DIALOGUE-BASED MANAGEMENT OF USER FEEDBACK IN AN AUTONOMOUS PREFERENCE LEARNING SYSTEM*

Juan Manuel Lucas-Cuesta, Javier Ferreiros
Speech Technology Group, Universidad Politécnica de Madrid, Spain
juanmak@die.upm.es, jfl@die.upm.es

Asier Aztiria
University of Mondragon, Spain
aaztiria@eps.mondragon.edu

Juan Carlos Augusto, Michael McTear
School of Computing and Mathematics, University of Ulster at Jordanstown, Northern Ireland, UK
jc.augusto@ulster.ac.uk, mf.mctear@ulster.ac.uk

Keywords: Spoken dialogue systems, Autonomous preference learning, User feedback, Human-computer interaction.

Abstract: We present an enhanced method for user feedback in an autonomous learning system that includes a spoken dialogue system to manage the interactions between the users and the system. By means of a rule-based natural language understanding module and a state-based dialogue manager we allow the users to update the preferences learnt by the system from the data obtained from different sensors. The design of the dialogue together with the storage of context information (the previous dialogue turns and the current state of the dialogue) ensures highly natural interactions, reducing the number of dialogue turns and making it possible to use complex linguistic constructions instead of isolated commands.

1 INTRODUCTION

Ambient Intelligence (AmI), defined as 'a digital environment that proactively, but sensibly, supports people in their daily lives' (Augusto, 2007), is the focus of intensive research within the Computer Science community, as AmI systems provide significant opportunities for computers to improve the standards of life for some segments of our society. Applications of AmI are nowadays being developed in a variety of areas: automotive, healthcare, etc. (Cook et al., 2009).

All those applications have a common overarching aim of serving people unobtrusively (Weiser, 1991; Aghajan et al., 2009). Spoken dialogue systems (McTear, 2004) are particularly well suited to cover this requirement as they allow humans to communicate with a system in a natural way, without demanding technical training. This provides a much more natural interaction mode than, for example, forcing users to use a PDA, a keyboard, a touch screen, remote controls, or other physical devices.

A natural interaction also implies the ability of the system to adapt to different users. It is thus important that the system can identify the habits and preferences of each user, and learn this information to modify its behaviour in order to offer each user the functionalities that are most appropriate. The learning of these usage patterns has to be carried out in a transparent way for the user. State-of-the-art learning algorithms would produce associations which are not entirely appropriate for the user. Therefore, it is important that such algorithms are coupled with a system which allows a user to fine-tune those learnt preferences.

1.1 Preference Learning

Learning the habits and preferences of each user is an essential feature in any AmI system. There exist different examples in the literature regarding this learning task. One of the first approaches concerned the use of Artificial Neural Networks for inferring rules for smart homes (Begg and Hassan, 2006). Other techniques, such as Fuzzy-Logic (Hagras et al., 2004)

*This work has been partially supported by the Spanish Ministry of Education under the Program of University Personnel Training (FPU), reference AP2007-00463.

and Case-Based Reasoning (Sadeh et al., 2005), have also been used but, as Müller pointed out (Müller, 2004), 'the overall dilemma remains: there does not seem to be a system that learns quickly, is highly accurate, is nearly domain independent, does this from few examples with literally no bias, and delivers a user model that is understandable and contains breaking news about the user's characteristics'.

1.2 User Feedback

It is important that the user gets information regarding the patterns learnt by the system, in such a way that he/she can accept the correct ones and refine or remove the wrong ones. To offer the user that information, the system needs an interaction interface. It is also important that the interaction between the user and the system takes place as intuitively for the human as possible. In that sense, it is a highly desirable goal to allow users to interact with the system by speaking with it, since speech is the most natural way of communication between humans.

A speech-based system needs to understand what the users say and to provide spoken answers to them. It also has to gather the actions that users want to carry out and deliver them to the proper set of actuators. We will refer to a system that can perform all these tasks as a Spoken Dialogue System (SDS, (McTear, 2004)).

To offer the user a flexible and natural interaction, it is possible to design the dialogue manager (i.e. the core of the SDS) as a finite state machine (FSM) that offers the users the different actions to perform following a given command. However, more interesting approaches offer the user more initiative when interacting with the system. For instance, a Bayesian Networks (BN) approach (Fernández et al., 2005), makes the system able to deal with incomplete information situations, such as ellipsis or anaphora, thus allowing the user to speak in a highly natural and friendly way.

