information. This way if this toolkit tags a sequence of words as a date, the system fills the slot of the day or hour with the detected value. If the user writes the word weather, the system will deduce that weather is the information kind that the user wants to know.

4.3.1 Freeling

As it has been said in 4.3, to make the concept understanding, the semantic analyzer Freeling has been used. This toolkit provides some apis that allow making fairly complete semantic analysis from a word sequence. It takes a word sequence as input and returns a tagged word sequence as output, indicating what kind of word is each one of the input sequence. In this application is especially useful for some reasons. First, the analysis provides us the root of the word (e.g. if the analyzed word is raining, the api will provide the root rain), which is very useful because allows rule simplification to relate words with concepts (It is especially useful in Spanish). On the other hand, it is quite powerful detecting dates, which directly transform in a format such as $[DayOfWeek\ /Day\ /Month\ /Year\ /Hour\ /Minute\ /am-pm]$, where the fields are filled with "??" value by default and are filled with the values found by the analysis. At last, it also detects proper nouns so it simplifies the search for cities.

4.4 Answer generator

As well as the concept understanding module, the natural language generator module has also been developed in a very simple way because it is not the goal of the project. The responses are written in a text file with just some fields such as *city* or *date*, and each action maps to a written

sentence. The natural language generator just replaces the fields on the sentences with the value provided by the dialogue manager.

4.5 Rule-based dialog policy

To test the system as a SDS of the first generation, a rule based strategy can be designed because of the low complexity of the state space. The advantages of designing a rule based strategy are the following:

• A example strategy is obtained, which will be very useful to test if the learned strategies trained later are valid.
• Using this system with real users, a corpus of user responses is generated which will be later used to design an user model.

This strategy starts the dialogue greeting the user and offering its help, giving the initiative to the user. After the greeting continues asking for the slots that have not been filled in a deterministic order until it can provide the information required. Then, the strategy used is to ask for the unfilled slots until the four are filled, following this order: City, day, hour, information. It is a direct mapping from states to actions of section 4.2:

• Beginning of the dialogue $\longrightarrow$ A1.
• 0xxx state$\longrightarrow$ A2.
• 10xx state$\longrightarrow$ A3.
• 110x state$\longrightarrow$ A4.
• 1110 state$\longrightarrow$ A5.
• 1111 state$\longrightarrow$ A6.

4.6 MDP based dialog policy

After developing the rule-based policy, an automatically learnt dialogue policy using a MDP has been designed. To obtain the optimal policy, given by equation 3, the first step is to obtain the transition matrix T and the reward matrix R.

4.6.1 Transition matrix

To compute the transition matrix $T(s', a, s) = p(s'|s, a)$ is necessary to have a dialogue corpus that provides data of transitions from every state choosing every action available. This can be done directly testing the system against a user model without having any dialogue data, and letting the equation 5 explore the state-action space as well as the transition matrix changes with new dialogue data. This procedure lets us know the evolution of the strategy (which can be measured with the total reward of each dialogue for example), but it does not do an exhaustive exploration, and maybe suboptimal actions could be chosen while optimal ones remain unexplored. A way to avoid this is to have dialogues with a SDS whose policy is to randomly choose an action from the state space, which leads

to a much wider space exploration at the cost of learning time. As it has been explained in 2.3, it is very hard for a person to maintain dialogue with a system with this policy so a user model has to be used to get the corpus this way. The advantages of this strategy are that every state-action pair is explored, but after this the system should be implemented with the previous VF based strategy. The problem with this is that an action which was optimal with the user model but it is not with real users could remain unexplored.
We then propose alternative proceedings to balance exploration and exploitation. These procedures are presented in section 4.7.1.

4.6.2 User models

Two user models have been developed to test the system. First, a very simple, user model has been designed, which tries to answer to each system sentence in the way a human user would do, but without recording any dialogue history. It is a model similar to the N-gram based model explained in 2.3, with the difference that it does not need to infer the action performed by the system with n-grams, it does it by rules because it knows the action space of the system. The user model has a set of actions related to each machine action possible, so it just chooses one action at random from this set (e.g. if the system ask for the city, the user model will provide a city at random, or it may provide a city and the day). As it does not record any dialogue history, it may be asked twice for the same data slot and provide different information in each interaction, but this fact is irrelevant because the system state will not change because of the way the systems updates the state. (For the system is irrelevant to have de day slot filled with 6 or with 9, its state just shows that the day slot is known). The interaction between System and user model is explained in figure 8 where s_t represents the system state at time t, a_t the system at time t, and a_u the user action at time t. Directed arrows represent direct influence and the user model is represented as a single node to indicate that its action decision is independent of the time step t.
The second user model is a bit more complex. It follows the same scheme of the first one but its action space has been taken from dialogues obtained testing the rule-based system with real users. This way we have an action space a bit more realistic, because it responds to each system sentence with actions that real users have taken. The problem is that those actions may have been taken in a different state of the dialogue so this model is still far from being realistic, but lets us getting a corpus big and realistic enough to calculate the transition function.

