Efficient Spoken Dialogue Domain Representation and Interpretation

Tobias Heinroth, Dan Denich, Alexander Schmitt, Wolfgang Minker

Ulm University – Institute of Information Technology
Albert-Einstein-Allee 43, 89081 Ulm (Germany)
{tobias.heinroth, dan.denich, alexander.schmitt, wolfgang.minker}@uni-ulm.de

Abstract

We provide a detailed look on the functioning of the OwlSpeak Spoken Dialogue Manager, which is part of the EU-funded project ATRACO. OwlSpeak interprets Spoken Dialogue Ontologies and on this basis generates VoiceXML dialogue snippets. The dialogue snippets can be interpreted by all speech servers that provide VoiceXML support and therefore make the dialogue management independent from the hosting systems providing speech recognition and synthesis. Ontologies are used within the framework of our prototype to represent specific spoken dialogue domains that can dynamically be broadened or tightened during an ongoing dialogue. We provide an exemplary dialogue encoded as OWL model and explain how this model is interpreted by the dialogue manager. The combination of a unified model for dialogue domains and the strict model-view-controller architecture that underlies the dialogue manager lead to an efficient system that allows for a new way of spoken dialogue system development and can be used for further research on adaptive spoken dialogue strategies.

1. Introduction

Nowadays Spoken Dialogue Systems (SDS) see use in a variety of applications: customer relationship management in call centres, command-and-control of devices and services in cars, or phone-based booking services to name but a few. However, for a vast majority of the users it is still uncommon to *talk* to a computer-based system. Compared to established interfaces such as keyboard or mouse SDS still lead a shadowy existence. Undoubtedly a reason for this social antagonism is the high complexity of human language making SDS error-prone. To cope with this problem, specialised algorithms within the area of speech recognition and synthesis have been implemented and are subject to ongoing advancement. On the level of spoken dialogue management, which is a further key aspect of SDS, there are still no satisfying solutions available. On the one hand, heavyweight rule-based frameworks such as the TrindiKit (Larsson and Traum, 2000) require strong assumptions regarding the set-up and adjustment. On the other hand, statistical approaches such as the Bayes Net Prototype implemented within the TALK Project (Young et al., 2006) rely on the availability of training data for the spoken dialogue manager (SDM), which appears to be a big disadvantage.

One reason for the absence of a powerful and generic SDM is that there is no standard for spoken dialogue domain descriptions defined yet. The most widespread technology to implement SDS and to define the underlying SDM is the W3C standardized VoiceXML (Oshry et al., 2007) description language. In this approach the XML-based definition is used to describe the dialogue flow, the utterances, and the grammars. Since JavaScript can also be used within a VoiceXML document, conditions on external data or similar sources can be expressed. It is possible to implement command-and-control and, by extending the scope of a grammar, mixed-initiative dialogue structures. However, more complex structures such as negotiative or task-oriented dialogue flows within changing domains still wait for an appropriate approach. It seems to be necessary to combine the expressiveness of the scientific systems mentioned above and the domain-related adaptation requirements we especially have within Intelligent Environments (IE).

In a former publication we have presented the architecture and the benefits of our proposed SDM called OwlSpeak that provides the fundamentals of adaptive spoken dialogue management (Heinroth et al., 2010). In the work at hand we present the inherent mechanisms of the SDM and focus on the interplay of dialogue manager and underlying domain model encoded in OWL.

The remainder of this paper is structured as follows: The next section gives an overview on the framework of OwlSpeak and provides some related publications. In Section 3 we provide an exemplary instance of a Spoken Dialogue Ontology to show how a dialogue domain can be represented. Section 4 details the methods of OwlSpeak's computational part, the Controller in order to show how a dialogue domain can be interpreted. The paper concludes and provides some future work in Section 5.

