

## Chapter 2

# Paradigms for Deployed Spoken Dialog Systems

**Abstract** This chapter covers state-of-the-art paradigms for all the components of deployed spoken dialog systems. With a focus on speech recognition and understanding components as well as dialog management, the specific requirements of deployed systems will be discussed. This includes their robustness against distorted and unexpected user input, their real-time-ability, and the need for standardized interfaces.

**Keywords** Components of spoken dialog systems • Confirmation • Dialog management • Language generation • Natural language call routing • Real-time systems • Rejection • Robustness • Rule-based grammars • Speech recognition • Speech understanding • Speech synthesis • Statistical classifiers • Statistical language models • Voice browsing • VoiceXML

### 2.1 A Few Remarks on History

After half a century of intensive research into automatic speech recognition (one of the first published functional speech recognizers was built at Bell Labs in 1952 [27]), in the 1990s, the technology finally achieved a performance (in terms of accuracy and speed) that could be applied to simple tasks in the telephony systems of companies with large customer care call volume. Solutions to phone-based self-service using touch-tone interaction already existed. Now, applications could be speech-enabled allowing for a much wider range of solutions helping companies like FedEx, American Airlines, or UPS to effectively expand their self-service customer support offerings [88]. Applications ranged from package tracking (with a tracking number specified by the caller) to stock quotes and flight schedule information.

Speech-enabled menus have clear advantages compared to touch-tone menus when it comes to:

- Input items distinguishing a large number of types (such as city names or stock listings) or
- Mixed initiative or over-specification – when spoken language understanding and dialog manager are designed accordingly, the caller can input information or formulate requests unexpected at the current point of the dialog, e.g.

S: Where would you like to depart from?

C: From JFK on January 5th.

Another milestone in the development of spoken dialog systems was the introduction of *natural language call routing* in 1997 by researchers at AT&T Labs [43]. Large companies (as AT&T itself) often have a single service hotline phone number that serves as a portal to a variety of services and call center departments with different skills. There can be dozens if not hundreds of different destinations conventionally requiring a multi-layered menu structure to get the caller to the right place as shown in the following dialog recorded from the service hotline of a cable provider:

S: Which one can I help you with: Your bill, tech support, an order, an appointment, or a change to your service?

C: Repeat.

S: Just say: *My bill*, or press 1. *Tech support*, or press 2. *An order*, or press 3. *An appointment*, or press 4. Or say *make a change to my service*, or press 5.

C: Tech support.

S: And which one are you having problems with: Internet, e-mail, cable TV, or telephone?

C: My telephone.

S: Are you having trouble with the dial tone on your phone?

C: Nay.

S: Then, do you have a question about a feature on your phone such as voicemail or call forwarding?

C: Yeah.

S: You can say *voice mail*, *caller ID*, or *call forwarding*. If it is not one of those, say *help me with something else*.

C: Repeat the message, please.

S: Just say: *Voice mail*, or press 1. *Caller ID*, or press 2. *Call forwarding*, or press 3. Or, if it is not one of those, say *help me with something else*, or press 4.

C: Help me with something else.

S: Here are some others to choose from: Call waiting, three-way calling, speed dialing, call return, anonymous call rejection, or custom ring. Again, if it is not one of these, say *help me with something else*.

C: Three-way calling.

S: I can explain you how three-way calling works or help you with problems with it. Just say *instructions* or *problems*. Or you can say *help me with something else*.

C: Problems.

In contrast, a conversation using a natural language call router with an open-ended question could have been

S: Briefly tell me what you are calling about today.

C: My three-way calling is not working.

Apparently, this type of prompting comes along with a much shorter handling time resulting in a number of substantial advantages:

- Handling fees are saved (considering the processing of millions of such calls, the shaving of just seconds for every call can result in a significant impact on the application's bottom line).
- By reducing the number of recognition events necessary to get a caller to the right place, the chance of recognition errors decreases as well (even though it is true that open-ended question contexts perform worse than directed dialog, e.g., 85% vs. 95% True Total<sup>1</sup>, the fact that doing several of the latter in a row exponentially decreases the chance that the whole conversation completes without error – e.g. the estimated probability that five user turns get completed without error is  $(95\%)^5 = 77\%$  which is already way lower than the performance of the open-ended scenario; for further reading on measuring performance, see Chap. 3). Reducing recognition errors raises the chance of automating the call without intervention of a human agent.
- User experience is also positively influenced by shortening handling time, reducing recognition errors, and conveying a smarter behavior of the application [35].
- Open-ended prompting also prevents problems with callers not understanding the options in the menu and choosing the wrong one resulting in potential misroutings.

The underlying principle of natural language call routing is the automatic mapping of a user utterance to a finite number of well-defined classes (aka categories, slots, keys, tags, symptoms, call reasons, routing points, or buckets). For instance, the above utterance

My three-way calling is not working

was classified as `Phone_3WayCalling_Broken`, in a natural language call routing application distinguishing more than 250 classes [115]. If user utterances are too vague or out of the application's scope, additional directed disambiguation questions may be asked to finally route the call. Further details on the specifics of speech recognition and understanding paradigms used in deployed spoken dialog systems are given in Sect. 2.3.

## 2.2 Components of Spoken Dialog Systems

As introduced in Sect. 1.1 and depicted in Fig. 1.1, spoken dialog systems consist of a number of components (speech recognition and understanding, dialog manager, language and speech generation). In the following sections, each of

---

<sup>1</sup>See Sect. 3.2 for the definition of this metric.

these components will be discussed in more detail focusing on deployed solutions and drawing brief comparisons to techniques primarily used in academic research to date.

## 2.3 Speech Recognition and Understanding

In Sect. 2.1, the use of speech recognition and understanding in place of the formerly common touch-tone technology was motivated. This section gives an overview about techniques primarily used in deployed systems as of today.

### 2.3.1 Rule-Based Grammars

In order to commercialize speech recognition and understanding technology for their application in dialog systems, at the turn of the millennium, companies such as Sun Microsystems, SpeechWorks, and Nuance made the concept of *speech recognition grammar* popular among developers. Grammars are essentially a specification “of the words and patterns of words to be listened for by a speech recognizer” [47,128]. By restricting the scope of what the speech recognizer “listens for” to a small number of phrases, two main issues of speech recognition and understanding technology at that time could be tackled:

1. Before, large-vocabulary speech recognizers had to recognize every possible phrase, every possible combination of words. Likewise, the speech understanding component had to deal with arbitrary textual input. This produced a significant margin of error unacceptable for commercial applications. By constraining the recognizer with a small number of possible phrases, the possibility of errors could be greatly reduced, assuming that the grammar covers all of the possible caller inputs. Furthermore, each of the possible phrases in a grammar could be uniquely and directly associated with a predefined semantic symbol, thereby providing a straightforward implementation of the spoken language understanding component.
2. The strong restriction of the recognizer’s scope as well as the straightforward implementation of the spoken language understanding component significantly reduced the required computational load. This allowed speech servers to process multiple speech recognition and understanding operations simultaneously. Modern high-end servers can individually process more than 20 audio inputs at once [2].

Similar to the industrial standardization endeavor on VoiceXML described in Sect. 2.6, speech recognition grammars often follow the W3C Recommendation SRGS (Speech Recognition Grammar Specification) published in 2004 [47].

### 2.3.2 Statistical Language Models and Classifiers

Typical contexts for the use of rule-based grammars are those where caller responses are highly constrained by the prompt such as:

- Yes/No questions (*Are you calling because you lost your Internet connection?*).
- Directed dialog (*Which one best describes your problem: No picture, missing channels, error message, bad audio...?*).
- Listable items (city names, phone directory, etc.).
- Combinatorial items (phone numbers, monetary amounts, etc.).

On the other hand, there are situations where rule-based grammars prove impractical because of the large variety of user inputs. Especially, responses to open prompts tend to vary extensively. For example, the problem collection of a cable TV troubleshooting application uses the following prompt:

Briefly tell me the problem you are having in one short sentence.

