"Help Me, I Need More User Tests!" User Simulations as Supportive Tool in the Development Process of Spoken Dialogue Systems

Florian Kretzschmar, Sebastian Möller

Quality & Usability Lab, Telekom Innovation Laboratories, TU Berlin

florian.kretzschmar@telekom.de, sebastian.moeller@telekom.de

Abstract

In this paper we present our experiences in developing a spoken dialogue system supported by tests with a user simulation. Since the code of dialogue systems with modest complexity can easily get unclear, it is almost impossible to deliver error-free systems without user tests in the development process. We show how we included our user simulation environment SpeechEval in the development process of three VoiceXML dialogue systems and discuss advantages and drawbacks compared to tests with real users.

Index Terms: spoken dialogue systems, user simulation, user tests

1. Introduction

Although human-computer interaction via speech is still perceived as critically error-prone and cumbersome by end users, the availability of high quality systems with natural language understanding capability is increasing throughout the last years. Higher processing power and faster network connections to speech servers make complex human-computer interaction via speech on mobile devices possible, which is endorsed by mobile phone providers like Apple or Google, who offer usage of large scale speech recognition on their servers. But with the increasing possibilities to offer complex natural-language dialogue systems, the necessity for extensive usability testing grows. For the developer it gets harder to deliver bug-free, user friendly systems that make use of the possibilities of state-of-the-art speech technology.

To illustrate the problem, Table 1 shows some information about a quite simple mixed-initative VoiceXML system the authors developed in three system versions which differ only in the confirmation strategy. The number of fields, grammars, variables and conditional clauses makes it almost impossible to develop such systems without user tests. Since tests with real users are expensive and time-consuming, it is common practice to conduct one test at the end of the development process instead of integrating the user tests into an iterative process of development and testing, which is however recommended in user-centered development[1]. User simulations are already proposed as a potential addition to traditional user tests, for instance by [2], but the simulation requires a dedicated model of the test system the simulation can interact with on text level, which is not usable for real user tests.

	Confirmation Strategy		
	Summary	**Explicit**	**Implicit**
Lines of code	703	797	1000
# Fields	9+	26+	21+
# Grammars	8	14	13
# Variables	25+	26+	29+
# Conditions	73+	89+	138+

Table 1: Source code statistics of the three VoiceXML system versions. The plus means, that the actual number depends on the dynamically generated parts.

We want to demonstrate in this paper, how a user simulation can help during the development of SDSs to find bugs that would otherwise probably be unnoticed, exemplified with the user simulation workbench SpeechEval[3]. We also want to show that it can help to compare system versions before they are tested with real users.

More information about the system we developed in three versions follows in Section 2. A short descriptions of the user simulation follows in Section 3. In Section 4 we show how we used the simulation in the development process of the three system versions, and in Section 5 we compare the dialogs generated with the user simulation on the final systems with data obtained with real users. In the final section we discuss the benefits and drawbacks of our method.

2. Test Systems

We developed three versions of an SDS that provide restaurant information. The language of the systems is German, but for sake of understandability everything in this paper is translated into English. All system versions provide the address of restaurants in Bochum, Germany, based on 5 information pieces the user has to provide: the day the caller wants to eat (today, tomorrow,

Slot	# Values	Examples
day	11	Monday,Today
time	35	Lunch, 6pm
location	62	City Centre, University
price	42	Low, 10 Euro
cuisine	20	Regional, Italian

Table 2: Slot names and the number of possible values for each slot.

etc.), the time (lunch, dinner), the city district, the price range (low, middle or high priced) and the cuisine (german, chinese, italian, etc.). Table 2 shows the slots and the total number of possible inputs per slot. All three versions implement a mixed-intiative dialogue where the user can give as much information as he/she wants in the first utterance, and the system asks for missing information afterwards. The version differ in their confirmation strategy: One system confirms all input at once after it has all information pieces (summary), one system confirms after each input (explicit), and the third system confirms all input implicitly as part of the next system prompt. We illustrate the difference with this example:

System: "What can I do for you?"
User: "I am looking for a chinese restaurant in the city centre."
System:

- Explicit: "You said chinese cuisine in the city centre, is that correct?"
- Implicit: "You said chinese cuisine in the city centre. When do you want to go?"
- Summary: "When do you want to go?" ... after all information is collected: "So you want to eat today evening in the city centre chinese cuisine in a lower price range. Is that correct?"

