17.5.1. Resolution of Relative Expression

There are two types of relative expression. The first type is the *reference*, relating linguistic expressions to real-world entities. This may involve disambiguation, by inference or direct user query. When a user says, "*Give me Eric's phone number,*" many people with first name *Eric* may exist in the database. The second type of relative expression is the *co-reference*. Co-reference occurs when different names or referring expressions are used to signify the same real-world entity. For example, in the sentence "*Nelson Mandela* has a long history of leadership within the African National Congress, but *he* is aging and nobody was surprised yesterday when *Mandela* announced his successor" the terms *Nelson Mandela* and *Mandela* refer to the same person.

In linguistics, there are three different types of co-references. The example above is an *ellipsis*, where the omitted word(s) can be understood from the context. The other type is *deixis*. A deixis refers to the use of a word such as that, now, tomorrow, or here, whose full meaning depends on the extralinguistic context in which it is used. Location deictic co-references are very common for multimodal applications where pointing devices (modalities) like pens can be used to indicate the real locations. The most common type of co-reference is *anaphora*, which is a special type of co-reference, where a word or phrase has an indirect, dependent meaning, standing for another word or concept previously introduced. The pronoun *he* in the sentence above is an anaphor referring to *Nelson Mandela* too.

Time deictic co-references like *tomorrow*, *next week, the week of 2/ 7/ 2000*, etc., are among the easiest category for resolution (requiring only simple domain knowledge). The resolution of other relative expressions usually requires deep natural language processing. We focus our discussion on anaphora resolution, since it represents the most challenge one among others and approaches of solving this problem are typical of the kind of methods appropriate for resolving a variety of other relative expressions.

17.5.1.1. Priority Entity Memory

We introduce a simple resolution method [60] that is based on semantic class type abstraction and priority entity memory. This method is straightforward and is very powerful to handle most cases even without complex natural language processing.

Whenever a conversion of a relative expression occurs, the consequent entity is added to the entity memory. The entity memory consists of *turn* and *discourse* memories. Either type of memory consists of a number of priority queues that are delineated by entity types. An entity can only be remembered into the queue of compatible types (e.g., through inheritance). When referred to, the memory item increases its priority in the queue. This treatment resembles the *cache language* model described in Chapter 11.

The turn memory is a cache for holding entities in each turn. There are two types of turn memories. The *explicit* memory holds the entities that are resolved directly from semantic objects. In contrast, the *implicit* memory is for entities that are deduced from relative

Amazon/VB Assets
Exhibit 1011
Page 908

expressions. In accessing the memory, the explicit turn memory takes precedent over the discourse memory, which in turn has a higher priority than the implicit. At the end of the system's turn, all the turn memory items are moved and sorted into the discourse memory.

The distinctions between the three kinds of memories and the rules to operate them are designed as a simple mechanism for most common but not all possible scenarios. It is worth noting that the design has a bias toward *direct* and *backward* reference. For example, in the expression "*Forward this mail to John, his manager, and his assistant,*" the second *his* will be evaluated as referring to *John*, not to *his manager*. The implicit memory, however, provides a back-off for expressions like "*Send email to John, his manager, and her assistant*" in which the pronoun *her* should be taken as indicating John's manager is a female and resolved accordingly. However, since we store only the entities and not the semantic objects into the memory, the mechanism is not suitable for forward or pleonastic references, as in the examples like "*Since his promotion last May, John has been working very hard*" or "*It being so nice, John moved the meeting outside.*" Fortunately, these natural language phenomena are rare in a spoken dialog environment.

It is sensible to confirm[3] the resolved entities with users due to possible resolution errors. In cases where many entities in the entity memory can be matched with a semantic object, a decision of not performing any resolution and directly inquiring the user for disambiguation may be a better solution. In general, name references can be resolved by a sequence of simple rules. In the example of "*Give me Eric's phone number*" the SLU system may just generate the query message "*What is Eric's last name*?" when many people in the entity memory have the same name *Eric*.

17.5.1.2. Resolution by NLP

Extensive understanding is crucial for perfect resolution for relative expressions (in particular, anaphora). Though morphology, lexical semantics, and syntax can be helpful for disambiguation, ultimately it is a problem of inference using real-world knowledge and dialog state or context. In a discourse model of focus, it is assumed that speakers usually center their attention on a single main topic called the *focus*. Some utterances introduce or reintroduce a focus; others elaborate on it. Focus elements typically change (by being suspended, interrupted, resumed, etc.) over the course of a dialog. Once a focus element has been introduced, anaphora is usually used to represent it, making dialog more efficient.

Anaphora resolution specifies the referent of a pronoun or other anaphoric expression. This association should be supported by inference about properties and probabilities in the real world. Anaphora resolution can be done with a simple *entity focus* principle. For example, in the very common *schedule a meeting* type of dialog application, an exchange such as that shown in Figure 17.18 is centered on the initial focus element—the proposed meeting—and anaphora are likely to relate to that central topic, at least early on in the exchange. The

[3] One might decide which confirmation strategies (explicit or implicit confirmation) to use based on the confidence of the resolutions. The details of confirmation strategies are described in Section 17.6.

Amazon/VB Assets
Exhibit 1011
Page 909

(1) I'd like to schedule a [meeting]$_i$ with [Christoph]$_j$.
(2) [It]$_i$ can be anytime after 4.
(3) Tell [him]$_j$ [he]$_j$ can [grab a cab over here]$_k$.
(4) [That]$_k$ should be only if he's running late.

Figure 17.18 A *schedule a meeting* dialog example showing different anaphora usage.

subscript indicates the co-reference to the same entity. The focus is the *meeting* proposed in (1). The pronoun *it* in (2), by the very simple mechanism discussed here, can be interpreted as referring to the meeting. Some grammatical knowledge and the semantic class type should help the system to resolve *him* in (3) as *Jim* rather than the *meeting*. In sentence (3) the focus has shifted to the action of *taking a cab*, to which *that* refers in sentence (4). The locative *here* in (3) must also be resolved to the speaker's location.

Most formal models of anaphora resolution originated from research into discourse and human-human dialog. They tend to be overpowered, in making elaborate provision for greater topic and reference variation than exists in typical computer speech dialog applications of the present time. On the other hand, while they can provide resolution for some complicated situations, they tend to be underpowered, in failing to deal robustly with the realities of imperfect speech recognition and parsing.

Some of the work on anaphora resolution in dialog relies on elaborate focus-tracking mechanisms [47]. These tend to be somewhat circular in nature, in that anaphoric reference resolution is required for the focus-tracking algorithms to operate, while the anaphoric resolution itself relies on the currently identified focus structure of the dialog or discourse. Rather than elaborate on these possibilities, we instead present a number of relatively straightforward heuristics for anaphora resolution, some of which have been developed based on textual studies, but which may be relevant to increasingly complex human-computer dialog in the future. The discussion here is limited to the resolution of intersentential and intrasentential pronominal anaphora. Full noun-phrase anaphora, where one synonymous noun phrase is co-referent with another, requires even more powerful grammatical and semantic resources.

