Monitoring and explaining reasoning processes in a dialogue system's input interpretation step

Daniel Sonntag and Christian Schulz

German Research Center for AI (DFKI),
Stuhlsatzenhausweg 3, 66123 Saarbruecken, Germany
sonntag@dfki.de
christian.schulz@dfki.de

Abstract. We implemented a generic speech-based dialogue shell that can be configured for and applied to domain-specific dialogue applications. A toolbox for ontology-based dialogue engineering provides a technical solution for the two challenges of engineering domain extensions for new question and answer possibilities and debugging functional modules. In this paper, we address the process of debugging and maintaining rule-based input interpretation modules. While supporting a rapid implementation cycle until the dialogue systems works robustly for a new domain (e.g., the dialogue-based retrieval of medical images), production rules for input interpretation have to be monitored, configured, and maintained. We implemented a special graphical user interface to monitor and explain reasoning processes for the input interpretation phase of multimodal dialogue systems. A particular challenge was the presentation of the software system's ontology-based interaction rules in a way that they were accessible to and editable for humans for maintenance, and, at the same time, allowed a real-time monitoring of their application in the running dialogue system.

1 Introduction

Dialogue systems combine many individual natural language processing components into a single, complex, artificial intelligence system. Because of the complexity, these systems cannot easily be constructed and maintained. Over the last several years, we therefore build up a new discourse and dialogue infrastructure to ease the task of dialogue construction for a specific domain thereby allowing for a rapid dialogue system engineering process. A dialogue runtime environment, the ontoogy-based dialogue platform (ODP) framework and its platform API (the DFKI spin-off company SemVox, see www.semvox.de, offers a commercial version), translates between the speech user input/output and conventional query/answer data structures in ontology-based representations (in the case of, e.g., a SPARQL backend repository) [16]. In order to create the needed interaction rules in ontological form for processing user requests, we implemented an integrated toolbox which builds upon the industry standard Eclipse and also integrates other established open source software development tools to support

dialogue application development, automated testing, and interactive debugging. The main challenges we encountered in supporting a rapid dialogue system engineering process, i.e., implementing a new dialogue for a new domain, can be summarised as follows:

- Engineering ontological domain extensions;
- Debugging functional modules such as natural language understanding (NLU), input fusion, dialogue management, and external text-to-speech (TTS) synthesis.

In this paper, we discuss the input fusion step in particular and show how the production rules that are used for input interpretation can be monitored, configured, and maintained. We basically provide two new contributions. First, we discuss a graphical user interface to monitor and explain reasoning processes for the input interpretation phase of multimodal dialogue systems. Second, we show the usefulness of the interface by a demonstration of its usage in a particular real-world industrial application example, the dialogue-based access to medical image data.

2 Background and Related Work

We use a distributed, ontology-based dialogue system architecture, where every major component can be run on a different host, increasing the scalability of the overall system. In earlier projects [20,10] we integrated different sub-components to multimodal interaction systems which we then extended to the consistent usage of ontology data structures [5]. Thereby, the dialogue system also acts as the middleware between the clients and the backend services that hide complexity from the user by presenting aggregated ontological data. Prominent examples of integration platforms include OOA [7], TRIPS [1], and Galaxy Communicator [12]; the W3C consortium also proposes inter-module communication standards like the Voice Extensible Markup Language VoiceXML[1] or the Extensible MultiModal Annotation markup language EMMA[2], with products from industry supporting these standards[3]. We will use an EMMA-related XML format we called PreML which is also used throughout the examples in [14]. A comprehensive overview of ontology-based dialogue processing, which is the basis of our integration platform, can be found in [13], pp.71-131. Figure 1 shows the ontology components the user works with. The graphical user interfaces (GUIs) for editing ontologies, speech recognition grammars, and interaction rules are implemented as Eclipse plugins in the ODP workbench (see screenshots in [17]) using the open source toolkit JUnit. The results of the interactive processes where the dialogue engineers are involved, are stored in RDF repositories as ontology data, domain-dependent specifications, and ontology-based rules sets for interaction and input interpretation rules of the specific application domain.