2. Framework

OwlSpeak was implemented within the framework of the EU-funded project ATRACO[1]. The aim of ATRACO is to contribute to the realization of activity spheres based on trusted ambient ecologies, see (Kameas et al., 2009; Pruvost et al., 2009; Heinroth et al., 2009). An ambient ecology usually resides within an IE and consists of:

- Entities (users, agents)
- Devices and services
- Local ontologies

An activity sphere (see Figure 1) is both the semantically rich description of the resources required to achieve a user aim or an entity goal and its instantiation in the context of a specific ambient ecology; thus, multiple spheres, each corresponding to a separate aim or goal, can be instantiated concurrently, using the resources of the same ambient

[1] Adaptive and TRusted Ambient eCOlogies - `http://www.atraco.org`

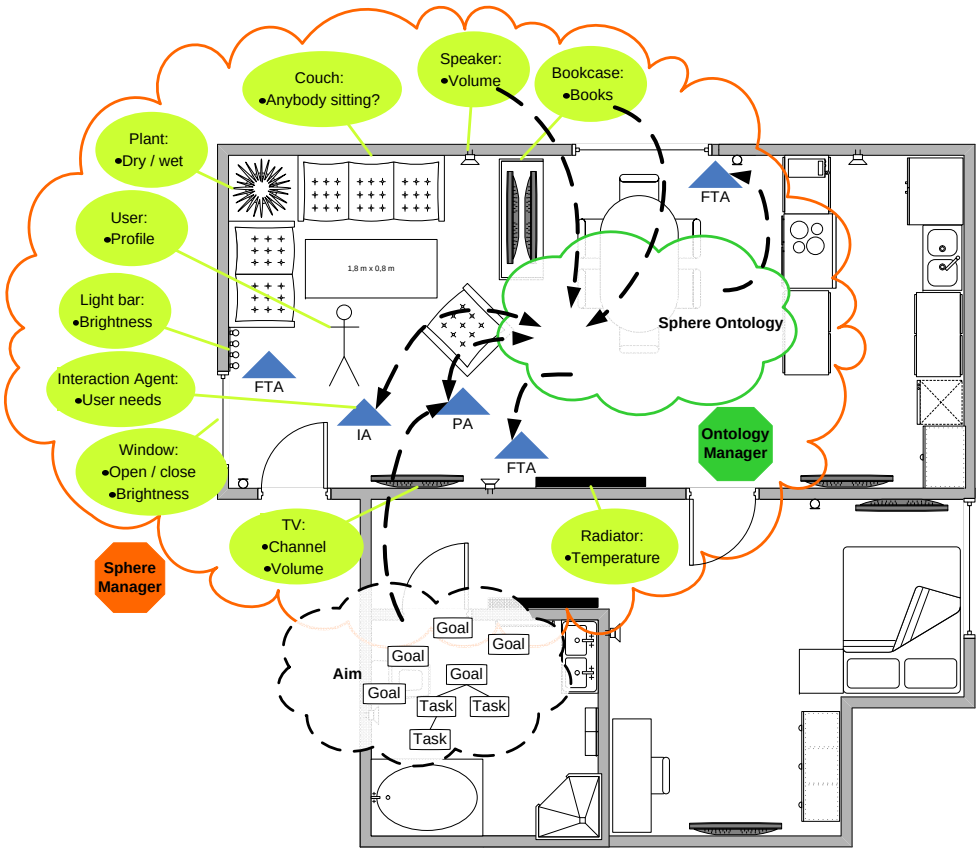

Figure 1: High-level view of one instance of ATRACO.

ecology at the same time. Each sphere is regarded as an autonomous instance of ATRACO and is supported by an independent ATRACO system; all spheres adopt the same ATRACO architecture. An activity sphere consists of:

- A description of an aim as a set of goals each modelled with an abstract task model
- Users / Devices / Services / IE, each having its local ontology
- Software Modules (sphere manager, ontology manager)
- Agents (Fuzzy Task Agent (FTA), Planning Agent (PA), Interaction Agent (IA)), each having its local ontology
- Policies (i.e. privacy, interaction, spoken dialogue, etc.)
- A sphere ontology

The sphere manager is responsible for creating, managing and dissolving spheres. The various agents are responsible for resolving conflicts, interacting with the user and in general realizing the concrete tasks in the task model. The sphere ontology is managed by the ontology manager. It results from merging the local ontologies of devices, services and IE required to achieve a specific goal and contains all necessary knowledge and information.