Syntactic conditions can be tested when a parse tree showing syntactic constituency is available. The most obvious syntactic filter for disallowing co-reference is simple grammatical feature agreement. For example, the following proposed co-indexed relation is not semantically possible in ordinary discourse, and the restriction is explicitly provided through the lexical morphology and syntax of the language:

The [girl]$_i$ thought [he]$_i$ was frightening.

Though the theoretical details can be complex [37], the basic intuition of syntax-based anaphoric resolution is that nonreflexive pronouns that are syntactically too close to a candidate co-referential NP (antecedent) are disfavored. For example, in a sentence such as:

[Bill's]$_i$ photo of [him]$_i$ is offensive.

Amazon/VB Assets
Exhibit 1011
Page 910

the coindexing of *Bill* with *him* is disallowed. By disallowed, we mean that your innate sense of proper English grammar and interpretation will balk at the proposed relation. The language provides a mechanism to override some proximity restrictions, as in the following repaired version:

```
[Bill's]i photo of [himself]i is offensive.
```

So, when is a pronoun *too close* to a possible antecedent? The most important syntactic concepts for determining anaphoric relations rely on structural attributes of parse trees. In fact, treatment of this problem represents a very large and highly argumentative subfield within theoretical linguistics. Nevertheless, any treatment of anaphora resolution on purely syntactic grounds is very likely to end with a list of conditions that can mostly be subsumed under some form of *x-bar* theory [25], as it is called in the theoretical linguistics.

17.5.2. Automatic Inference and Inconsistency Detection

Automatic inference can be carried out through the same framework of priority entity memory described in Section 17.5.1.1. During semantic evaluation, a partially filled semantic object is first compared with the entities in the memory based on the type compatibility. If a candidate is found, the discourse analysis module then computes a goodness-of-fit score by consulting the knowledge base and considering the position of the entity in the memory list. The semantic object is converted immediately to the entity from the memory if the score exceeds the threshold. In the process, all the actions implied by the entities are carried out following the order in which the corresponding semantic objects are converted.

In general, automatic inference can be implemented as description procedures attached to semantic slots as described in Section 17.3.1. In the example of an airline reservation system, a procedure or rule can be attached to automatically infer the destination city for the returning flight. The other powerful strategy for automatic inference is *slot inheritance.* When changing dialog turn for different semantic objects under the same service, the system may allow such slot inheritance to free users from repeating the same attributes. For example, after a user asks "*What is Peter Hon's office number*?" he may abbreviate his next query to "*How about Derek Acero's*?" Slot inheritance will allow the second semantic object regarding *Derek Acero* to inherit the *office number* slot even though it is not explicitly specified.

Inconsistency checking is crucial to initiate necessary events for *dialog repair.* A dialog may be diverted away from the ideal flow for various reasons (e.g., misrecognition, out-of-domain reference, conflicting information), many of which require domain- and application-specific knowledge to guide the dialog back to the desired course. This process is called dialog repair. Similar to automatic inference, inconsistency checking can be implemented as description procedures attached to semantic slots. In addition, inconsistency checking can also be triggered when semantic binding for a partially filled semantic object fails (e.g., indicated by a failed database lookup). The discourse analysis module is responsible only for

Amazon/VB Assets
Exhibit 1011
Page 911

sending the dialog repair events to the dialog manager, and it leaves the realization of the repair strategy to the corresponding event handler in the dialog manager.

For example, consider a query: "*Find me the cheapest flight from Seattle to Memphis on Sunday.*" The semantic binding fails because there is actually no flight available on Sunday from Seattle to Memphis based on the flight database. Thus, the discourse manager passes such event to the dialog manager, and the dialog manager will generate an appropriate message to let the users be aware of this fact.

17.6. DIALOG MANAGEMENT

For most applications, it is highly unlikely that a user can access or retrieve the desired information with just a single query. The query might be incomplete, imprecise, and sometimed inconsistent with respect to the discourse history. Even if the query is unambiguous, the speech recognition and sentence interpretation modules in a SLU system may make mistakes. Thus the SLU system needs to provide an interactive mechanism to perform clarification, completion, confirmation, and negotiation dialogs with users. By default, the objective of such a dialog is to help users accomplish the required tasks more efficiently. Being user-friendly is also one of the major objectives for dialog systems as discussed in Chapter 18. Since the goal of a SLU system is to provide a natural conversation interface for users, the ultimate SLU system should act like a real human, yet still possessing perfect memory and superfast computation. Based on these criteria, it is not hard to see why mix-initiative systems are preferred over system-initiative systems.

The dialog manager controls the interactive strategy and flow once the semantic meaning of the query is extracted and stored in the system's representation (discourse trees). The architecture of SLU dialog systems resembles the one used in event-driven GUI systems. In the same way that GUI events are assigned to graphical objects, the dialog events are assigned to semantic objects that encapsulate the knowledge for handling events under various discourse contexts. As mentioned in Section 17.5, the discourse tree with domain entity binding is passed along with necessary dialog events generated from the discourse analysis module to the dialog manager. The dialog manager acts as an intelligent domain knowledge handler that uses the semantic meaning of the query to check against domain-specific knowledge (including domain database and application logic) and generates the desired answer for the query or produces other necessary dialog strategy.

In this sense, the dialog manager functions as a GUI application that contains an event handler. The event handler handles dialog events passed from the discourse analysis module and generates appropriate responses to engage users to solve the problems. In addition, the dialog manager needs to implement the application logic to generate appropriate actions (e.g., make real airline and hotel reservation). In this section we discuss two modeling techniques for implementing application logic, and different dialog behaviors related to event handling.

Amazon/VB Assets
Exhibit 1011
Page 912

17.6.1. Dialog Grammars

Dialog grammars use constrained, well-understood formalisms such as *finite state* machines to express sequencing regularities in dialogs, termed *adjacency pairs*. The rules state sequential and hierarchical constraints on acceptable dialogs, just as syntactic grammar rules state constraints on grammatically acceptable strings. For example, an answer or a request for clarification is likely to follow a question, just as a finite state grammar might provide for a noun or an adjective, but not a verb, to follow a determiner such as *the*. In most dialog grammar systems, dialog-act types (*explain*, *complain*, *request*, etc. cf. Section 17.2.2) are categorized ,and the categories are used as terminals in the dialog grammar. This approach has the advantage that the formalism is simple and tractable. At every stage of processing the system has a basis for setting expectations, which may correspond to activating state-dependent language models, and for setting thresholds for rejection and requests for clarification.

In its essence, the dialog grammar model is exemplified by a rigid flowchart diagramming system control of the type and sequence of interaction. Figure 17.19 shows a finite state dialog grammar for an airline reservation SLU system. In this simple example, dialog-act categorization is omitted, and the interactions are controlled based on bare information items. This grammar makes simple claims: the interaction is basically question-answer; the topic queries are answered on-topic if possible, and presumably with a confirmation statement to catch the existence of a problem.

