

What If Everyone Could Do It?

A Framework for Easier Spoken Dialog System Design

Pierrick Milhorat
Institut Mines-Télécom
Télécom ParisTech
CNRS LTCI
Paris, France
milhorat@telecom-paristech.fr

Stephan Schlögl
Institut Mines-Télécom
Télécom ParisTech
CNRS LTCI
Paris, France
schlogl@telecom-paristech.fr

Jérôm Boudy
Institut Mines-Télécom
Télécom SudParis
Paris, France
jerome.boudy@telecom-sudparis.eu

Gérard Chollet
Institut Mines-Télécom
Télécom ParisTech
CNRS LTCI
Paris, France
chollet@telecom-paristech.fr

ABSTRACT

While Graphical User Interfaces (GUI) still represent the most common way of operating modern computing technology, Spoken Dialog Systems (SDS) have the potential to offer a more natural and intuitive mode of interaction. Even though some may say that existing speech recognition is neither reliable nor practical, the success of recent product releases such as Apple's *Siri* or Nuance's *Dragon Drive* suggests that language-based interaction is increasingly gaining acceptance. Yet, unlike applications for building GUIs, tools and frameworks that support the design, construction and maintenance of dialog systems are rare. A particular challenge of SDS design is the often complex integration of technologies. Systems usually consist of several components (e.g. speech recognition, language understanding, output generation, etc.), all of which require expertise to deploy them in a given application domain. This paper presents work in progress that aims at supporting this integration process. We propose a framework of components and describe how it may be used to prototype and gradually implement a spoken dialog system without requiring extensive domain expertise.

Author Keywords

SDS Design; Language Technology Components; WOZ.

ACM Classification Keywords

H.5.2 User Interfaces: Natural language; H.5.2 User Interfaces: Prototyping; D.5.2 User Interfaces: Voice I/O

General Terms

Human Factors; Design.

INTRODUCTION

Spoken Dialog Systems (SDS) are booming, with products such as Apple's *Siri*¹, Google's *Voice Search*² or Nuance's *Dragon Solutions*³ demonstrating how current (and future) technologies may change the way we interact with our devices. Even though a lot of these potential applications might also be achievable using traditional Graphical User Interfaces (GUI), a reasonably 'intelligent' computer system that (sufficiently) understands spoken input would simply convey a better user experience [23]. Yet, the design of this type of systems is complex and so we see a pressing demand for tools and techniques that better support this task. SDSs usually consist of several language technology components, ranging from speech recognition and generation to dialog management and artificial intelligence. Building functioning solutions may therefore require sufficient expertise in several different domains. While some tool-support for stand-alone components exists (e.g. [14, 27, 30, 8, 9, 26, 4]) only few attempts have been undertaken to generate a more holistic framework for SDS design (e.g. [15, 16, 7, 3]).

This paper discusses an SDS prototyping framework that has been implemented by our research group. The goal of our approach is to exclude 'hand-crafting' work as much as possible and use a combination of machine learning algorithms and Wizard of Oz (WOZ) experimentation [13] to build SDS solutions from scratch. Integrating existing open-source technology components with WOZ we aim for the creation of a flexible and easy to use prototyping environment, that can be used not only by speech engineers, but also by designers/researchers outside the signal processing community. The paper starts with an overview of the proposed framework architecture which is followed by a description of its different components. After that we discuss our current employment of the framework and conclude the paper with planned future directions.