The total number of individual collected utterances of this context was so large that the rule-based grammar resulting from the entire data used almost 100 MB memory which proves unwieldy in production server environments with hundreds of recognition contexts and dozens of concurrent calls. In such situations, the use of statistical language models and classifiers (statistical grammars) is recommendable. By generally treating an open prompt such as the one above as a call routing problem (see Sect. 2.1), every input utterance is associated with exactly one class (the routing point). For instance, responses to the above open prompt and their associated classes are:

```
Um, the Korean channel doesn't work well  ⇨ Channel_Other
The signal is breaking up  ⇨ Picture_PoorQuality
Can't see HBO  ⇨ Channel_Missing
My remote control is not working  ⇨ Remote_NotWorking
Want to purchase pay-per-view  ⇨ Order_PayPerView_Other
```

This type of mapping is generally produced semi-automatically as further discussed in Sect. 4.1.

The utterance data can be used to train a statistical language model that is applied at runtime by the speech recognizer to generate a recognition hypothesis [100]. Both the utterances and the associated classes can be used to train statistical classifiers that are applied at runtime to map the recognition hypothesis to a semantic hypothesis (class). An overview about state-of-the-art classifiers used for spoken language understanding in dialog systems can be found in [36].

The initial reason to come up with the rule-based grammar paradigm was that of avoiding too complex search trees common in large-vocabulary continuous speech recognition (see Sect. 2.3.1). This makes the introduction of statistical grammars for open prompts as done in this section sound a little paradoxical. However, it turns out that, surprisingly to the most common intuition, statistical grammars seem to always outperform even very carefully designed rule-based grammars when enough

training data is available. A respective study with four dialog systems and more than 2,000 recognition contexts was conducted in [120]. The apparent reason for this paradox is that in contrast to a general large-vocabulary language model trained on millions of word tokens, here, strongly context-dependent information was used, and statistical language models and classifiers were trained based only on data collected in the very context the models were later used in.

### 2.3.3 Robustness

Automatic speech recognition accuracy kept improving greatly over the last six decades since the first studies at Bell Laboratories in the early 1950s [27]. While some people claim that improvements have amounted to about 10% relative word error rate (WER<sup>2</sup>) reduction every year [44], this is factually not correct: It would mean that the error rate of an arbitrarily complex large-vocabulary continuous speech recognition task as of 2010 would be around 0.2% when starting at 100% in 1952. It is more reasonable to assume the yearly relative WER reduction being around 5% on average resulting in some 5% absolute WER as of today. This statement, however, is true for a trained, known speaker using a high-quality microphone in a room with echo cancellation [44]. When it comes to speaker-independent speech recognition in typical phone environments (including cell phones, speaker phones, Voice-over-IP, background noise, channel noise, echo, etc.) word error rates easily exceed 40% [145].

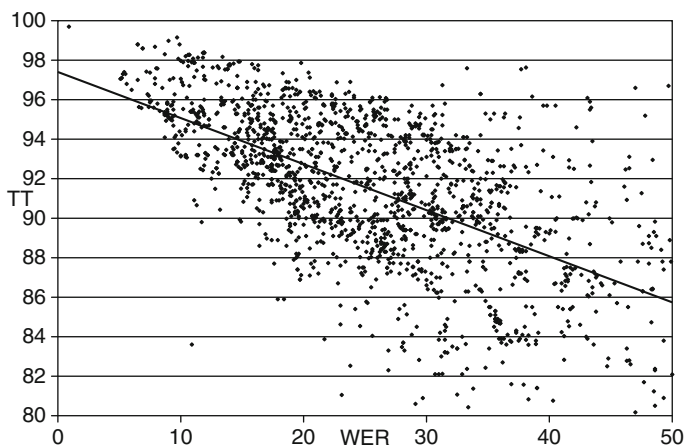
This sounds disastrous. How can a commercial (or any other) spoken dialog system ever be practically deployed when 40% of its recognition events fail? However, there are three important considerations that have to be taken into account to allow the use of speech recognition even in situations where the error rate can be very high [126]:

- First of all, the dialog manager does not use directly the word strings produced by the speech recognizer, but the product of the language understanding (SLU) component as shown in Fig. 1.1. The reader may expect that cascading ASR and SLU may increase the chance of failure since both of them are error-prone, and errors should grow rather than diminish. However, as a matter of fact, the combination of ASR and SLU has proven very effective when the SLU is robust enough to ignore insignificant recognition errors and still map the speech input to the right semantic interpretation.

Here is an example. The caller says *I wanna speak to an associate*, and the recognizer hypothesizes *on the time associate* which amounts to 5 word errors

---

<sup>2</sup>Word error rate is a common performance metric in speech recognition. It is based on the Levenshtein (or edit) distance [64] and divides the minimum sum of word substitutions, deletions, and insertions to perform a word-by-word alignment of the recognized word string to a corresponding reference transcription by the number of tokens in said reference.



**Fig. 2.1** Relationship between word error rate (abscissa) and semantic classification accuracy (True Total, ordinate)

altogether. Since the reference utterance has 6 words, the WER for this single case is 83%. However, the SLU component deployed in production was robust enough to interpret the sole presence of the word *associate* as an agent request and correctly classified the sentence as such resulting in no error at the output of the SLU module.

Figure 2.1 shows how, more globally, word error rate and semantic classification accuracy (True Total, see Sect. 3.2 for a definition of this metric) relate to each other. The displayed data points show the results of 1,721 experiments with data taken from 262 different recognition contexts in deployed spoken dialog systems involving a total of 2,998,254 test utterances collected in these contexts. Most experiments featured 1,000 or more test utterances to assure reliability of the measured values. As expected, the figure shows an obvious correlation between word error rate and True Total (Pearson’s correlation coefficient is  $-0.61$ , i.e. the correlation is large [98]). Least-squares fitting a linear function to this dataset produces a line with the gradient  $-0.23$  and an offset of 97.5% True Total that is also displayed in the figure. This confirms that the semantic classification is very robust to speech recognition errors reflecting only a fraction of the errors made on the word level of the recognition hypothesis.

Even though it may very well be due to the noisiness of the analyzed data, the fact that the constant offset of the regression line is not exactly 100% suggests that perfect speech recognition would result in a small percentage of classification errors. This suggestion is true since the classifier itself (statistical or rule-based), most often, is not perfect either. For instance, many semantic classifiers discard the order of words in the recognition hypothesis. This makes the example utterances

(1) Service interrupt

and

(2) Interrupt service

look identical to the semantic classifier while they actually convey different meanings:

(1) A notification that service is currently unavailable or a request to stop service

(2) A request to stop service

- It is well-understood that human speech recognition and understanding exploits three types of information: acoustic, syntactic, and semantic [45, 133]. Using the probabilistic framework typical for pattern recognition problems, one can express the search for the optimal meaning  $\hat{M}$  (or class, if the meaning can be expressed by means of a finite number of classes) of an input acoustic utterance  $A$  in two stages:

$$\hat{W} = \arg \max_W p(W|A) = \arg \max_W p(A|W)p(W) \quad (2.1)$$

formulates the determination of the optimal word sequence  $\hat{W}$  given  $A$  by means of a search over all possible word sequences  $W$  inserted in the product of the *acoustic model*  $p(A|W)$  and the *language model*  $p(W)$ . Similarly,

$$\hat{M} = \arg \max_M p(M|W) = \arg \max_M p(W|M)p(M) \quad (2.2)$$

expresses the search for the optimal meaning  $\hat{M}$  [36] based on the *lexicalization model*  $p(W|M)$  and the *semantic prior model*  $p(M)$  [78].

