

AIDED DESIGN OF FINITE-STATE DIALOGUE MANAGEMENT SYSTEMS

Olivier Pietquin – Thierry Dutoit

Faculté Polytechnique de Mons – TCTS Lab.
Parc Initialis – Av. Copernic, 1
B-7000 Mons – Belgium
E-mail : {pietquin,dutoit}@tcts.fpms.ac.be

ABSTRACT

Due to recent progresses in the field of speech and natural language processing, spoken dialogue systems are becoming more and more common. Nevertheless, the design of complete dialogue systems remains uneasy. On the one hand, developing such a system involves defining a dialogue strategy. Though automatic learning of dialogue strategies has been introduced in several researches, it stays hard to use in practice. On the other hand, system design implies some coding skills and is still a job for specialists despite the emergence of the VoiceXML language.

In this paper we describe a graphical interface dedicated to ease the development of dialogue systems. The user of the interface may be helped along his design thanks to automatically learned strategies.

1. INTRODUCTION

Nowadays Human-Computer Interfaces (HCI) are widely studied and become one of the major interests among the scientific community. Due to last decades' progresses in the field of speech technologies, like Automatic Speech Recognition (ASR) and Text-To-Speech synthesis (TTS) and in the field of Natural Language Processing (NLP), voice enabled interfaces and spoken dialogue systems are hopefully going to become more and more common.

With the growing influence of the Internet and the recent emergence of the XML technology, the w3c consortium defined the VoiceXML markup language to ease the development of such systems [1][2]. Indeed, the VoiceXML markup language describes an universal way to express dialogues featuring speech synthesis, automatic speech recognition and so on. Beyond that, the major strength of this language is to dissociate dialogues descriptions from the complex and low level programming generally induced by dialogue system development. This way, the dialogue description stands in a set of scripts whereas the low level programming takes place in a browser that will interpret the scripts and

produce the real sequences of interactions. This allows designers without particular skills to access voice technologies. Moreover, VoiceXML technology allows platform-independent and portable design in opposition with previous dialogue systems design tools [8][4].

With the concepts of VoiceXML scripts and browsers, dialogue systems design becomes analogous to web sites design. Yet, most of nowadays amateur web designers don't write HTML code anymore as graphical design tools became widespread during last few years. Thanks to those tools, a large number of amateur designers could *draw* their web site and brought their contributions to the growth of the Internet. In the same way, in the purpose of easing the dialogue designer's job and to allow amateur designers to create their own dialogues, we developed a graphical interface able to produce VoiceXML scripts.

On the other hand, the success of a spoken dialogue system mainly stands in its ability to provide the needed information to the human user within an acceptable time and behaving as naturally as possible. The behavior of a dialogue system is determined by its strategy. Designing a dialogue strategy, and so, defining the scheduling of voice interactions, is probably the most delicate task of the dialogue design process. Some researches in dialogue metrics and automatic strategy learning attempted to solve the problem of dialogue strategies design but results remain difficult to use in practice. They are usually used for evaluation and sometimes enhancement of already existing systems more than for designing new strategies from scratch. A fortiori, completely automatic design of dialogue strategies by artificial intelligence algorithms is still utopian today because of the partial subjectivity of the definition of an optimal strategy.

Therefore, in the aim of enabling non-specialist designers to create their own spoken dialogue systems, we propose to use the results of a reinforcement learning algorithm all along the design process to advise the designer. In this manner, designing a dialogue strategy will become a semi-automatic process involving itself a machine and a human operator.

In the following, the user of our interface will be called the *designer* or the *dialogue designer* while the term *user* will be associated to the user of the dialogue system.

2. STRATEGY LEARNING METHOD

As shown by Levin and Pieraccini [5], unsupervised learning techniques can be used in order to automatically learn optimal dialogue strategies, since supervised methods are not suitable. Indeed, a man-machine dialogue may be expressed as a Markov Decision Process (MDP) in terms of states, actions and strategy. Formalizing dialogues that way allows to apply Reinforcement Learning (RL) algorithms.