4.6.3 Reward matrix

The reward function design step is the most hand-made step of the design of an automatic policy, so we have tried to make it the most simple possible. Designing a very complex reward matrix is equivalent to design a rule-based strategy, and designing a very complex matrix would be very time consuming for big systems, so one of the characteristics of the reward matrix is its simplicity. Each action has a reward of -1, excepting 2 cases.

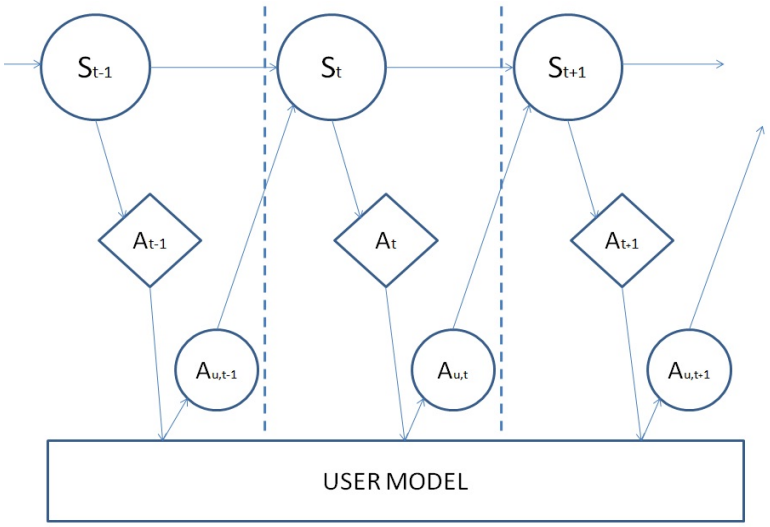

Figure 8: System with user model influence diagram

The case in which the system has obtained all the data required (1111 state) and the action A6 (Provide information) is chosen, the reward will be +10. In the case that action A6 is chosen in any other state, the reward will be -10.
Another reward function has been implemented, which is the same as the previous with the exception that choosing an action that does not change the actual state $s' = s$, the reward given is -2. The objective of this new matrix is to avoid actions which do not lead to a state change, and in consequence to the space exploration.

4.7 Experiments

First of all the MDP based strategy has been tested, learning it from the first user model and comparing the obtained strategy with the rule based one. The first way the strategy has been trained is letting the model "talk" with a SDS with random action selection as strategy, for 400 dialogues, to have a corpus from where compute the transition matrix. After that the optimal policy strategy has been applied (5) and the strategy obtained has been checked. Figure 10 represents the deterministic policies for the rule based system (SDSRB) and other 3 SDS, where the first row represents the states and the next one the action taken in each state by that system. SDS2 is a system where A1 action is only available as first action, so the action space has only 5 actions (The 6 before minus the A1 action), and the resulting policy is almost the same than the rule based one. This policy has been calculated with a discount factor $\gamma = 0.95$. Another policy with $\gamma = 0.80$ has been computed (SDS3).
After this training the strategy exploring the state-action space directly with the VF equation has been tested. This way the convergence may

be faster but the exploration is smaller. As we wanted to test more action selection ways that the optimal action selection, two more stochastic action selection algorithms, explained in the next sub-section have been designed and tested.