The ontology manager informs the various agents, when there is a change of state of the sphere ontology, so that the agents can directly take advantage of a homogeneous and always updated information pool. Agents, devices, and services autonomously maintain and update their local ontologies. Thus, the sphere ontology, being the result of aligning local ontologies, always reflects the most recent state of the sphere. The spoken dialogue manager OwlSpeak, which is detailed in Section 4 is part of the multimodal Interaction Agent and acts as the main component providing spoken interaction between user and ATRACO system.

The main concept underlying OwlSpeak is inspired by the Model-View-Controller paradigm. According to our approach VoiceXML is used as View; this presentation layer consists of a grammar to catch the next user utterance and/or provide a system prompt combined with a redirect link to access the next generated dialogue snippet. These VoiceXML snippets will be generated during the ongoing user-system conversation by the controller layer, which constitutes the computational part of OwlSpeak. The core of our approach constitutes a unified Model underlying the controller. It is represented by the so called Spoken Dialogue Ontology (see next Section) and also serves as the interface to the world outside the SDM, i.e., the ATRACO system.

3. Representation

Our approach is based on a unified model that represents a specific spoken dialogue domain: the Spoken Dialogue Ontology[2] (SDO) encoded in OWL. In the remainder of this paper we use the terms *concept* and *class* interchangeably. Figure 2 shows an SDO providing a model of a fictive dialogue that could run as shown in Table 1.

The basic concepts used within the SDO are *utterance*, to express what the system can say, *grammar*, to express what the user can say, and *semantic*, to express the meaning of

