A Multitasking Approach to Adaptive Spoken Dialogue Management

Tobias Heinroth, Dan Denich, and Wolfgang Minker

Institute of Information Technology
Ulm University, 89081 Ulm, Germany
{tobias.heinroth,dan.denich,wolfgang.minker}@uni-ulm.de

Abstract. Undoubtedly one of the key factors of a computed world, are the interfaces users ought to use. In this paper we present the adaptive spoken dialogue manager OwlSpeak to provide a spoken interface to a computed world, in our case to an Intelligent Environment. The most important feature of the dialogue manager is its ability to pause, resume, and to switch between more than one interactive task, which is a prerequisite to provide adaptive spoken dialogues. Especially within Intelligent Environments it is necessary to modify the status of an interface depending on the changing contexts of the environment and on the actual requirements the user may have. We present the implementation and evaluation of OwlSpeak as part of an existing Intelligent Environment that can be used by real subjects and show how multitasking can be utilised to cope with an adaptive speech interface.

Keywords: HCI, Intelligent Environments.

1 Introduction

Nowadays users have to deal with a variety of modalities that allow interaction with a computer. Computers and similar devices provide – besides keyboards and mice – visual, tactual, gesture-, and of course speech-based interfaces, to name but a few. The last-mentioned modality undoubtedly is the most natural communication medium for humans. The vision of talking naturally to a computer is still not realised. However, in several domains such as in-car tasks or telephony-based services Spoken Dialogue Systems (SDS) are getting more and more common. There are various requirements for SDS depending on the desired way of spoken interaction and on the handling of commands and controls the system provides. When using a mobile phone, for example, free text inputs such as negotiating or discussing are usually not necessary: a phone should merely understand commands like “Call Peter in the office!”. However even such commands are not as simple as they seem to be. For example, the mobile phone shouldn't start calling Peter if someone says “… you might call Peter in the office and ask him…” during a conversation, for example. Even so such examples are apparently artificial; the necessity of an SDS used as interface for a mobile phone is definitely a matter under discussion. Its primary functionality – establishing calls and dialling – can usually be performed by using a keypad and a display. In general for most SDS scenarios a graphical interface could also be utilised.

C. Stephanidis (Ed.): Universal Access in HCI, Part IV, HCII 2011, LNCS 6768, pp. 42 – 51, 2011.

However, besides the issue of necessity in general there are several arguments that make the case for SDS: one of their main benefits is their advance in efficiency within many scenarios (especially related to Intelligent Environments). If a user has to control 12 lights, for example, a graphical interface would intuitively consist of 12 buttons. If the same task has to be handled by an SDS fewer commands would be necessary. The user would just ask the system to "switch on the light" and would not have to cope with a graphical control panel. Furthermore, in case of ambiguities the SDS could check back which light the user wants to switch on. Within the scientific area of Intelligent Environments (IEs) SDS technologies thus result in one of the most efficient and natural interfaces between humans and computer-based systems. In this context we refer to IEs as networks of various different components such as sensors, actuators, and processors that are able to automatically exchange information about themselves and their surrounding without any human intervention. Thus a user can provide input to *component A* that analyses the request and provides the new information to *component B* that might process the request and/or route the information to a further *component C*. Within the context of IEs, proactive behaviour (warning, information, etc.), and negotiative dialogues speech is a promising modality, for many tasks such as the mentioned command-and-control of devices or services. In particular for elderly people, for disabled persons or people with serious diseases who cannot stand up without further ado SDSs are extremely useful since such a system can provide a centralised and at once natural interface that can easily be accessed.

The remainder of this paper is structured as follows: The next section provides an overview on the scientific field of SDS and of the related work. Section 2 presents some use cases and their related requirements with respect to multitasking by speech. In Section 3 we provide details on the implemented prototype and show how multitasking over several disjoint dialogue domains can be realised. Section 5 shows the results of the evaluation of the proposed approach. The paper concludes and provides some future work in Section 6.

2 Related Work

An SDS is a computer-based system that enables a user to bilaterally communicate with a machine via spoken language. The three most important layers of an SDS are the acoustic front-end, the semantic layer, and the logical layer, which is constituted by the Spoken Dialogue Manager (SDM). A speech recognition module and a speech synthesis module constitute the acoustic front-end that is usually accessed by the user via microphone(s) and speaker(s).

