This document is published in:

Corchado, E. et al. (eds.) (2011) *Soft Computing Models in Industrial and Environmental Applications*, *6th International Conference SOCO 2011*, (Advances in Intelligent and Soft Computing, 87), Springer, 173-182.
DOI: http://dx.doi.org/10.1007/978-3-642-19644-7_19

Soft Computing Models for the Development of Commercial Conversational Agents

David Griol, Javier Carbó, and José Manuel Molina

Group of Applied Artificial Intelligence (GIAA), Computer Science Department,
Carlos III University of Madrid
e-mail: {david.griol,javier.carbo,josemanuel.molina}@uc3m.es

Abstract. In this paper we present a proposal for the development of conversational agents that, on the one hand, takes into account the benefits of using standards like VoiceXML, whilst on the other, includes a module with a soft computing model that avoids the effort of manually defining the dialog strategy. This module is trained using a labeled dialog corpus, and selects the next system response considering a classification process based on neural networks that takes into account the dialog history. Thus, system developers only need to define a set of VoiceXML files, each including a system prompt and the associated grammar to recognize the users responses to the prompt. We have applied this technique to develop a conversational agent in VoiceXML that provides railway information in Spanish.

1 Introduction

A conversational agent can be defined as a software that accepts natural language as input and generates natural language as output, engaging in a conversation with the user. When designing this kind of agents, developers need to specify the system actions in response to user utterances and environmental states that, for example, can be based on observed or inferred events or beliefs. In addition, the dialog manager needs a dialog strategy that defines the conversational behavior of the system. This is the fundamental task of dialog management [11], as the performance of the system is highly dependent on the quality of this strategy. Thus, a great effort is employed to empirically design dialog strategies for commercial systems. In fact, the design

Funded by projects CICYT TIN2008-06742-C02-02/TSI, CICYT TEC2008-06732-C02-02/TEC, CAM CONTEXTS (S2009/TIC-1485), and DPS2008-07029-C02-02.

of a good strategy is far from being a trivial task since there is no clear definition of what constitutes a good strategy [15].

Once the dialog strategy has been designed, the implementation of the system is leveraged by programming languages such as the standard VoiceXML[1], for which different programming environments and tools have been created to help developers. These programming standards allow the definition of a dialog strategy based on scripted Finite State Machines. With the aim of creating dynamic and adapted dialogs, as an alternative of the previously described rule-based approaches, the application of soft computing models and statistical approaches to dialog management makes it possible to consider a wider space of dialog strategies [4, 16, 6]. The main reason is that these models can be trained from real dialogs, modeling the variability in user behaviors. The final objective is to develop conversational agents that have a more robust behavior and are easier to adapt to different user profiles or tasks.

The most extended methodology for machine-learning of dialog strategies consists of modeling human-computer interaction as an optimization problem using Markov Decision Process (MDP) and reinforcement methods [8]. The main drawback of this approach is due to the large state space of practical spoken dialog systems, whose representation is intractable if represented directly [17]. Partially Observable MDPs (POMDPs) outperform MDP-based dialog strategies since they provide an explicit representation of uncertainty [13]. However, they are limited to small-scale problems, since the state space would be huge and exact POMDP optimization is again intractable [17]. Other interesting approaches for statistical dialog management are based on modeling the system by means of Hidden Markov Models (HMMs) [1] or using Bayesian networks [10].

Additionally, speech recognition grammars for commercial systems have been usually built on the basis of handcrafted rules that are tested recursively, which in complex applications is very costly [9]. However, as stated by [12], many sophisticated commercial systems already available receive a large volume of interactions. Therefore, industry is becoming more interested in substituting rule based grammars with other soft computing techniques based on the large amounts of data available.

