A Flexible Wizard of Oz Environment for Rapid Prototyping

Stefan Scherer*, Petra-Maria Strauß†

* Institute of Neural Information Processing
† Institute of Information Technology
University of Ulm, 89069 Ulm, Germany
stefan.scherer@uni-ulm.de, petra-maria.strauss@uni-ulm.de

Abstract

This paper presents a freely available, and flexible Wizard of Oz environment for rapid prototyping. The system is designed to investigate the required features of a dialog system using the commonly used Wizard of Oz approach. The idea is that the time consuming design of such a tool can be avoided by using the provided architecture. The developers can easily adapt the database and extend the tool to the individual needs of the targeted dialog system. The tool is designed as a client-server architecture and provides efficient input features and versatile output types including voice, or an avatar as visual output. Furthermore, a scenario, namely restaurant selection, is introduced in order to give an example application for a dialog system.

1. Introduction

One of the major difficulties in designing a new dialog system is that researchers do not know the use cases of the system before the final version is ready, like answers to questions posed by the users the developers did not think of or unexpected wishes. Dozens of questions by the user can occur that the designers did not think of before. Therefore, it is necessary to conduct preliminary experiments with possible users in order to assess the full spectrum of interaction scenarios and required features, but with what system? Furthermore, it is necessary to receive feedback on usability issues at an early development stage in order to adapt the system accordingly. The most common answer to that question is to build a Wizard of Oz (WOZ) system, which is remotely controlled by one of the designers. Additionally, such WOZ systems are very useful in other research areas such as affective computing or emotion recognition, where differently by the wizard led dialogs are used to induce emotions. Other alternatives such as recording acted emotional speech are accompanied by a number of disadvantages, like overacted unrealistic emotions that may never occur in the targeted application (Burkhardt et al., 2005; Scherer et al., 2008). In these research fields it is necessary to gather realistic emotional data, e.g. emotional speech or facial expressions. A WOZ architecture can in these cases be used to induce emotions in the users of the dialog system, by directing the dialog in an emotion provoking way (Scherer et al., 2008; Strauss et al., 2008).

Building a WOZ system is very time consuming and therefore costly. Thus, in this work we propose a template architecture containing a client software, a server that is linked to the client, comprising all the data and rules, and possible output mechanisms, such as an avatar, text to speech, and runnable scripts, e.g. for a web browser. This work aims at providing dialog system developers a starting point for their WOZ environments saving valuable time. We do not aim at introducing a complete ready-to-use WOZ system, but consider it as very useful for rapid prototyping and a base for further customization. Due to the open source software the system may be extended in any direction.

This paper is organized as follows: Section 2. describes the requirements and the utilized software and hardware components as well as the necessary steps to customize the provided system, in Section 3. an example application for the WOZ system is presented, and Section 4. gives an overview of the implementation. Finally, Section 5. concludes and gives a perspective of further work.

2. Setup

The WOZ system is completely implemented in Java, providing a large flexibility in the choice of platforms. In Fig. 1 a general overview of the setup is shown. The central part of the setup is the wizard server which constitutes the dialog system itself, i.e. it runs on the computer the users directly interact with. It is a Java application running on the Sun Java Application Server available freely[1]. Additionally, SQL is running on the server. The developers are free to choose any distribution, such as the freely available MySQL[2]. If the intended application is not very large, as it was the case in the example scenario described in Section 3., one may alternatively use XML files to store the rules, vocabulary, and template texts, instead of SQL. The additional Resources folder contains easily exchangable avatar images, as well as all the files which can be displayed on the screen. The communication between server and client takes place via Ethernet.