4.7.1 Stochastic action selection

Iteration						
	$V_1^\pi(s)_{a_1}$	$V_1^\pi(s)_{a_2}$	...	$V_1^\pi(s)_{a_j}$	...	$V_1^\pi(s)_{a_{max}}$
	...	...	...	...	...	...
	$V_i^\pi(s)_{a_1}$	$V_i^\pi(s)_{a_2}$	...	$V_i^\pi(s)_{a_j}$	...	$V_i^\pi(s)_{a_{max}}$
	...	...	...	...	...	...
	$V_{n-1}^\pi(s)_{a_1}$	$V_{n-1}^\pi(s)_{a_2}$	...	$V_{n-1}^\pi(s)_{a_j}$	...	$V_{n-1}^\pi(s)_{a_{max}}$
	$V_n^\pi(s)_{a_1}$	$V_n^\pi(s)_{a_2}$	...	$V_n^\pi(s)_{a_j}$	...	$V_n^\pi(s)_{a_{max}}$

Action

Figure 9: Table for computing VF stochastically

The problem with the previous way of policy design is that when the policy is computed, it is strongly related to the behavior of the user model. Even if the system continues the exploration of the action-state space, it may get stuck in a suboptimal action chosen from the training with the user model which is not the optimal with real users. Two different strategy selection algorithms have been tested which explore the action-state space in a wider way.

•Semi-Random Value function selection: The first strategy is based on an idea taken from [12]. At each system turn, the system takes the optimal strategy or a random one with a probability β, which goes decreasing while the system takes more dialogue data. In our system this parameter has been set as $\beta = 1/\sqrt{2}$, where n is the number of dialogues in the training corpus. This way, the system does a wider exploration in the first steps of the training and converges to an almost optimal strategy when the training data is big enough. But in the long term following this strategy will be almost the same than following the optimal one, so some authors propose to have a bottom threshold for the probability, for example 0.1, so there is always a small part of exploration in the dialogue strategy.
•Global value function based action selection: The second way to try to avoid this is to make some kind of stochastic decision when taking an action, giving to each action a probability related to its value function. But the value functions cannot be calculated with 3, because even if the user takes the action in that time step stochastically the algorithm assumes that in the next step it will take the optimal one. our proposal has been doing it not computing the optimal value function, but computing it with the general value function equation 2, where in each recursivity, the policy $\pi(s, a)$ is computed from the values of the value function in the previous iteration. We have an example of how to compute it in 9.

This table corresponds to the VF computation for a single state s. The rows represent the recursivity iteration and columns represent the value function if the action a is chosen. To compute $\pi(s, a)$ in each step the action probability distribution is computed from the values of the previous iteration column.
Using this method results into a very random strategy. This is due to the fact that the VF values are not bounded, which makes the most VF have small difference between them compared to the worst or best action in each step (There is always an action with very different value of the others, usually A6 which has or a much bigger or a much lower value, so in the cases where its value its much lower than the others it makes the distribution of the rest of actions very similar). To solve this problem, implementing this system choosing only the best n actions to compute the policy has been tried, which has lead to a system with better performance but still very random in some cases. In the last case, the same system taking the best three actions and using the second reward function described in section 4.6.3 has been tried.

4.8 Results

State	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SDSRB	A2	A2	A2	A2	A2	A2	A2	A2	A3	A3	A3	A3	A4	A4	A5	A6
SDS1	A1	A1	A2	A1	A1	A1	A2	A2	A3	A1	A3	A1	A4	A4	A5	A6
SDS2	A2	A2	A2	A3	A2	A2	A5	A2	A4	A4	A5	A3	A5	A4	A5	A6
SDS3	A2	A2	A2	A3	A2	A4	A5	A2	A5	A3	A5	A3	A5	A4	A5	A6

Figure 10: Results for different systems

As there is not a function to measure the value of a dialogue manager other than the total mean reward, which is not a good measure because the reward is set by the designer, the evaluation of dialogue managers has to be done in an intuitive way. This means that a strategy will be good if it is user friendly when used with real users.
The results obtained with the systems trained with the random strategy are quite good (figure 10). In the optimal strategy, the main difference between this strategy and the learnt is that the learnt one (SDS1) abuses of the greeting action (A1). This means that with the data obtained from the user model the system reaches its goal quicker letting the user have the initiative. But this may only work with the defined user model because it responds to the A1 action providing by mean more than one datum, while when asked for a single slot it provides a mean of about one. The policy obtained from the interaction with the second user model was very similar to the obtained with the first. With the first stochastic algorithm the resulting strategy is almost the same because in the long term it works as the normal one.

With the global value function based action selection strategy using the second reward function the result has been very satisfactory with a transition function trained with the random method, and in the most cases it chooses the optimal action with a probability of about 0.8 (Optimal actions are considered the ones that SDS2 got), and suboptimal actions with the rest of probability. The system resulting this way is easy to use for real users and it can learn online because it continues searching in the solution space. When choosing a higher value for n, the more the state space is explored but the system may become more random and less user friendly. The problem with this procedure of choosing the action is that when trying to use this method with an untrained system (With no previous dialogues from where compute the transition matrix) it did not converge to a transition function which leaded to a dialogue strategy with stable reward. So even if the strategy explores more than the optimal one, it seems to be unstable in the long term so it is not valid.

