

Supporting a Rapid Dialogue System Engineering Process

Daniel Sonntag, Robert Nesselrath, Gerhard Sonnenberg, and Gerd Herzog

German Research Center for AI (DFKI), Stuhlsatzenhausweg 3, 66123 Saarbruecken,
Germany

Abstract. We implemented a generic 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 ontological domain extensions and debugging functional modules. We support a rapid implementation cycle until the dialogue systems works robustly for a new domain, e.g., the dialogue-based retrieval of medical images.

1 Introduction

The idea of the Semantic Web [Fensel et al., 2003] provides new opportunities for *semantically-enabled user interfaces*. The explicit representation of the *meaning* of data allows us to (1) transcend traditional keyboard and mouse interaction metaphors, and (2) provide representation structures for more complex, collaborative interaction scenarios that may even combine mobile and terminal-based interaction [Sonntag et al., 2009a]. Over the last years, we have adhered strictly to the developed rule “No presentation without representation.” The idea is to implement a generic, and semantic, dialogue shell that can be configured for and applied to domain-specific dialogue applications.¹

In this paper, we discuss the parts of a toolbox we implemented that supports a rapid dialogue engineering process to access multimedia repositories. (Following a domain knowledge acquisition methodology, the necessary domain knowledge can be acquired, e.g., the necessary medical knowledge about medical image contents [Sonntag et al., 2009b].) A workbench provides the support for task dialogue development and the knowledge engineering process for a domain-specific (multimodal) dialogue interface. This toolbox is based on the industry standard *Eclipse* and other established open source software development tools. We will describe a rapid dialogue engineering task and its challenges (section 2) and provide a technical solution for the two challenges of engineering ontological domain extensions (section 3) and debugging functional modules (section 4). The particular challenge we address is to speed up the implementation cycle until the dialogue systems works robustly. Section 5 provides a conclusion.

¹ This work is part of THESEUS-CTC (see www.theseus-programm.de) to implement dialogue applications for use case scenarios. It has been supported by the German Federal Ministry of Economics and Technology (01MQ07016). The responsibility for this publication lies with the authors.

2 Rapid Dialogue Engineering Task and Challenges

We implemented a situation-aware dialogue shell for semantic access to image media, their annotations, and additional textual material. 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. 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. Figure 1 provides a high-level view and rough sketch of the basic processing chain within the typical interaction cycle.

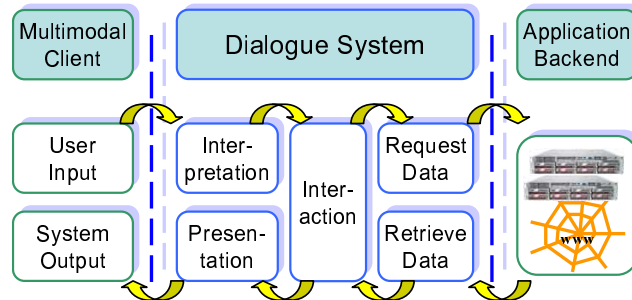


Fig. 1: Basic building blocks and core workflow of multimodal dialogue processing

The dialogue engineering task is to provide a dialogue-based access to answer questions about the domain of interest. The spoken dialogue input is used to generate SPARQL queries on ontology instances (using a Sesame repository, see www.openrdf.org) and following the guidelines in [Sonntag et al., 2007]). Prominent examples of integration platforms include OOA [Martin et al., 1999], TRIPS [Allen et al., 2000], and Galaxy Communicator [Seneff et al., 1999]; these infrastructures mainly address the interconnection of heterogeneous software components. In THESEUS, 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:

- providing a common basis for task-specific processing;
- accessing the entire application backend via a layered approach;
- engineering ontological domain extensions;
- debugging functional modules such as natural language understanding (NLU), fusion, and management (and external text-to-speech (TTS) synthesis).