[1] http://www.w3.org/TR/voicexml20/
[2] http://www.w3.org/TR/emma/
[3] http://www.voicexml.org

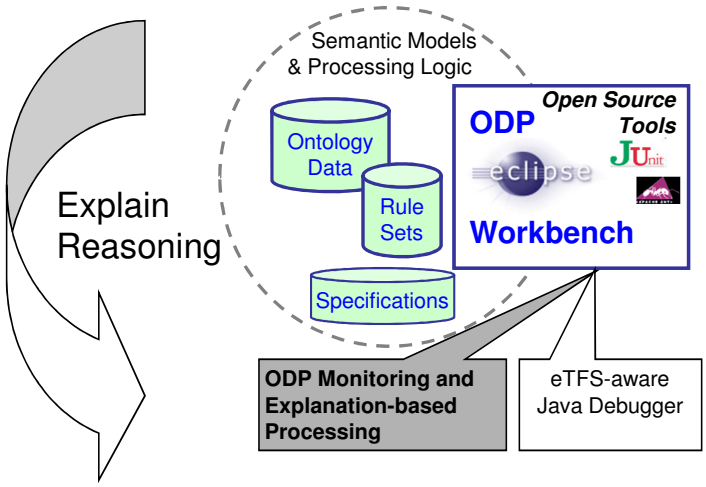

Fig. 1. Ontology components the user works with for explaining reasoning processes.

In [18], we tackled the challenge of how an application backend can automatically infer previously unknown knowledge (facts) and provide explanations for the inference steps involved. (Proper explanations are a main factor for increasing the level of user trust in end-to-end human-computer interaction systems.) In the medical application domain [16], we use a medical backend database system that stores RDF image meta data, e.g., patient names, dates of issue, and the like. The dominant query language for these RDF[4] repositories is the W3C recommendation SPARQL[5]. Examples of how SPARQL can be used in the context of integrating Linked Data for semantic dialogue and backend access can be found in [15]. A Java debugger has also been implemented; under the hood, it uses an application programming interface for the efficient representation of ontology-based data using extended typed feature structures (eTFS). As described in [11], the eTFS API is tightly integrated into a production rule system which enables a declarative specification of the processing logic in terms of production rules. This is the precondition for the ODP monitoring and explanation capabilities for internal, component-based, reasoning steps as described in the rest of the paper. Figure 2 provides rough sketch of the basic processing chain within the typical interaction cycle. The dialogue manager uses reasoning processes at each of the prominent processing stages, namely input interpretation, dialogue interaction, data requesting, the provision of answers in the application backend, the retrieval of data, and its presentation (which normally incorporates a NLU generation step [4]). In the context of explanations and production rule

[4] See http://www.w3.org/TR/rdf-primer/ and http://www.w3.org/TR/rdf-schema/.
[5] See http://www.w3.org/TR/rdf-sparql-query/.

reasoning for dialogue systems, in this contribution, we will focus on the input interpretation step which is highlighted in figure 2.

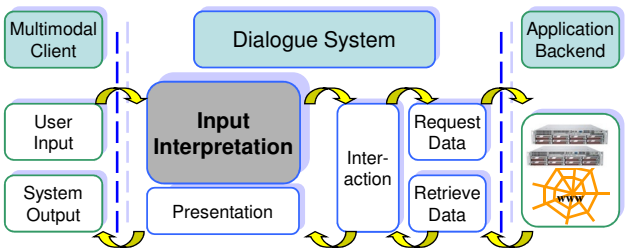

Fig. 2. Typical interaction cycle.

3 Explanation-based Interface for Production Rules

In this section, we explain the technical background of a GUI to monitor and explain reasoning processes for the input interpretation phase of multimodal dialogue systems. Essentially, the explanation-based GUI helps to deal with complex rules in a production rule system.

The production rule system provides rule-based fusion of different input modalities such as text, speech, and pointing gestures, and allows us to solve ambiguities in co-references. For example, when the user opens the patient records of several persons and utters "also the latest CT images" later in the dialogue, the production rule system is used to reason about the correct patient instance the user is referring to. We use the production-rules-based fusion and discourse engine which follows the implementation in [8].

Within the dialogue infrastructure, this component plays a major role since it provides basic and configurable dialogue processing capabilities. More processing robustness is achieved through the application of a special robust parsing feature in the context of RDF graphs as a result of the input parsing process and through the possibility to debug, edit, and maintain the production rules in a specific GUI. In addition, the semantic relationships between several catchwords from the spoken user input can be guessed (following [19]) according to the ontological domain model of the industrial (medical) application domain. For example, the query "any experts available?" searches for other radiologists with a specific portfolio that matches with the semantic disease annotation on the radiology images.

