TOWARDS A PROCESS OF BUILDING SEMANTIC MULTIMODAL DIALOGUE DEMONSTRATORS

Daniel Sonntag and Norbert Reithinger
German Research Center for AI (DFKI), Stuhlsatzenhausweg 3, 66123 Saarbruecken
sonntag@dfki.de, reithinger@dfki.de

Keywords: User/Machine Dialogue, Semantic Data Model, Multimodal Interaction, Prototype Development, Usability Planes

Abstract: A generic integration framework should allow us to build practical dialogue systems for specific use case scenarios. While domain-specific dialogue systems are simpler to achieve than more general, open-domain conversational dialogue systems, the integration into use cases and demonstration scenarios requires a lot of difficult integration work, especially in multimodal settings where different user devices such as touchscreens and PDAs are used. The challenges for those systems include, apart from the dialogue modelling task, the integration modelling for specific use case and demonstration scenarios. This paper reports on dialogue system prototype development based on ontology communication structures and we draw special attention to the process of how to build demonstration systems that include a task-oriented, information-seeking, or advice-giving dialogue as an important fragment of practical dialogue system development.

1 INTRODUCTION

Over the last several years, powerful commercial off-the-shelf solutions for speech recognition (ASR) or speech synthesis (TTS) have been produced. Even entire voice user interface platforms have become available. The German Voice Award tested about 120 systems with system-controlled functionality between 2005 and 2009. However, more complex semantic discourse and dialogue infrastructures, where the user can for example interrupt the system, have only moderate success so far in the entertainment or industrial sectors. There are several reasons for this. For example, a user-initiative dialogue system is a complex AI system that cannot be easily constructed since many natural language processing components have to be integrated into a common framework. This and similar problems have been addressed by distributed dialogue system integration frameworks. Prominent examples of research 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. 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 tried to make the interaction and demonstration most attractive and effective (dialogue and task performance) when considering the short implementation cycles in large-scale development and demonstration projects especially for multimodal systems. Thereby, hub-and-spoke dialogue frameworks play a major role (Reithinger and Sonntag, 2005). 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. All messages transferred between internal and external components are based on RDF data structures which are modelled in a discourse ontology (also cf. (Sonntag, 2010)).

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

2 DIALOGUE FRAMEWORK

The main architectural challenges we encountered in implementing a new dialogue application 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.

In our experience, these challenges can be solved by implementing the core of a dialogue runtime environment, an *ontology dialogue platform* (ODP) framework and its platform API (the SemVox spin-off company offers 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)—ranging from simple HTTP-based REST services to Semantic Web Services, driven by declarative specifications. The dialogue framework essentially addresses the first problem of how to integrate and harmonise different natural language processing (NLP) components and third-party components for ASR and TTS.

Using a 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 workbench or toolbox is required, as depicted in figure 1 (right). 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 to develop use case and demonstration specific prototypes. Hence, the automated dialogue application testing tools play a major role. In this paper, we will focus on the top-down conceptual workflow with which the application evaluation/testing step is integrated, the prototype development process (section 3).

It is important to point out that the architectural decisions are based on customisation and prototype development issues that arise when dealing with end-to-end dialogue-based interaction systems for industrial dissemination or demonstration scenarios. We use an RDF store (Jena TDB[4]) with the RDF version of Yago (Suchanek et al., 2007) which is a structured representation of Wikipedia contents for answering domain-specific or domain-independent questions, respectively. As an intermediate query representation language, a syntax based on the SPARQL Inferencing Notation (SPIN[5]) is used. This is in effect a structured version of SPARQL, the RDF query language. SPARQL originally uses a custom plain-text/string query syntax similar to SQL. The SPIN format instead uses an RDFS vocabulary to represent the individual SPARQL terms. The RDF(S) structured syntax allows us to use rule engines and other RDF(S)-based tools to modify the actual eTFS query to build a backend-specific SPARQL query.

[4]http://jena.sourceforge.net/TDB/

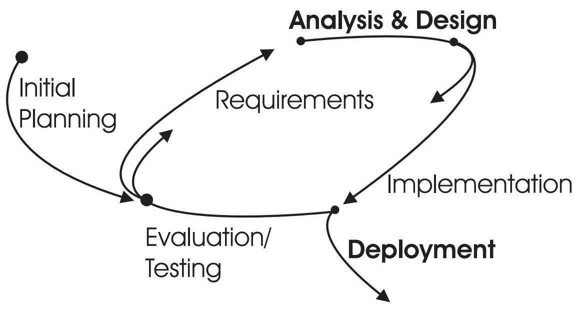

Figure 2: The prototype development process works best if done in partnership with representatives from the use cases for the evaluation step before deployment.