This two-stage approach has been shown to underperform a one-stage approach where no hard decision is drawn on the word sequence level [137]. In the latter case, a full trellis of word sequence hypotheses and their probabilities are considered and integrated with (2.2) [58, 84]. Despite its higher performance, the one-stage approach has not found its way into deployed spoken dialog systems yet because of primarily practical reasons, for instance:

- They are characterized by a significantly higher computational load (the search of an entire trellis requires extensively more computation cycles and memory than a single best hypothesis).
- Semantic parsers or classifiers may be built by different vendors than the speech recognizer, so, the trellis would have to be provided by means of a standardized API to make components compatible (see Sect. 2.6 for a discussion on standards of spoken dialog system component interfaces).

With reference to the different types of information used by human speech recognition and understanding discussed above, automatic recognition and understanding performance can be increased by providing as much knowledge as possible:

1. Acoustic models (representing the acoustic information type) of state-of-the-art speech recognizers are trained on thousands of hours of transcribed



speech data [37] in an attempt to cover as much of the acoustic variety as possible. In some situations, it can be beneficial to improve the effectiveness of the baseline acoustic models by adapting them to the specific application, population of callers, and context. Major phenomena which can require baseline model adaptation are the presence of foreign or regional accents, the use of the application in noisy environments as opposed to clean speech, and the signal variability resulting from different types of telephony connections, such as cell phone, VoIP, speaker phone, or landline.

2. In today's age of cloud-based speech recognizers [11], the size of language models (i.e. the syntactic information type) can have unprecedented dimensions: Some companies (Google, Microsoft, Vlingo, among others) use language models estimated on the entire content of the World Wide Web [18, 46], i.e., on trillions of word tokens, so, one could assume, there is no way to ever outperform these models. However, in many contexts, these models can be further improved by providing information characteristic to the respective context. For instance, in case of a directed dialog such as

Which one can I help you with: Your bill, tech support, an order, an appointment, or a change to your service?

the a priori probabilities of the menu items (e.g. *tech support*) are much higher than those of terms outside the scope of the prompt (e.g. *I want to order hummus*). These priors have a direct impact on the optimality of the language model.

Even if only in-scope utterances are concerned, a thorough analysis of the context can have a beneficial effect on the model performance. An example: Many contexts of deployed spoken dialog systems are yes/no questions as

I see you called recently about your bill. Is this what you are calling about today?

Most of the responses to yes/no questions in deployed systems are affirmative (voice user interface design best practices suggest to phrase questions in such a way that the majority of users would answer with a confirmation, as this has been found to increase the user confidence in the application's capability). As a consequence, a language model trained on yes/no contexts usually features a considerably higher a-priory probability for *yes* than for *no*. Thus, using a generic yes/no language model in contexts where *yes* is responded much less frequently than *no* can be disastrous as in the case where an initial prompt of a call routing application reads

Are you calling about [name of a TV show]?

The likelihood of somebody calling the general hotline of a cable TV provider to get information on or order exactly this show is certainly not very high (even so, in the present example, the company decided to place this question upfront for business reasons), so, most callers will respond *no*. Using the generic yes/no language model (trained on more than 200,000 utterances, see Table 2.1) in this context turned out to be problematic since it tended to cause

**Table 2.1** Performance of *yes* hypotheses in a *yes/no* context with overwhelming majority of *no* events comparing a generic with a context-specific language model

Language model	Training size	True Total of utterances hypothesized as <i>yes</i> (%)
Generic <i>yes/no</i>	214,168	27.3
Context-specific <i>yes/no</i>	1,542	77.4

substitutions between *yes* and *no* and false accepts of *yes* much more often than in regular *yes/no* contexts due to the wrong priors. In fact, almost three quarters of the cases where the system hypothesized that a caller responded with *yes* were actually recognition errors (27.3% True Total) emphasizing the importance of training language models with as much as possible context-specific information. It turned out that training the context-specific language model using less than 1% data than used for the generic *yes/no* language model resulted in a much higher performance (77.4% True Total).

- Last but not least, the amount and effect of speech recognition and understanding errors in deployed spoken dialog systems can be reduced by robust voice user interface design. There is a number of different strategies to this:

- *rejection and confirmation threshold tuning*

Both the speech recognition and spoken language understanding components of a spoken dialog system provide confidence scores along with their word or semantic hypotheses. They serve as a measure of likelihood that the provided hypothesis was actually correct. Even though confidence scores often do not directly relate to the actual *probability* of the response being correct, they relate to the latter in a more or less monotonous fashion, i.e., the higher the score, the more likely the response is correct. Figure 2.2 shows an example relationship between the confidence score and the True Total of a generic *yes/no* context measured on 214,710 utterances recorded and processed by a commercial speech recognizer and utterance classifier on a number of deployed spoken dialog systems. The figure also shows the distribution of observed confidence scores.

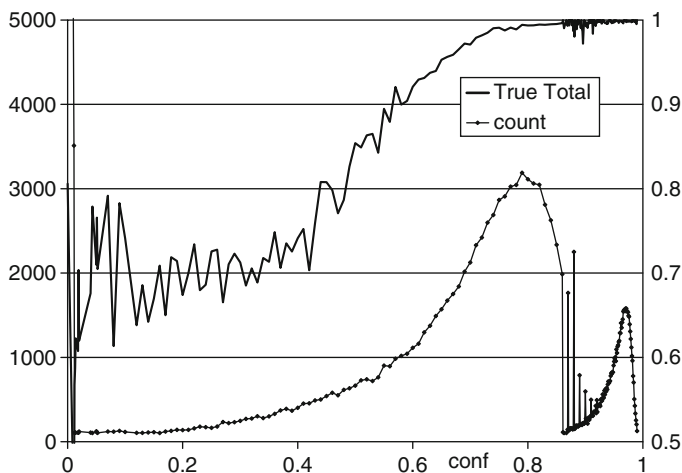
The confidence score of a recognition and understanding hypothesis is often used to trigger one of the following system reactions:

1. If the score is below a given *rejection threshold*, the system prompts callers to repeat (or rephrase) their response:

I am sorry, I didn't get that. Are you calling from your cell phone right now? Please just say *yes* or *no*.

2. If the score is between the rejection threshold and a given *confirmation threshold*, the system confirms the hypothesis with the caller:

I understand you are calling about a billing issue. Is that right?



**Fig. 2.2** Relationship between confidence score (abscissa) and semantic classification accuracy – True Total (ordinate, bold). The thin dotted line is the histogram of confidence values. The data is from a generic yes/no context

3. If the score is above the confirmation threshold, the hypothesis gets accepted, and the system continues to the next step.

Obviously, the use of thresholds does not *guarantee* that the input will be correct, but it increases the chance. To give an example, a typical menu for the collection of a cable box type is considered. The context's prompt reads

Depending on the kind of cable box you have, please say either *Motorola*, *Pace*, or say *other brand*.

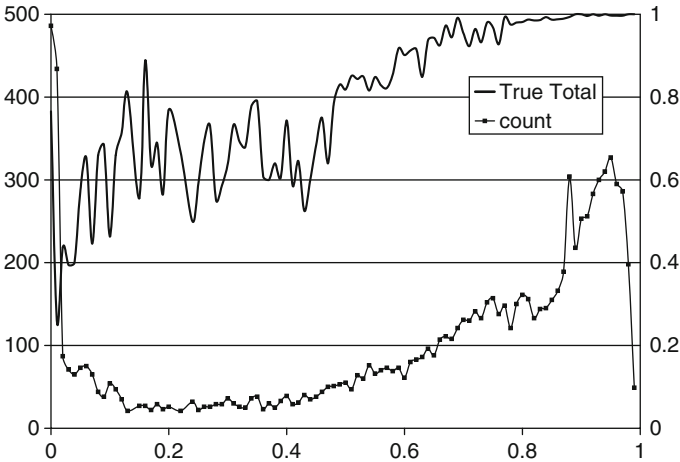
Figure 2.3 shows the relationship between confidence and True Total as well as the frequency distribution of the confidence values for this context. Assuming the following example settings<sup>3</sup>:

RejectThreshold = 0.07,  
ConfirmThreshold = 0.85,

the frequency distribution of the box collection context can be used to estimate the ratio of utterances rejected, confirmed, and accepted.