[2]Online available: `http://owlspeakonto.dyndns.info/OwlSpeakOnto.owl`

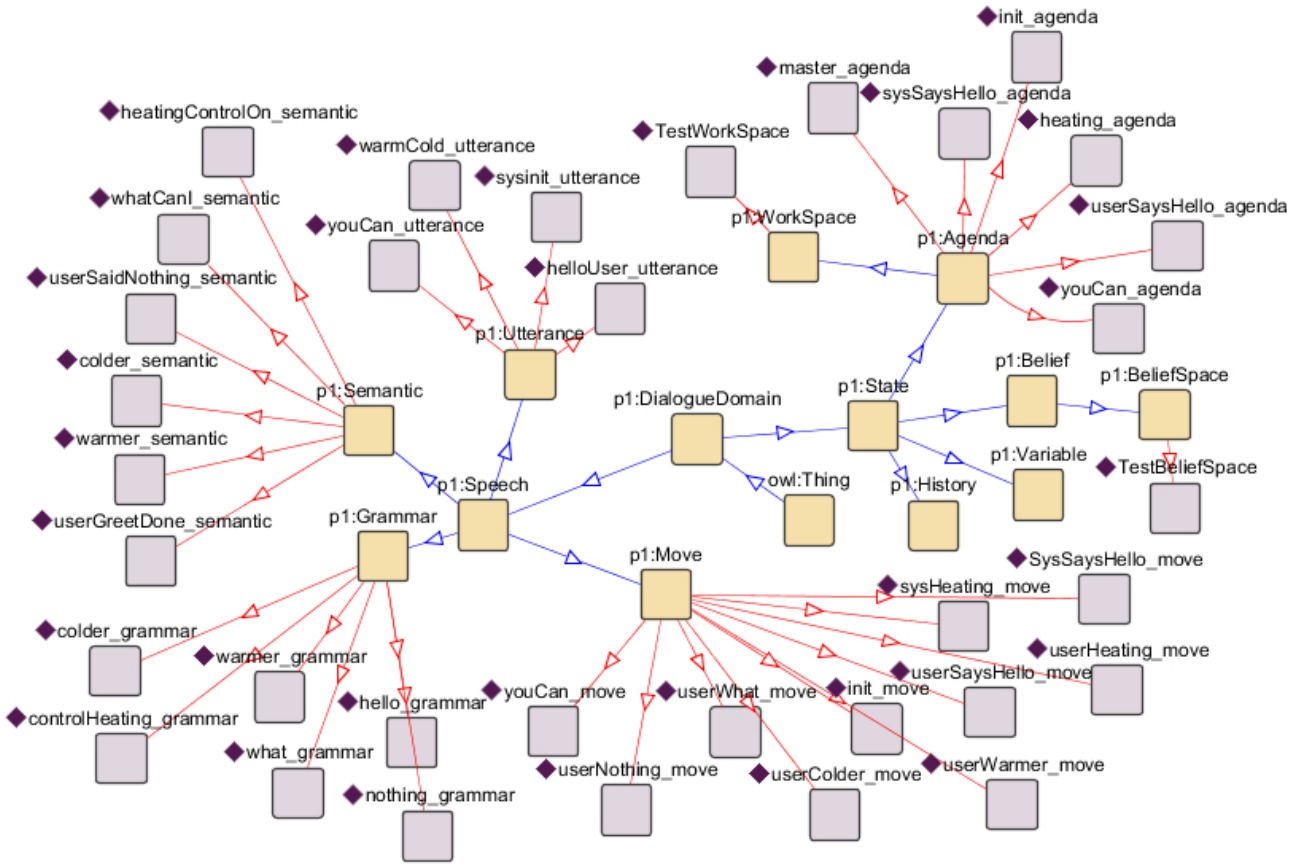

Figure 2: An exemplary dialogue domain modelled as a Spoken Dialogue Ontology.

what was said by either system or user. The *move* concept encapsulates these basic classes and usually represents pairings such as grammar plus semantic or grammar, utterance (i.e., confirmation), and semantic.

The two more general concepts are *agendas* and *moves*, which are strongly interrelated. The agenda concept is used to bundle several moves that belong to a specific dialogue turn. Furthermore it provides the links to all next possible agendas and lists (semantic) requirements that must be evaluated as true before the demanded agenda can be interpreted by the controller.

	Utterance	Semantic
U:	Hello.	userGreetingDone
S:	Hello user, what do you want to do?	-
U:	Control heating!	heatingControlOn
S:	Warmer or colder?	-
U:	Warmer!	heating(warmer)
...		

Table 1: An exemplary dialogue that is part of the presented SDO (*U* is user and *S* is system).

Therefore the agenda concept inherently requires the belief class for storing individuals of the class semantic, indicating that the respective semantic value is evaluated as true. This means that system and user agreed on a specific knowledge. As depicted in Figure 2 both the agenda and the belief classes have subclasses called *workspace* and *beliefspace*. They are used internally by the controller to temporarily store individuals that will be used during the ongoing dialogue. A designer creating an SDO wouldn't need to work on these classes. The History class is used to store all agendas that already have been processed. The variable class will be used for semantic values that provide information that might change during an ongoing dialogue such as counters or other mathematical expressions. However, this is a topic of future work.

A main benefit of the proposed SDO is that it can dynamically broaden or tighten its scope by modifying the individuals during the ongoing dialogue. Since the controller is able to work on a set of SDOs that can be enabled or deactivated while the system is running it is possible to interrupt a dialogue to utter something with a higher priority (such as a warning) and then proceed with the dialogue. Furthermore, because of the open nature of OWL ontologies, other components, namely ATRACO agents, may enrich the SDOs with new beliefs or add agendas that should be executed by the SDM.

4. Interpretation

In addition to defining the model layer by developing the structure of the SDO, the main effort is to implement sophisticated algorithms for interpreting the model and generating VoiceXML snippets. This and furthermore updating the SDO is the role of the controller. As shown in Figure 3 the controller has a dual hypostasis: implemented as a Java

Servlet instance it can be invoked to *work* on the SDO, i.e., to update its state or to *request* the actual dialogue turn to be carried out. A dialogue turn can either be a user or system turn or an exchange (see (ITU, 2005)).