The client is the front-end for the wizard. It transmits commands inducing queries, scripts, speech output, etc. to the server and receives the query results from the server.Each command is processed by using the keyboard or mouse attached to the client. Nothing but a simple Java string is transmitted over the Ethernet and interpreted by the server. During the initialization process, indicated by "on load" in Fig. 1, the client receives all the possible short commands and stores them in an auto complete combo box allowing the wizard to type the commands incredibly fast by only indicating fragments of the commands. Furthermore, no in-

[1] http://java.sun.com/
[2] http://www.mysql.com/

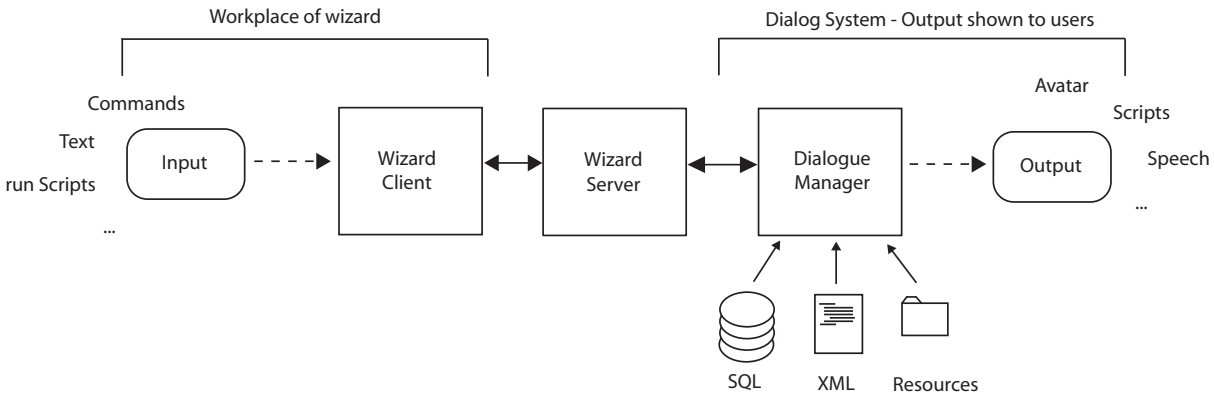


Figure 1: Schematic description of the WOZ system. Input is provided by the wizard to the client by typing and sending commands to the server. The server forwards the information to the dialogue manager which after performing the database queries etc. generates the system output. The server again communicates the output back to the client and wizard respectively.

valid or non-existent command can be typed by the wizard reducing errors enormously.
After the server interpreted such a command, it performs a database query and returns a list of possible matching database entries along with an appropriate template based system utterance, which can then be revised and adapted by the wizard, if needed, before being returned to the server for prompting. However, customized sentences for flexible speech output in any unforeseen dialog situation are also possible. The wizard can simply type in a text and send it to the server for prompting. Additionally, the client keeps track of all the transmitted and prompted utterances, which can be reloaded by double clicking with the mouse as well as using shortcuts on the keyboard.
In Fig. 1 the possible output modalities of the system are mentioned. For each prompt transmitted by the client the text to phoneme software txt2pho[3] is used to generate phoneme files interpreted by MBROLA to produce WAV-files to be played on the server. Further, the phoneme files are interpreted by the wizard server to produce harmonious avatar output. The avatar moves its mouth according to the given phonemes as seen in Fig. 2, and blinks randomly with one eye for some sort of a lively touch. As it will be explained in Section 3., scripts, such as starting a web browser and showing locations on maps or the schedule for the local bus are also possible and render the system quite flexible towards extensions.

Figure 2: Three different examples of phoneme based mouth positions.