The first two, more technical, challenges have been solved by implementing the core of a dialogue runtime environment, the ODP framework and its platform API (the DFKI spin-off company SemVox, see www.semvox.de, offers

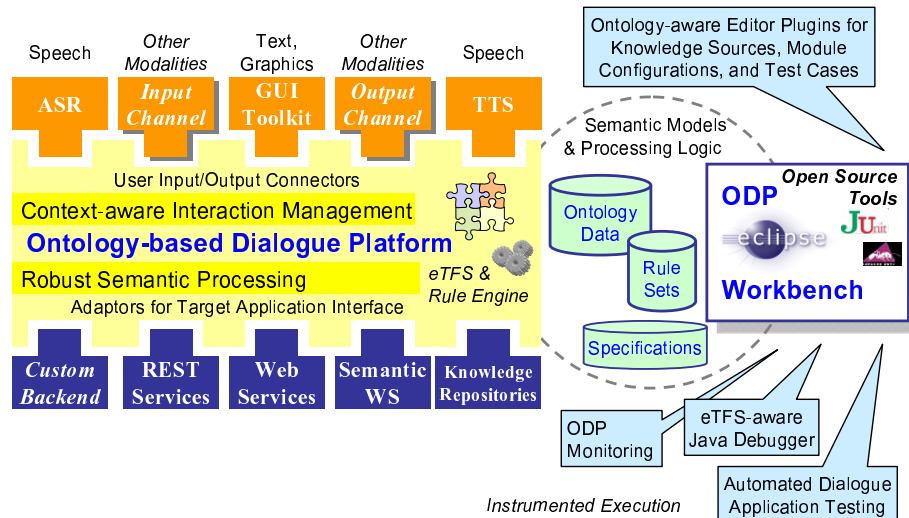


Fig. 2: Overall design of the multimodal dialogue shell: ODP framework and workbench

a commercial version), as well as providing configurable adaptor components. These translate between conventional answer data structures and ontology-based representations (in the case of, e.g., a SPARQL backend repository) or Web Services (WS)—ranging from simple HTTP-based REST services to Semantic Web Services, driven by declarative specifications [Sonntag and Möller, 2009]. However, using the dialogue framework for the implementation of domain dialogue requires domain extensions and the adaptation of functional modules while implementing a new dialogue for a new domain or use case. Hence, an integrated toolbox is required, as depicted in figure 2. The ODP workbench 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. A distinguishing feature of the toolbox is the built-in support for eTFS (extended Typed Feature Structures), the optimised ODP-internal data representation for knowledge structures. This enables ontology-aware tools for the knowledge engineer and application developer.

Implementing a Medical Domain Dialogue Dialogue engineering is an interactive process between the *dialogue engineer* and the *domain expert*. The dialogue engineer has to provide the domain user, e.g., a radiologist in the medical domain, with the desired dialogue competence (also cf. the THESEUS MEDICO use case).

The following medical dialogue illustrates the doctor’s practical interest in using a dialogue interface on top of a semantic image search engine. The dialogue concentrates around the questions 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.

1 **U**: “Show me the CTs, last examination, patient Peter Meier.”

2 **S**: Shows corresponding patient CT studies in DICOM picture series and MR videos.

3 **U**: “Show me the internal organs: lungs, liver, then spleen and colon.”

4 **S**: Shows corresponding patient image data according to referral record.

5 **U**: “Summarise the patient’s findings.”

6 **S**: Synthesises a summary of the patient’s findings.

3 Engineering Ontological Domain Extensions

The workbench for the ontology-based dialogue platform provides an editor to extend and modify the upper-level ontology with domain-specific content. The upper-level ODP ontology provides (1) interpretation structures resulting from a stepwise analysis of user input; (2) presentation structures that are employed for the declaration and generation of (multimodal) system output; (3) context information to improve the conversational competence in interaction management; and (4) dialogue task structures that function as templates for the implementation of domain-specific dialogue acts. We will focus on these templates.

Figure 3 (left) shows the class view of the domain-specific dialogue acts (as implemented in the Comet music retrieval system [Sonntag et al., 2009a] and in the dialogue system for the medical domain [Sonntag and Möller, 2009]); these are ontology class extensions of the *ODP#UserTask* class. Additional colour information about namespaces highlights the class embeddings concerning the GUI representation of the task goals, i.e., controlling GUI elements named *Spotlets* in the Comet subdomain (AddSpotlet, CloseSpotlet, UpdateSpotlet). Based on the dialogue management rules for the general ODP concepts, we can automatically handle the update of tasks and their results in the current display context.