One of the challenges regarding the realisation of real-life SDSs is their complexity. How much information must be taken into account in order to allow for a meaningful dialogue between the user and the system? How should this information be modelled in order to be both computer-readable and on the other hand easily assimilable into a probably on-going dialogue? Obviously the SDS and most notably the SDM must cope, depending on the scenario and the task to be tackled, with understanding and interpreting a maximum amount of information whilst keeping the complexity moderate. In order to realise adaptive and therefore intelligent spoken dialogue it is not only necessary to provide advanced voice recognition and speech

synthesis but also to incorporate an adaptive SDM residing at the core of such an SDS. Nowadays one of the most widespread technologies to implement SDMs is the W3C standardized VoiceXML description language. The idea behind this approach is to simplify the development of dialogues by providing a model description of the dialogue to be expressed. However, VoiceXML is not able to persistently store and therefore describe a specific state of a dialogue. Thus dialogue strategies that need functionalities such as pausing and resuming of parallel tasks can only be implemented difficultly.

Within the scientific field of IEs several international research projects have been concerned with spoken dialogue interaction for quite a long time [7, 3]. Three directions can be recognized that have been discussed in the recent past: heavyweight rule-based frameworks such as the TrindiKit [5], agent-based systems such as RavenClaw [1] and statistical approaches such as the Bayes Net Prototype implemented within the TALK Project [10]. The two former approaches require strong assumptions regarding the set-up and adjustment. In RavenClaw a dedicated agent would be needed for each task. This agent has to match the assumptions of the language recognition and logic-processing components of RavenClaw, hence it is very difficult to create agents that can cooperate with various other agents. In TrindiKit the dialogue flow is indirectly described by a rule-base. These rules have to be kept in a coherent state – if a new rule affects older ones the complete rule set must be updated. Once the rule-base is implemented the system performs well but the more complex the dialogue, the more complex the rule-base. All statistical approaches rely on the availability of training data, which appears to be a significant disadvantage as it is costly to collect corpora and to train the statistical models. Furthermore all these approaches are domain-dependent and it does not seem to be trivial to introduce new dialogues that may handle new domains.

3 Use Cases and Requirements

In the following we present a short example to describe our approach. We assume that in a fictive world where the user lives together with his IE a typical situation is the arrival at home. Besides a “greeting” task there are several other interactive tasks running in parallel – each providing the possibility of spoken interaction. Since one of the main duties an IE should handle is the control of tasks, i.e., the system should provide possibilities to facilitate the user's access to various functionalities, it is necessary to provide a (probably varying) set of spoken commands the system can interpret and execute. An example for such behaviour could be a user, after entering the apartment, telling the system to switch the lights on. Fig. 1 shows a set of three interactive tasks that may form an exemplary interactive situation.

Since the SDM adapts to the context it needs to be able to receive triggers from the outside world to change its state. The initial phase therefore is triggered by a “user enters room” event. This event might happen only once a day and/or when the user has left the room for a specified period depending on the configuration of the IE. In our example the SDM is set up to wait until the user greets the system (Task 1). It further activates a control task that listens to possible lighting control commands the user may utter (Task 3). Initial system studies in the iSpace at the University of Essex

revealed that the subjects preferred the SDS to be as unobtrusive as possible [4]. Thus we have designed the system to behave rather passively and not to proactively initialise a conversion if this can be avoided. By default, a control task such as Task 3 waits for user input and therefore behaves passively. However if the user initialises talking to the system by uttering a spoken command the system could take this opportunity to start dialogues that otherwise would have to be proactively initiated.