¹<http://www.apple.com/ios/siri/>

²<http://www.google.com/mobile/voice-search/>

³<http://www.nuance.com/dragon/>

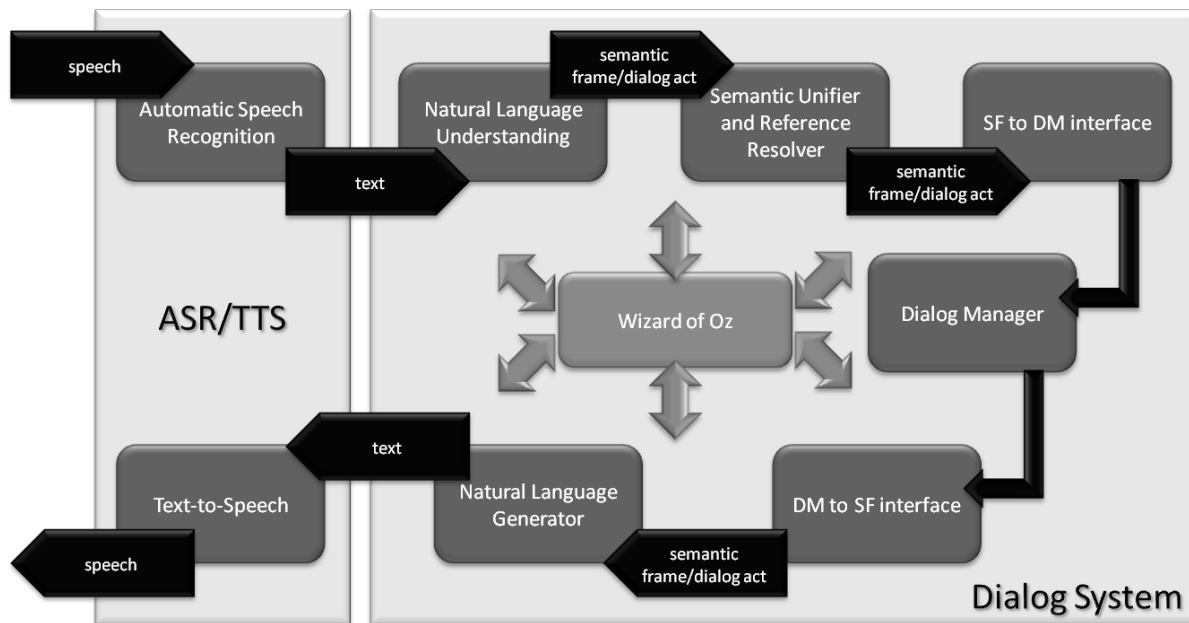


Figure 1. A framework architecture for Spoken Dialog System design

FRAMEWORK COMPONENTS

The overall architecture of our prototyping framework closely resembles the state of the art processing chain of a modern SDS (cf. Figure 1). On the input side we find the Automatic Speech Recognition (ASR) module, a Natural Language Understanding (NLU) component, as well as a novel component which we call the Semantic Unifier and Reference Resolver (SURR). The output side consists of a Natural Language Generation (NLG) component and a Text-to-Speech synthesis (TTS) module. The core of the system is represented by a Dialog Manager (DM) which is connected to the input and output chain via two formatting interfaces. These interfaces offer additional flexibility with respect to possible future framework extensions (e.g. a potential integration of multi-modality).

At runtime the ASR module decodes speech input, producing natural language text utterances and passes them on to the NLU component. The NLU component then extracts from these utterances so-called Semantic Frames (SF), which consist of a goal and 0 to n instantiated slots. Next, the SURR component filters these SFs, replaces relative values like dates, times or locations by absolute ones and resolves references. Then an interface component translates the SURR output into a format that can be processed by the DM. The DM, which represents the core of the overall system, is responsible for keeping track of the dialog progress, taking into account the given context, and consequently triggers the request for additional input; i.e. it is aware of the current tasks and therefore demands the relevant variables to be defined. It takes the output of the SURR and, based on the currently loaded task model, selects appropriate actions (i.e. it initiates utterances to be produced by the NLG component or commands to be sent to a back-end application). Again, a dedicated formatting interface is used to translate the DM output

into a format that can be interpreted by the NLG component. Such consequently produces the requested text utterance. Finally, the TTS module takes the NLG output and converts it into synthesized speech.

In order to offer this work flow we have integrated a set of open-source language technology components, augmented by various ‘home-made’ software modules, into a flexible SDS prototyping framework. The following sections will describe the different components of this framework and their roles in some more detail, and highlight which extensions and adaptations were necessary in order to create a cohesive interaction pipeline.