4 An Application Scenario for Explanations

In this section, we will describe an application scenario where the dialogue engineer can use our GUI to edit input fusion rules in a medical application scenario. The rules can be inspected and maintained because the engineer gets many explanations about how new rules behave in the *running* dialogue system. We focus on the aspect that the presentation of the software system's ontology-based interaction rules is in a way that they are accessible to humans for maintenance, and, at the same time, allow a real-time monitoring of their application in the running dialogue system.

4.1 Dialogue Example

Consider a radiologist who treats a patient which suffers from a specific type of cancer, e.g., lymphoma. The treatment of such severe diseases requires a lot of follow-up examinations where previous diagnoses have to be taken into account. The dialogue engineer has to provide the domain user, i.e., a radiologist in the medical domain, with the desired dialogue competence (also cf. the THESEUS MEDICO use case) to access the desired patient records, the corresponding image data, and comparative cases. In addition, the doctor should be able to complete a new finding while interacting with a multimodal dialogue system.

The following medical dialogue illustrates the doctor's practical interest in using a dialogue interface on top of an ontology-based search database for medical images. The dialogue concentrates around the questions of this user (U) about the media contents, i.e., the body parts and the anatomy shown in computer tomography (CT) picture series (also see DICOM, medical.nema.org) and magnetic resonance (MR) videos, and the the answers of the system (S).

Figure 3 shows the medical dialogue example and multiple CT image results which are displayed on an iPad. The user can freely speak to the iPad which is used as the interaction device in the distributed dialogue system architecture (also cf. http://www.dfki.de/RadSpeech).

4.2 Exploring Multiple Applicable Rules

We chose the OWL formalism recommended by W3C and identified a subset that suits the underlying eTFS representation for the specification of the input fusion ontology. The eTFS format unifies the properties of RDF/RDFS [6] and typed feature structures [3].

For the runtime knowledge processing step, we have implemented an eTFS API that provides useful methods to compare two types and access related concepts within the inheritance hierarchy quickly.

The API is essential for the reasoning engine of the ontology-based production rule system (PATE) [9]. PATE's architecture is centered around the idea of three separated data storage facilities: (i) the goal stack, (ii) the working memory, and (iii) the long-term memory. The working memory is responsible for the activated instances, the so-called working memory elements (WMEs), which are

1 **U**: “Show me the patient file, last examination, Gerda Meier.“
2 **U**: “Also the images, … CT”
3 **S**: “Shows the patient file and corresponding patient CT studies in DICOM pictures.“
4 **U**: “Show me the internal organs: lungs, liver, then spleen and colon."
5 **S**: Shows corresponding patient image data according to referral record.
6 **U**: “Summarise the patient's findings."
7 **S**: Synthesises a summary of the patient's findings.

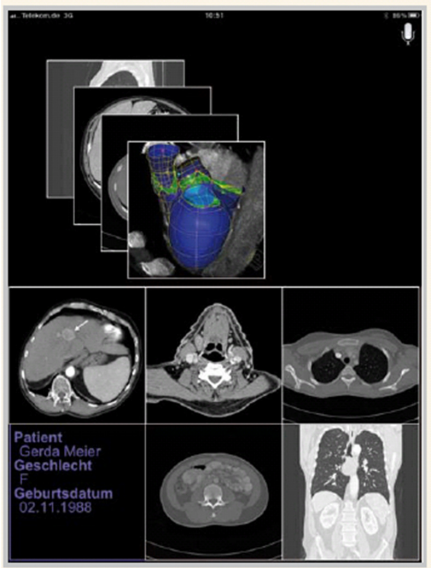

Fig. 3. Medical dialogue and system results on an iPad.

accessible for applying production rules. The long-term memory is responsible for the persistent storage for all instances of the type hierarchy the system has in the background. The purpose of the goal stack is to mimic the attentional focus during the dialogue process that is inspired by the cognitive models introduced in [2]. On which of the three stacks mentioned above the current WMEs are put, is determined by an activation value (which can also change after a rule has fired). In every processing state, there is always one single WME in focus (which has an impact on the rules accessible for firing). Only rules whose goal condition matches the pattern of the WME on top of the goalstack have a chance to fire. Without such editor functionalities, dialogue engineers would have to edit complex OWL files manually which is impractical.