The 11 shows the mean reward over the number of sample dialogues of

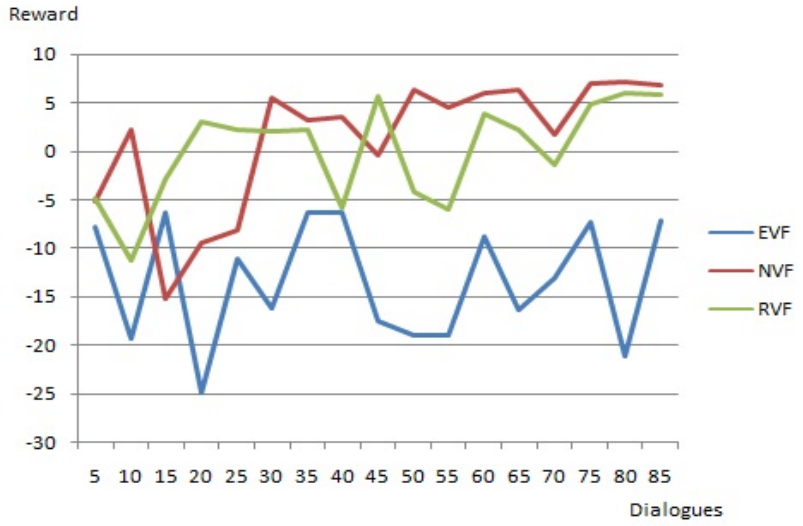

Figure 11: Reward over number of sample dialogues. NVF represents the normal value function, RVF the first stochastic value function and EVF the second stochastic value function.

the three strategies: Normal value function (NVF), first stochastic strategy (RVF), and second stochastic strategy (EVF). As can be seen both NVF and RVF converge to a strategy which has a reward of about 5. RVF gets higher reward faster but it is less stable than NVF, as is logical due to the random component of RVF. On the other hand, as it has been explained before, EVF is unstable and does not converge in a good strategy.

5 SDS for train information

After the developing of the weather forecast SDS, a system to provide train information has been developed. This new system is more complex than the previous one and its objective is to test the validity of the MDP based dialogue strategy methods in bigger systems. Some parts of this system had been previously designed in CITAR PROYECTO DIHANA but the dialogue management module has been fully redesigned as a MDP. The information this SDS can provide is the following:

•Departure and arrival times of trains.
•Prices.
•Travel duration.
•Train types.
•Services available in trains.

Almost the same steps followed to design the previous system have been taken, excepting the ones corresponding to the design of rule-based strategy.

5.1 State space

To design the state space of this system a similar strategy as the used in the weather forecast SDS has been used, where there are a list of concepts and attributes which are coded binary depending on if they are known or not. The concepts correspond to the information type the user wants to know and the attributes to the data given to provide those concepts. The list of concepts and attributes defined is the following:

Concepts:
•Departure time of a train.
•Arrival time of a train.
•Ticket price.
•Train type.
•Travel duration.
•Services of the train.

Attributes:
•Departure city.
•Arrival city.
•Departure time.
•Arrival time.
•Date.
•Train type.
•Travel type.
•Ticket type.
•Price.
•Services.

So the state space has been designed as a Boolean array which includes

all these concepts and attributes, a total of 15, plus an additional history slot which is used to not provide more than once the same information, plus an additional terminal state. This makes a space of $2^{17} + 1$ states, much bigger than the previous one. But this space has been only designed theoretically, because the computer used to develop the system did not have enough memory to work with a transition matrix of this size. In the testing we have worked with a system with 3 concepts and 6 attributes, which makes a space of $2^{10} + 1$ states.

5.2 Action space

The action space is very similar to the one described in 4.2, where the actions are oriented to gather the information required to fill the slots needed to provide information. In this case, as happens with the state space, the action space is bigger but it scales linearly with the size of the system, so it is not so big. In our case it has 18 actions, which are the following:

•A1 Provide departure time information.
•A2 Provide arrival time information.
•A3 Provide price information.
•A4 Provide train type information.
•A5 Provide travel duration information.
•A6 Provide services information.
•A7 Ask for departure city.
•A8 Ask for arrival city.
•A9 Ask for departure time.
•A10 Ask for arrival time.
•A11 Ask for date.
•A12 Ask for train type.
•A13 Ask for travel type.
•A14 Ask for ticket type.
•A15 Ask for price.
•A16 Ask for services.
•A17 Ask for information type.
•Start action: Greeting.

