Technical Support Dialog Systems: Issues, Problems, and Solutions

Kate Acomb, Jonathan Bloom, Krishna Dayanidhi, Phillip Hunter, Peter Krogh, Esther Levin, Roberto Pieraccini
SpeechCycle
535 W 34th Street
New York, NY 10001
{kate,jonathanb,krishna,phillip,peter,roberto}@speechcycle.com
esther@spacegate.com

Abstract

The goal of this paper is to give a description of the state of the art, the issues, the problems, and the solutions related to industrial dialog systems for the automation of technical support. After a general description of the evolution of the spoken dialog industry, and the challenges in the development of technical support applications, we will discuss two specific problems through a series of experimental results. The first problem is the identification of the call reason, or *symptom*, from loosely constrained user utterances. The second is the use of data for the experimental optimization of the Voice User Interface (VUI).

1 Introduction

Since the beginning of the telephony spoken dialog industry, in the mid 1990, we have been witnessing the evolution of at least three generations of systems. What differentiates each generation is not only the increase of complexity, but also the different architectures used. Table 1 provides a summary of the features that distinguish each generation. The early first generation systems were mostly informational, in that they would require some information from the user, and would provide information in return. Examples of those systems, mostly developed during the mid and late 1990s, are package tracking, simple financial applications, and flight status information. At the time there were no standards for developing dialog systems, (VoiceXML 1.0 was published as a recommendation in year 2000) and thus the first generation dialog applications were implemented on proprietary platforms, typically evolutions of existing touch-tone IVR (Interactive Voice Response) architectures.

Since the early developments, spoken dialog systems were implemented as a graph, called *call-flow*. The nodes of the call-flow typically represent actions performed by the system and the arcs represent an enumeration of the possible outcomes. Playing a prompt and interpreting the user response through a speech recognition grammar is a typical action. Dialog modules (Barnard et al., 1999) were introduced in order to reduce the complexity and increase reusability of call-flows. A Dialog Module (or DM) is defined as a call-flow object that encapsulates many of the interactions needed for getting one piece of information from the user, including retries, timeout handling, disambiguation, etc. Modern commercial dialog systems use DMs as their active call-flow nodes.

The number of DMs in a call-flow is generally an indication of the application complexity. First generation applications showed a range of complexity of a few to tens of DMs, typically spanning a few turns of interaction.

The dialog modality is another characterization of applications. Early applications supported strict directed dialog interaction, meaning that at each

	GENERATION		
	FIRST	**SECOND**	**THIRD**
Time Period	1994-2001	2000-2005	2004-today
Type of Application	Informational	Transactional	Problem Solving
Examples	Package Tracking, Flight Status	Banking, Stock Trading, Train Reservation	Customer Care, Technical Support, Help Desk.
Architecture	Proprietary	Static VoiceXML	Dynamic VoiceXML
Complexity (Number of DMs)	10	100	1000
Interaction Turns	few	10	10-100
Interaction Modality	directed	directed + natural language (SSLU)	directed + natural language (SSLU) + limited mixed initiative

Table 1: Evolution of spoken dialog systems.

turn the system would *direct* the user by proposing a finite—and typically small—number of choices. That would also result in a limited grammar or vocabulary at each turn.

The applications of the second generation were typically transactional, in the sense that they could perform a transaction on behalf of the user, like moving funds between bank accounts, trading stocks, or buying tickets. Most of those applications were developed using the new standards, typically as collections of VoiceXML documents. The complexity moved to the range of dozens of dialog modules, spanning a number of turns of interactions of the order of ten or more. At the same time, some of the applications started using a technology known as Statistical Spoken Language Understanding, or SSLU (Gorin et al., 1997, Chu-Carroll et al., 1999, Goel et al, 2005), for mapping loosely constrained user utterances to a finite number of pre-defined semantic categories. The *natural language* modality—as opposed to directed dialog—was initially used mostly for call-routing, i.e. to route calls to the appropriate call center based on a more or less lengthy description of the reason for the call by the user.