Automatic Speech Recognition Module

SDSs are different from other dialog systems in that speech represents their single interaction modality. Thus, an SDS’s first processing stage has to generate hypotheses about the orthographic content that is encoded in a user’s spoken input. Despite decades of research and commercial deployment this processing is still regarded as highly error-prone. Current best practice is to search for the best matching sequence of stochastic models using the digitized input signal. Mel-Frequency Cepstrum Coefficients (MFCC) (and their deltas) are widely used descriptors for such speech signal analyses (e.g. [6, 2, 1]). The distribution of the coefficients’ vectors for the contextualized phonemes or triphones (i.e. the smallest units of the processed sound signal) are usually encoded as Hidden Markov Models (HMM) [11], which were trained from already transcribed speech segments. These models constitute the first ingredient for building a working ASR module – the so called Acoustic Model (AM). Next, in order to construct words out of a sequence of phonemes, a Pronouncing Dictionary (PD) is required, which consists of the decomposition of a language’s words into phonemic

units. Finally, the last ingredient that is necessary to build the ASR module is a so-called Language Model (LM) which provides probabilities for given word sequences to appear in a sentence. Those probabilities are based on existing linguistic structures and encoded as n-grams. The combination of the three knowledge sources (i.e. AM, PD and LM) is then used by the recognition engine to produce one or several hypothesis of recognized text for a given (segmented) speech signal [1].

Given these requirements one may argue that building ASR systems for distinct application scenarios is time consuming and very much dependent on both the availability of required knowledge sources (i.e. AM, PD and LM), and the quality and amount of data that was used to construct them. Yet, existing Large Vocabulary Continuous Speech Recognition systems (LVCSR) often already cover a great amount of general purpose vocabulary (as long as their training has been performed on such data). Hence, extending such a general system (and its knowledge sources) to fit the vocabulary space of a specific application scenario may be quicker and more effective than building an entirely new recognizer from scratch. What is needed, however, are appropriate interfaces that allow for the adaptation of the general models so that they better facilitate the recognition of expected utterances related to a specific application scenario. Milhorat et al. [17] proposed a filtering method to favor such a recognition of ‘correct’ utterances while discarding mis-recognized or out-of-context ones. Results could then further be augmented with features like the dialog state, the dialog history and, a user’s personalized settings, and eventually be used to dynamically update/replace an LVCSR’s general engine configuration with a more specific, application dependent one.

In order to offer a solution that allows for such a dynamic adaptation of knowledge sources our prototyping framework integrates the Julius ASR engine, an LVCSR engine developed by the Kawahara Lab at Kyoto University [14]. The current setup supports the recognition of spoken input in English, French, Spanish and Dutch. In addition we have acquired the necessary databases to build recognizers for German and Italian. Using this setting we plan to create adapted language models for a number of application scenarios, including the speech-based operation of a calendar program, the use of communication services such as email and text messages, and the interaction with several health and well-being applications (e.g. a well-being diary).

Natural Language Understanding Component

Although all uni-modal dialog systems work with only one input modality (i.e. either direct text input or text recognized by an ASR component) the meaning representation they employ can differ greatly between solutions [9, 18]. The output that has to be produced by an integrated NLU component therefore depends on the purpose of the overall system as well as its DM formalism. Specific implementations can take on various forms and notations. For our prototyping framework we have chosen a frame-based semantic representation of language understanding. Semantic Frames (SF) are often used because of their versatility. An SF (cf. Figure 2) consists of a

goal (i.e. the user’s intent) and is further defined by a number of relevant parameters, represented by slot-value pairs. Given a textual input, the task of an SF-based NLU component is to select a matching SF (i.e. a goal and its parameters) from a predefined set of possibilities. It does this by applying a number of rules which are usually learned from an annotated corpus.



Figure 2. The example of a Semantic Frame (SF)

The NLU component we have integrated employs an algorithm developed by Jurcicek et al. [12]. It is based on sequential transformation rules which are applied to find a match between an input utterance and an SF. Rules consist of triggers and transformation operations. A trigger contains one or more conditions such as an n-gram or a skipping bigram in the user utterance, a goal value, or a slot-type in the (temporary) paired SF. The transformation is applied if all the conditions of a rule’s trigger match the input utterance-SF pair. An utterance to be processed is initialized with the default dialog act i.e. no slot and the most common goal as determined by the annotated training corpus. The training algorithm then looks for one rule that maximizes the value of the optimization function (i.e. it follows a transformation-based learning principle). In our case the optimization measure is the distance between each temporary SF and the ‘true’ SF in the corpus. This is computed as the sum of required addition, deletion and substitution operations (i.e. Levenshtein distance). Once this best rule is found, it is applied to the current state of the corpus and the algorithm is re-initiated for the resulting new training database. The process is stopped when the best rule’s increase of the optimization function is below a given threshold.