These actions could be gathered in 3 summary actions, *provide information*, *ask for slot* and *greeting*, but to simplify the design of the system it has been implemented as a set of 18 actions.

5.3 Semantic understanding

To develop the semantic understanding module, the help of previous work done in [16] has been used. Specifically, a corpus of tagged dialogues [14][15] where each word or group of words that correspond to a concept/attribute has been assigned to a text file corresponding to each concept/attribute. Then the semantic understanding is done with the help of these files (for each user utterance the module returns a list of concepts and attributes) and with the additional help of the Freeling api [10] for

date and hour detection as explained in 4.3. The semantic understanding rate of this module is far away from being perfect but as the system dispenses the ASR module, it somehow simulates the understanding error generated in a real system from the combination of ASR and semantic understanding.

5.4 Response generator

As in 4.4, the response generator has been designed in a functional way that permits the testing of the goal of the work, which is the dialogue management. In this case the module is even simpler because when it provides information it does not have a database from which obtain data, so it just prompts which data it should provide and supposes that the data will always be in the database.

5.5 MDP based dialog policy

When developing the algorithm that controls the MDP based dialog policy a problem has arisen; Due to the memory limitations of the computer where the project has been developed it could not work with a transition matrix of size $|S| \times |S| \times |A|$, where $|S|$ is the size of the state space and $|A|$ is the size of the action space, so the size of the transition matrix would be $((2^{16}+1)^2) * 18$, which exceeds the memory of the used computer. To test the system with a feasible state space the concept number has been reduced to three (*departure time, arrival time and price*) and the attribute number to six (*departure city, arrival city, departure time, arrival time, date and train/travel/ticket type*). The action space has been reduced in the same way to a total of eleven. This way, the transition matrix size will be $((2^{10}+1)^2) * 11$.

5.5.1 Transition matrix

To compute the transition matrix the same procedures that in 4.6.1 have been used, but this time much more sample dialogues are needed because of the bigger size of the state space. While in the previous SDS in 100 dialogues it had obtained a policy with stable reward in this system does not get a stable reward until it has about 2000 dialogues. This means that a huge corpus of dialogues is needed to train this kind of system which forces the usage of user models to train the system.

5.5.2 User models

The same user model explained in 2.3 has been used, changing the actions the user model can take for actions related to train information asking. The user model is so simple to test if a system of this size could be trained with a very simple user model, and to test how does a model trained with a simple model act when interacts with a real user which uses different conversation strategies.

5.5.3 Reward matrix

The design of the reward function has been more complex than the one in the weather forecast system, because in this case we have a wider variety of actions that in different states can give a large positive or negative reward. The reward matrix has been defined in the following way:

•Provide any information when the state meets the conditions: +10
•Provide any information when the state does not meet the conditions: -10
•Any other action: -1

The conditions the system needs to meet in each *provide information* action are explained in figure 12. This table shows the necessary attributes (X) and optional ones (O) to provide concept information. If the necessary attributes value is *true* when the action is performed the reward will be 10, otherwise the reward will be -10. The optional attributes modify the information provided by the system but the reward value is the same. To these attributes the slot value for the history information must be added, which will be in *true* if any value has changed since the last provide information action and in *false* if no value has changed.

Concept / Attribute	Dep. time	Arr. Time	Price
Dep. city	X	X	X
Arr. city	X	X	X
Dep. time			X
Arr. Time			X
Date	X	X	X
Type	O	O	X

Figure 12: Attributes needed to provide concept information

5.6 Experiments

The experiments done with this SDS have been the two which gave best results in the previous SDS, which are training the system with the user

model explained in 5.5.2 directly using the normal VF (5) and the "Semi-Random" variation explained in 4.7.1 to explore the action-state space and to get the corpus necessary to get the transition matrix. As the state space is much bigger this time the number of dialogues needed to get this corpora has been 2000, but even after this some states remain unexplored so more dialogues could be needed to get a more robust system. To compensate the increase of dialogues of the corpora the parameter used to decide if the action taken in the "Semi-Random" strategy is random has been set to $\beta = 1/\sqrt[4]{n}$, which will make the system eventually take random actions for a longer period of dialogues.