For customizing the system for one's own application, adaptations need to be made in the following parts of the system. First, it is necessary to setup an SQL database or XML file, in order to define categories, data items, and templates for system utterances. For an example refer to Section 3. Secondly, all necessary software needs to be setup according to the particular requirements. MBROLA for example is quite flexible with regard to the language the system is built in, such as German, English, or any other available language[4]. Further, the images for the avatar have to be designed or exchanged. Mainly there are three types of images that need to be transparent for the avatar: head, eyes, and different mouth positions. Additionally, one can define scripts such as running a web browser and assign them to certain triggers that need to be implemented for the respective needs. A template class which can be adapted to the individual needs of the developers (in order to run customized scripts, start programs, etc.) is part of the open source implementation.
The hardware requirements are within reasonable limits. The software should run on any standard computer. Unfortunately, txt2pho only exists for Linux. Further, the developers have to find a way to transmit the audio and optionally the video signals from the test room to the wizard's room. In our case we used wireless microphones and a webcam. The video signal of the webcam was streamed using VLC's broadcasting capabilities[5] and received at the client via a second installation of VLC on the client computer.

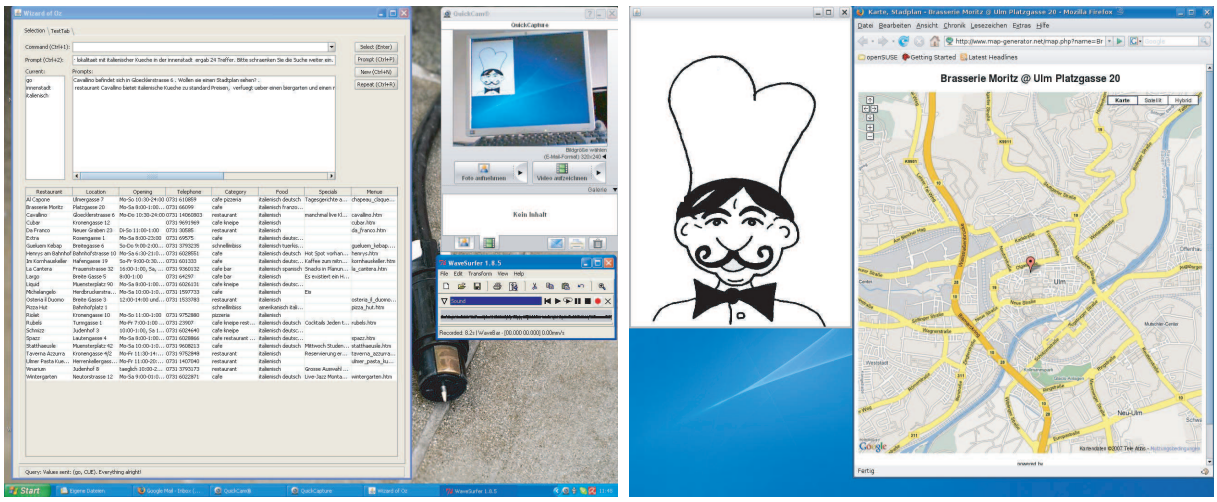
3. Example Scenario

This section gives an overview of an already implemented Wizard of Oz scenario using the system described above. The main scenario is a system, that helps two users find a desired restaurant in Ulm and surroundings (Strauss et al., 2006; Strauss et al., 2007). The system database contains around 100 restaurants described by several features, such as cuisine, type of locality, address, closest bus station, etc. In Fig. 4 an example dialog recorded within the restaurant selection scenario is shown. **U1** indicates the main user, who is directly communicating with the system (indicated

[3] `http://www.ikp.uni-bonn.de/dt/forsch/phonetik/hadifix/HADIFIXforMBROLA.html`

[4] For all available languages refer to: `http://tcts.fpms.ac.be/synthesis/mbrola.html`

[5] VLC is an easy to use multi platform video player freely available at: `http://www.videolan.org/vlc/`



(a) (b)

Figure 3: (a) Example screenshot of the Wizard client; (b) Example screenshot of the system (User perspective).

by **S**), and **U2** is the second user within the multi-user scenario. The second user is only allowed to talk indirectly via the main user with the dialog system. However, he plays an important role in the course of a dialog since the wishes and preferences of both users should be satisfied in the end.