While the model behind the first and second generations of dialog applications can be described by the form-filling paradigm, and the interaction follows a pre-determined simple script, the systems of the third generation have raised to a qualitatively different level of complexity. Problem solving applications, like customer care, help desk, and technical support, are characterized by a level of complexity ranging in the thousands of DMs, for a number of turns of dynamic interaction that can reach into the dozens. As the sophistication of the applications evolved, so did the system architecture by moving the logic from the client (VoiceXML browser, or voice-browser) to the server (Pieraccini and Huerta, 2005). More and more system are today based on generic dialog application server which interprets a dialog specification described by a—typically proprietary—markup language and serve the voice-browser with dynamically generated VoiceXML documents. Finally, the interaction modality of the third generation systems is moving from the strictly directed dialog application, to directed dialog, with some natural language (SSLU) turns, and some limited mixed-initiative (i.e. the possibility for the user to change the course of the dialog by making an unsolicited request).

2 Technical Support Applications

Today, automated technical support systems are among the most complex types of dialog applications. The advantage of automation is clear, especially for high-volume services like broadband-internet, entertainment (cable or satellite TV), and telephony. When something goes wrong with the service, the only choice for subscribers is to call a technical support center. Unfortunately, staffing a call center with enough agents trained to help solve even the most common problems results in prohibitive costs for the provider, even when outsourcing to less costly locations End users often experience long waiting times and poor service from untrained agents. With the magnitude of the daily increase in the number of subscribers of those services, the situation with human agents is bound to worsen. Automation and self-service can, and does, help reduce the burden constituted by the most frequent call reasons, and resort to human agents only for the most difficult and less common problems.

However, automating technical support is particularly challenging for several reasons. Among them:

- Troubleshooting knowledge is not readily available in a form that can be used for automation. Most often it is based on the idiosyncratic experience of the individual agents.
- End users are typically in a somewhat emotionally altered state—something for which they paid and that is supposed to work is broken. They want it repaired quickly by an expert human agent; they don't trust a machine can help them.
- The description of the problem provided by the user can be imprecise, vague, or based on a model of the world that may be incorrect (e.g. some users of internet cannot tell their modem from their router).
- It may be difficult to instruct non-technically savvy users on how to perform a troubleshooting step (e.g. *Now renew your IP address*.) or request technical information (e.g. *Are you using a Voice over IP phone service?*)
- Certain events cannot be controlled. For instance, the time it would take for a user to complete a troubleshooting step, like rebooting a PC, is often unpredictable.
- The acoustic environment may be challenging. Users may be asked to switch their TV on, reboot their PC, or check the cable connections. All these operations can cause noise that can trigger the speech recognizer and affect the course of the interaction.

On the other hand, one can leverage the automated diagnosis or troubleshooting tools that are currently used by human agent and improve the efficiency of the interaction. For instance, if the IP address of the digital devices at the user premises is available, one can ping them, verify their connectivity, download new firmware, and perform automated troubleshooting steps in the background without the intervention of the user. However, the interplay between automated and interactive operations can raise the complexity of the applications such as to require higher level development abstractions and authoring tools.

3 High Resolution SSLU

The identification of the call reason—i.e. the problem or the symptoms of the problem experienced by the caller—is one of the first phases of the interaction in a technical support application. There are two possible design choices with today's spoken language technology:

- **Directed dialog**. A specific prompt enumerates all the possible reasons for a call, and the user would choose one of them.
- **Natural Language:** An open prompt asks the user to describe the reason for the call. The utterance will be automatically mapped to one of a number of possible call reasons using SSLU technology.