This system is easily programmed. The challenge lies in providing tools to application authors to ease the tedium and minimize the errors in the construction of grammars, and to allow for more flexibility and spontaneous deviations from the expected transitions in the grammar. Such deviations may be important for novice users, who may more naturally tend to give their information (origin, destination, time) in one single utterance or in a different order.

In general, the dialog grammar approach has the following potential disadvantages

- The interaction may be experienced by a user as brittle, inflexible, and unforgiving, since it is difficult to support mix-initiative systems.
- Dialog grammars have difficulty with nonliteral language (indirection, irony, etc.).
- A speech act might be expressed by several utterances, complicating the grammar.
- A single utterance might express several speech acts, complicating the grammar.

To address these issues, more sophisticated approaches to enhance hand-built finite state dialog grammars have been attempted. For example, once can add statistical knowledge based on realistic data to dialog grammars. The statistical learning methods, like come CART, *n*-grams, or neural networks [3] can be used to learn the association between utterances and states in the training data.

Amazon/VB Assets
Exhibit 1011
Page 913

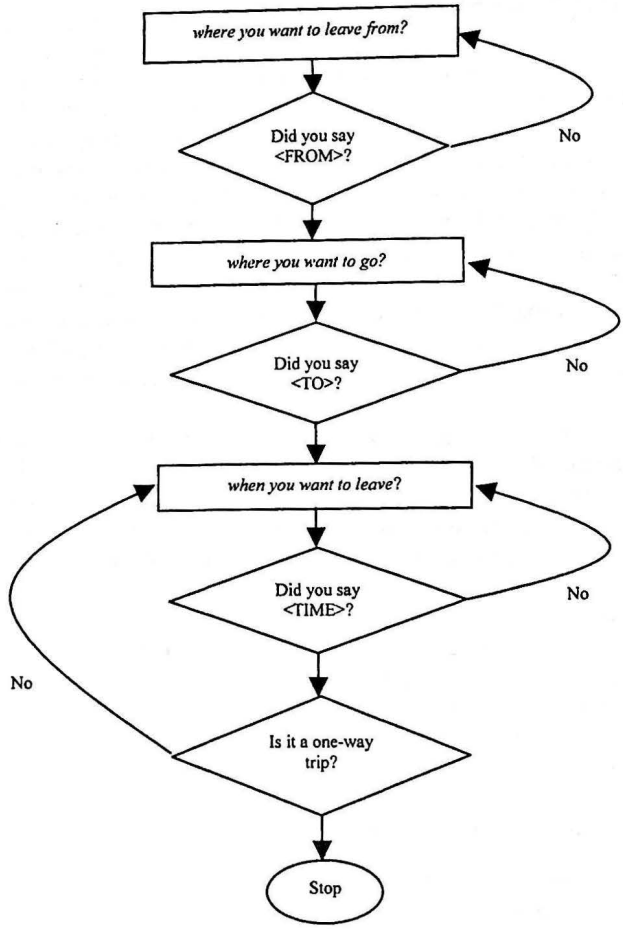


Figure 17.19 A finite state dialog grammar for airline reservation (after [19]).

17.6.2. Plan-Based Systems

Plan-based approaches [2, 41] seek to overcome the rigidity and shallowness of dialog grammars and templates. They are based on the observation that humans *plan* their actions to achieve various *goals*. Thus, plans and goals are in some degree of correspondence. A system operating under these assumptions needs to *infer goals*, *construct and activate plans*. A user may have a preconceived plan for achieving his/her goals or may need to rely on the system to supplement or construct appropriate plans.

Amazon/VB Assets
Exhibit 1011
Page 914

Plan-based systems are well studied in artificial intelligence (AI) [32, 65]. The mathematical foundation of the plan-based approach is inference. The behaviors of the system and the knowledge of the domain are programmed as a set of logical rules and axioms. The system interacts with the user to gather facts, which consequently trigger rules and generate more facts as the interaction progresses. As illustrated in Eq. (17.1), the goal of the dialog manager is to derive the action **A** based on discourse semantic S_n. Taking this view, the dialog manager is a natural outgrow of the semantic evaluation process. It is the step where the system's intent is computed. The outcome of the dialog manager is a message (via different rendering) the system conveys to the user.

In essence, a plan-based system is an embodiment of a state machine for which different discourse semantics are regarded as states. The difference, however, is that the *states* for the plan-based system are generated dynamically and not limited to a predetermined finite set. This capability of handling an unbounded number of states is a key strength of plan-based systems in terms of scalability.

Even a simple interaction can involve a variety of complex subgoals and pragmatic inferences. A partial plan for the airline reservation example in Section 17.6.1 is illustrated in Figure 17.20. One wants to know if a flight itinerary (F12) is an available one. The relationships among the goals and actions that compose a plan can be represented as a directed graph, with *goals*, *preconditions*, *actions,* and *effects* as nodes and relationships among these as arcs. These graphs illustrate the compositional nature of plans, which always include nested subplans, down to an almost infinite level of detail. The appropriate level of planning specification is thus a judgment call and must be application dependent.

The arcs are labeled with the relationship that holds between any two nodes. *SUB* shows that the child arc is the beginning of a subplan for the parent. At some point appropriate to the domain of the planning application, the SUBs will be suspended and represented as a single subsuming node. In Figure 17.20, ENABLE indicates a precondition on a goal or action. EFFECT indicates the result of an action. ENABLE indicates an enabling relationship between parent and child nodes.

Plan-based approaches incorporate a rich and deep model of rational behavior and, thus, in theory, permit a more flexible mode of interaction than do dialog grammar approaches. However, they can be complex to construct and operate in practice, due to reliance on logical and pragmatic inference, and due to the fact that no fully understood theoretical underpinning exists for their specification. The complexity of the domain of modeling often requires significant efforts from human experts to author the logical rules and axioms.

In plan-based theories of agent interaction, each dialog participant needs to construct and maintain a model of all participants' goals, commitments, and beliefs. Plans are, thus, a relatively abstract notion, leading to the hope that plans could be designed in an application-independent fashion, which would permit the development of *plan libraries*. Such libraries could be easily adapted to a variety of domains; just as specific entity models are derived from generic classes via inheritance in object-oriented programming.