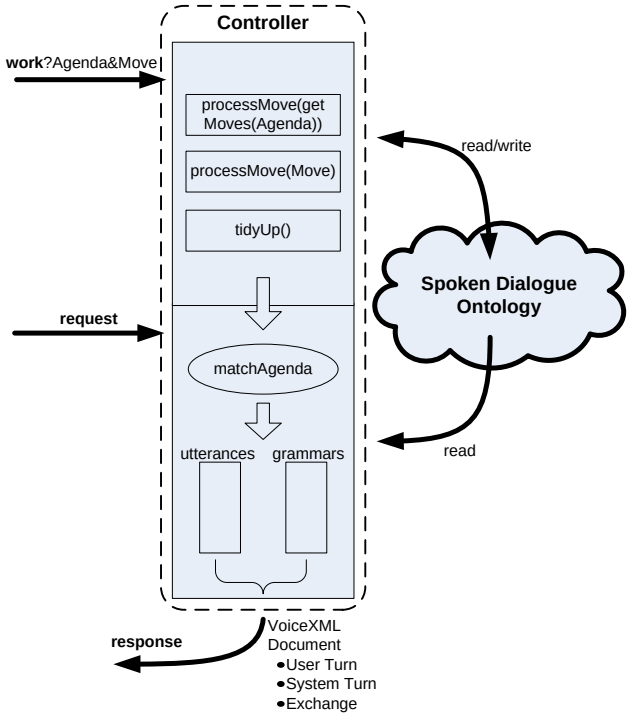

Figure 3: Detailed view on the Controller.

If the controller is invoked using the *request* command the *matchAgenda()* function (see Listing 1) derives a valid agenda that can be accessed (i.e., all of its requirements are evaluated as true). Taking this agenda into account two stacks are built: one for all utterances that should be used in the next turn and one for all grammars that are needed to understand what the user utters. The controller then generates a valid VoiceXML document that represents the next turn. Within this document the actual agenda is stored as a variable, since an expected *work* command will need that information as described in the next paragraph.

```
1  matchAgenda(Agenda){
2    if ((knowledgeIn(Beliefs)).containsAll(
         requiredKnowledge(Agenda))){
3      if(Agenda.hasMoves()){return Agenda;
4      }else{
5        nextAgenda = getChildren(Agenda);
6        delete(Agenda);
7        matchAgenda(nextAgenda);
8      }
9    }
10 }
```

Listing 1: The matchAgenda() function written as pseudo code.

If the controller is invoked using the *work* command, either one or two parameters can be transferred that the system will require: The actual agenda that should be processed and an optional move that was executed by the user. The controller verifies that the parameters are valid (i.e., are part of the SDO) and then starts to work. At first, it checks if the move (i.e., the recognized speech) is listed to be part of the actual agenda. This must be done since the system is able to understand global commands that do not directly belong to the ongoing dialogue as well.

If the user's input is not part of the actual agenda, the system will consider the dialogue as interrupted and will not process the actual agenda. However, it will process the global command and later on will resume the dialogue. If the move is listed within the current agenda, then the agenda should be considered to be processed.

To understand the processing functionality we have to focus on the concept of *beliefs*. Moves as part of an agenda can have a semantic value that provides the computational meaning of the respective move. For example, if the system asks "How do you do?", the user can answer "I'm fine" or "I'm sick". The semantic value of the former answer would be user(fine) and of the latter user(sick). Both values could be passed to a user profile ontology that is part of the ATRACO ecology and will be attached to the beliefs class, which is part of the SDO. The belief class is used to evaluate the requirements that are necessary to perform further agendas. For example, the user(sick) belief could enable the following question to be uttered by the system: "Do you want to arrange a doctor's appointment?".