Semantic Unifier and Reference Resolver

The Semantic Unifier and Reference Resolver (SURR) is not a standard SDS component but rather one of the features that was needed to fill the gap between the NLU component integrated with our prototyping framework and its DM component. In particular, it transforms the NLU output, which is out of context, so that it can be processed by the following DM. For example, if we want to add a valid event entry to a calendar application the system usually requires an event name (i.e. a title), a starting as well as an ending point in time (i.e. a date and a time) and maybe an optional note. A user, however, might interact with the system as follows:

- User: “Add the birthday of my daughter, on Saturday the 15th of November from 2 pm”
- System: *[asks the user for the ending-point-in-time slot’s value]* “When will it be finished?”
- User: “I think I’ll be there for 6 hours”

In this situation, even if the system would create an SF with a duration slot of 6 hours instead of an ending time, the DM would not be able to process the data as it requires a precise ending. What we see here is an SF space difference between the semantic interpreter (NLU) and the decision-making component (DM). To solve this mismatch we would need to augment the entire dialog task model, which consequently might also require significant changes to be made to the back-end application. Instead, however, our framework uses a dynamic mapping component (i.e. the SURR) that allows for a duration slot to be converted into an ending-point-in-time slot. We call this process the semantic unification. Furthermore we use what we call a reference resolution process to convert the ‘tomorrow’ that is used in the above example into ‘today’s date incremented by one day’. Both operations, semantic unification and reference resolution, are contained in the same tree structures which are searched by the SURR algorithm. These trees are handcrafted from situations that happen in experiments with real users, and then further expanded according to a designer’s/researcher’s ideas. For instance, after having implemented the ‘tomorrow’ branch one may think of adding the ‘yesterday’ one.

The current version of the SURR module is based on data collected through a set of initial experiments. It employs a tree-climbing algorithm that is applied to a structure of additive and converting branches. Every link between nodes represents a predicate. Figure 3 shows an example of a tree and Figure 4 its associated implementation. The initial function looks for 1 to n slot-value pair(s) for which a transforming predicate exists and subsequently applies the defined operation. The resulting (transformed) SF is then processed again and such is repeated until no further predicate match is found. The algorithm succeeds if the final SF contains only those pairs that are declared as roots. All parameters of an SF which cannot be replaced by a root slot (i.e. where the algorithm fails) are subsequently discarded.

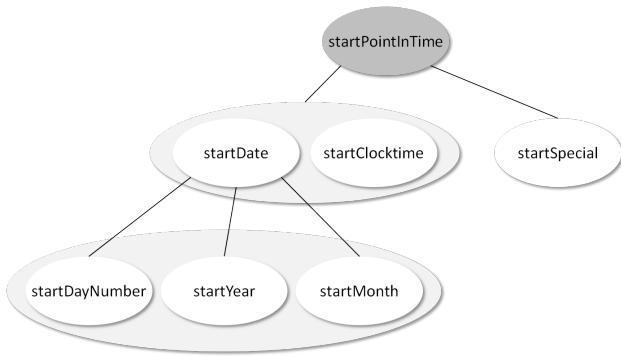


Figure 3. The tree structure of the Semantic Unifier and Reference Resolver (SURR)

Dialog Manager

To date several probabilistic DM components are available (e.g. [10, 28]). Yet, most of them require a significant amount of data to produce viable results, and their scalability is often limited to a few slots, user dialog acts and system actions. An alternative can be found in fully deterministic DM components whose functional breadth is pre-defined. Such,

```

/**root definitions*/
root([slot(startPointInTime, _)]).

/**start point in time(current date + current clocktime) -> start special(now)*/
rewrite([slot(startPointInTime, C)], [slot(startSpecial, now)]) :-
    append(A, B, C), append(D, E, A), append(F, G, E),
    slot(currentMonth, D), slot(currentDayNumber, F),
    slot(currentYear, G), slot(currentClocktime, B).

/**start point in time -> start date + start clocktime*/
rewrite([slot(startPointInTime, P)], [slot(startDate, D), slot(startClocktime, C)]) :-
    append(D, C, P).

/**start date -> start day number + start month + start year*/
rewrite([slot(startDate, D)], [slot(startMonth, M), slot(startDayNumber, Dn), slot(startYear, Y)]) :-
    append(X, Y, D), append(M, Dn, X).
  