Amazon/VB Assets
Exhibit 1011
Page 915

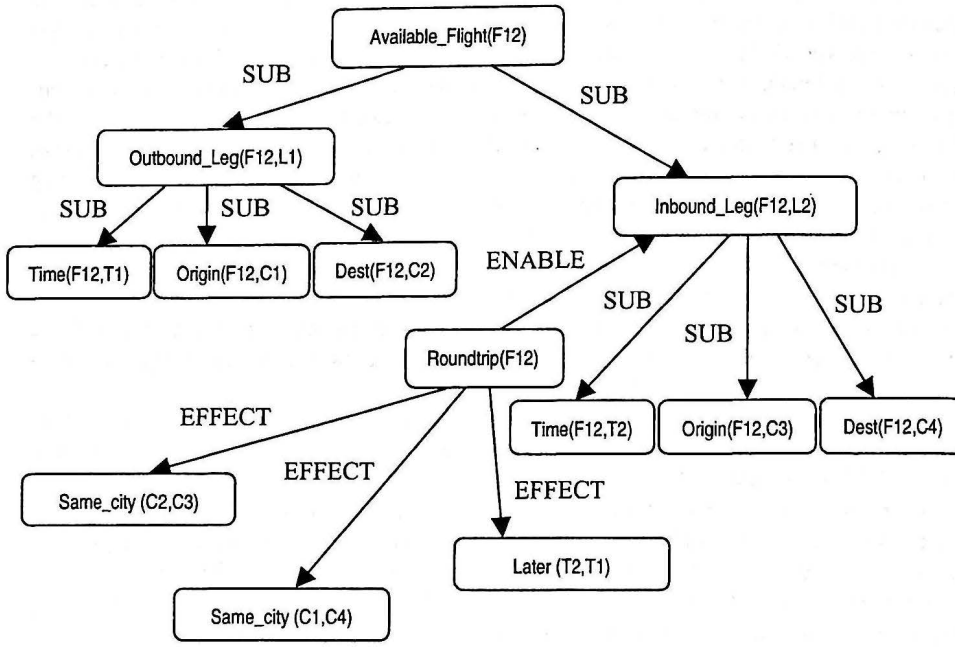


Figure 17.20 A partial plan for the airline reservation example in Figure 17.19 represented as a graph.

The following operational cycle exemplifies the plan approach, describing interaction of two agents, X (the helpful assisting agent) and Y (the client). Interaction is stated from X's point of view [10].

- Observe Y's act(s)
- Infer Y's plan (using X's model of Y's beliefs and goals)
- Debug Y's plan, finding obstacles to success of plan, based on X's beliefs
- Adopt the negation of the obstacles as X's goal
- Plan to achieve those goals and execute the plan

A flight itinerary that at least contains an *Outbound_Leg* subgoal and another possible *Inbound_Leg* subgoal is a round trip. Let's assume F12 is a round trip itinerary. At the *Inbound_Leg* node, the interesting question is how much of the underlying goal (*Time*(F12,T2), *Origin*(F12,C3) and *Dest*(F12,C4)) can be inferred by the information provided by the system from the dialog so far, or from other known conditions. For example, the destination of the *Inbound_Leg* can be inferred from the origin in the outbound leg. The origin city can be inferred similarly. Going one step further, you can also infer that the de-

Amazon/VB Assets
Exhibit 1011
Page 916

parture time for the inbound leg must occur after the departure time of the outbound leg (T1 < T2). Those three inferences are shown in the *Effect* arcs in Figure 17.20.

The goal inference could be a cooperative process, with the system making the minimal queries needed to verify and choose among alternative hypotheses. Or, it could be based on pure inference, with perhaps a confirmation step. Inference modeling can get very complicated. The technologies of inference are complex models of the beliefs, desires, and intentions of agents, making use of generic logical systems, which operate over the propositions corresponding to the nodes in a plan structure such as shown in Figure 17.20. Both user and system are assumed to be operating from partially shared world and discourse models consisting of beliefs about all relevant entities and their relationships. If utterances and speech acts are not in conflict with the constraints implied by the world models, communication and action can proceed. Otherwise, either the utterance itself must be further interpreted, supplemented, or clarified, or the world models need to be changed.

The natural expression of rational behavior, communication, and cooperation is some form of first-order logic. We define axioms and inference rules for *Belief* and *Intention*. If the modal operator for belief is B, axioms and inference rules for an agent i with respect to proposition schemata ϕ or ψ could be formalized in the following logical expression.

$$\begin{aligned}
&(B_i(\phi) \wedge B_i(\phi \Rightarrow \psi)) \Rightarrow B_i(\psi) \\
&B_i(\phi) \Rightarrow \neg B_i \neg \phi \\
&B_i(\phi) \Rightarrow B_i(B_i(\phi)) \\
&\neg B_i(\phi) \Rightarrow B_i(\neg B_i(\phi)) \\
&\neg B_i(\phi) \Rightarrow \neg B_i(B_i(\phi)) \\
&\forall x B_i(\phi) \Rightarrow B_i(\forall x \phi)
\end{aligned} \tag{17.5}$$

These describe appropriate conditions on beliefs of rational agents, such as entailment and consistency. Intentions, in turn, are formalized with respect to beliefs. For example, if an agent is to form an intention to bring about a state of affairs, it is reasonable that s/he believes this state of affairs is not currently in force:

$$I_i(\phi) \Rightarrow B_i(\neg \phi) \tag{17.6}$$

Other such axioms formalize related constraints on intentions, e.g., having an intention entails a commitment to achieving any preconditions, and belief in the possibility of doing so. Many more axioms involving all aspects of rational behavior, and formalizing, to some extent, the Gricean Maxims can be devised. For example, a kind of conversational cooperation occurs when a participant i is willing to come to believe what i believes his/her conversational partner j is attempting to communicate (at least for the limited operational domains in question!), unless i holds beliefs to the contrary:

$$B_i(I_j(B_i(\phi(j)))) \wedge \neg B_i(\neg \phi(j)) \Rightarrow B_i(\phi(j)) \tag{17.7}$$

When beliefs and intentions are modeled in this fashion, it may be possible to directly construct the core of a dialog engine based on rational principles as a theorem prover. Such a treatment is, however, beyond the scope of this discussion.

Amazon/VB Assets
Exhibit 1011
Page 917

A few desirable system behaviors that would naturally follow from limited inference and goal tracking can be briefly examined. Unlike the dialog grammar approach, a plan-based system allows digression, since the user's intention model has been built into the plan. When a system is confused about a user's input, a cooperative system could begin to perform the critical pragmatic steps that uniquely distinguish the conversational interface. A chain of inferring the user's goal, based on the system's axioms, dialog history, and current knowledge, would be triggered.

It is essential for a system to track the *dialog focus*, or temporary centers of attention, in order to understand things that are unspoken but assumed to be salient across utterances. In this case, the user's input is ambiguous—*June 22* is for outbound or inbound flight? If the dialog architecture provides a method of tracking focus, it may be simple to resolve the legs from an earlier query.