In order to come up with an estimate for the accuracy of the box collection activity including confirmation (if applicable), re-confirmation, re-collection, and so on, one has to take into account that, in every recognition context, there are input utterances out of the system's action scope. In response to the question about the box type, people may say

<sup>3</sup>See Chap. 4 on how to determine optimal thresholds.



**Fig. 2.3** Relationship between confidence score (abscissa) and semantic classification accuracy – True Total (ordinate, bold). The thin dotted line is the histogram of confidence values. The data is from a cable box collection context.

**Table 2.2** Distribution of utterances among rejection, confirmation, and acceptance for a box collection and a yes/no context. The yes/no context is used for confirmation and, hence, does not feature an own confirmation context. Consequently, one cannot distinguish between TACC and TACA but only specify TAC. The same applies to TAW and FA

Event	Box collection (%)	Yes/No (confirmation) (%)
TACC	43.29	80.89
TACA	35.17	
TAWC	2.10	0.52
TAWA	0.03	
FAC	3.78	1.14
FAA	0.09	
FR	6.94	5.90
TR	8.61	11.56

*I actually need a phone number*, or the recognizer might have caught some side conversation or line noise, etc. Hence, when asking for how successful the determination of the caller’s box type given the contexts’ speech understanding performance is at the end, one will have to use the full set of spoken language understanding metrics discussed in Chap. 3 as demonstrated in Table 2.2.

In a standard collection activity that allows for confirmation, re-confirmation, re-recollection, second confirmation, and second re-confirmation, there are 18 ways to correctly determine the sought-for information entity:

1. Correctly or falsely accepting<sup>4</sup> the entity without confirmation (TACA, FAA at collection),
2. Correctly or falsely accepting the entity with confirmation (TACC, FAC) followed by a correct or false accept of *yes* at the confirmation (TAC, FA).
3. Correctly or falsely accepting the entity with confirmation (TACC) followed by a true or false reject at the confirmation (TR, FR) followed by a correct or false accept of *yes* at the confirmation (TAC, FA).
4. ...

Instead of listing all 18 ways of determining the correct entity, the diagram in Fig. 2.4 displays all possible paths. Using the example performance measures listed in Table 2.2, one can estimate the proportional traffic going down each path and, finally, the amount ending up correctly (in the lower right box), see Fig. 2.5. Here, one sees the effectiveness of the collection/confirmation/re-collection strategy, since about 93% of the collections end up with the correct entity. The collection context itself featured a correct accept (with and without confirmation) of only 78.5%. This is an example for how robust interaction strategies can considerably improve spoken language understanding performance.

– *Robustness to specific input*

In recognition contexts with open prompts such as the natural language call router discussed in Sect. 2.1, often, understanding models distinguishing hundreds of classes [115] are deployed. Depending on the very specifics of the caller response, the application performs different actions or routes to different departments or sub-applications. In an example, somebody calls about the bill. The response to the prompt

Briefly tell me what you are calling about today.

could be, for example:

- (1) My billing account number.
- (2) How much is my bill?
- (3) I'd like to cancel this bill.

---

<sup>4</sup>The author has witnessed several cases where a speech recognizer falsely accepted some noise or the like, and it turned out that the accepted entity was coincidentally correct. For example:

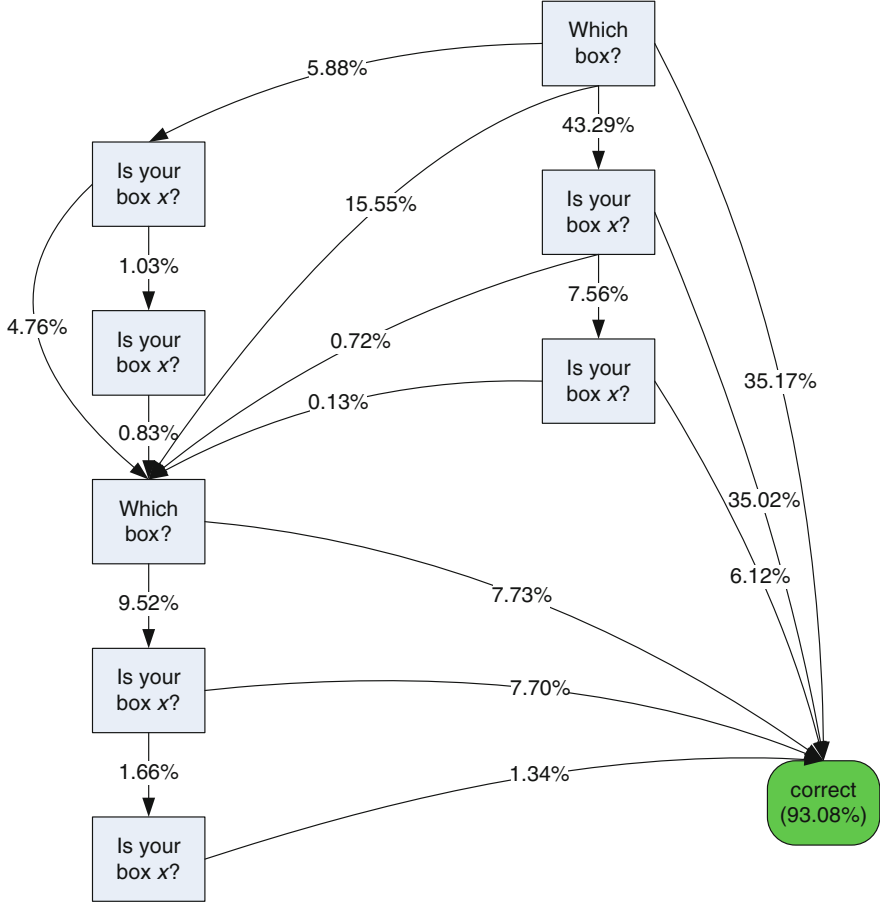
S: Depending on the kind of cable box you have, please say either *Motorola*, *Pace*, or say *other brand*.

C: <cough>

S: This was *Pace*, right?

C: That's correct.





**Fig. 2.5** The same as Fig. 2.4, but the path caption indicates the portion of traffic hitting the respective path

If, due to speech recognition and understanding problems, one of the specific responses (1–11) is classified as the generic one (12), this would be counted as an understanding error. The overall experience to the caller may, however, not be bad since the underlying high resolution of the context's classes is not known externally. An example conversation with this kind of wrong classification is

- A1: Briefly tell me what you are calling about today.
- C1: How much is my bill?
- A2: You are calling about your bill, right?
- C2: Yes.
- A3: Sure. Just say *get my balance*, or *make a payment*. Or say, *I have a different billing question*.
- C3: Get my balance.
- A4: <presents balance>

(If there would not have been recognition problems, Turns A3, and C3 would have been bypassed). When looking at a number of example calls of the above scenario, there were 1,648 callers responding *yes* to the confirmation question A2 as opposed to 1,139 responding *no* (41%). This indicates that the disturbing effect of a substitution of a class by a broader class can be moderate. For the sake of completeness, when the classifier returned the right class, 11834 responses were *yes* and only 369 were *no* (3%).

– *Miscellaneous design approaches to improve robustness*

There are several other voice user interface design techniques that have proven to be successful in gathering information entities such as [116]:

- Giving examples at open prompts:

Briefly tell me what you are calling about today.

can be replaced by

Briefly tell me what you are calling about today. For example, you can say *what's my balance?*

- Offering directed back-up menu:

Briefly tell me what you are calling about today.

can be replaced by

Briefly tell me what you are calling about today. Or you can say *what are my choices?*

- Clear instructions of which caller input is allowed (recommended in re-prompts):

Have you already rebooted your computer today?

can be replaced by

Have you already rebooted your computer today? Please say *yes* or *no*.