Rule Inspection The background knowledge of the dialogue system determines the relations of different concepts within the hierarchy. Our ontology says that the main condition in the rule represented by the concept *comet#QueryTask* subsumes the concept *medico#RetrievePatientImages*, which represents the ontological concept for the task in the second turn of the dialogue example in 3. If the dialog state has the attentional focus on an that eTFS structure, then the condition is sufficient to cause the rule to fire, because the unification would succeed in this constellation.

Further refinements of rule firing is provided by the possibility to involve the unification of side conditions in the rule with structures located inside the

working memory. This mechanism is especially beneficial in the context of tuning and debugging rules at runtime, which we will illustrate in more detail.

We implemented a *rule inspector*, a tool that provides debugging options along the process pipeline from the interpretation of an utterance to the adaption of ontological representations of commands, to actual retrieval actions in the backend, to the reallocation of the retrieved results in the presentation step. The main view of the rule inspector GUI (figure 4) shows the state of the production rule system after the calculation of the firing rule before applying the actions coded in the head of the production rule (action part). The two lists, the working memory contents and the goal stack that correspond to the *working memory* box and the *goal stack* box of the upper part in figure 4, respectively, show the rule's basic type and activation score. Additionally, a syntax-highlighted XML viewer allows us to inspect the WMEs during the dialogue process by clicking on the entries in the list, see the *wme inspector* in 4. The same debugging support is provided in the lower part of the GUI, where matching rules indicated in the *conflict* set' box, once selected, are displayed in an XML format together inside the *rule inspector* box with their probabilities to fire successfully. Rule firing can be invoked manually by the dialogue engineer while clicking on the *Step!* button.

New Rule Creation and Debugging Moreover, rule inspection is supported for all (other) production rules assigned to a processing module that are listed in the *Rule base* box in figure 4. This is crucial when the dialogue engineer adds a new rule to the already existing collection of rules. In this specific scenario, the engineer expects to enrich the existing interaction possibilities covered by the dialogue application.

However, often the expected behavior at a target state along the dialogue process fails to appear. At this point, the dialogue engineer can invoke in the rule inspector to get support in the process of finding out why a certain rule, here the newly created rule, does not fire.

The rule inspector performs the reasoning step to deduce and display the conflicting part of the new rule that does not semantically fit to the set of valid rules with regard to the current state of dialogue process.

Figure 4 captures the state where the dialogue process step shifts to 'request data' in the backend system after successful interpretation. This snapshot shows the "pause" in the processing stage after the utterance "Also the images" (cf. the example dialogue in figure 3) when the radiologist requests the patient's image data. The modality fusion component keeps track of the ongoing discourse context and resolves the correct patient reference 'Gerda Meier' shown in the *WME inspector* box of figure 4. The ontological concept *medico#RetrievePatientImages* represents the system task to be executed on the backend service, this means, the query instance will be transformed into the corresponding SPARQL query. In this particular case, however, the rule which would trigger the request to the desired backend service, is not an option for the rule engine because the conditions do not match. Here, the developer has the opportunity to navigate and select the desired rule and receive the information where unification fails, indicated in the