New user tasks and concepts for GUI representations are created with the Ontology Class Editor shown in figure 3 (right). With this editor, the names and namespaces of new classes are defined. Furthermore, new slots can be added or their ranges restricted. Our Eclipse plugin allows dialogue engineers to write down ontological domain extensions and test them in the running system in a unique programming environment (Eclipse) without the need for an external ontology tool such as Protégé (<http://protege.stanford.edu/>). This is particularly beneficial when debugging functional modules.

4 Debugging Functional Modules

In order to ease the task of writing domain-specific natural language understanding (NLU) rules in ontology form, we use *auto-completing* combo-boxes for writing down automatic speech recognition (ASR) and NLU rules. While typing in an ontology concept or slot, concept and slot names with matching prefixes

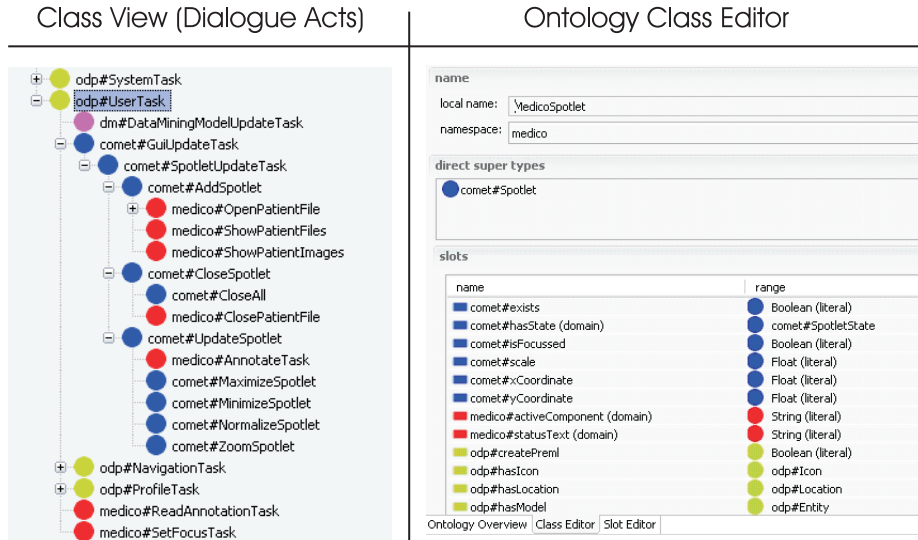


Fig. 3: Graphical User Interface of the Ontology Engineering Tool

are shown. In addition, this procedure helps validate the ontology structures the user creates. Figure 4 shows the online engineering and debugging cycle.

First, the user writes ASR rules in ontological form. Figure 4 (left) shows the rules for recognising and interpreting utterance (5) in the example dialogue: “Summarise the patient’s findings.” We interpret the utterance as a deictic one, where the determination of the (patient) referent is dependent on the context in which it is said. Here, of course, the context is the patient “Peter Meier”, stored in the discourse context.

Second, this missing information, named the *refProp* property, is added by the FUSION module in the process pipeline, after the message is sent over the ODP dialogue platform. Most importantly, a complex dialogue ontology instance (cf. figure 3 (left)) is built up by the modules in this processing pipeline, whereby the abstract ontology instance is easily created by the ODP upper class template. This means the user has to fill in only the slots which are specific for the domain task.

Third, the fully instantiated ontology instance can be inspected in another special Eclipse plugin. This plugin (figure 4 (right)) allows a dialogue engineer to inspect the ontological interpretation result when a user tests the running system. If an incorrect ASR or NLU rule is encountered, or the wrong dialogue task instance is built up, the dialogue engineer can directly change (revise) the corresponding rules and let the user “say again” (repeat her/himself) without interrupting the dialogue session. In this way, the debugging of functional modules, such as the speech recognition, interpretation, and fusion modules, can be extremely facilitated, especially when complex ontology structures are used for knowledge representation and message transfer.