- Offer touchtone alternatives (recommended in re-prompts):

Please say *account information, transfers and funds, or credit or debit card information*.

can be replaced by

Please say *account information* or press 1, *transfers and funds* or press 2, or say *credit or debit card information* or press 3.



## 2.4 Dialog Management

After covering the system components speech recognition and understanding, Fig. 1.1 points at the *dialog manager* as the next block. In Sect. 1.1, it was pointed out that it “host[s] the system logic[,] communicat[es] with arbitrary types of backend services [and] generates a response ... corresponding to ... semantic symbols”. This section is to briefly introduce the most common dialog management strategies, again with a focus on deployed solutions.

In most deployed dialog managers nowadays, the dialog strategy is encoded by means of a *call flow* that is a finite state automation [86]. The nodes of this automaton represent dialog activities, and the arcs are conditions. Activities can:

- Instruct the language generation component to play a certain prompt.
- Give instructions to synthesize a prompt using a text-to-speech synthesizer.
- Activate the speech recognition component with a specific language model.
- Query external backend knowledge repositories.
- Set or read variables,
- perform any type of computation, or
- Invoke another call flow as subroutine (that may invoke yet another call flow, and so on – this way, a call flow can consist of multiple hierarchical levels distributed among a large number of pages, several hundreds or even more).

Call flows are often built using WYSIWYG tools that allow the user to drag and drop shapes onto a canvas and connect them using dynamic connectors. An example sub-call flow is shown in Fig. 2.6.

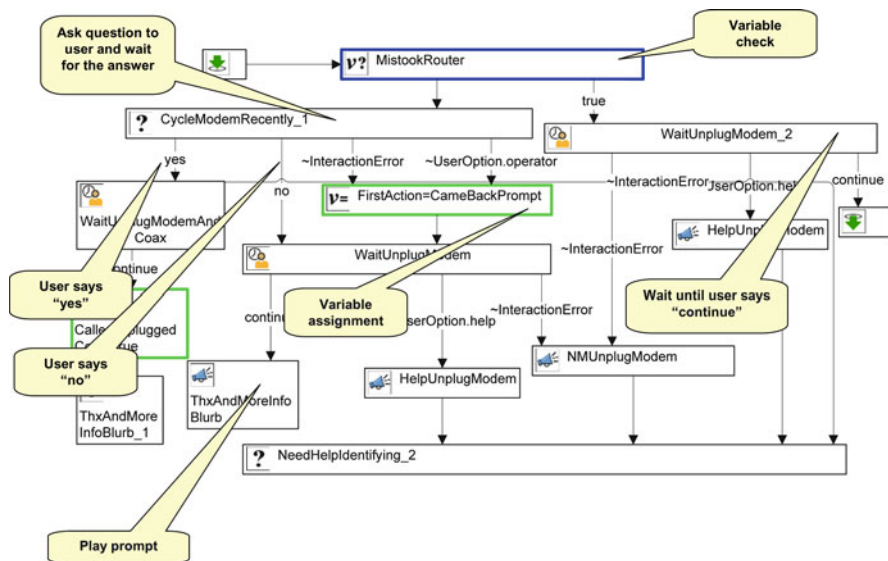


Fig. 2.6 Example of a call flow page

Call flow implementations incorporate features to handle designs getting more and more complex including:

- Inheritance of default activity behavior in an object-oriented programming language style (language models, semantic classifiers, settings, prompts, etc. need to be specified only once for activity types used over and over again; only the changing part gets overwritten; see Activities *WaitUnplugModem*, *WaitUnplugModem\_2*, *WaitUnplugModemAndCoax* in Fig. 2.6 – they only differ in some of the prompt verbiage).
- Shortcuts, anchors, gotos, gosubs, loops.
- Standard activities and libraries collecting, for instance, phone numbers, addresses, times and dates, locations, credit card numbers, e-mail addresses, or performing authentication, backend database lookups or actions on the telephony layer.

Despite these features, complex applications are mostly bound to relatively simple human-machine communication strategies such as yes/no questions, directed dialog, and, to a very limited extent, open prompts. This is because of the complexity of the call flow graphs that, with more and more functionality imposed on the spoken language application, quickly become unwieldy. Some techniques to overcome the statics of the mentioned dialog strategies will be discussed in Chap. 4.

Apart from the call flow paradigm, there are a number of other dialog management strategies that have been used mostly in academic environments:

- Many dialog systems aim at gathering a certain set of information from the caller, a task comparable to that of filling a form. While one can build call flows to ask questions in a predefined order to sequentially fill the fields of the form, callers often provide more information than actually requested, thus, certain questions should be skipped. The *form-filling* (aka slot-filling) call management paradigm [89, 108] dynamically determines the best question to be asked next in order to gather all information items required in the form.
- Yet another dialog management paradigm is based on *inference* and applies formalisms from communication theory by implementing a set of logical principles on rational behavior, cooperation, and communication [63]. This paradigm was used in a number of academic implementations [8, 33, 103] and aims at optimizing the user experience by:
  - Avoiding redundancy.
  - Asking cooperative, suggestive, or corrective questions.
  - Modeling the states of system and caller (their attitudes, beliefs, intentions, etc.).
- Last but not least, there is an active community focusing on *statistical approaches* to dialog management based on techniques such as:
  - *Belief systems* [14, 139, 144]  
This approach models the caller's true actions and goals (that are hidden to the dialog manager because of the fact that speech recognition and understanding

are not perfect). It establishes and updates an estimate of the probability distribution over the space of possible actions and goals and uses all possible hints and input channels to determine the truth.

- *Markov decision processes/reinforcement learning* [56, 66]

In this framework, a dialog system is defined by a finite set of dialog states, system actions, and a system strategy mapping states to actions allowing for a mathematical description in the form of a Markov decision process (MDP). The MDP allows for automatic learning and adaptation by altering local parameters in order to maximize a global reward. In order to do so, an MDP system needs to process a considerable number of live calls, hence, it has to be deployed, which, however, is very risky since the initial strategy may be less than sub-optimal. This is why, very often, simulated users [7] come into play, i.e. a set of rules representing a human caller that interacts with the dialog system initializing local parameters to some more or less reasonable values. Simulated users can also be based on a set of dialog logs from a different, fairly similar spoken dialog system [48].

- *Partially observable Markov decision processes* [143]

While MDPs are a sound statistical framework for dialog strategy optimization, they assume that the dialog states are *observable*. This is not exactly true since caller state and dialog history are not known for sure. As discussed in Sect. 2.3.3, speech recognition and understanding errors can lead to considerable uncertainty on what the real user input was. To account for this uncertainty, partially observable Markov decision processes (POMDPs) combine MDPs and belief systems by estimating a probability distribution over all possible caller objectives after every interaction turn. POMDPs are among the most popular statistical dialog management frameworks these days. Despite the good number of publications on this topic, very few deployed systems incorporate POMDPs. Worth mentioning are those three systems that were deployed to the Pittsburgh bus information hotline in the summer of 2010 in the scope of the first Spoken Dialog Challenge [13]:

- AT&T's belief system [140].
- Cambridge University's POMDP system [130].
- Carnegie Mellon University's benchmark system [95] based on the Agenda architecture, a hierarchical version of the form-filling paradigm [102].

## 2.5 Language and Speech Generation

(*Natural*) *language generation* [26] refers to the production of readable utterances given semantic concepts provided by the dialog manager. For example, a semantic concept could read

CONFIRM: Modem=RCA

i.e., the dialog manager wants the speech generator to confirm that the caller's modem is of the brand RCA. A suitable utterance for doing this could be

You have an RCA modem, right?

Since the generated text has to be conveyed over the audio channel, the *speech generation* component (aka speech synthesizer, text-to-speech synthesizer) transforms the text into audible speech [114].