This processing mentioned above takes the individuals of the belief class into account and can be split into the following functions:

- Store the semantic values of the agenda as *beliefs*.
- Remove semantics that are marked to be contrary to the agenda's semantics.
- Store all possible agendas that could be performed during the next turn
- Remove the executed agenda itself.

The system will then process the move that was executed by the user in a similar way the agenda was processed. Finally, the *work* command calls the function that implements the request command, since the system now can generate a new VoiceXML document on the basis of the updated SDO.

5. Conclusions

In this paper we have provided a focussed look on the functioning of the controller that performs the computational part of the OwlSpeak SDM. OwlSpeak is integrated within the ATRACO system and will be evaluated within the intelligent environment *iSpace* at the University of Essex. The developed SDO is used as model to represent a specific dialogue domain that can adaptively be modified during an ongoing dialogue. The controller is able to interpret a set of these models and generates VoiceXML dialogue snippets that are processed by an independent VoiceXML-based speech Server such as TellMe[3] or Voxeo Prophecy[4].

We estimate our approach to be efficient because we combine all information needed to perform a specific dialogue within one unified dialogue domain. Since all SDOs are of the same structure they may be treated as one virtually

[3] http://studio.tellme.com/
[4] http://www.voxeo.com/prophecy

aligned SDO and interpreted in a similar way only one single SDO would be treated. To resolve conflicts we aim at the implementation of a general "system SDO" that provides resolution dialogues such as "Which device should be activated?" if a grammar matches two different devices that can be accessed with similar grammars. The idea of a unified model for spoken dialogue representation is not only efficient during runtime but also facilitates the development, i.e., the design of spoken dialogues. Specialised tools are planned to be implemented for this purpose.

Acknowledgements

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 and from the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

6. References

Tobias Heinroth, Alexander Schmitt, and Gregor Bertrand. 2009. Enhancing speech dialogue technologies for ambient intelligent environments. In *5th International Conference on Intelligent Environments (IE09)*, volume 2 of *Ambient Intelligence and Smart Environments*, pages 42–49. IOS Press, July.

Tobias Heinroth, Dan Denich, and Alexander Schmitt. 2010. Owlspeak - adaptive spoken dialogue within intelligent environments. In *IEEE PerCom Workshop Proceedings*. presented as part of SmartE Workshop.

ITU. 2005. Parameters describing the interaction with spoken dialogue systems. ITU-T Recommendation Supplement 24 to P-Series, International Telecommunication Union, Geneva, Switzerland. Based on ITU-T Contr. COM 12-17 (2009).

Achilles Kameas, Christos Goumopoulos, Hani Hagras, Michael Gardner, Tobias Heinroth, Wolfgang Minker, Apostolos Meliones, Dimitrios Economou, Yacine Bellik, and Gaëtan Pruvost. 2009. A Pervasive System Architecture that supports Adaptation using Agents and Ontologies. In *The 10th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN)*, pages 148–153, Los Alamitos, CA, USA, December. IEEE Computer Society.

Staffan Larsson and David Traum. 2000. Information State and Dialogue Management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*, 6:323–340.

M. Oshry, R. Auburn, P. Baggia, M. Bodell, D. Burke, D. Burnett, E. Candell, J. Carter, S. Mcglashan, A. Lee, B. Porter, and K. Rehor. 2007. Voice extensible markup language (voicexml) version 2.1. Technical report, W3C Voice Browser Working Group, June.

Gaëtan Pruvost, Achilles Kameas, Tobias Heinroth, Lambrini Seremeti, and Wolfgang Minker. 2009. Combining agents and ontologies to support task-centred interoperability in Ambient Intelligent Environments. In *Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 55–60, Washington, DC, USA, November. IEEE Computer Society.

Steve Young, Jason Williams, Jost Schatzmann, Matt Stuttle, and Karl Weilhammer. 2006. D4.3: Bayes net prototype - the hidden information state dialogue manager. Technical report, TALK - Talk and Look: Tools for Ambient Linguistic Knowledge, IST-507802, 6th FP.