As an attempt to improve the current technology, we propose to merge soft computing models with VoiceXML. Our goal is to combine the flexibility of statistical dialog management with the facilities that VoiceXML offers, which would help to introduce soft computing models for the development of commercial (and not strictly academic) conversational agents. To this end, our technique employs a soft computing model based on neural networks that takes into account the history of the dialog up to the current dialog state in order to predict the next system response. The soft computing model is learned from a labeled training corpus for the task and is mainly based on the modelization of the sequences of the system and user dialog acts and the definition of a data structure, which takes into account the data supplied by the user throughout the dialog, and makes the estimation of the model from the training data manageable. Expert knowledge about deployment of VoiceXML applications, development environments and tools can still be exploited using our

[1] http://www.w3.org/TR/voicexml20/

technique. The only change is that transitions between dialog states is carried out on a data-driven basis (i.e., is not deterministic). In addition, the system prompts and the grammars for ASR are implemented in VoiceXML-compliant formats (e.g., JSGF or SRGS).

The remainder of the paper is as follows. Section 2 describes our proposal to model the dialog and develop a module to predict the next system response based on a soft computing model. Section 3 presents a detailed explanation of how to construct this module to develop a commercial railway information conversational agent, and the results of its evaluation. Finally, our conclusions are presented.

2 Our Proposal to Introduce Soft Computing Models in Commercial Conversational Agents

As stated in the introduction, our approach to integrate soft computing methodologies in commercial applications is based on the automatic learning of the dialog strategy using a soft computing dialog management methodology. In most conversational agents, the dialog manager makes decisions based only on the information provided by the user in the previous turns and its own dialog model. For example, this is the case with most conversational agents for slot-filling tasks. The methodology that we propose for the selection of the next system response for this kind of task is detailed in [5]. We represent the dialogs as a sequence of pairs (A_i, U_i), where A_i is the output of the dialog system (the system answer) at time i, expressed in terms of dialog acts; and U_i is the semantic representation of the user turn (the result of the understanding process of the user input) at time i, expressed in terms of frames. This way, each dialog is represented by:

$$(A_1, U_1), \cdots, (A_i, U_i), \cdots, (A_n, U_n)$$

where A_1 is the greeting turn of the system, and U_n is the last user turn. From now on, we refer to a pair (A_i, U_i) as S_i, the state of the dialog sequence at time i.

In this framework, we consider that, at time i, the objective of the dialog manager is to find the best system answer A_i. This selection is a local process for each time i and takes into account the previous history of the dialog, that is to say, the sequence of states of the dialog preceding time i:

$$\hat{A}_i = \underset{A_i \in \mathscr{A}}{\operatorname{argmax}} P(A_i | S_1, \cdots, S_{i-1})$$

where set $\mathscr{A}$ contains all the possible system answers.

The main problem with dealing to resolve the latter equation is regarding the number of all possible sequences of states, which is usually very large in a practical domain. To solve this problem, we define a data structure in order to establish a partition in the space of sequences of states (i.e., in the history of the dialog preceding time i). This data structure, that we call *Dialog Register* (*DR*), contains the

information provided by the user throughout the previous history of the dialog. After applying the above considerations and establishing the equivalence relation in the histories of dialogs, the selection of the best A_i is given by:

$$\hat{A}_i = \underset{A_i \in \mathscr{A}}{\operatorname{argmax}} P(A_i | DR_{i-1}, S_{i-1})$$

Each user turn supplies the system with information about the task; that is, he/she asks for a specific concept and/or provides specific values for certain attributes. However, a user turn could also provide other kinds of information, such as task-independent information (for instance, *Affirmation*, *Negation*, and *Not-Understood* dialog acts). This kind of information implies some decisions which are different from simply updating the DR_{i-1}. Hence, for the selection of the best system response A_i, we take into account the *DR* that results from turn 1 to turn $i-1$, and we explicitly consider the last state S_{i-1}. Our model can be extended by incorporating additional information to the *DR*, such as some chronological information (e.g. number of turns up to the current turn) or user profiles (e.g. user experience or preferences).

2.1 Soft Computing Approach Proposed for the Implementation of the Dialog Manager

We propose to solve the latter equation by means of a classification process. This way, every dialog situation (i.e., each possible sequence of dialog acts) is classified taking into a set of classes $\mathscr{C}$, which groups together all the sequences that provide the same set of system actions (answers). Thus, the objective of the dialog manager at each moment is to select a class of this set $c \in \mathscr{C}$, and the answer of the system at that moment is the answer associated with this selected class.