5.7 Results

The results obtained are shown in figure 13, where the vertical axis shows the mean reward over 50 dialogues and the horizontal the number of dialogues. The two results shown represent the evolution of the reward for the normal value function (NVF in red) and for the semi-random value function (SRVF in blue). As can be seen, in this SDS the NVF converges much faster to a higher reward than the SRVF, the contrary that happened in the weather forecast system. This may be because of the $\beta = 1/\sqrt[4]{n}$ parameter used in the SRVF, which makes the system take too many random actions, or because of the simplicity of the user model used which makes the NVF find the optimal path to an high reward dialogue very easily, because the model has very little behavior variations. Anyway both systems have converged to a suitable dialogue strategy what means that these methods are suitable to train a MDP based SDS.
But it must be taken into account that these systems are only trained to interact with users that have a similar behavior of the user model, when the real user tries a strategy that has not been followed by the user model the system will probably fall into an unexplored state and will not take the optimal action. This means that the behavior of the system is strongly related to the user model behavior. And may not get the same reward when tested with real users.

6 Conclusions

After revising the bibliography and implementing the two systems explained before these are the conclusions obtained:

1.) The performance of the dialogue manager is strongly related to the rest of modules of the whole SDS, especially to the understanding module. Authors try to design domain independent dialogue managers, but as most of the times the semantic interpretation of an utterance depends on the dialogue state, the dialogue state should be designed to be used by the understanding and dialogue modules, or both modules be developed together. And the understanding module can not be designed domain independent. The rest of modules do also have big importance in the systems performance. (e.g. if the natural language generator module can not make the user understand the action taken by the dialogue manager,

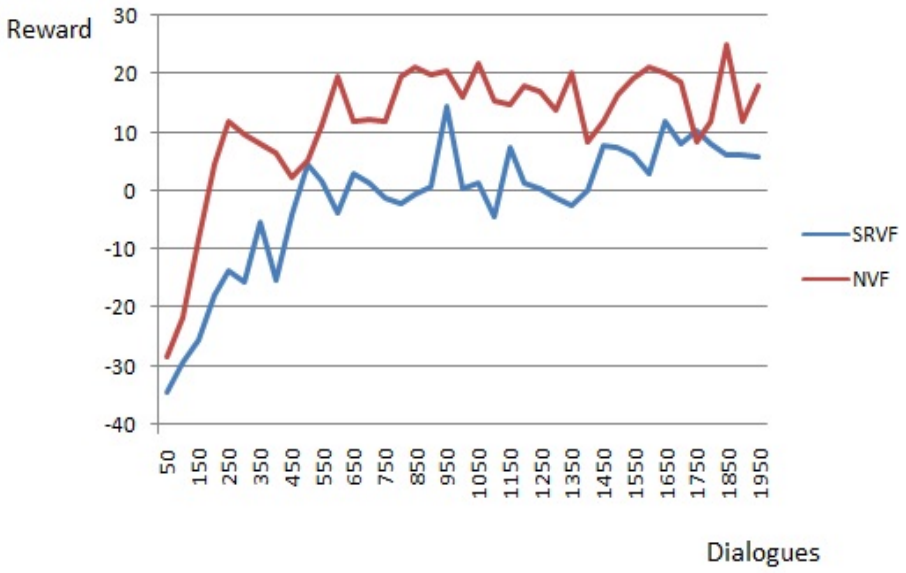

Figure 13: Reward over number of sample dialogues

this will affect the users next utterance.)

2.) One of the greatest difficulties of the development of a SDS is the design of a state space which gathers the most information possible about its environment but making it small enough to be computationally feasible. A compromise between environment information and state complexity must be taken, and this many times leads to poor state representations or state representation that can not represent all the environment information needed to choose the strategy.

3.) The design of a realistic user model is one of the keys to make the automatic learned strategy work. But it is really difficult to design a model that can simulate human behavior given that human behavior differs greatly from a person to another. Then it can happen that a system trained with one model works well for a person but it does not for another.

4.) In this work only "slot-filling" systems have been tested, which are the easiest to model as a MDP. This kind of system usually work with a system-initiative strategy which may be the easiest to learn for a MDP based dialogue manager, but other kind of systems such as problem solving systems or systems where the user wants to explore options rather than get a specific goal, which need to give more user-initiative, should also be designed. These systems are harder to design because of the state

space design and specially because of the reward function design, and may not learn the optimal strategy so easily.