```

Figure 4. An example implementation of the Semantic Unifier and Reference Resolver (SURR)

however, requires greater knowledge of the supported dialog space and is therefore only suitable for well defined interaction domains. Since the goal of our framework is to support the development of dialog systems for specific application scenarios we decided to integrate Disco [21, 22], a representative of the later approach. It requires a task model compliant with the ANSI CEA-2018⁴ standard, which essentially demands a recursive decomposition of tasks into atomic actions. Disco integrates a so-called inference engine which, if provided with one or more task models, is able to manage a mixed-initiative dialog. It processes a hierarchy of tasks (applying plan recognition), guiding the user towards the completion of macro tasks (consisting of several sub-tasks). Planning is performed automatically, supported by static task models and a dynamic focus stack. Task models contain the task structure, the temporal constraints for the dialog and the data flow within the models. They are implemented in XML and usually require expertise to be built. In order to help with their creation we investigated two notation languages. These languages aim at the automatic extraction of suitable task models based on the description of the given back-end application, where the back-end applications is represented by a form-filling service with attached commands. The first such language is a set of first-order logic formulas. It enables the designer/researcher to specify incompatibilities between slots, as well as optional and mandatory slot attributes. While such certainly helps the design process its application is somewhat limited. Additional manual edits are still required in order to support all the essential information a task model might need to encode. Hence a second, more advanced language is currently under development, which supports conditional relationships between slots, reusable sub-application descriptions, and computed values. Here, an application’s command is described as a form containing an ID, a set of slots, sub-forms (i.e. links to other forms) and an action triggered by the completion of the form. Sub-form attributes are boolean optional, ignore and default, which respectively set the linked form to non mandatory, ignored (in the case the applicable condition is not fulfilled), or default (in the case of ambiguity). This process may allow for a richer formalism and should enable the designer/researcher to focus more thoroughly on the actual application.

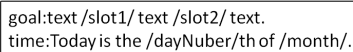
⁴[http://www.ce.org/Standards/Standard-Listings/R7-Home-Network-Committee/CEA-2018-\(ANSI\).aspx](http://www.ce.org/Standards/Standard-Listings/R7-Home-Network-Committee/CEA-2018-(ANSI).aspx)

Formatting Interfaces (SF to DM and DM to SF)

In addition to the earlier highlighted semantic ambiguities which exist between NLU and DM (we tackle them with the described SURR component), we often also find certain formatting incompatibilities between those two components (as well as between the DM and the following NLG component). Such is usually caused by the use of different input/output interface standards or diverging forms of knowledge representation. Generally the task of a DM component is to trigger output-dialog-acts and accompanying actions based on input provided by the NLU. The anticipated input as well as the produced output are, however, context dependent so that the current dialog state is often required for better disambiguation. To tackle this problem we have introduced two formatting interfaces; one of which translates an SF (delivered by the SURR) into a context-specific input-dialog-act (i.e. factoring in the current dialog state), and a second one that takes the output-dialog-act delivered by the DM and translates it back into an SF (i.e. the format that can be processed by our NLG component). While these interfaces do not modify the actual input/output content they can be regarded as necessary formatting components, implemented as an overlay to the actual DM. As such, they also offer more flexibility with respect to the modularization of our framework (Note: A future replacement of single components might require additional input/output formatting).

Natural Language Generation Component

While NLG is generally an important aspect of an SDS it is currently not our main area of interest. Our framework therefore only implements a very basic generation engine. It uses the output of the DM (i.e. the output that has been converted by the output formatting interface described above) to select a human-readable response sentence form a set of possible templates. Each template uses a goal ID that is matched with the dialog act produced by the DM. The SDS designer has to provide at least as many templates (cf. Fig. 5) as dialog acts exist. In case there are more possible templates for a given dialog act, the NLG component randomly selects one and forwards it to the TTS.



```
goal:text /slot1/ text /slot2/ text.
time:Today is the /dayNuber/th of /month/.
```

Figure 5. Natural Language Generator templates

Text-to-Speech Synthesis Module

Finally, in order to generate speech from the text fragments produced by the NLG component, our framework integrates the OpenMary TTS [19, 25]; a state-of-the-art, open source, synthesis platform which supports several languages. We currently use the platform to produce speech output in German, Italian, French and English.