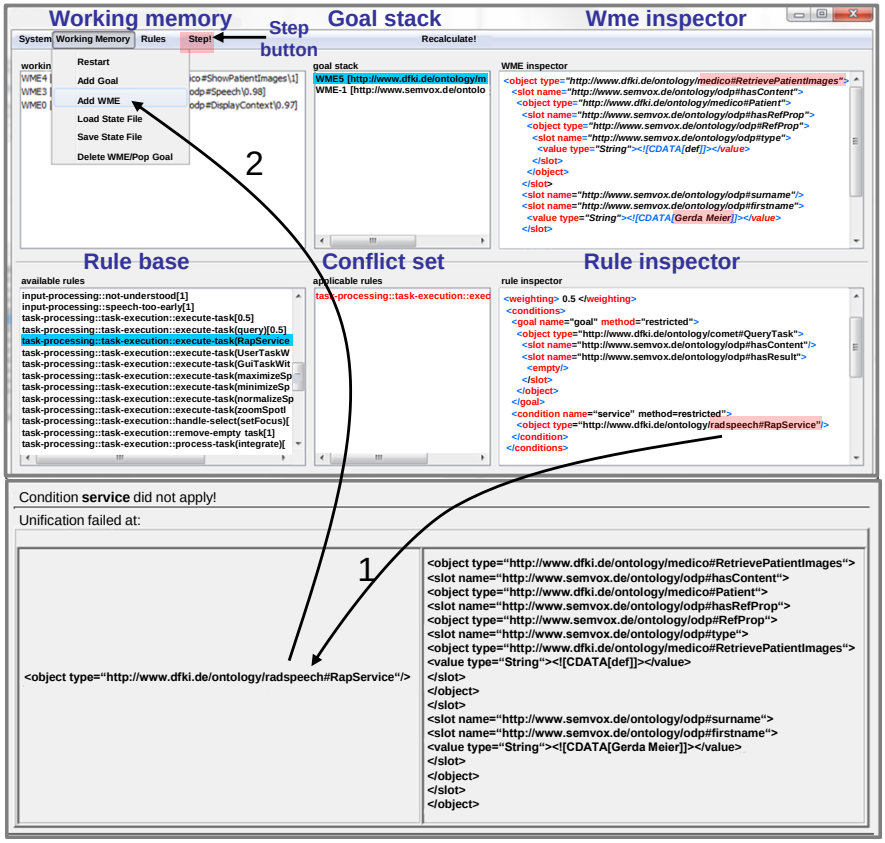

Fig. 4. Display, explanation, and editing in case of a unification conflict.

lower box in figure 4. Therein the dialogue engineer may consult the rule inspector for individual parts of the eTFS structures that collide. In particular, the newly added rule requires an instance of the concept *radspeech#RapService* on the working memory in order to match the current configuration of the dialogue state, indicated by the arrow (1). The GUI support of detecting the conflicting parts during unification provides the necessary explanations to the developer about the changes in the dialogue state and the actions he or she can perform (at runtime!) in order to cause the newly added rule to fire.

In addition to the functionality to give advice to the developer at runtime, the rule inspector offers the opportunity to add the required instance on the working memory manually. Thus, by adding the *radspeech#RapService* instance to the working memory, see arrow (2), the debugging process will be resumed towards the newly created and correct interaction possibility. Previously, the

new backend for the query "also the images" could not be accessed because the rule for the *RapService* did not fire. Instead, we inspected the more general rule (executeTask(query)) to fire (with the help of the GUI) and successfully adapted the conditions to rerank the firing rules so that the desired rule is included and the *RapService* could be accessed in this dialogue state. In the course of adding this new behaviour to an already existing dialogue application, the explanation possibilities of the inspection editor offer powerful support to view low-level data, and edit/reason about new rules which can be done at runtime with the help of only one graphical user interface tool.

5 Conclusion and Future Work

Based on an integration platform for off-the-shelf dialogue solutions and internal dialogue modules (ODP platform), we described the parts of the discourse and dialogue infrastructure that allow for a explaining reasoning processes in the (ontology-based) input interpretation step. We focussed on a medical application scenario where we demonstrated how to maintain rules, explore multiple applicable rules, inspect rules, and create new rules that can be reasoned about and therefore explained at runtime. In this way, we provided a solution to the particular challenge to present ontology-based interaction rules in a way that they are accessible to and editable for humans for maintenance, and, at the same time, allow a real-time monitoring of their application in the running dialogue system. In future work, we plan to include reasoning processes that suggest rules to the user in addition to explaining why user-generated rules do not behave correctly.

Acknowledgements Thanks go out to Robert Nesselrath, Yajing Zang, Markus Löckelt, Matthieu Deru, Simon Bergweiler, Alassane Ndiaye, Norbert Pfleger, Alexander Pfalzgraf, Jan Schehl, Jochen Steigner and Colette Weihrauch for the implementation and evaluation of the dialogue infrastructure. This research has been supported by the THESEUS Programme funded by the German Federal Ministry of Economics and Technology (01MQ07016).