3. User Simulation

The user simulation works on the speech level. This means, that it recognizes system prompts via ASR, analyses the system prompt through natural language understanding, and an action planner component determines the suitable user utterance on the dialogue act level. A natural language generation component then generates the specific user utterance which is synthesized by a TTS component.

This method has a few advantages and drawbacks: The simulation works as soon as a callable system mockup exists. Information about dialog acts, slot and their values is directly integrated in the grammars for the ASR of the simulation, thus it can be used during the development of any SDS without much additional work. Nevertheless, there is some work to be done before the simulation can be used. First, the developer has to define grammars with the system prompts and corresponding meta information for the speech recognizer of the simulation. Furthermore the simulation benefits from real user utterances for training. Although it works without training data, the diversity of generated dialogs will be much higher when the simulation can vary the utterances based on real data from the same system or other systems of the same domain. One example: A simulation is being run with the goal to get a restaurant at a specific location. The simulation recognizes the question "Where do you want to eat?", which is the dialogue act request_info(location). If there is no prior training with real user utterances, the NLG component always uses the location from the predefined goal as answer. If the simulation is trained with for instance 100 answers to this question, it takes randomly an answer from the corpus, which results in more diverse dialogs.

The database consists of mappings between dialogue acts and user utterances that are being made generic where possible. The user utterance "I want to eat chinese food in the city centre." is stored as "I want to eat FOODTYPE food in LOCATION." If it is not possible to insert other slot values than the original, the utterance is stored as is, for instance "I do not want to spend much." We filled the database with over 1000 utterances from a previous experiment with a predecessor system from 2005 [4] and used this data to train our simulation. For a more detailed description of the user simulation we refer to Scheffler et al.[5]

4. Development Process

A feasible approach to the development of modern spoken dialogue systems involves user testing during the development to constantly find bugs that are not obvious to the developer. Since tests with real users are hardly possible multiple times during the development, a user simulation is a reasonable substitute. The system can constantly be tested for bugs (i.e. syntactical errors in the source code), recognition accuracy and correct dialogue flow just by generating a number of dialogs once a new function is implemented.

We used this process for the development of the three system versions described in Section 2. First we developed the systems until we considered them finished, and ran the user simulation on them with a typical task: Find a chinese restaurant in the city centre. We found several problems in the dialogs by listening to the dialogs, and by analyzing the log files, both the ones of the simulation and the ones of the systems. After we fixed the problems, we ran the simulation again, and repeated this until no problems occurred anymore throughout 20 dialogues.

We found the following types of problems that are further explained in the next paragraphs: Unfilled/wrongly filled variables, dead ends, wrong database queries, wrong dialogue flow due to erroneous if-conditions and problematic grammars (i.e. many misunderstandings).

Variables hold dynamic content like user answers so that they can be integrated into prompts, for instance con-

firmations, and can be used in conditional if-clauses for determining the next system action. Because variables are dynamically filled, it is not easy to foresee if a variable will be correctly filled at the time it is used. For instance, after the first prompt in a mixed-intiative system, it is not clear which slots the user fills with his utterance, or sometimes it is necessary to fill a variable at one stage, and delete it after another. This could lead to bugs due to non- or wrongly filled variables. With the help of the user simulation, a lot (if not all) of these errors can be identified. In the development of the systems introduced in Section 2, we found several bugs of this kind, especially after mis- and non-understandings, and after switching between several VXML files.

Dead ends in the dialogs occur, when the dialogue does not switch to the next dialogue step when it is supposed to do so. Again, the increasing number of if–conditions can make it hard to see if the dialogue behaves like intended by the developer. A call with the user simulation can be automatically terminated after a given amount of time, making it easy to identify loops and dead ends by just looking at the dialogs that were terminated due to the dialogue length. We identified this kind of errors especially at dialogue states where the next action - depending on the user answer - can be a previous step, for instance corrections.