Wizard of Oz Component

One last important aspect of our proposed framework architecture is the integration of a Wizard of Oz (WOZ) component. WOZ constitutes a prototyping method that uses a human operator, the so-called wizard, to simulate a system (or

part of it) in order to collect relevant interaction data [5]. To support this task we have integrated the WebWOZ prototyping platform [24]; a tool that permits the wizard to replace one or several components of an SDS. Such should offer an easy and efficient solution for various sorts of data gathering. For example, the training corpus for our NLU component consists of possible inputs and its matching outputs. Replacing this component by a human wizard who transforms spoken input into relevant dialogue acts (i.e. SFs), may alleviate the fastidious work of manually searching and annotating corpus data that matches a given application domain.

CURRENT FRAMEWORK EMPLOYMENT

The framework described above is currently used to build a multi-lingual SDS for an application scenario situated in the ambient assisted living domain. Experiments are conducted in which the WOZ component acts as a substitution for the ASR as well as the NLU component. Doing this we are able to collect various types of interaction data (mainly training data that is used for building and improving the NLU component and user experience data that helps to obtain initial end-user feedback). While our initial sessions are in French, experiments in German and Italian are planned for the next couple of month. Once sufficient data for a language is collected, one only needs to re-configure the ASR and re-train the NLU to integrate it with the system. Such demonstrates the flexibility we are aiming for with our framework composition. Another aspect of this flexibility is reflected by the amount of control the human operator (i.e. the wizard) can take over. Set-ups in which 1-*n* parts of the framework are simulated/augmented/controlled should allow for accurate refinements of faulty or weak components as well as support user studies at any stage of the development process; an aspect which, we believe, may enable also non-experts to use our framework as a means for designing and building novel SDS solutions.

CONCLUSION AND FUTURE WORK

We presented a flexible SDS prototyping framework that aims to support the easy and quick construction of voice user interfaces for different application scenarios. The implementation of this framework is achieved through the integration of a set of interchangeable open-source language technology components. While the different components are not by default ready to be used with any application domain, their configuration and adaptation to fit a specific purpose requires only little knowledge and expertise.

Future work will focus on the adaptability and flexibility of the presented framework, particularly exploring its employment by non-expert users. Furthermore we will investigate possible ways of improving single framework components. For example, we aim for increasing the robustness of the ASR by using the feedback produced by post-processing components (i.e. NLU, SURR, DM). Another planned improvement is the use of parametric HMMs [29, 20]. Those can be controlled by a set of external shared parameters and therefore would match more closely the acoustic phenomena of spoken language. Finally, we are also investigating the use of several speech recognition hypothesis.

ACKNOWLEDGEMENTS

The research presented in this paper is jointly supported by vAssist, a project funded by the European Ambient Assisted Living Joint Programme and the National Funding Agencies from Austria, France and Italy (AAL-2010-3-106), and ARHOME, a French national research project.