The wizard follows the conversation and enters commands denoting the preferences mentioned by the users, e.g. "Italian" to specify Italian cuisine. Each input induces a database query upon which the selection of matching restaurants together with an appropriate system utterance are returned from the server. Besides determining user constraints for the query, the wizard can also select particular restaurants and supply any information about them which is contained in the database, e.g. the command "info" returns a string containing interesting information about a selected restaurant. The wizard may then prompt the automatically generated system responses that are returned to the client after the interpretation of every command.

A third type of commands trigger certain scripts to be started, like e.g. "showmap" results in opening a web browser and pointing to the address of the restaurant using Google maps. Fig. 3 shows the system as seen from the wizard's (a) and the user's perspective (b). The users see on the left of the screen the avatar and on the right side a map in a web browser pointing to the selected restaurant.

The wizard's perspective shows on the upper right side the image of the screen as seen by the users which is forwarded through the Ethernet using a webcam in order to check the output screen of the server. An audio recording tool records the conversation in the other room (lower right side). Finally, there is the client tool remotely controlling the server (on the left side). The design of the tool is very functional - there are only a few buttons that can be controlled by mouse or keyboard. The commands are entered into the field in the top row which is implemented as a drop down box listing all possible commands that are, as said before, automatically completed when typing the beginning of a word. The line below the command box depicts the system utterances before prompting, i.e. the automatically generated system responses for the wizard to check and edit if need, or custom utterances are entered by the wizard. Furthermore, the history of the prompts as well as the current constraints are shown above the result list, from which one can choose entries using the mouse or keyboard to load them as the selected restaurant which then enables the wizard to access the information about this restaurant.

U1: Computer, could you please give us a Mexican restaurant in the city centre of Ulm.
S: The query for Mexican food resulted in two hits. My suggestions are restaurant Flamenco and restaurant Peppers.
U2: Ask him where they are.
U1: Where are the Mexican restaurants?
S: Restaurant Flamenco offers Mexican and Spanish food for moderate prices, possesses a terrace and is located in the city centre next to the Danube. Restaurant Peppers offers ...
U1: In which street is the restaurant?
S: Flamenco is located at number eight Inselgasse.
U1: Great! How are the prices in this restaurant?
S: *Menu pops up on screen.*

Figure 4: Excerpt from one of the recorded dialogs. (Strauss et al., 2007)

In Fig. 5 an example XML entry for a restaurant and a respective grammar element also used for the query statements in the parsed XML tree or the SQL database are shown. The dots in the listings indicate that there are many more features available for the restaurants. The grammar part identifies all possible variations of words and synonyms that the system should understand and which are used for the database query. Only the XML file has to be adapted to define the domain, not the program code itself.

4. System architecture

The implementation of the WOZ system is kept as simple as possible. The system comprises three parts: the Wizard Client as shown in Fig. 1 only contains one important central class named *WizardClientForm* resembling the graphical user interface (GUI) of the client application. The

Restaurant entry:

```
<restaurant>
    <name>Asia Wan am Muensterplatz</name>
    <adresse>Muensterplatz 14</adresse>
    <telefonnummer>0731 1537371</telefonnummer>
    ...
    <preisklasse>gehoben</preisklasse>
    ...
</restaurant>
```

Grammar entry of price category ("preisklasse"):

```
<rule id="preisklasse">
    <one-of>
        <item> <tag>val='gehoben'</tag>gehoben</item>
        ...
        <item> <tag>val='standard'</tag>standard</item>
    </one-of>
</rule>
```

Figure 5: Example restaurant and category entry, in this case price category.

permitted commands (which are loaded upon initialization) and system utterances are selected and sent to the *WizardWebServer* (Fig. 1) as common Java *Strings*. Furthermore, the query results from the server are received, and displayed in a table to give an overview of possible restaurants and additional information comprising cuisine, category, location, etc. The *WizardWebServer* adopts the task of communicating the commands and query results between the *WizardClient* and the *DialogManager*. The coarse architecture of the *DialogManager* is shown in Fig. 6.