2.1. Formalization of dialogue

We can use the formalism of finite MDPs to describe a human-machine dialogue system as it is possible to define a finite state set $\mathbf{S} = \{s_i\}$ and a finite action set $\mathbf{A} = \{a_i\}$:

- \mathbf{S} is the set of dialogue states. Each particular dialogue state s_i generally describes the information obtained thus far by the system throughout its interaction with the human user. As a result, \mathbf{S} is finite; otherwise the interaction would never stop.
- \mathbf{A} is the set of all the possible system's actions. Typical dialogue system's actions $\{a_i\}$ are greetings, spoken utterances (constraining questions, confirmations, relaxation, presenting data etc.), database queries, closing the dialogue etc.

Using finite MDPs formalism also involves assuming the Markov property all through the dialogue. It is met if the transition probability to next state s_{t+1} given the history of the interaction, depends exclusively on the current state s_t and on the action a_t taken by the system when in state s_t :

$$P(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) = P_T(s_{t+1} | s_t, a_t)$$

To save us having problems linked to the Markov property, the state representation may include some information about the history of the interaction.

2.2 Simulating the dialogue environment

An RL agent learns an optimal strategy, which is a mapping between states and actions, through a trial-and-error process with its environment. As a result an MDP is defined by its state and action sets but also by the one-step dynamics of the environment that is formally characterized by two quantities:

- the probability to step from state s at time t to state s' at time $t+1$ by performing action a when in state s :

$$P_{sas'} = P_T(s_{t+1} = s' | s_t = s, a_t = a)$$

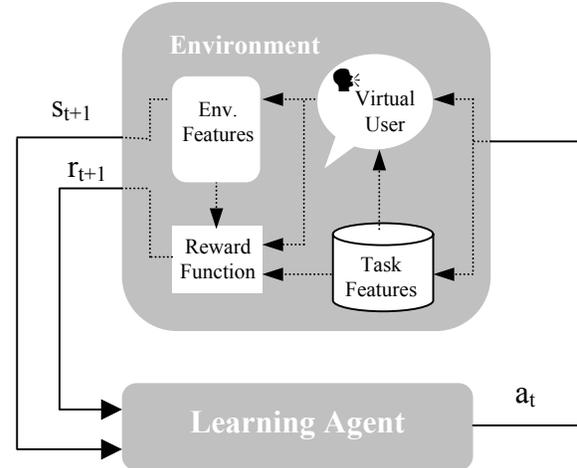


Figure 1: Environment Simulation

- the expected reward obtained when stepping from state s at time t to state s' at time $t+1$ having performed action a when in state s :

$$R_{sas'} = E[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s']$$

If it is relatively difficult to estimate those quantities, it is even more difficult to make the system interact with a real environment which would involve a true human user through all the learning process. Indeed, this process could last for thousands of dialogues. Thus, we developed a simulated dialogue environment (Fig. 1).

This environment is composed of four blocks.

- The *virtual user*, thanks to a set of probabilities [6][7], produces answers to the learning agent solicitations.
- The *task features* block is the only part of the environment to be different for each specific dialogue design. It can be a database, the list of slots to be filled in a form-filling application etc.
- The *environment features* block is optional and is used to introduce particular features of a dialogue environment. For example, it may be the model of an ASR system introducing recognition errors [7].
- The *reward function* block builds the immediate reward r_t of each interaction. It is a weighted sum based on

$$r_t = W_t N_t + W_{ASR} N_{ASR} - W_s f(U_s)$$

where N_t is set to 0 if the last state is reached and to 1 otherwise (it's a way to measure the duration of the dialogue), N_{ASR} is the number of possible recognition errors in the current turn and $f(U_s)$ is a function of the user's satisfaction. The weights W_x are adjustable and positive. Some terms can be added like $W_{dba} N_{dba}$ in the case of a database consultation application, where N_{dba} is the number of database accesses. The task dependent block would then be a database.