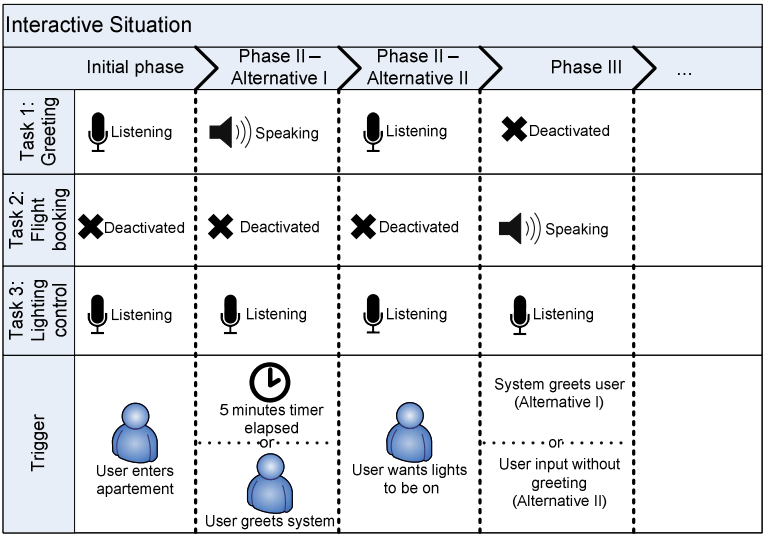

Fig. 1. An interactive situation that may occur with two alternatives

Fig. 1 presents two alternatives showing how the situation could proceed in Phase II: Alternative I contains two triggers that might allow the system to perform Task 1; the five-minutes-timer elapsed since the user entered the room or – probably the more usual case – the user greets the system. As mentioned above the reason for such a five-minutes-timer is that the system should act as unobtrusively as possible. Note that Task 3 is still active since the system is meant for handling more than one interactive task in parallel. If one of the two triggers is actuated the system would greet the user and add a semantic value such as "userInitiatedConversation" to the knowledge base (see Section 4). This would allow Phase III to start. Table 1 shows a possible conversation that might occur when making use of the proposed set of dialogues.

Alternatively Phase II could pass off conditioned by the user telling the system to switch the lights on. This would make Task I obsolete – the system shouldn't greet the user in response to such a spoken command. It would be more natural if the system skips the greeting task and activates the proactive Task 2 "Flight booking" instead. Fig. 1 shows Phase III constituted by the additionally activated Task 2 and the still running Task 3. The preceding greeting task has either become obsolete or has already been processed. Since the user (or the IE) can dynamically activate or deactivate the tasks the SDM may perform, it is possible at any time to end a conversation with the system or to start a dialogue the user respectively the system has not been aware of.

Table 1. A dialogue snipped that might occur during an interactive situation

Speaker	Utterance
Suki	Hello Julia!
Julia	Hi Suki!
Suki	Switch the lights on!
Julia	Do you want to start booking the flight now?

The example above describes a typical situation that may consist of more than one task in parallel. Obviously multitasking is a main source of and a main reason for adapting spoken dialogues in general. In [2] we have introduced three classes of adaptation: Device Adaptation, Event Adaptation, and Task Adaptation. In this paper we focus on two classes that influence the proposed multitasking approach. Herewith the first important class is *Device Adaptation.* Environmental changes may vary, depending on the surrounding and the situation of the user (kitchen, living room, car, etc.) and the availability of devices and services. This requires the capability to continuously change grammars, utterances, and system commands to a changing device population and changing user focus within the IE.

Event Adaptation also relates to environmental changes. Since various tasks within IEs are to be accomplished it is necessary to move the actual focus of an on-going dialogue to other (contingently more urgent) dialogues. These may consist of informative system utterances, alerts, or short yes-no-questions. Afterwards the on-going dialogue would have to be resumed. We have recognised two types of events that require adaptation: external events and internal events. While the former ones always need an entity that throws the specific event, the latter ones may be initiated by the dialogue manager itself. Reasons for initiating an internal event can be various and sundry: fixed priorities, dynamic priorities (i.e., changing over time), semantics, and depending on the progress (positive or negative) made within an on-going dialogue.