Databases are a necessary part in many modern spoken dialogue systems. The incorporation of information from databases includes querying the server for the data with the correct query, and building dynamic system prompts based on all the different information the database can return, or reacting adequately when there is no result from the database. If the developer starts the user simulation with a variety of goals, different queries to the database can be tested. The developers can then identify erroneous queries in the logs of the database, or false system prompts resulting from unexpected results from the database in the dialog logs. We found with this method bugs in MySQL-Select-Statements and in the PHP script we used to dynamically create the restaurant list.

Modern SDSs have to understand possibly thousands of different sentences. Thus it is not uncommon that words that are hard to differentiate are part of the grammars, for instance the (made up) example “cheap” and “sheep”. Problematic words is not easy to identify, especially when statistical grammars are used. In our development, we found out that the German words “ändern” and “beenden” or “chinesisch” and “indisch” were often mixed up when uttered by the simulation’s TTS. Although words that are problematic for synthesized speech are not necessarily hard to distinguish when spoken by real users, it can draw the attention of the developer to these phrases or words, and further testing with some real testers can be targeted towards these problems.

	Sum		**Expl**		**Impl**	
	Real	**Sim**	**Real**	**Sim**	**Real**	**Sim**
# Dialogs	10	20	10	20	10	20
Dur.(sec.)	123	160	136	98	135	116
# Turns	16.6	25	26.2	16.4	19	15.0
# Err/Dlg	0.7	2.6	1.6	1.1	2	1.9
Task Succ.	0.95	1.0	1.0	0.77	0.9	0.7

Table 3: Dialogue parameters from real and simulated users averaged over each system.

5. Comparison with Real Users

We used the user simulation on the final systems after the development finished. To be able to compare the evaluation with a real user tests, we also let 10 real users interact with the systems. Since the three systems are basically the same except for the confirmation strategies, we ruled out the learning effect to be expected in an subjective test by varying the order in which the systems are being presented. The (simulated and real) users did the same task (“Find a cheap chinese restaurant in the city centre.”) with all three systems. Afterwards we calculated the dialogue parameters for each system as shown in Table 3.

The user simulation produced more errors (i.e. mis- and non-understandings) than the real users on all systems. This can be explained by the corpus we used for training the utterances. It has a lot of longer sentences, and fill words like “hm” are also transcribed. The simulation utters the same amount of information in the first utterance as the real users in the corpus, which means it utters one slot with the probability of 0.51, two with 0.13, three with 0.11, four with 0.15 and five slots with a probability of 0.1. The utterances with 4 to 5 information pieces (for instance “I, uhm, want to go today to a, uhm, cheap chinese restaurant in the city centre.”) tend to be misrecognized, because synthesized by a TTS they often sound worse than uttered by real users. The dialogs of the simulated users are - despite having more errors - shorter for the systems with explicit and implicit confirmation. This can partly be due to the very short reaction time of the simulation. The answer of the simulated user is uttered directly after the system prompt finished, whereas the real users often need time to answer. Also the task success shows, that there were problems in the interaction with the explicit and implicit confirmation systems. A closer look on the dialogs revealed, that the simulation tried to correct an error at a certain point in the dialogue by saying “no”, but at that point the system wrongly terminated the dialogue with that answer.

One important question is how many system states the simulation tests. The analysis of the simulated data revealed, that the simulation indeed visited all possible system states for all systems *before* the list of restaurants was generated, but only for the system with implicit confirmation the simulation had to change slot values *after* the first

presentation of restaurants. This is because the user simulation always disconfirms misunderstandings on the systems with summary confirmation and explicit confirmation directly after the system asks for confirmation. Since both system version asks for confirmation of all slots before it presents the restaurant information, no errors have to be resolved afterwards, although it would be possible. On the other hand, the simulation fails to resolve all errors when interacting with the system with implicit confirmation before the restaurant list is generated, because it only resolves one false slot at a time. This leaves false slots, if two or more slots were wrongly recognized at once. In that case, the simulation tries to change the wrongly filled slots after the presentation of the restaurant list . Thus, although the simulation produces diverse dialogs, it is still necessary to check manually if the simulation ran through all dialogue states that need to be tested. See Section 6 for an idea how to resolve this issue.