References

1. Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L., Stent, A.: An Architecture for a Generic Dialogue Shell. Natural Language Engineering 6(3), 1–16 (2000)
2. Anderson, J.R., Lebiere, C.J. (eds.): The Atomic Components of Thought. Lawrence Erlbaum Associates, Mahwah, NJ (1998)
3. Carpenter, B.: The Logic of Typed Feature Structures. No. 32 in Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge, UK (1992)
4. Engel, R.: SPIN: A Semantic Parser for Spoken Dialog Systems. In: Proceedings of the 5th Slovenian and First International Language Technology Conference (IS-LTC 2006) (2006)

5. Fensel, D., Hendler, J.A., Lieberman, H., Wahlster, W. (eds.): Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential. MIT Press (2003)
6. Manola, F., Miller, E.: RDF primer. W3C recommendation, W3C (February 2004), published online on February 10th, 2004 at http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
7. Martin, D., Cheyer, A., Moran, D.: The Open Agent Architecture: a framework for building distributed software systems. Applied Artificial Intelligence 13(1/2), 91–128 (1999), citeseer.ist.psu.edu/martin99open.html
8. Pfleger, N.: FADE - An Integrated Approach to Multimodal Fusion and Discourse Processing. In: Proceedings of the Dotoral Spotlight at ICMI 2005. Trento, Italy (2005)
9. Pfleger, N., Schehl, J.: Development of advanced dialog systems with PATE. In: Proceedings of Interspeech 2006—ICSLP: 9th International Conference on Spoken Language Processing, Pittsburgh, PA, USA. pp. 1778–1781 (2006), http://www.isca-speech.org/archive/interspeech_2006/i06_1598.html
10. Reithinger, N., Fedeler, D., Kumar, A., Lauer, C., Pecourt, E., Romary, L.: MIAMM - A Multimodal Dialogue System Using Haptics. In: van Kuppevelt, J., Dybkjaer, L., Bernsen, N.O. (eds.) Advances in Natural Multimodal Dialogue Systems. Springer (2005)
11. Schehl, J., Pfalzgraf, A., Pfleger, N., Steigner, J.: The BabbleTunes System. Talk to Your IPod! In: Proceedings of the 10th International Conference on Multimodal Interfaces (ICMI) (2008)
12. Seneff, S., Lau, R., Polifroni, J.: Organization, Communication, and Control in the Galaxy-II Conversational System. In: Proceedings of Eurospeech'99. pp. 1271–1274. Budapest, Hungary (1999)
13. Sonntag, D.: Ontologies and Adaptivity in Dialogue for Question Answering. AKA and IOS Press, Heidelberg (2010)
14. Sonntag, D., Deru, M., Bergweiler, S.: Design and Implementation of Combined Mobile and Touchscreen-Based Multimodal Web 3.0 Interfaces. In: Proceedings of the International Conference on Artificial Intelligence (ICAI). pp. 974–979 (2009)
15. Sonntag, D., Kiesel, M.: Linked data integration for semantic dialogue and backend access. In: AAAI Spring Symposium on Linked Data Meets Artificial Intelligence (2010)
16. Sonntag, D., Möller, M.: Unifying semantic annotation and querying in biomedical image repositories. In: Proceedings of International Conference on Knowledge Management and Information Sharing (KMIS) (2009)
17. Sonntag, D., Sonnenberg, G., Nesselrath, R., Herzog, G.: Supporting a rapid dialogue engineering process. In: Proceedings of the First International Workshop On Spoken Dialogue Systems Technology (IWSDS) (2009)
18. Sonntag, D., Theobald, M.: Explanations in dialogue systems through uncertain RDF knowledge bases. In: Roth-Berghofer, T., Tintarev, N., Leake, D.B., Bahls, D. (eds.) Proceedings of the Fifth International Workshop on Explanation-aware Computing (ExaCt 2010). CEUR Workshop Proceedings, vol. 650, pp. 1–12. CEUR-WS.org, Lisbon, Portugal (2010)
19. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data. In: ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering. pp. 405–416. IEEE Computer Society, Washington, DC, USA (2009)
20. Wahlster, W.: SmartKom: Symmetric Multimodality in an Adaptive and Reusable Dialogue Shell. In: Krahl, R., Günther, D. (eds.) Proceedings of the Human Computer Interaction Status Conference 2003. pp. 47–62. DLR, Berlin, Germany (2003)