Our approach of developing a complete spoken dialogue system makes use of a FSM strategy, enhancing the vocabulary that the recognizer is able to handle. However, although we provide the users with the ability to talk in a more natural way, the initiative of the dialogue still relies on the system, which controls the dialogue flow. To address this, we offer the user the chance to skip several of the states, if he/she has enough experience in interacting with the system.

1.3 Autonomous Learning System

Our approach aims at obtaining user patterns from sensor data. For that we have developed a system, called *PUBS* (Patterns of User Behaviour System) (Aztiria et al., 2008), which discovers user's common behaviours and habits from data recorded by sensors. *PUBS* is made up of three different modules. A Language ($\mathcal{L}_{PUBS}$) to represent the learnt patterns in a clear and non ambiguous way; an Algorithm ($\mathcal{A}_{PUBS}$) that discovers patterns in data collected by sensors; and an interaction system (I_{PUBS}) which allows the users to have a basic interaction with the system.

Our original HCI interface, I_{PUBS}, was developed using Sphinx-4 (Walker et al., 2004) for the Automatic Speech Recognition (ASR) task, and FreeTTS (Walker et al., 2002) to provide speech-based feedback to the user. This system had two main drawbacks. On one hand, the user had to know the way *PUBS* stored and represented the information. On the other hand, the system-initiative approach for managing speech interaction allowed the user to talk in a very restricted way, using only isolated words (i.e. *accept*, *reject*, or closed answers), or simple commands, such as device names or numbers.

Given that we want to provide the users with a more natural and flexible speech interaction, our aim is to develop a dialogue-based interaction system, substituting the baseline keyword-based approach but reusing the recognition and synthesis tools. The new interaction system, which will be referred to as I^2_{PUBS}, is presented in the next Section.

2 DIALOGUE-BASED USER FEEDBACK

The spoken dialogue system we have developed, I^2_{PUBS}, makes use of the Sphinx-4 recognition software and the FreeTTS synthesizer, as we have said before. Therefore, our goal is to design the natural language understanding (NLU) module, as well as the dialogue manager (DM).

Although we have kept the ASR module, we have modified the language models it makes use of, allowing more complex sentences. We have collected a set of sentences of the application domain (i.e. the control of the patterns learnt by the system), and used them to train an *n*-gram based language model.

Each of the new modules works as follows. Once the ASR has recognized the input utterance (i.e. identified the words which form the utterance), the NLU extracts the semantics of those words. We will refer to this semantics as the *dialogue concepts*. Using these concepts, the DM obtains the user intentions, that is, the commands that the user has said to the system. If this information is enough to carry out the task the user has requested, the system will perform it. Otherwise, it will store this information, together with the current dialogue state, in the *Context Information Memory*, and will ask the user for the remaining data.

2.1 Natural Language Understanding

The NLU module extracts the semantics from the recognized utterance. By semantics we refer to the 'meaning', related to the application domain, of the different words or phrases which form the utterance. We represent the semantic information as *dialogue concepts*. Each concept consists of an *attribute* (a concept identifier) and the *value* that the user has assigned to the attribute. In our domain, the values can be literals (i.e. the name of a room), numbers (i.e. the value measured by a sensor) or binary variables (i.e. whether the user has requested some information).

The NLU module develops a two-step procedure. First of all, each recognized word is categorized according to different criteria. Once every word is categorized, the understanding step converts the categorized words into dialogue concepts. Both the categorization and the understanding submodules perform their tasks using a set of rules defined by an expert.

We have defined a set of 20 different categories including a *GARBAGE* tag, which is used to label those words that do not carry semantic information.

A given word can be labeled with more than one category, depending on its context (i.e., the closest words in the utterance). We have developed a bottom-up strategy to categorize the recognized utterance. We first categorize the words with a single category (i.e. numbers, rooms, or days). Then the analysis is extended to the neighbourhood of each word and the already-tagged words. The combination of words or categories can thus generate new categories.

Once all the words of the recognized utterance have been uniquely labeled, the understanding module applies a set of expert-defined rules to assign the dialogue concepts (attributes and values) to the sentence. We have defined 16 different concepts, which are extracted by applying a specific subset of rules.

2.2 Dialogue Manager

The dialogue manager that we have developed for I^2_{PUBS} is a finite state machine (FSM). Depending on the current state and using the information the user provides (which the understanding module has extracted), the system takes a decision about what action the user wants to fulfill. The system tries to perform this action, using also the information stored in a context memory, providing all this information is enough for fulfilling the required action. Otherwise, the system will ask the user for the information it needs.