Directed dialog would be the preferred choice in terms of accuracy and cost of development. Unfortunately, in most technical support applications, the number of call-reasons can be very large, and thus prompting the caller through a directed dialog menu would be impractical. Besides, even though a long menu can be structured hierarchically as a cascade of several shorter menus, the terms used for indicating the different choices may be misleading or meaningless for some of the users (e.g. *do you have a problem with hardware, software, or networking?*). Natural language with SSLU is generally the best choice for problem identification.

In practice, users mostly don't know what the actual problem with their service is (e.g. *modem is wrongly configured*), but typically they describe their observations—or *symptoms*—which are observable manifestations of the problem. and not the problem itself (e.g. symptom: *I can't connect to the Web*, problem: modem wrongly configured). Correctly identifying the symptom expressed in natural language by users is the goal of the SSLU module.

SSLU provides a mapping between input utterances and a set of pre-determined categories. SSLU has been effectively used in the past to enable automatic call-routing. Typically call-routing applications have a number of categories, of the order of a dozen or so, which are designed based on the different *routes* to which the IVR is supposed to dispatch the callers. So, generally, in call-routing applications, the categories are known and determined prior to any data collection.

One could follow the same approach for the problem identification SSLU, i.e. determine a number of a-priori *problem* categories and then map a collection of training *symptom* utterances to each one of them. There are several issues with this approach.

First, a complete set of categories—the problems—may not be known prior to the acquisition and analysis of a significant number of utterances. Often the introduction of new home devices or services (such as DVR, or HDTV) creates new problems and new symptoms that can be discovered only by analyzing large amounts of utterance data.

Then, as we noted above, the relationship between the problems—or broad categories of problems—and the manifestations (i.e. the symptoms) may not be obvious to the caller. Thus, confirming a broad category in response to a detailed symptom utterance may induce the user to deny it or to give a verbose response (e.g. Caller: *I cannot get to the Web.* System: *I understand you have a problem with your modem configuration, is that right?* Caller: *Hmm…no. I said I cannot get to the Web.*).

Finally, caller descriptions have different degrees of specificity (e.g. *I have a problem with my cable service* vs. *The picture on my TV is pixilated on all channels*). Thus, the categories should reflect a hierarchy of symptoms, from vague to specific, that need to be taken into proper account in the design of the interaction.

As a result from the above considerations, SSLU for symptom identification needs to be designed in order to reflect the *high-resolution* multitude and specificity hierarchy of symptoms that emerge from the analysis of a large quantity of utterances. Figure 1 shows an excerpt from the hierarchy of symptoms for a cable TV troubleshooting application derived from the analysis of almost 100,000 utterance transcriptions.

Each node of the tree partially represented by Figure 1 is associated with a number of training utterances from users describing that particular symptom in their own words. For instance the top-most node of the hierarchy, "TV Problem", corresponds to vague utterances such as *I have a problem with my TV* or *My cable TV does not work*. The" Ordering" node represents requests of the type *I have a problem with ordering a show*, which is still a somewhat vague request, since one can order "Pay-per-view" or "On-demand" events, and they correspond to different processes and troubleshooting steps. Finally, at the most detailed level of the hierarchy, for instance for the node "TV Problem-Ordering-On Demand-Error", one finds utterances such as *I tried to order a movie on demand, but all I get is an error code on the TV.*

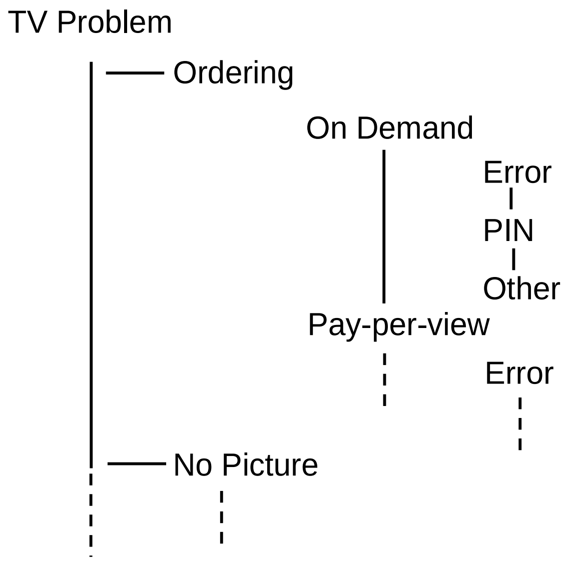

Figure 1: Excerpt from the hierarchical symptom description in a cable TV technical support application