The cost of an entire dialogue session (\mathbf{R}_d) is then expressed as the sum of all the immediate rewards r_t obtained along the interaction:

$$R_d = \sum_{t=0}^{t=T_f} r_t$$

After an infinite number of dialogue sessions, we obtain:

$$\bar{R}_d = E[R_d] = E\left[\sum_{t=0}^{t=T_f} r_{t+1}\right] = \sum_{t=0}^{t=T_f} E[r_{t+1}] \rightarrow \sum_{t=0}^{t=T_f} R_{sas'}$$

Since the optimal strategy π^* is the strategy that minimizes the expected value of the dialogue cost, the simulated environment can be used to learn the optimal strategy.

As the learning agent interacts with a simulated environment and so, is in a pure learning process, we chose the “Exploring Starts Montecarlo” algorithm [9]. Indeed, it doesn’t have to follow any consistent strategy but has to explore all the state-action pairs as fast and as many times as possible.

3. CHOICE OF THE DIALOGUE MANAGEMENT MODEL

Generally, spoken dialogue systems are divided in three main categories. Actually, such a system may be system-led, user-led or mixed-initiative according to who controls the dialogue flow. Most common systems are system-led, in which the system asks a sequence of precise questions to the user, and mixed-initiative systems in which the system and the user share the control to cooperate in order to achieve the user’s goal. In the case of user-led systems, the system answers to direct user’s questions.

To handle those types of control sharing, two kinds of dialogue management can be investigated: self-organized or finite-state methods. In the case of finite-state methods, the dialogue is represented like a state-transition network where transitions between dialogue states specifies all legal paths through the network. All possible dialogues have to be known and described, the structure is quite fixed. Self-organized methods are of several kinds (theorem-proving, object-oriented and event-driven methods are the most common) and provide a better flexibility because the scheduling of the dialogue is dynamically updated according to some computation based on the current dialogue state.

Nevertheless, we chose the finite-state representation of dialogues for several reasons:

- Finite-state method is easier to understand and more intuitive for the designer as a visual, global and ergonomic representation of self-organized management would be difficult to realize. In short, it is more “user-friendly”.
- It is much more straightforward to give a visual representation of finite-state as a dialogue is described as a set of states linked by transitions.

- As said before, we use a Reinforcement Learning algorithm to automatically generate a dialogue strategy. This kind of artificial intelligence method is based on Markov Decision Processes that are a particular case of Finite State Machines. Thus, the mapping is easier to realize.
- Only user-led systems can’t be described by a finite-state model. Most of dialogues are generally system-led or mixed-initiative and, anyway, the VoiceXML language doesn’t allow user-led behavior.
- As discussed in [8], lots of applications like form-filling, database querying or directory interrogation are more successfully processed this way. Those applications are the most popular.
- System-led behavior is easy to describe as a state-transition network.
- Even if self-organized methods are definitively more compliant, nested sub-dialogues may help to gain in flexibility in the finite-state methods.
- As we will discuss in the next paragraph, we can imagine a mapping between a state-transition network and a VoiceXML script structure while a VoiceXML implementation of a self-organized system is hard to realize.

4. FROM STATE-TRANSITION NETWORKS TO VOICEXML STRUCTURES

Our interface is designed to generate VoiceXML scripts. It is thus necessary to define a mapping between the state-transition network drawn by the user and the structure of the script to be generated. Actually, we decided to start from the W3C VoiceXML 2.0 working draft [2] to design a graphical interface that reflected the main features of the specification and not to impose fixed VoiceXML structures to the features of a usual dialogue design GUI, with the purpose of keeping the code editable. Of course, all the features couldn’t be integrated yet to keep the interface easy of use. Here are some decisions we took in order to realize this mapping:

- Each drawn network corresponds to a form in the VoiceXML script.
- Each state in the network can be mapped to a form item that may be an input item (<field>, <record>, <transfer> or <subdialog> items) or a control item (generally a <block> item).
- A name is associated to each state. This name is also the id of the corresponding form item in the script.
- There is one “start” node. The properties of this node may include global variables, grammars or event handlers for the current dialogue.
- There may be several “exit” nodes as several ends may be possible for each dialogue. Each exit node corresponds to a <block> in the generated script. This

<block> can be used to utter an ending prompt to the user, redirect the dialogue to another form or document or to submit some variables of the current dialogue to an external script.