As the architecture of the system is based on a FSM, most of the initiative belongs to the system, as in I_{PUBS}. However, our system allows the user a certain degree of control of the dialogue. For instance, the user does not need to follow each step of the dialogue. If the users have enough knowledge about the system, they can avoid different states, making the dialogue more flexible. Furthermore, the users have the chance to stop the current dialogue if they perceive some wrong behaviour of the system. The state machine of the dialogue manager is shown in Fig. 1.

The full process of the dialogue management can be viewed as the following algorithm:

```
state := BEGIN;
synthesize(greeting);
while (state != END) do
   if (recognition_expected) do
      recognize();
      understand(output_recognition);
   end if;
   dialogue_management(output_understanding);
   synthesize(output);
end while;
```

At the beginning of the interaction, the system synthesizes a welcome and sets the state to BEGINNING. If the user requests some information, the system returns the number of devices and the number of patterns learnt for each device. Now the user can ask for a specific pattern or for a set of them. Depending on that, the system will update the dialogue memory with the patterns the user has requested. In any case, the next state of the dialogue is ACTION_PATTERN.

For each pattern the user asks for, the system synthesizes all the information related to it (that is, its event, condition and action). The user can then perform an action over the pattern (i.e. accept, reject or refine it). Accepting a pattern means that the user finds it useful, so it is stored for using it proactively. A rejection implies that the pattern is not useful, so it is deleted. If the user wants to refine a pattern it means that the pattern is useful for the user, but it need to be tuned according to the user's desires.

In the two first cases, the system updates the state of the pattern, confirming the accepted patterns and deleting the rejected ones. As the process for the current pattern has finished, in the next state (CHECK_PATTERNS) the DM will check whether there are more patterns to be shown to the user.

When the user wants to modify a pattern, the system (in the state MODIFY_PATTERN) asks the user which of the three pattern fields he/she wants to modify. The user can ask for a single field, or for any combination of them (i.e. the event and the condition, or even the three fields at the same time).

Each field has a specific main state (MOD-

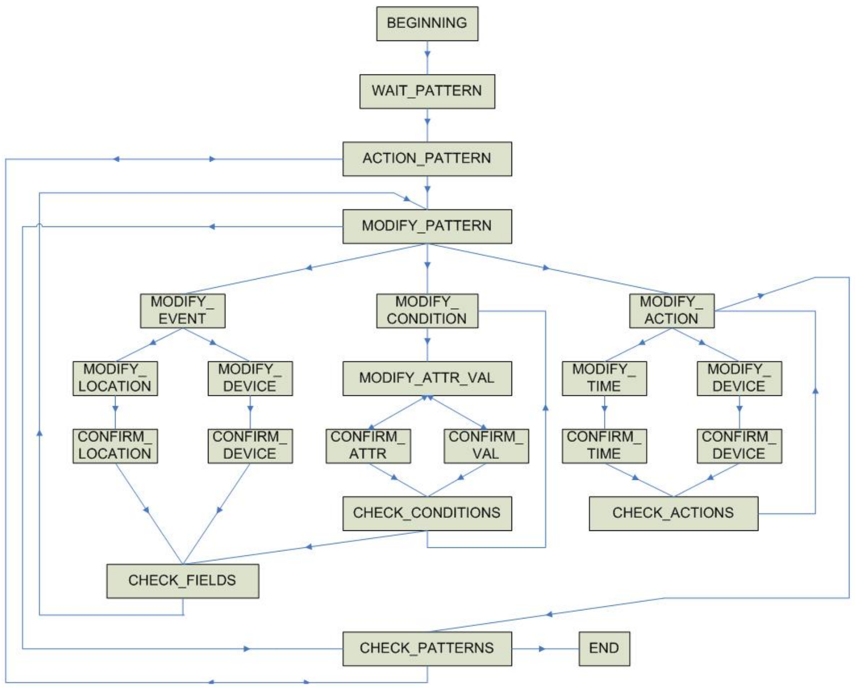

Figure 1: State machine of the Dialogue Manager.

IFY_EVENT, MODIFY_CONDITION, and MODIFY_ACTION), which controls the different elements that form the field. If the user wants to modify the event, he/she can modify the device which triggers the event, or the location in which he/she has to be detected. If the user decides to refine the condition, he/she can act over the attribute of the condition, or the value this attribute takes. For modifying the action, the user can select the device to act over, the action to perform (to switch on/off the selected device), or the moment in which the action will take place.