The classification function can be defined in several ways. We have evaluated four different definitions of such a function: a multinomial naive Bayes classifier, n-gram based classifier, a classifier based on grammatical inference techniques and a classifier based on neural networks [5]. An approach that uses Support Vector Machines for the classification process can be found in [2].

The best results were obtained using a multilayer perceptron (MLP) [14] where the input layer holds the input pair (DR_{i-1}, S_{i-1}) corresponding to the dialog register and the state. The values of the output layer can be seen as an approximation of the a posteriori probability of belonging to the associated class $c \in \mathscr{C}$.

As stated before, the *DR* contains information about concepts and attributes provided by the user throughout the previous history of the dialog. For the dialog manager to determine the next answer, we have assumed that the exact values of the attributes are not significant. They are important for accessing the databases and for constructing the output sentences of the system. However, the only information necessary to predict the next action by the system is the presence or absence of concepts and attributes. Therefore, the information we used from the *DR* is a codification of

this data in terms of three values, $\{0,1,2\}$, for each field in the *DR* according to the following criteria:

- **0**: The concept is unknown or the value of the attribute is not given.
- **1**: The concept or attribute is known with a confidence score that is higher than a given threshold. Confidence scores are given during the recognition and understanding processes. and can be increased by means of confirmation turns.
- **2**: The concept or attribute is activated with a confidence score that is lower than the given threshold.

3 Development of a Railway Information System Using the Proposed Technique

To test our proposal, we have used the definitions taken to develop the EDECAN dialog system, which was developed in a previous study to provide information about train services, schedules and fares in Spanish [6, 5]. A corpus of 900 dialogs was acquired for this project. In this corpus, the system generates a total of 51 different prompts, which were labeled taking into account three levels of labeling defined for the labeling of the system dialog acts. The first level describes general acts which are task independent. The second level is used to represent concepts and attributes involved in dialog turns that are task-dependent. The third level represents values of attributes given in the turns. The following labels were defined for the first level: *Opening*, *Closing*, *Undefined*, *Not-Understood*, *Waiting*, *New-Query*, *Acceptance*, *Rejection*, *Question*, *Confirmation*, and *Answer*. The labels defined for the second and third level were the following: *Departure-Hour*, *Arrival-Hour*, *Price*, *Train-Type, Origin*, *Destination*, *Date*, *Order-Number*, *Number-Trains*, *Services*, *Class*, *Trip-Type*, *Trip-Time*, and *Nil*. The 51 different system prompts have been automatically generated in VoiceXML using the proposed technique. For example, Figure 1 shows the VXML document to prompt the user for the origin city and the obtained grammar for ASR.

The *DR* that we have defined for our railway information system is a sequence of 15 fields, corresponding to the five possible queries that users can carry out to the system (*Hour, Price, Train-Type, Trip-Time, Services*) and the ten attributes that they can provide to complete these queries (*Origin, Destination, Departure-Date, Arrival-Date, Departure-Hour, Arrival-Hour, Class, Train-Type, Order-Number, Services*).

This way, every dialog begins with a dialog register in which every value is equal to 0 and the greeting turn of the system, as it is showed following.

. . . . . .

S_1: *Welcome to the railway information system. How can I help you?*

A_1: (*Opening*:*Nil*:*Nil*)

DR_0: 00000-1000001000

. . . . . .

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
   xmlns:xsi="http://www.w3.org/2001/
   XMLSchema-instance"
   xsi:schemaLocation="http://www.w3.org/2001/vxml
   http://www.w3.org/TR/voicexml20/vxml.xsd"
   version="2.0" application="app-trains.vxml">
 <form id="origin_form">
  <field name="origin">
     <grammar type="application/srgs+xml"
           src="/grammars/origin.grxml"/>
     <prompt>Tell me the origin city.</prompt>
     <filled>
       <return namelist="origin"/>
     </filled>
  </field>
 </form>
</vxml>
```