Language and speech generation as described above are typical components of academic spoken dialog systems [94]. Without going into detail on the technological approaches used in such systems, it is apparent that both of these components come along with a certain degree of trickiness. Since language generation has to deal with every possible conceptual input provided by the dialog manager it is either based on a set of static rules or relies on statistical methods [39, 60]. Both approaches can hardly be exhaustively tested and lack predictability in exceptional situations. Moreover, the exact wording, pausing, or prosody can play an important role for the success of a deployed application (see examples in [116]). Rule-based or statistical language generation can hardly deliver the same conversational intuition like a human speaker. The same criticism applies to the speech synthesis component. Even though significant quality improvements have been achieved over the past years [57], speech synthesis generally lacks numerous subtleties of human speech production. Examples include:

- Proper stress on important words and phrases:

S: In order to check your connection, we will be using the *ping* service.

- Affectivity such as when apologizing:

S: Tell me what you are calling about today.

C: My Internet is out.

S: I am sorry you are experiencing problems with your Internet connection. I will help you getting it up and running again.

- Conveying cheerfulness:

S: Is there anything else I can help you with?

C: No, thank you.

S: Well, thank *you* for working with me!

Even though there is a strong trend towards affective speech processing evolving over the last 5 years potentially improving these issues [85], the general problem of speech quality associated with text-to-speech synthesis persists. Highly tuned algorithms trained on large amounts of high-quality data with context awareness still produce audible artifacts, not to speak of certain commercial speech synthesizers that occasionally produce speech not even intelligible.

All the above arguments are the reasons why deployed spoken dialog systems hardly ever use language and speech generation technology. Instead, the role of the voice user interface designer comprises the *writing and recording of prompts*.

That is, every single system response is carefully worded and then recorded by a professional voice talent in a sound studio environment. At run-time, the spoken dialog system simply plays the pre-recorded prompt producing optimal sound quality<sup>5</sup>. Dynamic contents (such as the embedding of numbers, locations, e-mail addresses, etc.) can be implemented in a concatenative manner with pre-recorded contents as well. Only in instances where the nature of the presented contents is unpredictable or of a prohibitive complexity (such as with last names in a phone directory application on a large and frequently changing set of destinations), speech synthesis has no alternative.

In spite of the clear advantage of the prerecorded prompt approach, it features the clear disadvantage that every single prompt needs to be formulated and recorded covering every possible situation that can arise in the course of every dialog activity including, e.g.:

- The announcement prompt (the introductory part of the activity).
- Re-announcement prompts.
- Announcement-interrupted prompt (when the caller interrupts the announcement).
- Question prompt.
- Hold prompt (a caller asks the system to hold on).
- No-input, no-match, etc. prompts for the hold role.
- Hold-return prompt (resumes the interaction after a hold).
- No-input prompts (when the caller does not say anything).
- No-match prompts (when the caller caused a reject).
- Confirmation prompts (when the speech input needs to be confirmed).
- No-input, no-match, etc. prompts for the confirmation role.
- N-best prompts (when more than one recognition hypothesis is used for the confirmation).
- Help prompt (when the caller asked for more information).
- Operator prompt (when the caller asked for an agent).
- Expert prompt (when the caller is an expert user).
- Repeat prompt (when the caller asked to repeat the information), or
- Technical-difficulty prompt.

Consequently, deployed systems of regular complexity usually require thousands, sometimes tens of thousands of pre-recorded prompts. For example, the Internet troubleshooting application described in [6] currently comprises 10,573 prompts with a total duration of 33 h. As a result, the professional recording of prompts plays a major role for the overall cost and time of building an application. Presumably trivial projects such as switching the voice talent or localizing an existing spoken dialog system to another language [118] can become prohibitive.

---

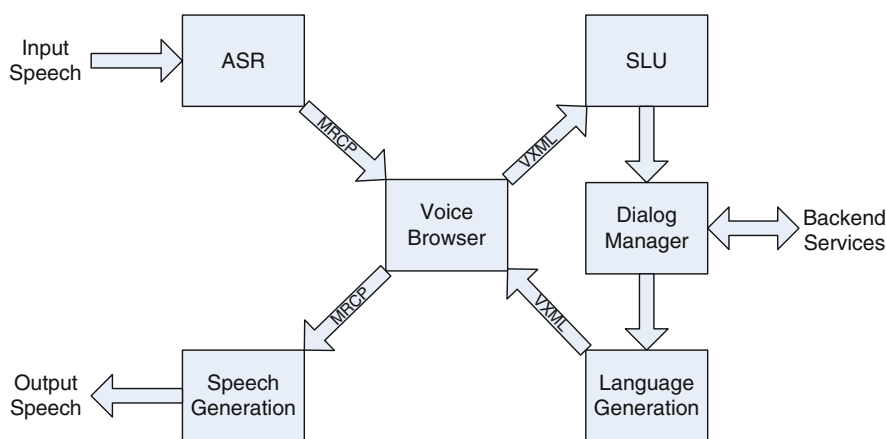
<sup>5</sup>This approach occasionally tricks callers in that they assume to be talking to a live person.

## 2.6 Voice Browsing

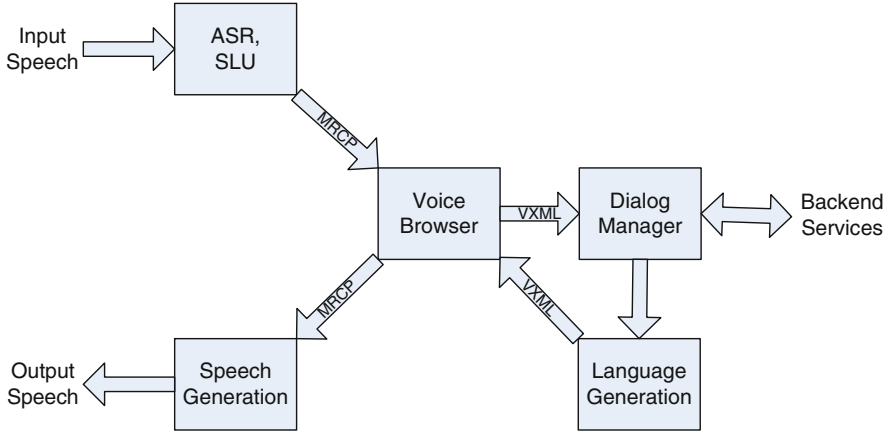
It became obvious to the speech industry that there was a need for standardized speech interfaces for spoken dialog systems only after the market saw an uptake in the number of speech applications that were introduced into the market, accompanied by the burgeoning number of speech vendors and consumers of such commercial spoken dialog systems. Given that speech recognizers, text-to-speech systems, telephony infrastructure, dialog managers, backend infrastructure, and the actual applications are potentially built by different companies in the first place, by standardizing how these components talk to each other, architecting and building solutions became much easier.

A great step towards the modularization of spoken dialog system components was the introduction of a proxy component, the *voice browser* [61]. It takes over the communication layer between speech recognition and synthesis on the one hand and language understanding and generation on the other as shown in Fig. 2.7. In an alternative architecture, speech recognition and understanding are coupled, so the voice browser communicates directly with the dialog manager (see Fig. 2.8).