5.) The design of the reward function is the most unexplored step on the design of a SDS, and is one of the most important steps if not the most important. A system in the long term will always act as reward function indicates, so if a system has a bad performance the most likely reason will be a poorly designed reward function. Anyways the design of this function is very complex. A good reward matrix should give high reward when the user is happy with the system action, but to be completely sure of the user satisfaction you need some kind of feedback, which is impossible to have when you are predicting future reward. Then it may be that an action taken in a specific state that gave high user satisfaction in one dialogue does not give the same satisfaction in another one, or in a posterior utterance of the same dialogue. (e.g. if the same information is provided 2 times, or if information is provided when one of the attributes needed has been taken erroneously.) Then, a system can give the same reward in two actions which give contrary satisfaction to the user. This may also be problem of a bad designed space state which does not differ between two different situations, so the space state and reward function are closely related.

References

[1] Delgado, R. L.-C. and Araki, M. 2006 –*Front Matter, in Spoken, Multilingual and Multimodal Dialogue Systems: Development and Assessment*–, John Wiley Sons, Ltd, Chichester, UK.

[2] Lawrence R. Rabiner, 1989 –*A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*—, Procedings of the IEE,Vol 77, No.2

[3] Steve Young, 1998. —*Probabilistic Methods in Spoken Dialog Systems*—, Royal Society.

[4] Esther Levin, Roberto Pieraccini, Wieland Eckert, 2000 —*A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies*—, IEEE Transactions On Speech and Audio Processing, Vol 8, No.1

[5] Antoine Raux, Maxine Eskenazi, 2009 —*A Finite-State Turn-Taking Model for Spoken Dialog Systems*—, Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the ACL, pages 629-637

[6] Esther Levin, Roberto Pieraccini, Wieland Eckert, 1997. —*User Modeling For Spoken Dialogue System Evaluation*—, IEEE.

[7] Heriberto Cuayahuitl, Steve Renals, Oliver Lemon, Hiroshi Shimodaira, 2009. —*Evaluation of a Hierarchical Reinforcement Learning Spoken Dialogue System*—, Elsevier - Computer Speech and Language 24, pages 395-429

[8] Kate Acomb, Jonathan Bloom, Krishna Dayanidhi, Phillip Hunter, Peter Krogh, Esther Levin, Roberto Pieraccini, 2007. —*Technical Support Dialog Systems: Issues, Problems, and Solutions*—, Bridging the Gap: Academic and Industrial Research in Dialog Technologies Workshop Proceedings, pages 25-31

[9] Tim paek, Roberto Pieraccini, 2008. —*Automating spoken dialogue management design using machine learning: An industry perspective*—, Elsevier - Speech Communication 50, pages 716-729

[10] Llus Padr, 2011.—*Analizadores Multilinges en FreeLing*—, Linguamatica.

[11] Jason D. Williams, Steve Young, 2007. —*Partially observable Markov decision processes for spoken dialog systems*—, Computer Speech and Language 21, pages 393-422

[12] Jost Schatzmann, Steve Young, Karl weilhammer, Matt Stuttle 2006. —*A Survey of Statistical User Simulation Techniques for Reinforcement-Learning of Dialogue Management Strategies*—, The Knowledge Engineering Review, vol 00:0, pages 1-24

[13] Blaise Thomson, Jost Schatzmann, Steve Young, 2008. —*Bayesian Update of Dialogue State for Robust Dialogue Systems*—, IEEE.

[14] Benedi, J.-M., Lleida, E., Varona, A., Castro, M.-J., Galiano, I., Justo, R., Lopez de Letona, I., Miguel, A., 2006. —*Design and acquisition of a telephone spontaneous speech dialogue corpus in spanish: Dihana*—, Fifth International Conference on Language Resources and Evaluation (LREC). pages 1636-1639.

[15] David Griol, Lluos F. Hurtado, Encarna Segarra, Emilio Sanchis, 2008. —*A statistical approach to spoken dialog systems design and evaluation*—, Speech Communication vol. 50, pages 666-682.

[16] M. Ines Torres, Arantza del Pozo, Raquel Justo, Hector Olmedo, 2010. —*DIALSDK: desarrollo de una plataforma base para la implementacion de sistemas de dialogo*—,Project granted by the Basque Governement UE09+/37