In the next section we will detail how the modular architecture of the prototype and the idea of a unified knowledge base describing the spoken dialogue and its state provide a fertile ground to realise adaptive behaviour in the described manner. One focus of the conducted research has been set on Event Adaptation and its two-sided mode of execution in practice. As mentioned above there are two kinds of events the SDM has to react on: external and internal events. The simplest case would be an external entity sending an event (e.g., an alert message) to the SDM, which immediately reacts and therefore interrupts any on-going dialogue in order to utter the alert message. However such behaviour is not always comfortable and comprehensible for the user. Thus we propose to not only react on external events but to incorporate also internal events. In case the alert message mentioned above is not time-critical it should not directly interrupt an on-going dialogue but should be suppressed until an internal event triggers the system to utter to message. For example, a trigger the system could use is the time that elapses since the external event occurred. If a specific threshold is reached (i.e., the alert gets more urgent during time) there is either no further need to suppress the message since the on-going dialogue has already been terminated or finally there is no other way than to interrupt the on-going dialogue.

4 The OwlSpeak Prototype

In order to meet the main requirements for multitasking mentioned in the previous section we have implemented the proposed Spoken Dialogue Manager OwlSpeak[1]. The Passive View variation of the Model-View-Presenter (MVP) pattern [8] has been used. This architectural approach allows for a maximum of flexibility regarding the switching between independent or interrelated tasks. The underlying idea of MVP is that an application or system should be divided into three logical parts, the Model, the View and the Presenter. The user only interacts with the View layer. Contrary to MVC the Presenter mediates between Model and View – the Model conveys no functionality, i.e., it is not an application but solely encodes the knowledge that is used by the Presenter. The term Model in this case refers to a *Domain Model*. Therefore especially for multitasking systems that provide a direct interface to the user (i.e., user interfaces) MVP is perfectly suitable.

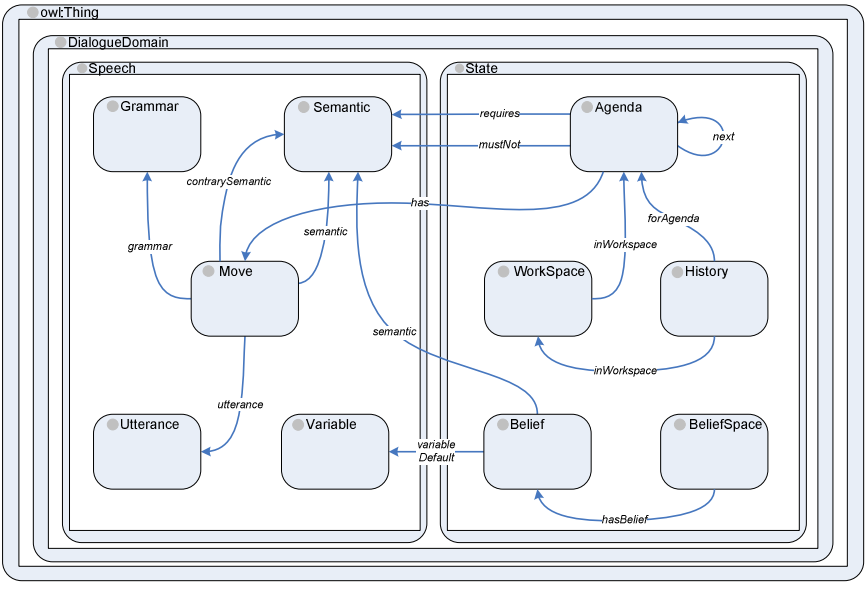

Fig. 2. Overview of the classes and main relations of the Spoken Dialogue Ontology

To be able to communicate with a user or with other external entities the application needs a knowledgebase that describes facts and the relation between such facts. A fact could, for example, be the name of a person or an ID number. A relation could be "has", which could be used for person "A" has ID number "4711". Without such knowledge a system would not be able to generate useful output, i.e., act as *knowledge source* nor to understand input that is provided by external entities, thus acting as *knowledge sink*. The term Domain Model could therefore be specified as the knowledge a system needs in order to be able to interact with the context and the user

[1] http://sourceforge.net/projects/owlspeak/

in a meaningful way. There are many ways to establish such a knowledgebase; SQL databases or XML files could be utilised, to name but a few. A more sophisticated option is to make use of ontologies to provide a common understandable knowledge base [6].