In the experimental results reported below, we trained and tested a hierarchically structured SSLU for a cable TV troubleshooting application. A corpus of 97,236 utterances was collected from a deployed application which used a simpler, non hierarchical, version of the SSLU. The utterances were transcribed and initially annotated based on an initial set of symptoms. The annotation was carried out by creating an *annotation guide* document which includes, for each symptom, a detailed verbal description, a few utterance examples, and relevant keywords. Human annotators were instructed to label each utterance with the correct category based on the annotation guide and their work was monitored systematically by the system designer.

After a first initial annotation of the whole corpus, the annotation consistency was measured by computing a cluster similarity distance between the utterances corresponding to all possible pairs of symptoms. When the consistency between a pair of symptoms was below a given threshold, the clusters were analyzed, and actions taken by the designer in order to improve the consistency, including reassign utterances and, if necessary, modifying the annotation guide. The whole process was repeated a few times until a satisfactory global inter-cluster distance was attained.

Eventually we trained the SSLU on 79 symptoms arranged on a hierarchy with a maximum depth of 3. Table 2 summarizes the results on an independent test set of 10,332 utterances. The result shows that at the end of the process, a satisfactory batch accuracy of 81.43% correct label assignment what attained for the utterances which were deemed to be in-domain, which constituted 90.22% of the test corpus. Also, the system was able to correctly reject 24.56% of out-of-domain utterances. The overall accuracy of the system was considered reasonable for the state of the art of commercial SSLUs based on current statistical classification algorithms. Improvement in the classification performance can result by better language models (i.e. some of the errors are due to incorrect word recognition by the ASR) and better classifiers, which need to take into account more features of the incoming utterances, such as word order[1] and contextual information.

Utterances	10332	100.00%
In domain	9322	90.22%
Correct in-domain	7591	81.43%
Out of domain	1010	9.78%
Correct rejection out-of-domain	249	24.65%

Table 2: Accuracy results for Hierarchical SSLU with 79 symptoms.

[1] Current commercial SSLU modules, as the one used in the work described here, use statistical classifiers based only on bags of words. Thus the order of the words in the incoming utterance is not taken into consideration.

3.1 Confirmation Effectiveness

Accuracy is not the only measure to provide an assessment of how the symptom described by the caller is effectively captured. Since the user response needs to be confirmed based on the interpretation returned by the SSLU, the caller always has the choice of accepting or denying the hypothesis. If the confirmation prompts are not properly designed, the user can erroneously deny correctly detected symptoms, or erroneously accept wrong ones.

The analysis reported below was carried out for a deployed system for technical support of Internet service. The full symptom identification interactions following the initial open prompt was transcribed and annotated for 895 calls. The SSLU used in this application consisted of 36 symptoms structured in a hierarchy with a maximum depth of 3. For each interaction we tracked the following events:

- the first user response to the open question
- successive responses in case of re-prompting because of speech recognition rejection or timeout
- response to the yes/no confirmation question)
- successive responses to the confirmation question in case the recognizer rejected it or timed out.
- Successive responses to the confirmation question in case the user denied, and a second best hypothesis was offered.

Table 3 summarizes the results of this analysis.