Focus is a useful concept in dialog understanding. The basic idea is similar to the entity memory tracking in anaphora resolution (see Section 17.5.1.1)—at any given point in a conversational exchange, a few items are at the center of attention and are given preference in disambiguation. Other items are in the background but may be revitalized as centers of attention at some later point. A static area can be used to contain items that are assumed background knowledge throughout the exchange. The main goal of conversation can initialize the stack. As subgoals are elaborated, new focus sets are pushed on the stack, and when these subgoals are exhausted, the corresponding focus object is popped from the stack and earlier, presumably broader topics are resumed. Focus shifts that are not naturally characterized as refinements of a broader current topic may be modeled by initiating a new independent focus stack. Focus shifts may be cued by characteristic linguistic signals, such as cue words and phrases (*well now, ok!, by the way, wait!, hey*, etc.). In many cases, focus structure tracks the recursively embedded plan structures, such as that shown in Figure 17.20.

17.6.3. Dialog Behavior

Even though the behavior of the dialog manager is highly dependent on the domain knowledge and the applications, some general styles of dialog behavior are worth investigating. The first important dialog behavior is the *dialog initiative* strategies. System initiative systems have the advantage of narrowing the possible inputs from users, while paying the price for extreme inflexibility. Although user initiative strategy is often adopted for GUI-based systems, it is seldom implemented for SLU systems, since total flexibility is translated into high perplexity (resulting low system performance). For many applications, a flexible mixed initiative style is preferred over a rigidly controlled one. Although it is possible to implement a mixed initiative system using either dialog grammars or plan-based approach, the latter is more flexible because it can handling an unbounded number of states.

Most often, the response generated by the dialog manager is either a *confirmation* or a *negotiation*. Confirmation is important due to possible SLU errors. There are two major confirmation strategies—explicit or implicit confirmations. An *explicit* confirmation is a response solely for confirmation of what the system has heard. On the other hand, an *implicit*

Amazon/VB Assets
Exhibit 1011
Page 918

confirmation is a response containing new input query and embedded confirmation with the hope that the user can catch and correct the errors if the embedded confirmation is wrong. The examples in Figure 17.21 illustrate both confirmation strategies.

SLU systems usually use a confidence measure as to when to use explicit and implicit confirmation. Obviously, explicit confirmation is used for low-confidence semantic objects while implicit confirmation is for high-confidence ones.

A negotiation response can arise whether a semantic object is fully filled or not. In the case of underspecification, there are some attributes of the semantic objects that cannot be inferred by the discourse manager. Possible actions range from simply pursuing the unfilled attributes in a predefined order, to gathering the entities in the knowledge base sorted by various keys. For cases of ill specification, an entity that matches the semantic object attributes does not exist. The planner can simply report such fact, or suggest removal or replacement of certain attributes, depending on how much domain knowledge is to be included in the planning process.

Often in the design process, we find it desirable to segregate a dialog into several self-contained sessions, each of which can employ specialized language, semantic, and even behavior models to further improve the system performance. Basically, these sessions are subgoals of the dialog, which usually manifest themselves as *trunk* nodes on the discourse tree. We implement a tree stack in which each trunk node is treated as the root for a discourse tree. The stack is managed in a first-in last-out fashion, as currently no digression is allowed from one subdialog to another. So far, the no-digression rule is considered to be a reasonable trade-off for dynamic model swapping.

Consider the example domain of travel itinerary planning [13]. At the top level is the *scenario*, which is the intended output of the interaction. The scenario is the entire itinerary, consisting of reservations for flights, hotels, rental cars, etc., all booked for the user at workable, coordinated times and acceptable prices and quality levels. A scenario might be: a flight out of the user's home city of Boston, from Logan airport, on April 2, at 4:00 PM on a particular flight, connecting in Dallas-Ft. Worth to another flight to a regional airport, an overnight hotel stay, a meeting the next day in the morning, a drive to a second local afternoon meeting, a flight from the regional airport in the evening to LA for a late meeting, another overnight stay in LA, a morning meeting at the hotel, and a return flight back to Boston later that same morning.

```
I:  I would like to fly to Boston.
R1: Do you want to fly to Boston?          (explicit confirmation)
R2: When do you want to fly to Boston?     (implicit confirmation)
```

Figure 17.21 With the input *I would like to fly to Boston*, explicit confirmation response R1 *Do you want to fly to Boston*? only allows the user to confirm the destination, while implicit confirmation response R2 *When do you want to fly to Boston*? allows the user to provide departure-time information and have a chance to confirm the destination as well.

Amazon/VB Assets
Exhibit 1011
Page 919

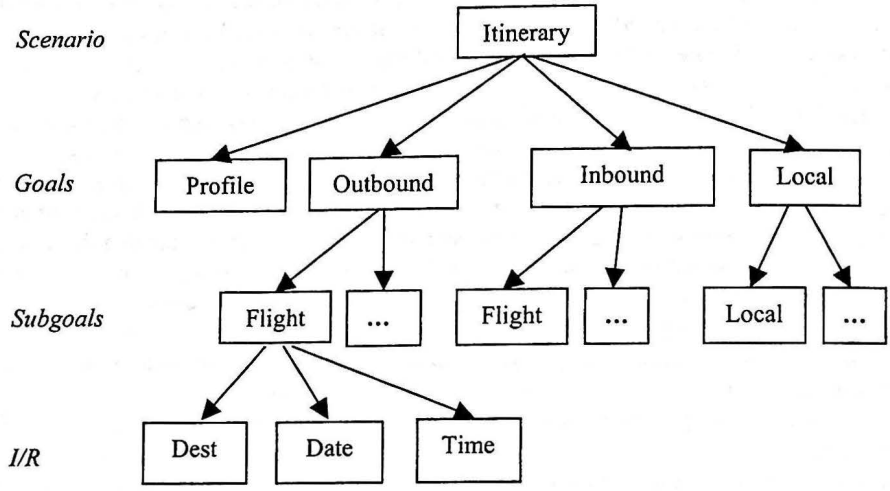


Figure 17.22 A dialog structure hierarchy for travel.

Creating the finished itinerary for this scenario involves *goals*, generated by the system or the user. Goals might include: access user travel profile, book outbound and inbound flights, and make local arrangement (hotel reservation and car rental). Goals in turn may subsume *subgoals*. Subgoals are concerned with the details of planning. These would include establishing particular desired cities and airports for the flights, price investigation, queries about hotel location and quality, etc. The subgoals in their turn are generally realized via speech acts forming I/R pairs. A simplified schematic of the structure of the itinerary structure described above might appear as shown in Figure 17.22.

This structure lends itself to a variety of control mechanisms, including system-led and mixed initiative. For example, the system may ask guiding questions such as "*Where would you like to go*?" followed by "*What day would you like to leave?*" or the system could begin processing from the user's point of view by accepting an utterance like *I want to go from Boston to LA,* corresponding to the *Dest* node of a flight on the outbound flight, and responding with a query about the next needed item, e.g., *What day would you like to leave*? This system can also accommodate a user who may wish to talk about his or her hotel reservation immediately after making the outbound flight reservation, before arranging the inbound flight.