The underlying knowledge base of OwlSpeak is modelled using OWL ontologies, so called Spoken Dialogue Ontologies (SDOs). We have implemented a tree shaped structure to arrange the data-bearing individuals using a defined set of classes. The root of each knowledge base is DialogueDomain, which has the two subclasses Speech and State. We divide the ontology into these two main branches since we want to distinguish between knowledge that corresponds to the static structure and knowledge that corresponds to the dynamic state of the actual dialogue. Fig. 2 shows an overview of all classes populating the SDO together with the relations interlinking them. OwlSpeak makes use of a specific number of dialogue representations. These representations serve as Domain Models. Each representation provides knowledge about both dialogue flow and state of a specific spoken conversation. Depending on contextual information various sets of SDOs can be activated or deactivated. It is furthermore possible to add new representations for dialogues during runtime and therefore extend the knowledgebase, i.e., the Model.

5 Evaluation

The main question of the evaluation was *how do users cope with the multitasking capabilities of the prototype*? A challenge herewith was that it is exceedingly difficult to evaluate an SDM without evaluating the SDS that provides ASR and TTS, which certainly are both strongly perceived by the user. To solve this issue we divided the subjects into two groups each of them conducting the same spoken dialogue but using different multitasking strategies. All in all 26 mixed-gender subjects in the age between 17 and 59 years participated in the evaluation. The first group (Group A) conducted the main dialogue, a travel booking task, and received several reminders afterwards. The results of this group were used as baseline. The second Group (Group B) received the reminders dynamically during the on-going dialogue and therefore has been engaged in a more comprehensive multitasking conversation. The main issue of the evaluation setup was how to measure which of the approaches performs better and what does "better" mean in this context. Established SDS evaluation approaches such as PARADISE [9] utilise metrics such as "task completion", "repetition rate", and "error rate", which are only partly useful for rating the SDM itself. We have decided that a proper way of evaluating the multitasking capabilities of the prototype is to find out if it is easier for Group A or for Group B to recognize the reminders and to remind them after the dialogue has been concluded. During the evaluation the subjects had to imagine that they are talking to their IE, which is able to correspond with the travel agency in order to book a flight and a hotel for the next holiday. They furthermore received the following reminders:

- "Your friend Oliver has his birthday on the 15th of August. You might buy him a present."
- "I should remind you to rent a movie for tonight."
- "According to the weather report there will be heavy rain today."

All participants concluded the dialogues successfully. The overall performance of the system was rated positively. Both groups rated the system nearly equally on a scale from 1 (very bad) to 10 (very good) as good (6,35) in average. However as mentioned above these numbers are not sufficient to measure the performance, usability or functionality of the SDM. The user perceives the whole SDS and therefore merely rates it. However these numbers cannot be left aside: If the subjects had rated the whole system as bad it wouldn't make any sense to look for results that help us rating the SDM. A bad SDS would inevitably lead to a bad rating of the included SDM. Thus in our case an SDS rated as good serves as a fertile ground to measure the performance of the SDM, which was the main topic of the experiment.

Table 2. The number of subjects per group who gave right answers per category

Information	A	B	A+B
Friend's name	0	0	0
Day of Birth	8	5	13
Rent movie	6	2	8
Buying present	9	3	12
Watching movie	6	2	8
Rainy weather	8	9	17

Since the subjects were not explicitly told to take care of any additional information but only to complete the “travel booking” task, it was expected that only a few test persons would retain all information the system provides. Since Group A received the three reminders after the main tasks have been concluded we furthermore expected that Group A would experience a slight advantage. To be able to rate the outcome of the questionnaire we counted a point for each information the specific subjects kept in mind. Thus a subject who didn't remind any of the additional information would gain 0 points and a test person who reminded all of the provided information would gain 6 points: for the right name of the friend, the right date, renting a movie, buying a present, watching a movie, and the rainy weather. Table 2 shows the number of subjects per group who gave the right answers to the specific questions. We assume that the friend's name – Oliver – was only poorly synthesised, thus no one was able to remind it. It is obvious that the static group outperforms the dynamic group in all but one category. The information about the weather came up at the end of both static and dynamic dialogue. The subjects of Group B had much more problems perceiving the information that was provided during the on-going main booking dialogues. A reason for this could be that the test persons were nearly totally occupied by managing the travel booking task. Since the system did not provide any assistance in switching to a “reminder” task the users blanked out all information that was not related to the main aim: concluding the booking.