The first row reports the number of calls for which the identified symptom was correct (as compared with human annotation) and confirmed by the caller. The following rows are the number of calls where the identified symptom was wrong and the caller still accepted it during confirmation, the symptom was correct and the caller denied it, and the symptom was wrong and denied, respectively. Finally there were 57 calls where the caller did not provide any confirmation (e.g. hung up, timed out, ASR rejected the confirmation utterance even after re-prompting, etc.), and 100 calls in which it was not possible to collect the symptom (e.g. rejections

Accepted correct	535	59.8%
Accepted wrong	118	13.2%
Denied correct	22	2.5%
Denied wrong	63	7.0%
Unconfirmed	57	6.4%
No result	100	11.2%
TOTAL	**895**	**100.0%**

Table 3: Result of the confirmation analysis based on the results of 895 calls

of first and second re-prompts, timeouts, etc.) In both cases—i.e. no confirmation or no symptom collection at all—the call continued with a different strategy (e.g. moved to a directed dialog, or escalated the call to a human agent). The interesting result from this experiment is that the SSLU returned a correct symptom 59.8 + 2.5 = 62.3% of the times (considering both in-domain and out-of-domain utterances), but the actual "perceived" accuracy (i.e. when the user accepted the result) was higher, and precisely 59.8 + 13.2 = 73%. A deeper analysis shows that for most of the wrongly accepted utterances the wrong symptom identified by the SSLU was still in the same hierarchical category, but with different degree of specificity (e.g. Internet-Slow vs. vague Internet)

The difference between the actual and perceived accuracy of SSLU has implications for the overall performance of the application. One could build a high performance SSLU, but a wrongly confirmed symptom may put the dialog off course and result in reduced automation, even though the perceived accuracy is higher. Confirmation of SSLU results is definitely an area where new research can potentially impact the performance of the whole system.

4 Experimental VUI

Voice User Interface (VUI) is typically considered an art. VUI designers acquire their experience by analyzing the effect of different prompts on the behavior of users, and can often predict whether a new prompt can help, confuse, or expedite the interaction. Unfortunately, like all technologies relying on the anecdotal experience of the designer, in VUI it is difficult to make fine adjustments to an interface and predict the effect of competing similar designs before the application is actually deployed. However, in large volume applications, and when a global measure of performance is available, one can test different non-disruptive design hypotheses on the field, while the application is running. We call this process *experimental VUI*.

There have been, in the past, several studies aimed at using machine learning for the design of dialog systems (Levin et al., 2000, Young 2002, Pietquin et al, 2006). Unfortunately, the problem of full design of a system based uniquely on machine learning is a very difficult one, and cannot be fully utilized yet for commercial systems. A simpler and less ambitious goal is that of finding the optimal dialog strategy among a small number of competing designs, where all the initial designs are working reasonably well (Walker 2000, Paek et al 2004, Lewis 2006). Comparing competing designs requires carrying on an exploration based on random selection of each design at crucial points of the dialog. Once a reward schema is defined, one can use it for changing the exploration probability so as to maximize a function of the accumulated reward using, for instance, one of the algorithms described in (Sutton 1998).

Defining many different competing designs at several points of the interaction is often impractical and costly. Moreover, in a deployed commercial application, one needs to be careful about maintaining a reasonable user experience during exploration. Thus, the competing designs have to be chosen carefully and applied to portions of the dialog where the choice of the optimal design can make a significant difference for the reward measure in use.

In the experiments described below we selected the symptom identification as a point worth exploring. in an internet technical support application We then defined three prompting schemas

- Schema A: the system plays an open prompt
- Schema B: the system plays an open prompt, and then provides some examples of requests
- Schema C: The system plays an open prompt, and then suggests a command that provides a list of choices.

The three schemas were implemented on a deployed system for limited time. There was 1/3 probability for each individual call to go through one of the above schemas. The target function chosen for optimization was the average automation rate.