6. Discussion

In this paper we described, how an end-to-end user simulation can effectively help spoken dialogue system developer to find bugs in the implementation phase of a system. We developed three systems and integrated the user simulation in an iterative process of developing parts of the system, and testing them with simulated users. We found a variety of problems which are mostly not directly visible due to the usage of dynamic data, i.e. variables, databases, and complex dialogue flows involving a high number of if-then-else conditions. The problems we found by using the simulation include wrong dialogue flows, wrongly filled variables, errors in database queries and errors in the PHP scripts that generate dynamic VXML files. As an advantage over user simulations that need special system model to be able to interact with the system, the SpeechEval workbench can be used with the same prototypes that are used for tests with real users, since it uses TTS and ASR to communicate with the test system.

Nevertheless, some drawbacks have to be discussed: First of all, the user simulation has to be prepared with ASR grammars that consist of the system prompts and annotated slots. Since it recognizes the system prompts through an ASR component, and most of the natural language understanding is done in the grammars, the developer has to put some efforts into tuning the grammars of the simulation to achieve optimal recognition rates. But this is mostly an effort before the first usage of the simulation, since the grammars have to be only adjusted later on for new system versions.

A second drawback is, that the quality of the simulated dialogues is partly dependent on training data. If training data from interactions of real users with the same system or a system from the same domain is available, the likeliness of generating varying dialogues is much higher, hence the likeliness of finding problems is higher. This is especially a problem for completely new systems where such a corpus is not existent. Although it would be possible to hand-craft such a corpus by an expert, the realism of user test data can presumably not be achieved.

The third issue we want to discuss is the number of tested system states. One advantage of simulations compared to real user tests is, that the amount of generated data is theoretically infinite. One can generate thousands of dialogues to test a system. But the simulation is most valuable, when all possible system states are indeed triggered by the simulation. Although we had a corpus of real utterances which helps the simulation to produce diverse dialogues, some system states could not be visited due to the strategy of the simulation. In future versions, a solution to the problem might be an option to try to resolve a variable number of errors later in the dialogue, if the test system allows to do that.

We believe that the proposed procedure using the user simulation during the development - and tests with real users at the end of the development - can reduce the development time and costs significantly. In the future, we will try to quantify this amount of time and cost reduction by quantifying the amount and quality of errors we can find with the help of the simulated dialogues, and we will compare the pre-trained user simulation with a non-trained system, to also take into account that no dialogue data is available for completely new systems without predecessor versions. We also want to distinguish between different types of simulated users that differ in their behaviour in the future. The interaction behaviour can differ in the length of the utterances of the user, or using barge-in or not, or the strategy how errors are handled by the user. It is desirable to be able to adjust the behaviour of the simulated users, for example to simulate different cooperativeness levels [6].

7. References

[1] Nielsen, J. “Usability Engineering”, Academic Press, Boston, MA, 1993

[2] Engelbrecht, K.-P., “Estimating Spoken Dialog System Quality with User Models”, PhD Thesis, Quality & Usability Lab, TU Berlin, 2011

[3] Möller, S., Schleicher, R., Butenkov, D., Engelbrecht, K.-P., Gödde, F., Scheffler, T., Roller, R. and Reithinger, N., “Usability engineering for spoken dialogue systems via statistical user models”, First International Workshop on Spoken Dialogue Systems Technology (IWSDS 2009), Kloster Irsee, Germany, Dec. 2009

[4] Möller, S., “Quality of Telephone-Based Spoken Dialogue Systems”, Springer, New York, 2005

[5] Scheffler, T., Roller, R., Reithinger, N., “Speecheval - Evaluating spoken dialog systems by user simulation”, Proceedings of the 6th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems, Pasadena, CA (2009) 93–98

[6] López-Cózar, Ramón and Callejas, Zoraida and McTear, Michael, “Testing the performance of spoken dialogue systems by means of an artificially simulated user”, in Artificial Intelligence Review, 26(4):291–323, Springer Netherlands, 2006