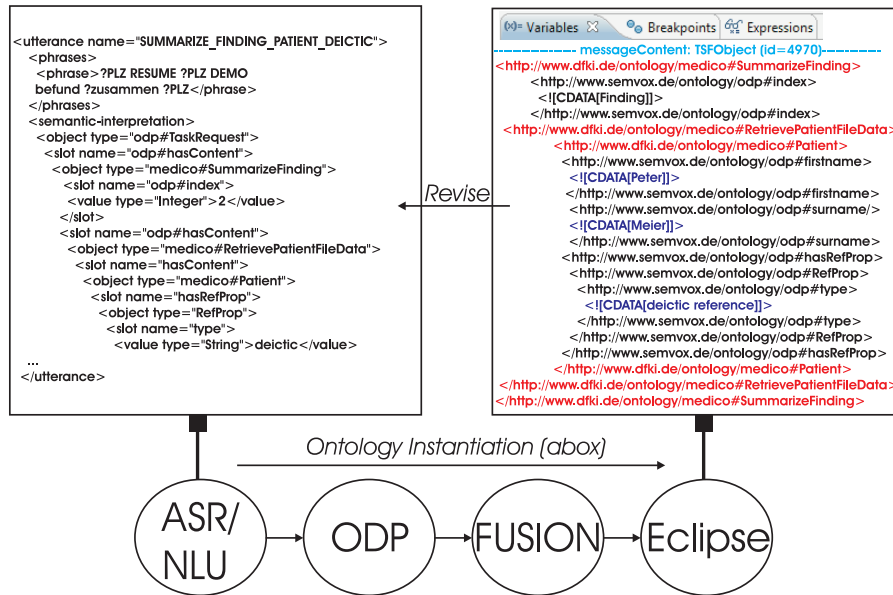


Fig. 4: Online Engineering and Debugging Cycle

4.1 Editing the GUI Ontology Model

A central building block for component development and an integral part of the ODP middleware is the included application programming interface for the efficient representation of ontology-based data using extended Typed Feature Structures (eTFS). As described in [Pfleger and Schehl, 2006], 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. The eTFS structures are also used to encode semantic presentation structures. The display context comprises of a more complex GUI ontology model. This means we specify the functionality of all visible GUI components in a GUI model; this model is ontology-based and can be extended by an appropriate authoring tool, the *display context editor*. Figure 5 shows the display context of a specific GUI frame which displays a SIE which stands for Semantic Interface Element (the generic term for the Comet Spotlets).

The screenshot in figure 5 shows how the SIE's functionality model can be extended by multiple object types available in ODP. An auto-completion box depicts all available options. In this case, the ASR push-to-talk functionality is available. This means we can easily add a push-to-talk functionality to any SIE object displayed on the screen. This is particularly beneficial for testing different ASR activations which are more user-centred than traditional buttons on the screen. This applies to other physical devices which are often not convenient in the application domain, too. The GUI Ontology Model also includes the semantic model of the referred entities (e.g., the patient or the patient file) which offers

additional flexibilities in defining the rule conditions or effects, especially for information update rules.