Figure 2 shows the effect on the cumulated average automation rate for each one of the competing design. The exploration was carried out until the difference in the automation rate among the three designs reached statistical significance, which was after 13 days with a total number of 21,491 calls. At that point in time we established that design B had superior performance, as compared to A and C, with a difference of 0.68 percent points.
Event though the gain in total automation rate (i.e. 0.68 percent points) seems to be modest, one has to consider that this increase is simply caused only by the selection of the best wording of a single prompt in an application with thousands of prompts. One can expect to obtain more important improvements by at looking to other areas of the dialog where experimental VUI can be applied and selecting the optimal prompt can have an impact on the overall automation rate.

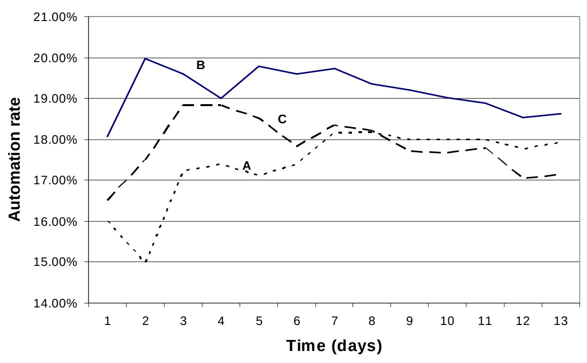

Figure 2: Daily average automation rate for competing designs.

5 Conclusions

We started this paper by describing the advances achieved in dialog system technology for commercial applications during the past decade. The industry moved from the first generation of systems able to handle very structured and simple interactions, to a current third generation where the interaction is less structured and the goal is to automate complex tasks such as problem solving and technical support.We then discussed general issues regarding the effective development of a technical support application. In particular we focused on two areas: the collection of the symptom from natural language expressions, and the experimental optimization of the VUI strategy. In both cases we described how a detailed analysis of live data can greatly help optimize the overall performance.

6 References

Barnard, E., Halberstadt, A., Kotelly, C., Phillips, M., 1999 "A Consistent Approach To Designing Spoken-dialog Systems," *Proc. of ASRU99 – IEEE Workshop*, Keystone, Colorado, Dec. 1999.

Gorin, A. L., Riccardi, G.,Wright, J. H., 1997 Speech Communication, vol. 23, pp. 113-127, 1997.

Chu-Carroll, J., Carpenter B., 1999. "Vector-based natural language call routing," *Computational Linguistics*, v.25, n.3, p.361-388, September 1999

Goel, V., Kuo, H.-K., Deligne, S., Wu S., 2005 "Language Model Estimation for Optimizing End-to-end Performance of a Natural Language Call Routing System," ICASSP 2005

Pieraccini, R., Huerta, J., Where do we go from here? Research and Commercial Spoken Dialog Systems, Proc. of 6th SIGdial Workshop on Discourse and Dialog, Lisbon, Portugal, 2-3 September, 2005. pp. 1-10

Levin, E., Pieraccini, R., Eckert, W., A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies, IEEE Trans. on Speech and Audio Processing, Vol. 8, No. 1, pp. 11-23, January 2000.

Pietquin, O., Dutoit, T., A Probabilistic Framework for Dialog Simulation and Optimal Strategy Learning, In IEEE Transactions on Audio, Speech and Language Processing, 14(2):589-599, 2006

Young, S., Talking to Machines (Statistically Speaking), Int Conf Spoken Language Processing, Denver, Colorado. (2002).

Walker, M., An Application of Reinforcement Learning to Dialogue Strategy Selection in a Spoken Dialogue System for Email . *Journal of Artificial Intelligence Research, JAIR*, Vol 12., pp. 387-416, 2000

Paek T., Horvitz E.,. Optimizing automated call routing by integrating spoken dialog models with queuing models. Proceedings of HLT-NAACL, 2004, pp. 41-48.

Lewis, C., Di Fabbrizio, G., Prompt Selection with Reinforcement Learning in an AT&T Call Routing Application, Proc. of Interspeech 2006, Pittsburgh, PA. pp. 1770-1773, (2006)

Sutton, R.S., Barto, A.G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.