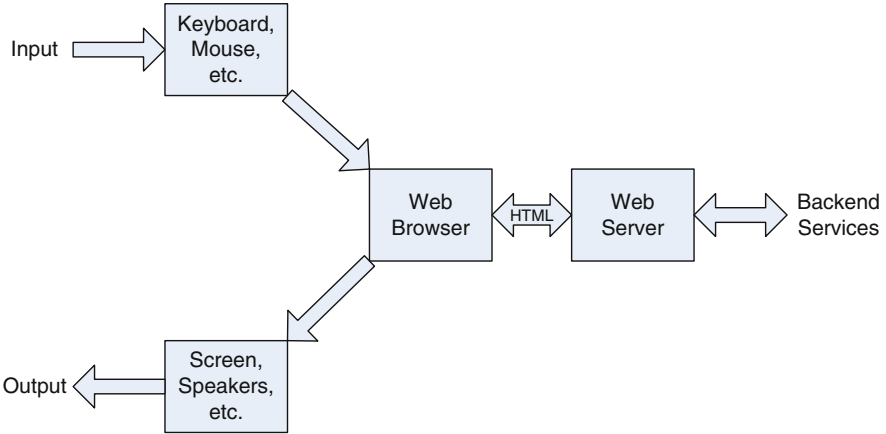
As its name suggests, the voice browser plays a role similar to a web browser which most often communicates with a human client on the one hand and a web server on the other. In this analogy (see Fig. 2.9), speech recognition (and, potentially, understanding) functions as input device of the voice browser which, in the web world, are keyboard, mouse, camera, and other input channels communicating with the web browser. Output device of the voice browser is the speech synthesizer that replaces the screen, loudspeakers, and other output channels used by a web browser. On the internal side, a voice browser communicates with the dialog manager (or with the spoken language understanding and generation components that are directly controlled by the dialog manager) playing the role of



**Fig. 2.7** General diagram of a spoken dialog system with voice browser



**Fig. 2.8** General diagram of a spoken dialog system with voice browser; ASR and SLU coupled



**Fig. 2.9** General diagram of a web browser

the web server in the web-based world. In fact, modern implementations of dialog managers are web applications making use of standard web servers such as Apache or Internet Information Services as well as common programming environments as Java Servlets, PHP, or .NET. Very much like their web counterparts, the components of Figs. 2.7 and 2.8 can be distributed over local and wide area networks communicating via HTTP and other standard protocols (in fact, the applications the author was working on in the past years – see e.g. [118, 120, 121, 123, 124] for details – were hosted on infrastructure in New York, New Jersey, Pennsylvania, California, Georgia, among others).

Inspired by the strength of standardization in the web world where the Hypertext Markup Language (HTML) serves as primary markup language for web pages, and

almost all available browsers and web content generators adhere to this standard, in 1999, a forum based on a selection of the most advanced speech research laboratories (AT&T, IBM, Lucent, and Motorola) was founded to develop a markup language for spoken dialog systems [109]. Based on the general definition of the Extensible Markup Language (XML), the new standard was branded VoiceXML, and soon after releasing Version 1 in 2000, control was handed over to the World Wide Web consortium (W3C) that made VoiceXML a W3C Recommendation in 2004 [72].

VoiceXML specifies, among other features:

- Which prompts to play (TTS or pre-recorded audio files).
- Which language or classification models (aka grammars, see Sect. 2.3) to activate (speech and touch tone).
- How to record spoken input or full-duplex telephone conversations.
- Control of the call flow.
- Telephony features for call transfer or disconnect.

VoiceXML was meant to open the entire feature space of the World Wide Web to the domain of spoken dialog systems. In this way, it was to:

- Minimize the number of transactions between voice browser and dialog manager (see Sect. 2.7 on how crucial and demanding real-time ability can be in distributed spoken dialog systems) – simple dialog systems can be implemented as a single VoiceXML page.
- Separate application code (VoiceXML) from low-level platform code (that can be in whatever programming language, or come along as a precompiled application).
- Allow for portability across different VoiceXML-compliant platforms (for both voice browsers and dialog managers).
- VoiceXML can be static (like static HTML), or dynamic (produced by dynamic web content generators such as PHP, CGI, Servlets, JSP, or ASP.NET).

Certainly, the most important step towards the modularization of spoken dialog systems was the specification of VoiceXML as the interface between dialog manager and voice browser. However, the internals of the voice browser itself, which had been originally introduced to serve as a proxy for proper communication between dialog manager and speech recognition and generation, still required well-defined interfaces. Again, this was because vendors of browser, ASR, and TTS in a single bundle could be multilateral, and there was a high demand for standardization to make components compatible with each other [19]. The response to this demand was the Media Resource Control Protocol (MRCP) published in 2006 by the Internet Society as an RFC (Request for Comments) [106]. MRCP controls media resources like speech recognizers and synthesizers and uses streaming protocols such as the Session Initiation Protocol (SIP), widely deployed in Voice-over-Internet-Protocol telephony [51].



## 2.7 Deployed Spoken Dialog Systems are Real-Time Systems

The heavy use of distributed architecture (see Fig. 3.2 for a high-level diagram of a deployed spoken dialog system's architecture including infrastructure to measure performance) requires a lot of attention to the real-time ability of the involved network machinery. In order to understand what *real-time processing* means in the context of deployed spoken dialog systems, one can use human-to-human phone conversations as a standard of comparison.

The average pause length between interaction turns is about 250 ms [15,42], and the average tolerance interval, i.e., the time after which the conversational partner feels obliged to speak, is approximately 1 s for American English speakers [50]. This means that the time lag between the moment when the caller stops and that when the system starts speaking should not be considerably longer than 1 s. If this requirement is not fulfilled, callers tend to repeat themselves assuming the system missed their response to a prompt (Turn 1). This repetition, however, may fall into the time scope of the next interaction turn (Turn 2) and, hence, may be interpreted as the response to the question of Turn 2. It is possible that the caller only heard snippets (or possibly nothing at all) of Turn 2's prompt, since, often, question prompts allow for so-called barge-in: Callers can respond at any time during the prompt and do not have to wait until the end of a possibly lengthy prompt allowing expert users to quickly navigate through a speech menu.

Table 2.3 displays an example conversation taken from a call routing application. The application was tuned to minimize handling time (around 37 s on average) producing substantial cost savings considering a volume of about 4 million calls per month.

This conversation features major glitches mainly because of the system taking too long to respond:

- The caller utters a response (3), waits for 1.3 s (4) to decide that the system either did not hear or is still listening, and qualifies her former response by saying *Technical* (5, 6). At this moment, the speech recognizer has already stopped listening, and the dialog manager is preparing the next context. In fact, the first 200 ms of the caller response (*Tech*) still fall into Context 1. The remaining part of the utterance (*nical*) coincides with the next context's system prompt that does not get played at all for being interrupted by the caller, and the corrupted utterance is interpreted in the scope of Context 6. The system receives a response that is out-of-scope for Context 6 (the fragment *nical* cannot be interpreted) and, consequently, re-prompts (8) by saying *I didn't get that...*
- The caller assumes the system is still in Context 1 and did not understand her response, so, she repeats her former input (9), pauses again for 1.2 s (10) without any system response and qualifies her answer by saying *Tech support* (11). The latter, however, again coincides with a system response to Input 9 (*Phone, sure*) and gets ignored since the system is not listening during this indirect confirmation prompt.

**Table 2.3** Example conversation in a call router application showing problems arising due to latency. Gray parts of the system prompt are not played due to barge-in by the caller

ID	Time/s	System	Caller
1	0	Briefly tell me what you are calling about today. For example: <i>I want to order new services.</i>	
2	4.7	<2.5 s silence>	
3	7.2		Telephone.
4	8.0	<1.3 s silence>	
5	9.3		Tech...
6	9.5	Which one can I help you with: your bill, tech support, an order, an appointment, or a change to your service?	...nical.
7	10	<1.9 s silence>	
8	11.9	I didn't get that. Just say <i>my bill</i> or press 1, <i>tech support</i> or press 2, <i>an order</i> or press 3, <i>an appointment</i> or press 4. Or say <i>make a change to my service</i> or press 5.	
9	18.1		Telephone.
10	18.9	<1.2 s silence>	
11	20.1	Phone, sure.	Tech support.
12	21.4	<0.8 s silence>	
13	22.2	Just say <i>my bill</i> or press 1, <i>tech support</i> or press 2, <i>an order</i> or press 3, <i>an appointment</i> or press 4. Or say <i>make a change to my service</i> or press 5.	
14	31.8		Tech support.
15	32.7	<0.8 s silence>	
16	33.5		Tech sup...
17	34.0	Are you having trouble with the dial tone on your phone?	...port.
18	34.4	<3.5 s silence>	
19	37.9	I didn't get that. If you're having trouble with the dial tone say <i>yes</i> , otherwise, say <i>no</i> .	
20	40.5		Tech support. Tech support.
21	43.8	<1.9 s silence>	
22	45.7	OK. Let me get someone on the line to help you.	
23	48.0	<1.0 s silence>	
24	49.0		Thank you.