For each element of each field, once the user has modified the desired setting, the system starts a confirmation step, providing spoken feedback to the user to ensure that the recognition and understanding steps worked properly. If the user confirms his/her selection, the system will update the state of the pattern. Furthermore, the dialogue manager will check (at the dialogue memory) if any field of the current pattern remains to be modified. In this case, the system will repeat the previous steps for the new field (in the state CHECK_FIELDS). Otherwise, the procedure for the pattern is finished. The system sets CHECK_PATTERNS as new state, and checks whether the user asked for more patterns, in which case it has to start again the dialogue steps for the new pattern. When the user has evaluated all the desired patterns, the system moves to the END state and says goodbye to the user, finishing thus the interaction.

If the user detects some wrong behaviour of the system (a misrecognition of his/her utterance, or an incorrect understanding, which leads to a non-expected state), he/she can abort the current dialogue by asking the system to cancel the spoken interaction. In each state, if the user wants to cancel the dialogue, the system interrupts the normal dialogue flow by directly reaching the END state of the FSM. In this situation, the system informs the user about what the problem has been, and finishes the interaction.

The system has a context memory for storing the information related to every dialogue turn, as well as the current dialogue state. Using this information, the system can decide which action to take and which state to reach. The dialogue memory acts thus as a short-term memory, (i.e. it only takes into account the information relevant for the pattern presented to the user). Therefore, the information stored in the context memory includes the device the pattern belongs to, the index of the pattern (provided that there is more than one pattern learnt for this device), and the current values of the three fields of the pattern (event, condition and action). The system can easily check which of these fields the user has already modified, and which ones remain to be modified.

Table 1: Initial dialogue turns.

Dialogue turn	Next state	Context memory
S. Hello, welcome to the feedback system. How can I help you?	BEGINNING	∅
U. How many patterns have you learnt?		
S. There are three devices. Device Lamp A has five different patterns. Device Lamp B has two different patterns. Device Lamp C has two different patterns. Which pattern of which device do you want to listen to?	WAITING_PATTERN	
U. I want to listen to every pattern of Lamp A.		
S. Pattern Lamp A. user is in V A Condition 0 time is earlier than seven a m twelve then turn on Lamp A What do you want to do with this pattern?	ACTION_PATTERN	device: Lamp A pattern: 0
U. Accept it.	REMAINING _PATTERNS	
S. Pattern Lamp A. (...)	ACTION_PATTERN	device: Lamp A pattern: 1

3 INTERACTING WITH THE SYSTEM

The examples we present next show different dialogue turns of both the user (*U.*) and the system (*S.*), as well as the next state of the dialogue manager, and the information stored in the context memory.

Table 1 shows the beginning of the interaction with the system. Once the system synthesizes the greeting message, the user requests some information. This turn makes the system wait for information about a specific pattern, or a set of patterns.

The user can ask not only for a single pattern, but also for every pattern of a device, or even for every pattern of all the devices. The context memory will store the pattern presented and its associated device.

Once a pattern has been presented to the user, he/she has to decide what to do with it. If the user accepts or rejects it, the system confirms or deletes the pattern. Then the system checks if there are more patterns to show to the user. In this case, it updates the context memory to present the next pattern.

Table 2 shows a situation in which the user wants to refine a pattern. He/she can select any field of the pattern, or a combination of them. In any case, the system updates the context memory.

At any time, the system tries to offer the user the information he/she would need to fulfill the current action. This way, when the user wants to modify a device, the system shows a list of the available devices, avoiding an additional dialogue turn in which the user probably asks the system about the available devices. We can thus complete the dialogue with fewer turns.

4 CONCLUSIONS

We propose an enhanced speech interface to allow user feedback in a learning system by means of a SDS. Our new approach not only deals with isolated commands, but also allows speakers to use more natural linguistic constructions, achieving more dynamic and user-friendly human-machine dialogues.

The inclusion of a context database which stores the information regarding previous dialogue turns enables the system to tackle incomplete utterances. This way the naturalness of the dialogues is increased.

Currently we are designing a framework to evaluate the performance of I^2_{PUBS}. Our aim is to develop a set of feasible scenarios in which a user interacts with the system fulfilling different actions regarding the patterns learnt (selecting and asking for different patterns; accept, refine, or delete them, and so on). We can measure the goodness of our approach by analyzing both objective and subjective metrics. Among the objective ones, the most interesting metrics are the recognition rates and the concept confidence, together with several dialogue variables: the number of dialogue turns the system needs to fulfill a specific action, or the number of turns in which the system needs to ask the context memory for previous information. The subjective metrics can be estimated by means of a questionnaire in which we ask for different aspects of the system, such as how natural the dialogue is per-

Table 2: Example of pattern modification.