```
#JSGF V1.0;
grammar origin;
public <origin> = [<desire>]
[<travel> <city> {this.destination=$city}]
[<proceed> <city> {this.origin=$city}];
<desire> = I want [to know] | I would like
[to know] | I would like | I want | I need
| I have to;
<travel> = go to | travel to | to go to
| to travel to;
<city> = Murcia | Vigo | Sevilla | Huelva |
Cuenca | Lugo | Granada | Salamanca |
Valencia | Alicante | Albacete | Barcelona
| Madrid;
<proceed> = from | going from | go from;
```

Fig. 1 VoiceXML document to require the origin city (left) and grammar to capture the associated value (right)

Each time the user provides information, this is used to update the previous *DR* and to obtain the new one. For instance, given a user turn providing the origin city, the destination city and the date, the new dialog register could be as follows.

. . . . . .

U_1: *I want to know timetables from Valencia to Madrid.*

Task Dependent Information: (*Hour*) [0.7] *Origin*:Valencia [0.2] *Destination*:Madrid [0.9]

Task Independent Information: None

DR_1: 10000-2100000000

. . . . . .

In this case, the confidence score assigned to the attribute *Origin* (showed between brackets in the previous example) is very low. Then, a "2" value is added in the corresponding position of the DR_1. The concept (*Hour*) and the attribute *Destination* are recognized with a high confidence score, adding a "1" value in the corresponding positions of the DR_1. Then, the input of the MLP is generated using DR_1, the codification of the labeling of the last system turn (A_1), and the task-independent information provided in the last user turn (none in this case). The output selected for the MLP would consist in the case of requiring the departure date. This process is repeated to predict the next system response afterwards each user turn.

4 Evaluation of the Developed Conversational Agent

A 5-fold cross-validation process was used to carry out the evaluation of the developed conversational agent. This way, the corpus containing 900 dialogs from the railway information domain was randomly split into five subsets of 1,232 samples (20% of the corpus). Our experiment consisted of five trials. Each trial used a different subset taken from the five subsets as the test set, and the remaining 80% of the

corpus was used as the training set. A validation subset (20%) was extracted from each training set.

In order to successfully use neural networks as classifiers, a number of considerations had to be taken into account, such as the network topology, the training algorithm, and the selection of the parameters of the algorithm. Using *April*, topology and algorithm parameters (i.e. learning rate and momentum) are estimated with an exhaustive search, using as stop criteria the MSE obtained in each epoch for the validation set. The gradient is computed in incremental mode, this way the weights are updated after each input, also a momentum is added to the backpropagation so that the networks can overcome local minimums. Different experiments were conducted using different network topologies of increasing number of weights: a hidden layer with 2 units, two hidden layers of 2 units each, two hidden layers of 4 and 2 units, a hidden layer with 4 units, etc. Several learning algorithms were also tested: the incremental version of the backpropagation algorithm (with and without momentum term) and the quickprop algorithm. The influence of their parameters such as learning rate or momentum term was also studied.

To train and evaluate the neural networks, we used the *April* toolkit, developed by the Technical University of Valencia [3]. *April* is an efficient implementation of the Backpropagation (BP) algorithm to train neural networks with general feedforward topology. *April* employes Lua [7], an extensible procedural embedded programming language especially designed for extending and customizing applications with powerful data description facilities. In our case, we have used it to describe the network topologies and the experiments. Besides, *April* adds a *matrix* and a *dataset* class which allow the straightforward definition and manipulation of huge sets of samples more flexibly than simply enumerating the pairs of inputs and outputs. The toolkit also includes additional features like softmax activation function, tied and constant weights, weight decay, value representation, and reproducibility of experiments. The Appendix section shows an excerpt from the script used to train and test our topologies with the *April* toolkit.

We firstly tested the influence of the topology of the MLP, by training different MLPs of increasing number of weights using the standard backpropagation algorithm (with a sigmoid activation function and a learning rate equal to 0.2), and selecting the best topology according to the mean square error (MSE) of the validation data. The minimum MSE of the validation data was achieved using an MLP of one hidden layer of 32 units. We followed our experimentation with MLPs of this topology, training MLPs with several algorithms: the incremental version of the backpropagation algorithm (with and without momentum term) and the quickprop algorithm. The best result on the validation data was obtained using the MLP trained with the standard backpropagation algorithm and a value of LR equal to 0.3.

We propose three measures to evaluate the prediction of the next system prompt by the soft computing methodology. These measures are calculated by comparing the answer automatically generated by this module for each input in the test partition with regard to the reference answer annotated in the evaluation corpus. This way, the evaluation is carried out turn by turn. These three measures are: i) *%exact*:

percentage of answers provided by the dialog manager that are equal to the reference system answer in the corpus; ii) *%coherent*: percentage of answers provided by the dialog manager that are coherent with the current state of the dialog although they do not follow the original strategy; iii) *%error*: percentage of answers provided by the dialog manager that would cause a dialog failure. Table 1 shows the results of the evaluation.

Table 1 Evaluation results obtained with the dialog manager developed for the railway conversational agent

	%exact	*%coherent*	*%error*
System answer	84.11%	93.76%	4.24%

The results of the *%exact* and *%coherent* measures show the satisfactory performance of the developed dialog manager. The codification defined to represent the state of the dialog and the good operation of the MLP classifier make it possible for the answer generated by the manager to agree with exactly the reference response in the corpus in 84.11% of cases. The percentage of answers generated by the MLP that can cause a system failure is only 4.24%. Finally, an answer that is coherent with the current state of the dialog is generated in 93.76% of cases. These results also demonstrate the correct operation of the proposed soft computing methodology.

5 Conclusions

In this paper, we have described a technique for developing interactive conversational agents using a well known standard like VoiceXML, and considering a soft computing dialog model that is automatically learned from a dialog corpus. The main objective of our work is to reduce the gap between academic and commercial systems by reducing the effort required to define optimal dialog strategies and implement the system. Our proposal works on the benefits of statistical methods for dialog management and VoiceXML, respectively. The former provide an efficient means to exploring a wider range of dialog strategies, whereas the latter makes it possible to benefit from the advantages of using the different tools and platforms that are already available to simplify system development.

We have applied our technique to develop a conversational agent that provides railway information, and have shown that it enables creating automatically VoiceXML documents to prompt the user for data, as well as the necessary grammars for ASR. The results of its evaluation shows the prediction of a coherent system answer in most of the cases. As a future work, we plan to study ways for adapting the proposed soft computing model to more complex domains and corpora.

References

1. Cuayáhuitl, H., Renals, S., Lemon, O., Shimodaira, H.: Human-Computer Dialogue Simulation Using Hidden Markov Models. In: Proc. of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU 2005), San Juan, Puerto Rico, pp. 290–295 (2005)
2. Denecke, M., Yasuda, N.: Does this answer your Question? Towards Dialogue Management for Restricted Domian Question Answering Systems. In: Proc. of the 6th SIGdial, Workshop on Discurse and Dialogue, Lisbon, Portugal, pp. 65–76 (2005)
3. Espana-Boquera, S., Zamora-Martinez, F., Castro-Bleda, M., Gorbe-Moya, J.: Efficient BP algorithms for general feedforward neural networks. In: Mira, J., Álvarez, J.R. (eds.) IWINAC 2007. LNCS, vol. 4527, pp. 327–336. Springer, Heidelberg (2007)
4. Georgila, K., Henderson, J., Lemon, O.: User Simulation for Spoken Dialogue Systems: Learning and Evaluation. In: Proc. of the 9th Interspeech/ICSLP, Pittsburgh, USA, pp. 1065–1068 (2006)
5. Griol, D., Hurtado, L., Segarra, E., Sanchis, E.: A Statistical Approach to Spoken Dialog Systems Design and Evaluation. Speech Communication 50(8–9), 666–682 (2008)
6. Griol, D., Riccardi, G., Sanchis, E.: A Statistical Dialog Manager for the LUNA project. In: Proc. of Interspeech/ICSLP 2009, pp. 272–275 (2009)
7. Ierusalimschy, R.: Programming in Lua. Published by Lua.org (2003)
8. Levin, E., Pieraccini, R., Eckert, W.: A stochastic model of human-machine interaction for learning dialog strategies. IEEE Transactions on Speech and Audio Processing 8(1), 11–23 (2000)
9. McTear, M.F.: Spoken Dialogue Technology: Towards the Conversational User Interface. Springer, Heidelberg (2004)
10. Paek, T., Horvitz, E.: Conversation as action under uncertainty. In: Proc. of the 16th Conference on Uncertainty in Artificial Intelligence, San Francisco, USA, pp. 455–464 (2000)
11. Paek, T., Pieraccini, R.: Automating spoken dialogue management design using machine learning: An industry perspective . Speech Communication 50(8–9), 716–729 (2008)
12. Pieraccini, R., Suendermann, D., Dayanidhi, K., Liscombe, J.: Are We There Yet? Research in Commercial Spoken Dialog Systems. In: Matoušek, V., Mautner, P. (eds.) TSD 2009. LNCS, vol. 5729, pp. 3–13. Springer, Heidelberg (2009)
13. Roy, N., Pineau, J., Thrun, S.: Spoken dialogue management using probabilistic reasoning. In: Proc. of the 38th Annual Meeting of the Association for Computational Linguistics (ACL 2000), Hong Kong, China, pp. 93–100 (2000)
14. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: PDP: Computational models of cognition and perception, pp. 319–362. MIT Press, Cambridge (1986)
15. Schatzmann, J., Weilhammer, K., Stuttle, M., Young, S.: A Survey of Statistical User Simulation Techniques for Reinforcement-Learning of Dialogue Management Strategies. Knowledge Engineering Review 21(2), 97–126 (2006)
16. Williams, J., Young, S.: Partially Observable Markov Decision Processes for Spoken Dialog Systems. Computer Speech and Language 21(2), 393–422 (2007)
17. Young, S., Schatzmann, J., Weilhammer, K., Ye, H.: The Hidden Information State Approach to Dialogue Management. In: Proc. of 32nd IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Honolulu, Haway, USA, vol. 4, pp. 149–152 (2007)

Appendix: Training and Testing a MLP with the *April* Toolkit