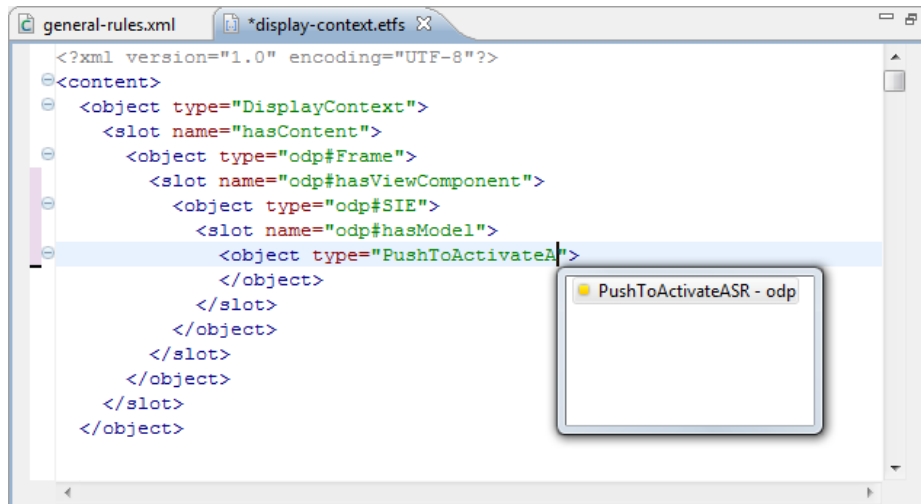


Fig. 5: Display Context Editor

The medical scenario should illustrate this: the screenshot (figure 6) shows how the dialogue system is embedded into the use case specific application environment. The user can ask questions using speech and refer to displayed elements as deictic expressions. The current display context frame sets the dialogue context in the form of a simple information state in order to resolve (deictic) references. (The information state is not discussed in this paper. The fusion rules, however, are explained in more detail in section 4.2). The current display context is the dotted frame in figure 6. The displayed SIEs make up the display context (cf. figure 6). While using the display context editor, the push-to-talk functionality can be easily attached to a SIE. In this way, implicit ASR activation can be tested according to clinical usability issues. The clinician can, for example, click on a CT image where several landmarks are displayed to activate the ASR. Then, he or she selects a landmark and asks specific questions such as “Summarise the patient’s findings here.” In this example, the user is able to implicitly activate the ASR (please note that this is independent of the individual integration pattern of the user). Finally, the output of the FUSION step is transferred to the backend system. The generated SPARQL queries (section 4.3) are then processed by the backend system in order to retrieve the medical images of the patient in the current dialogue focus.

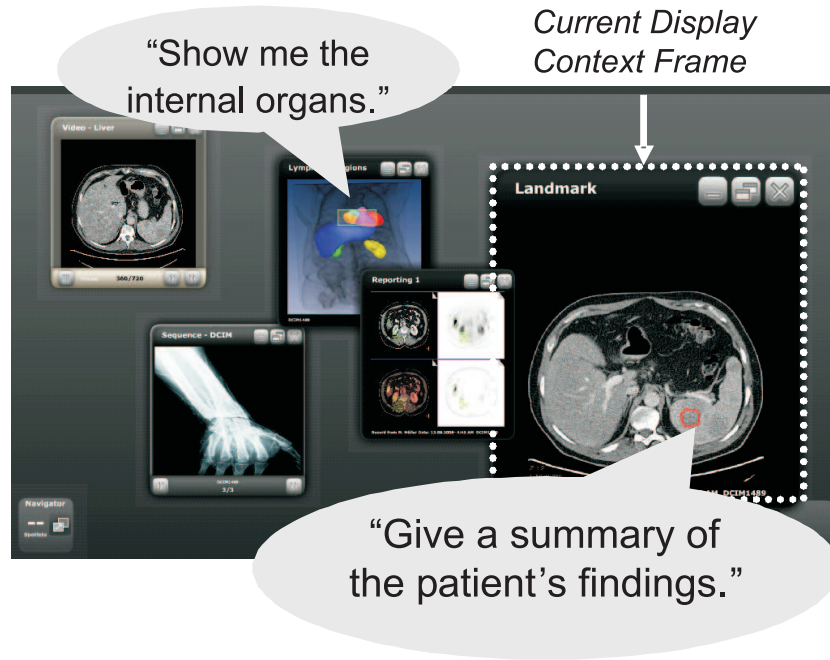


Fig. 6: Multimodal (Touchscreen) Installation for the Radiologist

4.2 Maintaining Fusion Rules and Deictic References

A modality fusion component keeps track of the ongoing discourse, completes different types of anaphora, and merges input from different modalities. We use a production rule system, FADE, which is part of the ODP distribution. With the increasing number of involved rules for dialogue input fusion, developing rules can become error-prone and controlling their effects quite difficult. To support rule authoring for maintaining the correct fusion behaviour of the system while adapting to a new domain, we implemented a graphical user interface to monitor and edit fusion requests. With the help of a dialogue fusion input editor, the user can adapt system states or rule contents during runtime.

The fusion input editor in figure 7 shows the state of the dialogue system after calculating the set of *applicable rules* (figure 7, left) for resolving deictic references in utterance (5) “Summarise the patient’s findings.” + *Click on specific landmark* in a medical image. The task is to find the correct patient reference and the correct deictic reference for the word “findings” which corresponds to the image region the user clicked on.