17.7. RESPONSE GENERATION AND RENDITION

Response generation, also known as the *message generation*, is the process in which the message is physically presented to the user. This is the stage that significantly involves human-factor issues, as discussed in Chapter 18. It is more susceptible to application-specific or user interface considerations. For example, to handle a message requesting the user to

Amazon/VB Assets
Exhibit 1011
Page 920

select a sizable list of alternatives, a system with a suitable visual display might choose to present the whole list, while a speech-only system might require a more clever way. In this section we mainly focus on speech output modality and provide some thoughts on other popular output modalities.

A conversational interactive system requires a speech-output capability. The speech output may be comprised of system requests for clarification, disambiguation or repeat of garbled input; confirmation; prompting for missing information; statements of system capabilities or expectations; and presentation of results. At the lowest level, this is done via a text-to-speech engine, as discussed in Part III (Chapters 14, 15, and 16) and shown as a component in Figure 1.4. However, most text-to-speech engines have been designed for a read speech style. Moreover, such systems typically perform only shallow syntactic and semantic analysis of their input texts to recover some text features that may have prosodic correlates. Because the topic space of a task-oriented dialog system is narrower, there are opportunities to tune prosodic and other attributes of the speech output for better quality.

There are two major concerns in voice-response rendering. First is the creation or selection of the content to be spoken, and second is the rendition of it, which may include special prosodic markups as guidance to a TTS engine.

17.7.1. Response Content Generation

The response content can be explicitly tied to the semantic representation of the domain task and objects. The semantic class could incorporate custom prompts for specific slots or even for whole semantic classes. Whenever the dialog manager finds that specification for a particular slot is missing from a semantic object, it can check if it contains prompts. If prompts are present, one could be selected at random for presentation to the user.

Response prompts can be embedded in semantic representation. Prompts are usually provided for each slot to provide direction for users to fill the slot in the next dialog term. For example, the semantic class `ByName` defined in Figure 17.9 can be enhanced with the prompts in Figure 17.23.

Prompts could be associated with conditions. For example, in a flight information system, a conditional prompt can be inserted into the semantic class definition to inform users of the flight arrival time based on whether the flight has landed or not, as shown in Figure 17.24.

Other systems may include some categorization of prompts for different functions. For example, at the task level of an airline reservation system, the categorized message list might appear as shown in Figure 17.25. The grammar format makes provision for convenient authoring of messages that can be specified and accessed by functional type at runtime. The `BEMsg` is a special type of message. In this particular architecture, communication with the database engine (cf. the boxes application and database in Figure 17.2) is controlled by messages that are authored in the task specification. The URL attribute indicates a database

Amazon/VB Assets
Exhibit 1011
Page 921

```
<!-- semantic class definition for ByName that has type
PERSON too -->
<class type="PERSON" name="ByName">
    <slot type="FIRSTNAME" name="firstname"
          prompt="Please specify the last name for [firstname]/>
    <slot type="LASTNAME" name="lastname"
          prompt="Please specify the first name for [lastname]/>  />
    <cfg>
          .....
    </cfg>
</class>
```

Figure 17.23 Semantic class ByName in Figure 17.9 is enhanced with prompts specified for the case of missing a particular slot information.

access. The </rclist> is the set of possible return codes from the back-end application (as it attempts to perform the specified command from the message). Again, every return condition is associated with a message by the task specification author. Those shown here include a simple confirmation of a successful completion, as well as a warning for *flight sold out* and a generic failure of transaction message.

```
<class type="FLIGHT" name="Flight">
    <slot type="FLIGHTNO" name="flight_no">
    <slot type="TIME" name="sch_time">
    <slot type="TIME" name="actu_time">
    <slot type="CITY" name="dep_city">
    <slot type="CITY" name="arr_city">
    <slot type="AIRLINE" name="airline">
    <prompt condition= "$SYS_TIME > [actu_time]">
          Flight [flight_no] is landed at [actu_time]
    </prompt>
    <prompt condition= "default">
          Flight [flight_no] is scheduled to land at [sch_time]
    </prompt>
    <cfg>
          .....
    </cfg>
</class>
```

Figure 17.24 A semantic class Flight contains a conditional prompt to inform users when invalid [depart_time] is detected.

Amazon/VB Assets
Exhibit 1011
Page 922

```
<messages>
  <msg id="Help"> Please specify the flight time, origin and destination </msg>
  <msg id="Cancel"> Canceling itinerary... </msg>
  <msg id="Confirm"> Buying ticket from [origin] to [dest] on [time]? </msg>
  <msg id="BEMsg" url="http://server/...?op=buy&time=[time]&flight=[flight]..">
    <rclist>
       <rc id="OK"> Complete buying </rc>
       <rc id="SO"> The flight is sold out </rc>
       <rc id="ERROR"> Cannot complete transaction </rc>
    </rclist>
  </msg>
</message>
```

Figure 17.25 An example of categorization of prompts for an airline reservation SLU system.

Such systems can incorporate other kinds of categorization as well. For example, a system might provide a battery of responses to a given task or subtask situation, varying depending on a speech recognition confidence metric. Thus a set of utterances ordered by decreasing confidence might appear as:

```
You want to fly to Boston?
Did you say Boston?
Could you repeat that, please?
Please state a flight reservation.
```

Systems of this type are sometimes referred to as *template systems* for response generation. They have the advantages of direct authoring and simplicity of implementation and may provide very high quality if the message templates of the application can be played with matching digitized speech utterances or carrier phrases in the synthesizer.

The specificity and application-dependent qualities of template-based systems are sometimes perceived as weaknesses that could potentially be overcome by more general, flexible, and intelligent systems. In these systems the *message generation* box could subsume discrete modules, as shown in Figure 17.26. The semantic representation would typically be akin to logical forms (see Chapter 2) expressed via semantic frames or conceptual graphs. The representation would include abstract expression of content as well as speech-act type and other information to guide the tactical or low-level aspects of utterance generation, such as word choice, sentence type choice, grammatical arrangement, etc.

Amazon/VB Assets
Exhibit 1011
Page 923

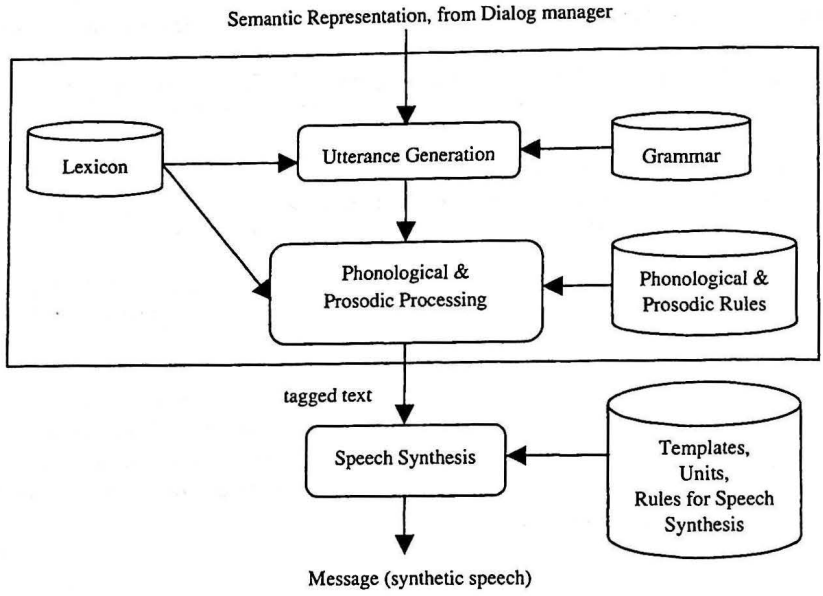


Figure 17.26 Natural language generation and rendition modules.