- After another silence to load the next prompt (12), the system starts speaking (13) offering menu options including the one just ignored (Tech support). The patient caller repeats herself (14), waits for 0.8 s (15) and repeats herself once again (16, 17). In the meantime, the system has already interpreted Response 14 and moves on to the next context while the speaker already started speaking (16). Again, the prompt gets interrupted right away, and the recognizer only captures the second part of the response (*port*) which cannot be successfully interpreted.
- Consequently, the system apologizes and replays the question (19). The caller assumes she is still in Context 13, and, therefore, interrupts the prompt repeating her former response twice (20). Since her input still does not answer the question, the system gives up according to the application's policy and escalates to a human operator (22).

The reader may want to argue that the speech understanding problems could have been reduced by:

1. Overcoming technical hurdles making the system listen without even slight interruptions (thereby avoiding the cut user inputs 5/6 and 16/17).
2. Revisiting the barge-in behavior of certain prompts (e.g. forcing the caller to listen to the first seconds of 6 and 17).

(1) is in the responsibility of the technology vendors (i.e. the developers of speech recognizer and voice browser) which, as discussed above, are usually companies different from the ones building the applications, making it a hard problem to tackle. (2) is in the court of the voice user interface designers, but there are also a number of drawbacks to forcing callers to listen to extended prompts, inter alia, an increase of average handling time and the fact that speech input may not be acknowledged at all (exemplified by Turn 11 in Table 2.3), in turn resulting in potential understanding problems.

Generally, a significant reduction of latency most probably would have saved the above sample conversation to begin with. To understand what it takes to make deployed spoken dialog systems in a distributed environment real-time-able, one needs to look at all the actions performed between the moment when a caller's speech is over and when the system response starts playing (considering the architecture shown in Fig. 2.8):

As shown in Table 2.4, there are three types of contributors to the overall latency, constant ( $C$ ), server-load-dependent ( $S$ ), and network-dependent ( $N$ ) ones. The single constant contributor, the complete recognition time-out (i.e. the duration the recognizer waits after the caller stops speaking until deciding that the utterance is over), cannot be altered without compromising recognition and understanding accuracy due to false end-point detection (in fact, there is extensive scientific work dedicated to the determination when to take turn based on various clues such as prosody, syntax, semantics, or pragmatics [53, 82, 131]). Latency caused by server overload can be reduced by carefully balancing load among available servers or by upgrading the stock of available computational resources connecting additional machines. Finally, the network needs to be laid out to accommodate guaranteed response times of a magnitude lower than 100 ms round-trip delay (consider that a single voice browser/dialog manager turn can involve up to seven network transactions or even more depending on the specific communication protocol). This response time may not exceed a certain maximum threshold (e.g., 100 ms) even in case of occasional high-load situations.

To get a rough idea of the required network capacity in such a real-time system, the example scenario referred to in Fig. 1.4 is considered where:

- In peak situations, a customer service hotline receives some  $n = 20,000$  calls per hour.
- Every single of these calls is processed by the call routing application mentioned earlier in this chapter.

**Table 2.4** Steps performed by a deployed spoken dialog system between a caller stops talking and the system starts responding.  $C$  is a constant contribution to latency, while  $S$  and  $N$  are variable durations depending on server load and network speed, respectively

step	$C S N$
Complete recognizer time-out (this is the time the recognizer waits until deciding that the speaker utterance is over and that the silence is not a natural speaking pause) (ASR)	$C$ (1,000 ms)
Completing speech recognition and delivering the recognition hypothesis (ASR)	$S$
Classifying the recognition hypothesis and delivering the semantic hypothesis (SLU)	$S$
Returning recognition and semantic hypotheses over the network to the voice browser	$N$ (<5 ms LAN; <100 ms WAN)
The voice browser decides whether to ignore the recognition event based on the semantic hypothesis (in so-called <i>hot-word</i> contexts, the application is to ignore all user inputs but a number of predefined classes in order not to interrupt the conversation unnecessarily – see e.g. Context 11 in Table 2.3)	$S$
In regular contexts, the voice browser forwards recognition and semantic hypotheses over the network to the dialog manager	$N$
The dialog manager processes the voice browser's output, navigating the call flow, accessing backend services if required, and preparing the system's response (language generation)	$S$ (3 s with, 100 ms without backend)
The dialog manager sends the next request to the voice browser over the network providing information about what prompt to play, which speech recognition and understanding models to load, and a number of additional parameters such as time-outs, sensitivity, confidence thresholds, etc. (for details about these, see Sect. 2.3)	$N$
The dialog manager request gets compiled (or interpreted) by the voice browser	$S$
All required prompts (audio files) are requested over the network (they are usually located on a separate media server). Alternatively, the prompt text is sent over the network to a text-to-speech module	$N^a$
If applicable, the text-to-speech module generates an audio signal (speech generation)	$S$
The audio signal or file is sent back to the voice browser (or directly to the prompt player) over the network	$N^a$
Speech recognition and understanding models are requested over the network (they are usually located on a separate media server)	$N^a$
Speech recognition and understanding models are sent back to the voice browser (or directly to the speech recognizer) over the network	$N^a$
ASR and SLU modules are compiled by providing speech recognition and understanding models	$S^a$
ASR starts listening	–
The prompt starts playing	–

<sup>a</sup>Indicates that this contribution does not apply when server file caching is active

- One call requires 19.1 transactions between voice browser and dialog manager on average (measured on data from July 2010).
- A single transaction averages at 3,463 bytes sent from the dialog manager and 700 bytes the other way (measured on data from July 2010).

**Table 2.5** Network throughput produced by a number of applications hosted on two data centers (one for the voice browsers, one for the dialog managers) connected by a single wide area network connection

Application	Customer	Throughput/(Mbit/s)
Call router	A	2.81
Internet troubleshooting	A	1.80
Cable TV troubleshooting	A	0.80
Digital phone troubleshooting and FAQ	A	0.03
FAQ (about settings and new cable equipment)	A	0.07
Customer survey after speaking to a human agent	A	0.78
Call-back application after outage clearance	A	0.02
Internet troubleshooting	B	0.21
Cable TV troubleshooting	B	0.33
Sum		6.84

Using these values, one can compute the average load for the dialog manager outbound connection as

$$L = 20000 \cdot 19.1 \cdot 3463 \text{ bytes/hour} = 2.81 \text{ Mbit/s.} \quad (2.3)$$

While this amount sounds non-critical assuming that reliable high-speed Internet connections are available for at least 10 Mbit/s, one has to consider that there may be other applications sharing the same network connection. Specifically, as the example application is a call router, it routes callers to human operators or other spoken dialog systems. When these other systems' voice browsers and dialog managers are hosted in the same facilities as those of the call router, most often, they will share the network connection. In the case of the present example, Table 2.5 shows which applications were sharing the network connection with the call router and which expected throughput each of them produced.

Moreover, transactions are not evenly distributed during the 1-h time frame. Similar to what was discussed in Sect. 1.2, one can calculate the likelihood that transactions overlap in time, and, based on that, what the expected network latency caused by overlapping transactions would be.

<http://www.springer.com/978-1-4419-9609-1>

Advances in Commercial Deployment of Spoken Dialog  
Systems

Suendermann, D.

2011, XII, 69 p. 20 illus., 7 illus. in color., Softcover

ISBN: 978-1-4419-9609-1