The example ontology structure in the *rule inspector* (figure 7, right) shows the state of the fusion reference module before resolving the missing deictic information of the summarisation request. (It is the rule with the highest weighting that ranks first and is displayed.) The patient object, embedded into the task request, is annotated with a deictic *refProp* property. This rule gives an overview of

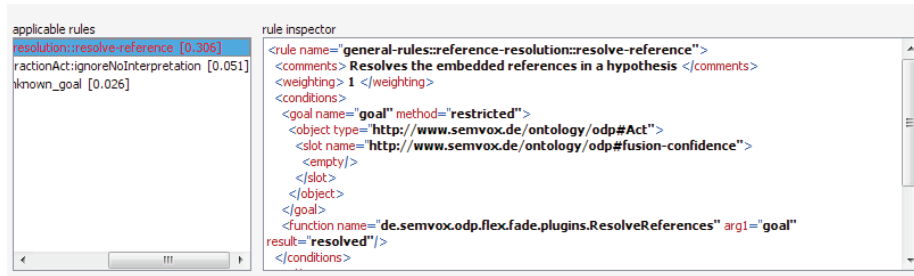


Fig. 7: Dialogue Fusion Input Editor

the rule conditions and actions and allows developers to follow the actions which are performed in the next fusion steps (in this example a reference resolution of an unresolved embedded object, the patient).

4.3 Generating SPARQL Queries

Many systems are available that translate natural language input into structured ontological representations (e.g., AquaLog [Lopez et al., 2007]), port the language to specific domains, e.g., ORAKEL [Cimiano et al., 2007], or use reformulated semantic structures NLION [Ramachandran and Krishnamurthi, 2009]. AquaLog, e.g., presents a solution for a rapid customisation of the system for a particular ontology; with ORAKEL a system engineer can adapt the NLU component in several cycles thereby customising the interface to a certain knowledge domain; and NLION uses shallow natural language processing techniques (i.e., spell checking, stemming, and compound detection) to realise a single semantic concept or an ontology property. All of them support the translation to SPARQL queries in principal. However, all of them deal with written keywords or simple semantic relations, e.g., *X isDefinedAs Y*. They do not focus on the much more complex ASR/NLU process needed for natural speech input while using a dialogue system. In addition, these systems directly transfer the input to the desired SPARQL queries without dealing with the complex influences of message passing in dialogue frameworks or input fusion. In our dialogue domain, however, these influences demand a complex online engineering and debugging cycle. The result should be an instantiation of a complex query object from which a SPARQL query can be derived.

The dominant query language for RDF repositories (which we use in the example) is the W3C recommendation SPARQL². Similar RDF-based query languages are worth mentioning such as RDQL³, SERQL⁴, or iTQL used by

² <http://www.w3.org/TR/rdf-sparql-query/>

³ <http://www.w3.org/Submission/RDQL/>

⁴ <http://www.openrdf.org/doc/sesame/users/ch06.html>

Kowari⁵. All these languages are based on the notion of RDF triple patterns which can be connected via several query operators such as “union” or “filter”.

We used SPARQL queries because they are the de facto standard and are supported by the two semantic repository APIs we use, namely Jena and Sesame (Sesame for a remote medical repository). In addition, the SPARQL 2 specification provides support for operators such as “group by” or aggregate functions (e.g., COUNT, MIN, MAX, SUM). In the context of (unstructured) natural language input, SPARQL also provides convenient operator extension, i.e., the “filter” operator, to specify free text searches and even regular expressions based on operations for regular expressions. (This includes the ability to ask SQL “like”-operator style expressions.) Although operator extensions that deal with regular expressions are only seldom available at public SPARQL endpoints (the execution of such queries is rather computation-intensive), they can be easily used when addressing internal or private remote repositories of moderate size, as is the case for the medical example repository.