Natural language generation from abstract semantic input is a deep and complex field. Let us briefly consider a slightly more abstract form of template-selection mechanism that could gracefully either accommodate a simple set of static, authored response utterances or, alternatively, serve as a form of semantic input to a generalized, NLP-based utterance generation module. Imagine that instead of simply providing lists of prompt strings with embedded slot identifiers, a system of parameterization can be used [24]. The parameters could be at varying levels of abstraction and would function as descriptors of static content when preauthored prompts were being used, or would serve as a kind of input semantic representation when a general natural language was used. The set of parameters might include attributes of utterances such as the following:

- *Utterance type*: mood of the sentence, i.e., declarative, *wh*-question, yes/no question, or imperative.
- *Dialog or speech act*: confirmation, suggestion, request, command, warning, etc.
- *Body*: some characteristic lexical content for the utterance, apart from any situation-dependent words and concepts. This could serve as a hint to a generator. In many cases this would be the main verb of a sentence and might

Amazon/VB Assets
Exhibit 1011
Page 924

also include characteristic cue words, especially for functional transitions, e.g., *however*, *now*, etc.

- *Given*: information that is understood from the discourse history. This is usually represented as pronouns or other anaphora in the generated utterance.
- *New*: anything that is in the informational foreground, due to lack of prior mention, but may not be precisely the purpose of the prompt, per se. New material typically receives some kind of prosodic prominence in speech.

Examples of these parameter indices for templates from a theater ticket-reservation domain might appear as in Table 17.4. The basic idea of the parametric approach is that such a level of medium abstraction allows for flexibility in the choice of deployment tactics. If a full set of static prompts and response utterances is available for all cases, then this approach reduces to a template system, though it does provide the potential for separation of grammars and prompt files. If, however, a natural language generation component is available for dynamic message generation, a parameter set like that above can serve as input.

Table 17.4 Sentence generation indices for an airline reservation SLU system.

Act	Type	Body	Given	New	Example
Meta [sorry]	Decl	*no*	-	-	*No, sorry.*
Verify	Y/N-Q			Boston	*Boston?*
Request-info	WH-Q	*fly*	you	thing	*When do you want to fly?*
Request-info	WH-Q	*want*	you	airline tomorrow	*Which airline would you like to fly tomorrow?*
Stmt[sorry]	Decl	*sold out*	it	-	*Sorry it is sold out.*
Stmt	Decl	*sold out*	USAir	-	*Sorry, USAir is sold out.*

17.7.2. Concept-to-Speech Rendition

Once the response content is generated, the SLU system needs to render it into a waveform to play to the users. The task is naturally assigned to a text-to-speech component. However, the response generated in the previous session is more than text message. It contains the underlying semantic information, because it is usually embedded in the semantic representation as shown in Figure 17.23 and Figure 17.24. This is why the speech rendition is often done through a *concept-to-speech* module. A concept-to-speech system can be considered as a text-to-speech system with input text enhanced with domain knowledge tags. With these extra tags, a concept-to-speech system should be able to generate tailored speech output to better convey the system intention.

Chapter 15 discussed the role of prosody in human perception. When messages are generated, it is expected that they are supplemented with hints as to their information structure. At a minimum, the message generation component can identify which parts of the utterance constitute the *theme*, which is material understood, previously mentioned, or

Amazon/VB Assets
Exhibit 1011
Page 925

somehow extending a longer thread of coherence in the dialog, from the *rheme*, which is the unique contribution of the present utterance to the discourse [36]. If such a distinction is marked on the generated utterances, or templates, it can be associated with characteristic pitch contour, prosodic phrasing, and other effects (see Chapter 15).

For example, in the question-answer pair shown in Figure 17.27 (from ordinary human conversation), the theme and rheme components are bracketed. The theme of the answer consists of a mention of Mary, and the act of driving, both carried forward from the question. The theme consists of new information, the answer to the question, embedded in a kind of placeholder noun phrase. Clearly, the input to the message generation component requires some indication of which entities of the input semantic representation are linked to discourse history.

Q: Which car did Mary drive?
A: (Mary drove)$_{th}$ (the RED car.)$_{rh}$

Figure 17.27 A question-answer pair with theme and rheme components marked.

Prosodic rules are triggered by information structure. In general, a theme in the early part of a statement may be realized with a rise-fall-rise pitch contour, often with turning points in the contour aligned with lexically stressed or other salient syllables of the words in the theme. Rheme marking by pitch contour is also essential for naturalness, and a common rheme tune in English declaratives is a slight rise up to the final lexically stressed syllable, followed by a fall to the bottom of the speaker's pitch range. The actual alignment of pitch extrema will depend on the position of focus, or maximum contrast and information value, within either the theme or the rheme.

In Figure 17.27, the word *RED* is in focus within the rheme. If the question had implied a contrast between Mary's car and other people's cars, it would be acceptable to esta b-lish a focus on *Mary* in the theme as well, marked by a pitch accent (see Chapter 15). Sometimes the portion of either theme or rheme that is not in focus (e.g., *drove* or *car*) is called the *ground* [54, 55].

The response generator could add such rheme-theme information that may be used to trigger more specialized prosodic rules. For example, one experimental system is based on a message generator that dynamically creates concise descriptions of individual museum objects during a tour, while attempting to maximize correlations to objects a museum visitor has already seen [21]. During the response generation phase, simple entities and factual statements are combined, first into a semantic graph and then into a text, in which the rhetorical functions of utterances and clauses, and their relations to one another, are known. This information can be passed along to a synthesizer in the form of markup tags within the text. A synthesizer can then select appropriately interesting pitch contours that indirectly reflect rhetorical functions.

In a dialog system, other attributes beyond rheme-theme kinds of information structure, such as speech-act type, may have characteristic intonation patterns. This might include a regretful-sounding contour (perhaps sampled from real speaker data) applied when apologizing (*Sorry, that flight is sold out*) or a cheerful-sounding greeting. Although the concept-to-speech module can be implemented as just a text-to-speech system that take the advan-

Amazon/VB Assets
Exhibit 1011
Page 926

tage of the extra semantic knowledge to generate appropriate prosody, the most natural speech rendition is still to play back a prestored waveform for the entire message. This is why the concept-to-speech module usually relies heavily on playback of template waveform. However, it is obvious that we can't record every possible message like "*Flight [flight_no] is schedule to land at [sch_time]*" in Figure 17.24. Instead, a carrier sentence can be recorded and the slots can then be replaced with real information. The slot can be synthesized with an adapted TTS, which essentially eliminates the need for a front end in the TTS system.