3 PROTOTYPE DEVELOPMENT PROCESS

Figure 2 shows a typical software development process which takes usability issues into account. It is a process with iterative steps, meaning the cycle is repeated but in a cumulative fashion. Please note that in our recommended form, the *analysis & design* and *implementation* steps, and the *evaluation/testing* and *requirements* steps timely overlap (work in each step is finished before work in the next step can finish instead of finishing the steps before the next step starts). The deployment step is often performed when the internal *evaluation/testing* cycle stops. A better approach is to involve representatives from the use cases for the evaluation/testing steps before the system is deployed at their organisation for testing. We will use this cycle as an abstract view of our prototype development process and focus on the *analysis & design* and *deployment* stages.

[5]http://spinrdf.org/

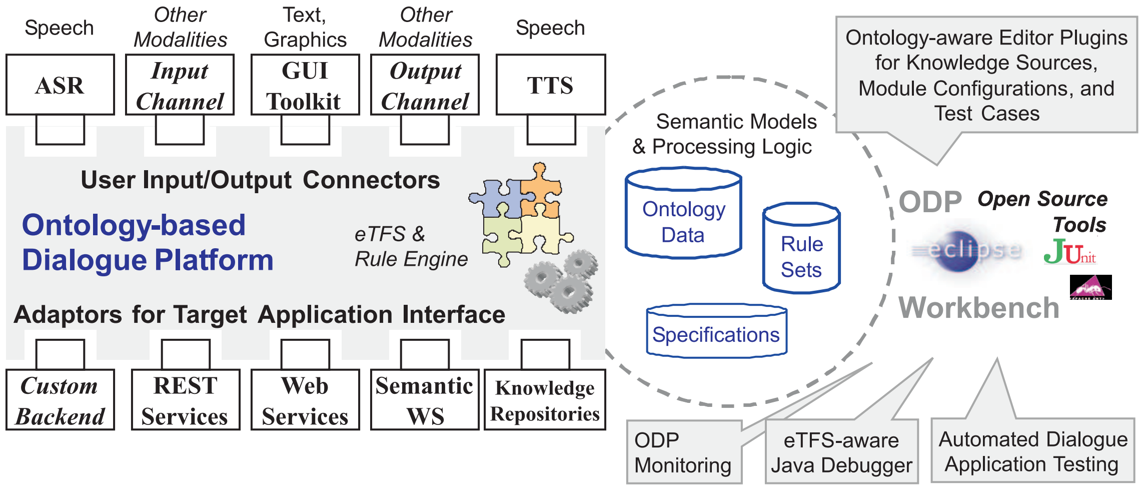

Figure 1: Conceptual architecture of the ontology-based dialogue platform (ODP) and our Eclipse workbench/toolbox. The workbench is used to specify the semantic dialogue management models and processing logic for specific demonstrator scenarios.

3.1 Analysis & Design

The analysis and design step of the prototype development process assumes the existence of several universal principles of design and a usability strategy which supports the iterative software development process.

Design Principles. We use design principles because demonstrator systems are mysteriously tied to the subconscious instincts and perceptions of the test user. We identified at least four such principles which we think apply to the prototype development process:

1. the 80/20 rule;
2. broad accessibility;
3. aesthetic effect;
4. alignment.

(1) The *80/20 rule* is a principal design rule which originally stated that eighty percent of the effects in large systems are caused by twenty percent of the variables, which suggest a link to normally distributed events (Juran, 1993). When applied to our demonstration scenarios, the rule means that 80 percent of a demonstrator's usage involves only 20 percent of the provided (dialogical) interaction competence or interface features. Accordingly, a semantic model should highlight the important features in specific usage contexts. (2) The principle of *Broad accessibility* asserts that research and demonstrator systems should be accessible and usable by people of diverse abilities and backgrounds. Hence, a global semantic user model can be used, and adaptations or modifications can be generally avoided. Of course, this principle cannot be applied generally without a certain controversy especially when designing business applications. Therefore, the accessibility concept has been extended to four standard characteristics of accessible designs: perceptibility, operability, simplicity, and forgiveness.[6] Interestingly, the perceptibility characteristic plays a major role when a trained presenter gives the presentation to a group of system evaluators; redundant presentation methods (e.g., textual, graphical and iconic) enhance the transparency of the system process. Likewise, forgiveness is implemented by ways to prevent usage errors (e.g., control buttons can only be used in the correct way) or by means to undo an action (e.g., an undo button or the speech command “Please go back to ...”). This characteristic is particularly welcome in situations where the prototype is presented to a greater audience, e.g., when exhibited at a fair. (3) The *aesthetic effect* describes the phenomenon that aesthetic designs are (subconsciously) more easily perceived and more effective at fostering a positive attitude toward the demonstrator system (also cf. works on apparent usability, as, e.g., in (Kurosu and Kashimura, 1995)). (4) Finally, the *alignment* principle concerns the relative placement of visual affordances on a graphical screen according to a semantic spatial model. Broadly speaking, related elements in the design should be aligned with related elements to create a sense of unity, cohesion or semantic relatedness (as is, for example, the case with medical pictures of body regions which exhibit