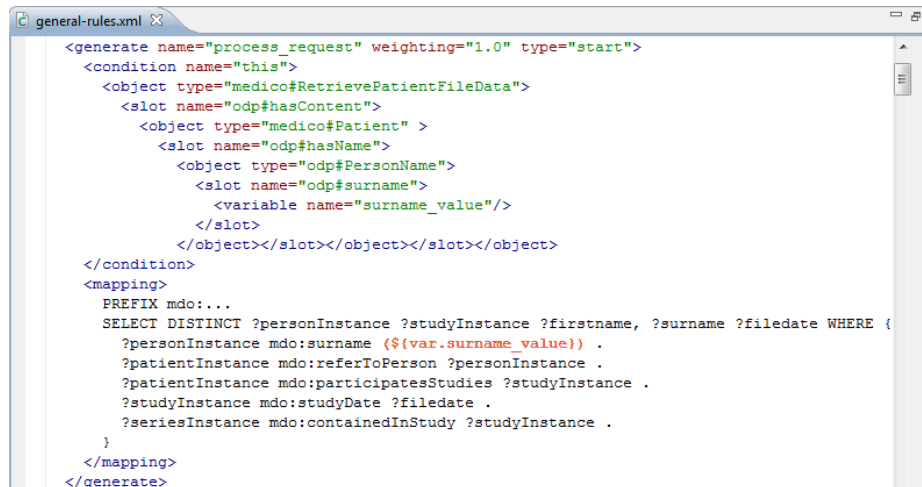
The *SPARQL Query Editor*, displayed in figure 8, allows dialogue engineers or retrieval experts to specify what should happen when processing a user request. For example, the process of retrieving a patient data file has several conditions as slots to be filled before the technical retrieval process makes sense. In our example, the patient name slots have to be filled. Subsequently, the mapping to the SPARQL query can be defined, i.e., the surname variable can be added to a SPARQL SELECT query. The interesting thing about this editor is that a dialogue engineer and a SPARQL retrieval expert can work together seamlessly: the dialogue engineer provides questions about the important retrieval objects. Hereupon, the retrieval expert writes some SPARQL queries for the specific domain. Finally, the dialogue engineer can use these example queries to insert variables for new conditions, e.g., the surname of a patient. In this way, the dialogue engineer can execute many application-specific SPARQL queries with only limited knowledge of the SPARQL dialect of a semantic search engine.

5 Conclusion

Over the last several years, the market for speech technology has seen significant developments [Pieraccini and Huerta, 2008] and powerful commercial off-the-shelf solutions for speech recognition (ASR) or speech synthesis (TTS). Entire voice user interface platforms (VUI) have also become available. Based on an integration platform (ODP) for such off-the-shelf solutions and internal dialogue modules, we described some parts of an Eclipse-based toolbox for ontology-based dialogue engineering and provided a technical solution for the two challenges of engineering ontological domain extensions and debugging functional modules such as the ASR/NLU component, the fusion component, and the (very domain-specific) SPARQL generation process.

Further application scenarios and more complex prototype systems need to be addressed. The integrated Eclipse-based toolbox for ontology-based dialogue

⁵ <http://www.kowari.org/>



```
general-rules.xml
<generate name="process_request" weighting="1.0" type="start">
  <condition name="this">
    <object type="medico#RetrievePatientFileData">
      <slot name="odp#hasContent">
        <object type="medico#Patient" >
          <slot name="odp#hasName">
            <object type="odp#PersonName">
              <slot name="odp#surname">
                <variable name="surname_value"/>
              </slot>
            </object>
          </slot>
        </object>
      </slot>
    </object>
  </condition>
  <mapping>
    PREFIX mdo:...
    SELECT DISTINCT ?personInstance ?studyInstance ?firstname, ?surname ?filedate WHERE {
      ?personInstance mdo:surname ( ${var.surname_value} ) .
      ?patientInstance mdo:referToPerson ?personInstance .
      ?patientInstance mdo:participatesStudies ?studyInstance .
      ?studyInstance mdo:studyDate ?filedate .
      ?seriesInstance mdo:containedInStudy ?studyInstance .
    }
  </mapping>
</generate>
```

Fig. 8: SPARQL Query Editor

engineering will continue to be expanded with more tools and predefined generic knowledge sources, facilitating the creation of new multimodal dialogue applications. From the experience we gained from the two use case prototypes (which have the *comet* and *medico* namespaces in the ODP framework) we are now in the process of extending the functional dialogue shell modules for a more complex dialogue behaviour which includes the extensions of ASR/NLU, fusion, and SPARQL queries, respectively. For example, the context editor allows a dialogue engineer to specify rules to empirically evaluate different integration patterns and adapt the fusion rules accordingly. In addition, efficient error recovery in the dialogue assumes a complex state model of the characteristics of the retrieval environment. Different access and reliability models have to be updated regularly. In both examples, machine learning plays a major role. This would close the gap between the manual dialogue engineering support as described here and automatic—learned—procedures to adapt to new dialogue domains and dialogue situations while supporting a rapid implementation cycle until the dialogue systems works robustly for a new domain or situation.