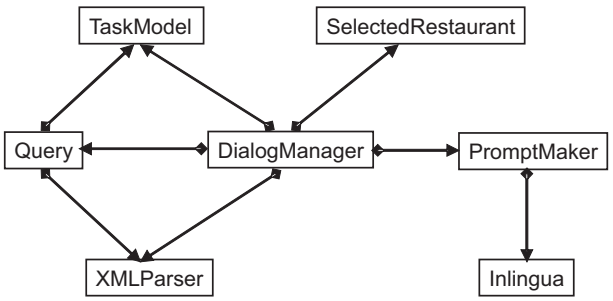


Figure 6: Class diagram of the system architecture around the DialogManger.

According to the input supplied by the *WizardWebServer* the *DialogManager* induces different actions. The user constraints are stored in the *TaskModel* which defines the search criteria for the database query. The *Query* class, induced by the *DialogManager*, performs the query on the data supplied by the the *XMLParser* which constitutes the connection to the database[6]. The *DialogManager* analyses the list of query results and sends it back to the *WizardClientForm* (via the *WizardWebServer*). In the case that the query resulted in one single restaurant, or upon the wizard selecting a particular restaurant, this restaurant is loaded in *SelectedRestaurant*, i.e. the *DialogManager* retrieves the data from the *XMLParser* and is now able to promptly supply any information available about this restaurant or perform further possible actions such as displaying the menu, showing the location on a map, or providing bus connections. System utterances are generated by the *PromptMaker* in communication with *Inlingua* which is responsible for the language specific properties, such as providing different cases, gender specific endings, singular or plural endings, etc. for German. The necessary operations are listed in *Inlingua* and the whole words or endings are listed in property files that are loaded on startup. It is quite easy to expand the system with different languages like English, by providing the particular property files. Finally, the *DialogManager* induces the system output such as speech synthesis, scripts, and avatar.

[6]In this example the data is stored in an XML file.

5. Conclusions

This paper introduced an environment for rapid prototyping of a Wizard of Oz setup for the development of dialog systems. The benefits are clearly the lowered amount of work that is necessary for implementing such an environment. The only tasks that are left for the developers is to customize the database, the avatar, and additional scripts according to the targeted application and domain. The presented work provides a setup in a useful balance of flexibility and already implemented features. The environment provides simple exchangable speech output in many different languages, simple visual output by using an avatar, and optional script extensions, such as starting programs on the server. The client is an easy to use and effective tool for remotely controlling the server. However, there are still interesting ideas left that will be implemented in the near future, such as automatic database generation using existing datasets, e.g. from web pages.

6. References

F. Burkhardt, A. Paeschke, M. Rolfes, W. Sendlmeier, and B. Weiss. 2005. A database of german emotional speech. In *Proceedings of Interspeech 2005*.

S. Scherer, F. Schwenker, and Palm G., 2008. *Emotion recognition from speech using multi-classifier systems and RBF-ensembles*, chapter 3, pages 49–70. Studies in Computational Intelligence. Springer.

P.-M. Strauss, H. Hoffmann, H. Neumann, W. Minker, G. Palm, S. Scherer, F. Schwenker, H. Traue, and U. Weidenbacher. 2006. Wizard-of-oz data collection for perception and interaction in multi-user environments. In *Proceedings of International Conference on Language Resources and Evaluation (LREC)*.

P.-M. Strauss, H. Hoffmann, and S. Scherer. 2007. Evaluation and user acceptance of a dialogue system using wizard-of-oz recordings. In *Proceedings of Intelligent Environments 07*.

P.-M. Strauss, H. Hoffmann, W. Minker, H. Neumann, G. Palm, S. Scherer, H. Traue, and U. Weidenbacher. 2008. The pit corpus of german multi-party dialogues. In *Proceedings of LREC 2008*.