- To each transition corresponds a <goto> tag in the script. The “nextitem” attribute of the <goto> tag is used to redirect the dialogue to the correct item in the current form.
- If there are several transitions starting from a single state, a condition is associated to each transition and an <if> tag will be generated in the script with a corresponding “cond” attribute.

To each state is associated a specific window that enables the user to fill-in particular features of the state. For example, to specify prompts and speech grammars of a <field> state, generate the executable content of a <block> state, to detail the list of parameters to pass to a subdialog. Double clicking on a <subdialog> state or on an exit state redirecting to an other dialogue will also allow the user to open a new design window in order to draw a new network.

5. AIDED DESIGN

In the second section of this paper, we described an automatic strategy learning technique. We can imagine several methods to use the results of the learning. Of course, we can display a state-transition network that represent the complete optimal strategy learned by the RL system. This is possible when there are few states in the network and the interface is able to produce such a network in this case. Nevertheless, when the state space is large and the number of transitions grows, for example in the case of a mixed-initiative dialogue (because the user can be over-informative and provide more information than asked by the system), considering all the possible paths will result in a unmanageable display. This is augmented if repair mechanisms are included in the dialogue. That is why we opted for an aided design.

Indeed, as the learned strategy is a mapping between states and actions, the user is advised to perform a given action when he draws a new state in the interface. According to previously designed states and corresponding actions in the network, the interface infers the current state. This can anyway be corrected by the designer. If several actions are possible, the designer can choose the one he wants or none. After his choice, the designer will have to complete the options of the chosen action (prompts and grammars when the action corresponds to a user query, for example).

6. CONCLUSIONS AND FURTHER WORKS

By creating this interface, our goal was to ease the design of spoken dialogue systems. To do so, we tried to

integrate some techniques to help novice users like state-transition networks representation of dialogues and strategy learning. Nevertheless, some specific troubles of dialogue design remain. Writing speech grammars, for example, is still a major problem. As speech grammars can be seen as finite state machines, we think to about integrating a graphical tool for grammars design and testing in our interface. Other features, like direct recording of prompts or even better, automatically generated prompts, can be investigated.

7. REFERENCES

- [1] The VoiceXML Forum : <http://www.voicexml.org>
- [2] VoiceXML 2.0 Working Draft (W3C) : <http://www.w3.org/TR/voicexml20/>
- [3] S. Sutton, D. G. Novick, R. A. Cole, M. Fanty, ‘Building 10,000 spoken-dialogue systems’, in *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Philadelphia, 1996
- [4] S.R. Klemmer, A.K. Sinha, J. Chen, J. A. Landay, N. Aboobaker, and A. Wang, ‘SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces.’ In *CHI Letters, The 13th Annual ACM Symposium on User Interface Software and Technology: UIST 2000*.
- [5] R. Pieraccini, E. Levin, W. Eckert ‘A Stochastic Model of Human Machine Interaction for Learning Dialog Strategies’. *IEEE Transactions on Speech and Audio Processing, Vol. 8 pp 11-23. 2000*.
- [6] R. Pieraccini, E. Levin, W. Eckert ‘User Modeling for Spoken Dialogue Systems,’ *Proc. IEEE ASR Workshop*, Santa Barbara, 1997.
- [7] O. Pietquin, S. Renals ‘ASR System Modeling for Automatic Evaluation And Optimization of Dialogue Systems’, *Proceedings of the International Conference on Acoustics Speech and Signal Processing, ICASSP 2002*, Orlando 2002.
- [8] M. McTear. ‘Modelling Spoken Dialogues with State Transition Diagrams: Experiences with the CSLU Toolkit’. *Proc 5th International Conference on Spoken Language Processing*, Sydney, Australia, 1998
- [9] R. S. Sutton, A.G. Barto ‘Reinforcement Learning : An Introduction’ *MIT Press* 1998