Dialogue turn	Next state	Context memory
S. Pattern Lamp A. (...)	ACTION_PATTERN	device: Lamp A
What do you want to do with this pattern?		pattern: 0
U. Modify it.		
S. What do you want to modify in this pattern?	MODIFY_PATTERN	
U. Could I modify the event and the action?		
S. User location was V A.	MODIFY_EVENT	field: event
Which location do you want to be the new one?		
Possible user locations are V A, V Bedroom.	MODIFY_LOC	
U. Please set V Bedroom as new user location.		
S. Do you want to confirm V Bedroom as new user location?	CONFIRM_LOC	
U. Yes.		
S. The action carried out was turn on the Lamp A.	CHECK_FIELDS	
What do you want to modify in the current action?	MODIFY_ACTION	field: action
You can modify the device and the action.		
U. I would like to modify the device and the action.		
S. Possible devices are Lamp A, Lamp B, Lamp C.	MODIFY_ACTION	
Which device and action do you want to set?		
U. Switch on Lamp B.		
S. Do you want to confirm On as new action	CONFIRM_ACTION	
and Lamp B as new device?		
U. Yes.		
S. Thank you for your collaboration.	END	$\emptyset$

ceived, the ease of interaction with the system, or the advantages over the original approach.

Using user-related information (i.e. knowledge about the identity of the user who has caused the learning of each pattern) would be a highly attractive task. This information would allow the system to show each user only those patterns related to him/her. We could also modify on-line the behaviour of the dialogue system, adapting it to each user's preferences.

Neither I_{PUBS} nor I^2_{PUBS} can know beforehand the devices in the environment, so that grammars for the recognizer could be dynamically created and loaded. A simple way of doing that consists of mixing a static grammar and different dynamic ones (Lucas-Cuesta et al., 2009). The flexibility of Sphinx-4 will ensure a good performance of the recognized sentences as long as the vocabulary matches the dynamic grammar.

REFERENCES

Aghajan, H., Delgado, R. L.-C., and Augusto, J. (2009). *Human-Centric Interfaces for Ambient Intelligence*. Academic Press, Elsevier.

Augusto, J. C. (2007). Ambient Intelligence: The Confluence of Pervasive Computing and Artificial Intelligence. In Schuster, A., editor, *Intelligent Computer Everywhere*, pages 213–234. Springer Verlag.

Aztiria, A., Augusto, J. C., and Izaguirre, A. (2008). Autonomous Learning of User's Preferences Improved through User Feedback. In Gottfried, B. and Aghajan, H. K., editors, *BMI*, volume 396 of *CEUR Workshop Proceedings*, pages 72–86. CEUR-WS.org.

Begg, R. and Hassan, R. (2006). Artificial Neural Networks in Smart Homes. *Designing Smart Homes: The Role of Artificial Intelligence*, pages 146–164.

Cook, D. J., Augusto, J. C., and Jakkula, V. R. (2009). Ambient Intelligence: applications in society and opportunities for AI. *Pervasive and Mobile Computing*, 5:277–298.

Fernández, F., Ferreiros, J., Sama, V., Montero, J. M., Segundo, R. S., and Macías-Guarasa, J. (2005). Speech interface for controlling a Hi-Fi audio system based on a Bayesian Belief Networks approach for dialog modeling. In *Proc. INTERSPEECH, Lisbon, Portugal, September 2005*, pages 3421–3424.

Hagras, H., Callaghan, V., Golley, M., Clarke, G., and Pounds-Cornish, A. (2004). Creating an ambient-intelligence environment using embedded agents. *IEEE Intelligent Systems*, 19(6):12–20.

Lucas-Cuesta, J. M., Fernández, F., and Ferreiros, J. (2009). Using Dialogue-Based Dynamic Language Models for Improving Speech Recognition. In *Proceedings of the 10th Annual Conference of the International Speech Communication Association (Interspeech), Brighton, UK*, pages 2471–2474.

McTear, M. (2004). *Spoken Dialogue Technology: Toward the Conversational user Interface*. Springer Verlag.

Müller, M. E. (2004). Can user models be learned at all? Inherent problems in machine learning for user modeling. *The Knowledge Engineering Review*, pages 61–88.

Sadeh, N., Gandom, F., and Kwon, O. (2005). Ambient Intelligence: The MyCampus experience. Technical report, CMU-ISRI-05-123.

Walker, W., Lamere, P., and Kwok, P. (2002). FreeTTS - A Performance Case Study. Technical report, Sun Microsystems Laboratories (TR-2004-139).

Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., Wolf, P., and Woelfel, J. (2004). Sphinx-4: A Flexible Open Source Framework for Speech Recognition. Technical report, Sun Microsystems Laboratories (TR-2002-114).

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3):94–104.