[6]Also cf. the Web Content Accessibility Guidelines 1.0, available at http://www.w3.org/TR/WCAG10/.

a semantically grounded spatial relationship according to the distribution of organs in a body). All principles should help to diminish the degree of misuse and misunderstanding based on the described factors while allowing for flexible presentation forms.

Usability Strategy. General usability guidelines express that a usable product is easy to learn, efficient to use, provides quick recovery from errors, is easy to remember, enjoyable to use, and visually pleasing. While these guidelines provide a nice abstract list of optimisation parameters, experience shows they are only seldom used to go through the prototype development cycle, as (Cronholm, 2009) points out. The analysis and design principles described in the previous paragraph offer more technical advice when following the development cycle. We used the analysis and design guidelines in combination with usability guidelines that consider five different planes (Garrett, 2002) in the development process. Every plane has its own issues that must be considered. From abstract to concrete, these are (1) the strategic plane, (2) the scope plane, (3) the structure plane, (4) the skeleton plane, and (5) the surface plane. In accordance with the software development process in figure 2, defining the users and their needs on the strategic plane is the first step in the design process. It is also useful to create personas that represent a special user group, e.g., the representatives of a specific business case. On the scope plane, then, you have to define the system's capacity (e.g., what a user should be able to say when using a multimodal Internet terminal for music download and exchange) and then the requirements for the technical dialogue components.

3.2 Deployment

The deployment step is the activity that makes a demonstrator system available for use in the use cases. Software deployment activities normally include the release, the modification of a software system that has been previously installed, etc. However, we focus on the deployment process that results in a new software distribution, demonstrator, or selected demo event during the project runtime. In large scale integration projects (i.e., projects with a runtime of three to five years) the distribution of these events is of particular interest. The requirements analysis and revised analysis at the beginning of the development process is in a timeframe of several months. This allows one or two runs through the prototype development process before revised requirements are specified (also cf. figure 2).

4 CONCLUSION

We described a specific dialogue system prototype development process and drew special attention to the process of how to build demonstration systems that include a task-oriented, information-seeking, or advice-giving dialogue. All implementations follow our ontology-based dialogue system framework (ODP) in order to provide a common basis for task-specific semantic-based processing.

The prototype development process includes design principles and a usability strategy. In our experience, obeying the *aesthetic effect* principle has a positive side effect on the implementation of the *broad accessibility* principle. This is particularly welcome in public demonstration scenarios.

REFERENCES

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.

Cronholm, S. (2009). The usability of usability guidelines. In *Proceedings of the 19th Australasian conference on Computer-Human Interaction Conference*.

Garrett, J. J. (2002). *The Elements of User Experience*. American Institute of Graphic Arts, New York.

Juran, J. M. & Gryna, F. M. (1993). *Quality planning and analysis : from product development through use*. McGraw-Hill, New York.

Kurosu, M. and Kashimura, K. (1995). Apparent usability vs. inherent usability: experimental analysis on the determinants of the apparent usability. In *CHI '95: Conference companion on Human factors in computing systems*, pages 292–293, New York, NY, USA. ACM.

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.

Reithinger, N. and Sonntag, D. (2005). An integration framework for a mobile multimodal dialogue system accessing the Semantic Web. In *Proceedings of INTERSPEECH*, pages 841–844, Lisbon, Portugal.

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, D. (2010). *Ontologies and Adaptivity in Dialogue for Question Answering*. AKA and IOS Press, Heidelberg.

Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: A Core of Semantic Knowledge. In *16th international World Wide Web conference (WWW 2007)*, New York, NY, USA. ACM Press.