REFERENCES

1. Anusuya, M. A., and Katti, S. K. Front end analysis of speech recognition: a review. *International Journal of Speech Technology* 14, 2 (2011), 99–145.
2. Baker, J. M., Deng, L., Glass, J., Khudanpur, S., Lee, C.-h., Morgan, N., and O'Shaughnessy, D. Research Developments and Directions in Speech Recognition and Understanding, Part 1. *IEEE Signal Processing Magazine* 26, 3 (2009), 75–80.
3. Bohus, D., Raux, A., Harris, T. K., Eskenazi, M., and Rudnicky, A. I. Olympus: an open-source framework for conversational spoken language interface research. In *Proc. of ACL-HLT* (2007).
4. Churcher, G. E., Atwell, E. S., and Souter, C. Dialogue management systems: a survey and overview. *Research Report Series - University of Leeds School of Computer Studies* (February 1997).
5. Dahlbäck, N., Jönsson, A., and Ahrenberg, L. Wizard of oz studies - why and how. In *Proc. of ACM IUI* (1993), 193–200.
6. Davis, S. B., and Mermelstein, P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics Speech and Signal Processing* 28, 4 (1980), 357–366.
7. Galibert, O., Illouz, G., and Rosset, S. Ritel: an open-domain, human-computer dialog system. In *Proc. of INTERSPEECH* (2005), 909–912.
8. He, Y., and Young, S. Semantic processing using the Hidden Vector State model. *Computer Speech & Language* 19, 1 (2005), 85–106.
9. He, Y., and Young, S. Spoken language understanding using the hidden vector state model. *Speech Communication* 48, 3-4 (2006), 262–275.
10. Henderson, J., and Lemon, O. Mixture model POMDPs for efficient handling of uncertainty in dialogue management. In *Proc. of ACL-HLT* (2008), 73–76.
11. Juang, B.-H., and Rabiner, L. R. Hidden Markov models for speech recognition. *Technometrics* 33, 3 (1991), 251–272.
12. Jurčiček, F., Mairesse, F., Gašić, M., Keizer, S., Thomson, B., Yu, K., and Young, S. Transformation-based Learning for semantic parsing. *Evaluation* (2009), 2719–2722.
13. Kelley, J. F. An empirical methodology for writing User-Friendly Natural Language computer applications. In *Proc. of ACM CHI* (1983), 193–196.
14. Lee, C., Jung, S., and Lee, G. G. Robust dialog management with n-best hypotheses using dialog examples and agenda. In *Proc. of ACL-HLT* (2008), 630–637.
15. Leuski, A., Pair, J., and Traum, D. How to talk to a hologram. In *Proc. of IUI* (2006), 360–362.
16. Leuski, A., Patel, R., Traum, D., and Kennedy, B. Building effective question answering characters. *Proc. of SIGDIAL* (2009), 18–27.
17. Milhorat, P., Istrate, D., Boudy, J., and Chollet, G. Hands-free speech-sound interactions at home. In *Proc. of EUSIPCO* (2012), 1678–1682.
18. Mori, R. D., Béchet, F., Hakkani-Tur, D., McTear, M., Riccardi, G., and Tur, G. Spoken language understanding: A survey. In *Proc. of IEEE ASRU* (2007).
19. Pammi, S., Charfuelan, M., and Schröder, M. Multilingual voice creation toolkit for the MARY TTS platform. In *Proc. of LREC* (2010).
20. Radenen, M., and Artieres, T. Contextual hidden markov models. In *Proc. of ICASSP* (2012), 2113–2116.
21. Rich, C. Building task-based user interfaces with ANSI/CEA-2018. *Computer* 42, 8 (2009), 20–27.
22. Rich, C., and Sidner, C. L. Using collaborative discourse theory to partially automate dialogue tree authoring. In *Proc. of IVA* (2012), 327–340.
23. Schalkwyk, J., Beeferman, D., Beaufays, F., Byrne, B., Chelba, C., Cohen, M., Garret, M., and Strophe, B. Google search by voice : A case study. *Visions of Speech: Exploring New Voice Apps in Mobile Environments, Call Centers and Clinics* 2 (2010), 1–35.
24. Schlögl, S., Doherty, G., Karamanis, N., and Luz, S. WebWOZ: a wizard of oz prototyping framework. In *Proc. of ACM EICS* (2010), 109–114.
25. Schröder, M., and Trouvain, J. The German text-to-speech synthesis system MARY: A tool for research, development and teaching. *International Journal of Speech Technology* (2003).
26. Seneviratne, V., and Young, S. The hidden vector state language model. In *Proc. of INTERSPEECH* (2005), 1–4.
27. Stolcke, A. SRILM-An extensible language modeling toolkit. In *Proc. of ICSLP* (2002).
28. Williams, J. D., and Young, S. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech & Language* 21, 2 (2007), 393–422.
29. Wilson, A. D., and Bobick, A. F. Parametric hidden markov models for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21, 9 (1999), 884–900.
30. Young, S., Evermann, G., Gales, M. J. F., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., and Woodland, P. C. The HTK Book (for HTK Version 3.4), 2006.