During a free discussion that followed each experiment several participants stated that they had consciously ignored all information that didn’t directly relate to the main task. However if the subjects had been informed beforehand that they will have to answer specific questions about the dialogue history we would have gain a totally different result since it is far from practice that a user would not be confronted with such reminders spontaneously. A main reason for the overall bad result – only the

rainy weather was reminded by nearly all participants – could be that we had to use an English-language SDS. This could have been problematic for the German native speakers that participated in the experiment. However we expect that even with a German speaking system the discrepancy between the two groups approach would still be observable.

5 Conclusion

The current version of the OwlSpeak Spoken Dialogue Manager already fulfils several requirements that arise from the multiple task-based situations that occur within IEs. It is able to pause and resume active tasks, add and remove dialogue domains, permanently save the state of a dialogue, and furthermore it provides more than one active spoken dialogue in parallel.

Those functionalities allow for adaptive spoken dialogues that cover several disjoint or partly overlapping domains. However it seems to be obvious that it is a different task for the user to use the spoken interface to solve a *single* task (e.g., book a flight) with the help of an SDS or to use an SDS to interface with an IE that provides a variety of different controls and tasks. Thus the main question that has to be answered before multitasking SDSs will be used in everyday life is *how do users cope with the multitasking capabilities of a system such as the proposed prototype?*

Compared to GUIs users are already accustomed to their multitasking capabilities: techniques such as taskbars or widgets are totally adopted by the users. However when it comes to spoken interaction we are far away from such a wide spread user acceptance. Enhancing the common usage of SDSs by adding the functionality of multitasking could be an important step towards wider application of spoken interfaces. The evaluation revealed several questions that have to be answered before. How can a change of focus from one task to another be signalised by the system or by the user? How can the system distinguish between user inputs that might relate to different tasks? How can the user distinguish between system outputs that might relate to different tasks? Future work would be to find and evaluate technical solutions to answer these questions. We plan to compare different dialogue strategies that might be used to signalise a task change: for example, remarks before changes occurs, auditory icons, or more complex sub-dialogues may proof useful for the users. Furthermore we are currently implementing sophisticated functionalities that enable OwlSpeak to detect task changes that are initialised by the user.

Acknowledgment. The research leading to these results has received funding from the European Community's 7th Framework Programme (FP7/2007-2013) under grant agreement n° 216837.

References

[1] Bohus, D., Rudnicky, A.I.: The ravenclaw dialog management framework: Architecture and systems. Computer Speech & Language 23, 332–361 (2009)

[2] Heinroth, T., Denich, D., Schmitt, A.: Owlspeak - adaptive spoken dialogue within intelligent environments. In: 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp. 666–671 (March 2010)

[3] Heinroth, T., Kameas, A., Pruvost, G., Seremeti, L., Bellik, Y., Minker, W.: Human-Computer Interaction in Next Generation Ambient Intelligent Environments. Intelligent Decision Technologies 5(1) (2011)
[4] van Helvert, J., Hagras, H., Kameas, A.: D27 - prototype testing and validation. Tech. rep., The ATRACO Project (GA no 216837, 7th FP) (2009)
[5] Larsson, S., Traum, D.: Information state and dialogue management in the trindi dialogue move engine. Natural Language Engineering, 323–340 (2000)
[6] McGuinness, D.L., van Harmelen, F.: Owl web ontology language. W3C Recommendation (2004)
[7] Minker, W., López-Cózar, R., McTear, M.: The role of spoken language dialogue interaction in intelligent environments. Journal of Ambient Intelligence and Smart Environments 1(1), 31–36 (2009)
[8] Potel, M.: MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java. Tech. rep., Taligent Inc (1996)
[9] Walker, M.A., Litman, D.J., Kamm, C.A., Abella, A.: Paradise: a framework for evaluating spoken dialogue agents. In: Proceedings of the Eighth Conference on European Chapter of the Association for Computational Linguistics (1997)
[10] Young, S., Williams, J., Schatzmann, J., Stuttle, M., Weilhammer, K.: D4.3: Bayes net prototype - the hidden information state dialogue manager. Tech. rep., TALK - Talk and Look: Tools for Ambient Linguistic Knowledge, IST-507802, 6th FP (2006)