One of these situations is the dialogue-based access (as THESEUS service) to a semantic search index. For example, Sindice, <http://sindice.com/>, provides a lookup index of Semantic Web resources. Together with newer inference capabilities [Delbru et al., 2008], Sindice will provide advanced search capabilities in the near future which can be used to answer very detailed questions about the desired entities (e.g., clinical trails). Whereas the dialogue-based access to keyword-based search engines has only moderate success, semantic (ontology-based) interpretations of dialogue utterances may become the key advancement in semantic search, thereby mediating and addressing dynamic semantic search engines which are already freely available.

References

- [Allen et al., 2000] Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L., and Stent, A. (2000). An Architecture for a Generic Dialogue Shell. *Natural Language Engineering*, 6(3):1–16.
- [Cimiano et al., 2007] Cimiano, P., Haase, P., and Heizmann, J. (2007). Porting natural language interfaces between domains: an experimental user study with the orakel system. In *IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces*, pages 180–189, New York, NY, USA. ACM.
- [Delbru et al., 2008] Delbru, R., Polleres, A., Tummarello, G., and Decker, S. (2008). Context Dependent Reasoning for Semantic Documents in Sindice. In *4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2008)*.
- [Fensel et al., 2003] Fensel, D., Hendler, J. A., Lieberman, H., and Wahlster, W., editors (2003). *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press.
- [Lopez et al., 2007] Lopez, V., Uren, V., Motta, E., and Pasin, M. (2007). Aqualog: An ontology-driven question answering system for organizational semantic intranets. *Web Semant.*, 5(2):72–105.
- [Martin et al., 1999] Martin, D., Cheyer, A., and Moran, D. (1999). The Open Agent Architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1/2):91–128.
- [Pfleger and Schehl, 2006] Pfleger, N. and Schehl, J. (2006). Development of Advanced Dialog Systems with PATE. In *Proceedings of the International Conference on Spoken Language Processing (Interspeech 2006 - ICSLP)*, Pittsburgh, PA.
- [Pieraccini and Huerta, 2008] Pieraccini, R. and Huerta, J. M. (2008). Where do we go from here? In Dybkjr, L. and Minker, W., editors, *Recent Trends in Discourse and Dialogue*, volume 39 of *Text, Speech and Language Technology*, pages 1–24. Springer, Dordrecht.
- [Ramachandran and Krishnamurthi, 2009] Ramachandran, V. A. and Krishnamurthi, I. (2009). Nlion: Natural language interface for querying ontologies. In *COMPUTE '09: Proceedings of the 2nd Bangalore Annual Compute Conference on 2nd Bangalore Annual Compute Conference*, pages 1–4, New York, NY, USA. ACM.
- [Seneff et al., 1999] Seneff, S., Lau, R., and Polifroni, J. (1999). Organization, Communication, and Control in the Galaxy-II Conversational System. In *Proceedings of Eurospeech '99*, pages 1271–1274, Budapest, Hungary.
- [Sonntag et al., 2009a] Sonntag, D., Deru, M., and Bergweiler, S. (2009a). Design and implementation of combined mobile and touchscreen-based multimodal web 3.0 interfaces. In *Proceedings of the International Conference on Artificial Intelligence (ICAI)*.
- [Sonntag et al., 2007] Sonntag, D., Engel, R., Herzog, G., Pfalzgraf, A., Pfleger, N., Romanelli, M., and Reithinger, N. (2007). *SmartWeb Handheld—Multimodal Interaction with Ontological Knowledge Bases and Semantic Web Services*, volume 4451 of *Lecture Notes in Computer Science*, pages 272–295. Springer.
- [Sonntag and Möller, 2009] Sonntag, D. and Möller, M. (2009). Unifying semantic annotation and querying in biomedical images repositories. In *Proceedings of the International Conference on Knowledge Management and Information Sharing (KMIS/IC3K)*.
- [Sonntag et al., 2009b] Sonntag, D., Wennerberg, P., Buitelaar, P., and Zillner, S. (2009b). *Cases on Semantic Interoperability for Information Systems Integration: Practices and Applications*, chapter Pillars of Ontology Treatment in the Medical Domain. Information Science Reference.