```
------------------------------------------------------------
-- READING DATA FROM FILES
------------------------------------------------------------
train_input, train_output = read_file("train.txt")
test_input,  test_output, d_output = read_file("test.txt")
------------------------------------------------------------
-- PARAMETERS DEFINITION
------------------------------------------------------------
num_hidden1      =   110
num_hidden2      =   110
defined_learning_rate  =    0.03
defined_momentum       =    0.02
weight_decay      =    0.0
seed_network       =  123
seed_shuffle   =  456
num_train       =  100
------------------------------------------------------------
-- NEURAL NETWORK CREATION
------------------------------------------------------------
num_inputs = train_input:patternSize()
num_outputs = numclasses
rand1 = random(seed_network)
rand2 = random(seed_shuffle)
if num_hidden2 == 0 then
  stringnetwork = string.format("%d inputs %d logistic %d softmax",
                  num_inputs, num_hidden1, num_outputs)
else
  stringnetwork = string.format("%d inputs %d logistic %d logistic %d softmax",
                  num_inputs,num_hidden1, num_hidden2, num_outputs)
end

printf("# Neural network with topology: %s\n",networkstring)
printf("# Data test with %d patterns\n",test_input:numPatterns())
Neuralnetwork = Mlp(stringnetwork)
Neuralnetwork :generate(rand1, -0.7, 0.7) -- generates weights of the network

datatrain = {
   learning_rate  = defined_learning_rate,
   momentum       = defined_momentum,
   weight_decay   = weight_decay,
   input_dataset  = train_input,
   output_dataset = train_output,
   shuffle        = rand2
}
datatest = {
   input_dataset  = test_input,
   output_dataset = test_output,
}
------------------------------------------------------------
-- TRAINING AND SAVING THE NEURAL NETWORK
------------------------------------------------------------
printf("# epoch mse_train mse_test\n")
for epoch = 1,num_train do
  errortrain = MLP:train(datatrain)
  errortest  = MLP:validate(datatest)
end
Neuralnetwork:save("neuralnetwork_top1.txt")
```