One problem of this approach is that the same prosody is used for a word regardless of where it appears, which results in lower naturalness, because prosodic context is important for natural speech. Enhanced quality can be achieved by having different instances of those slot words, depending their contexts. For example, we can have different *one* recordings depending on whether it is the first digit on a flight number, the second, or the last. Determining the number of different contexts where a slot needs to be recorded is typically done much like the context-dependent acoustic modeling discussed in Chapter 9. This technique increases the naturalness, at the expense of increasing the number of necessary recordings.

17.7.3. Other Renditions

So far, we have assumed that a dialog system may be used only in a speech-only modality. Although such systems have found many applications, multi+modal interaction may be more compelling, as discussed in Chapter 18. In fact, voice output might not be the best information carrier in such an environment. For example, the latest wireless phones are equipped with an LCD screen that allows for e-mail and Web access. If a high-resolution screen is available, the renditions mechanism will likely be visually oriented.

When renditions become visually oriented, the message generation component needs to be replaced by a graphic display component. Since GUI has been the dominant platform for deploying major computer applications today, the behavior and technique of such a display component is well studied and documented [17]. The SLU system needs only to pass the semantic representation from the dialog management module to a GUI rendering module. Of course, the GUI rendering module should also be equipped with domain knowledge to generate best rendering to convey the dialog message. MiPad [22] is such an example and is discussed in Chapter 18.

17.8. EVALUATION

How do we define a quantitative measure for understanding? Evaluation of understanding and dialog is a research topic on its own. We review a number of research techniques being pursued.

17.8.1. Evaluation in the ATIS Task

An application used for development, testing, and demonstration of a wide variety of dialog systems is the Air Travel Information Service (ATIS) task, sponsored by the DARPA Spoken Language Systems program [20]. In this task, users ask about flight information and

Amazon/VB Assets
Exhibit 1011
Page 927

make travel arrangements. To enable consistent evaluation of progress across systems, a corpus of data for this task has been collected and shared among research sites.

The application database contains information about flights, fares, airlines, cities, airports, and ground services, organized in a relational schema. Most user queries, though they may require some system interaction in order to specify fully, can be answered with a single relational query. The ATIS data collection is done using the wizard-of-oz framework.[4] A user interacts with the system as though working with a fully automated travel planner. Hidden human *wizards* were used in the data-collection process to provide efficient and correct responses to the subjects. A typical scenario presented as a task for a subject to accomplish by means of the automated assistant is as follows:

> Plan the travel arrangements for a small family reunion:
>
> First pick a city where the get-together will be held. From three different cities (of your choice), find travel arrangements that are suitable for the family members who typify the *economy*, *high class*, and *adventurous* life styles.

After data collection, each query was classified as context dependent or context independent. A context-dependent query relies partially on past queries for specification, such as "*Is that a non-stop flight?*" Many of the system tests based on ATIS require not only accuracy of speech recognition (the user's spoken query), but also semantic interpretation sufficient to construct an SQL query to the database and correctly complete the desired transaction. Evaluation of ATIS was based on three benchmarks: SPREC (speech recognition performance), NL (natural language understanding for text transcription of spoken utterances), and SLU (spoken language understanding). For SLU systems we are interested only in the last two benchmarks.

With the help of constrained domain of ATIS, correct understanding can be translated into correct database access. Since database access is usually done via SQL database query, the evaluation of understanding can be performed in the domain of generated SQL queries. However, it is still ambiguous when someone would like to query flights around 11:00 a.m. For the purpose of understanding, how wide a time frame is *around* considered to be?

Many examples of queries contain some ambiguities. For instance, when querying about the flights between city X and Y, should the system display only the flights from X to Y; or flights in both directions. To alleviate the ambiguity, each release of ATIS training corpus was accompanied by a *Principles of Interpretation* document that has standard definitions of the meaning of such terms like *around* (means within a 15-minute window) and *between* (means only *from*).

Once the correct understanding is represented as an SQL query, ATIS can be easily evaluated by comparing the SQL queries generated by SLU systems against the standard labeled SQL queries. The utterances in ATIS are classified into three types:

[4] The wizard-of-oz data collection framework is described in Chapter 18.

Amazon/VB Assets
Exhibit 1011
Page 928

- A—semantically independent of earlier utterances, so per-turn semantic interpretation can uniquely identify the semantic intent.
- D—semantically dependent upon earlier utterances, so discourse knowledge is required to provide full interpretation.
- X—unevaluatable, so a response such as *No answer* or *I don't understand you, could you repeat yourself* is considered a right answer.

The other debatable item is whether a *No answer* output for type A and D utterances should be treated equally as a false SQL query. In the original 1991 ATIS evaluation, a false SQL query for type A and D utterances is penalized twice as heavily as a *No answer* output for type A and D utterances. However, the decision was dropped for the 1993 ATIS evaluation. ATIS decided not to evaluate dialog component for three reasons. First, dialog alters users' behavior during data collection. Users' utterances are highly contingent on the performance of the wizard-of-oz system, so the data collected has little use for systematic training and testing. Second, the SLU systems would likely have to be tested by real subjects. Third, the evaluation of dialog behavior is highly subjective, since effectiveness and user friendliness are generally vaguely defined.

17.8.2. PARADISE Framework

The evaluation of a dialog system is subjective in nature and is typically done in an end-to-end fashion. In such a framework, objective criteria like number of dialog turns and system throughput, and subjective measures like user satisfaction, are typically used.

One of the most sophisticated systems for evaluating dialog systems ever developed is the PARAdigm for Dialog System Evaluation (PARADISE) [57]. The designers of this framework took a comprehensive view of the many potential factors affecting dialog evaluation, in particular the distinction between measuring success of transaction (quality) and cost of the dialog, both in human and system terms. A decision-theoretic method, as shown in Figure 17.28, is used to explicitly weight these various disparate factors to achieve a unified measure. In addition, the PARADISE metrics can derive discrete scores for subdialogs, which is useful for diagnosis, comparison across systems, and tuning.

A simple measure for task success can be the following question: "*Was all the needed information exchanged, in the correct direction (user to system, system to user) at each step*?" PARADISE provides a framework for defining, for any interaction in a limited domain, a simplified representation of the minimal required information and its directionality. In PARADISE terms, this is an *attribute-value matrix* (AVM) showing the names and instantiations of required elements at dialog completion. This could be derived from reference frames for each required concept in a dialog exchange, with mandatory slots marked for legal completions. Once such reference frames or matrices are available, different dialog strategies that address the same function can be compared over many instantiations (test dialog sessions), using statistical measures that assess confusability and length.

Amazon/VB Assets
Exhibit 1011
Page 929

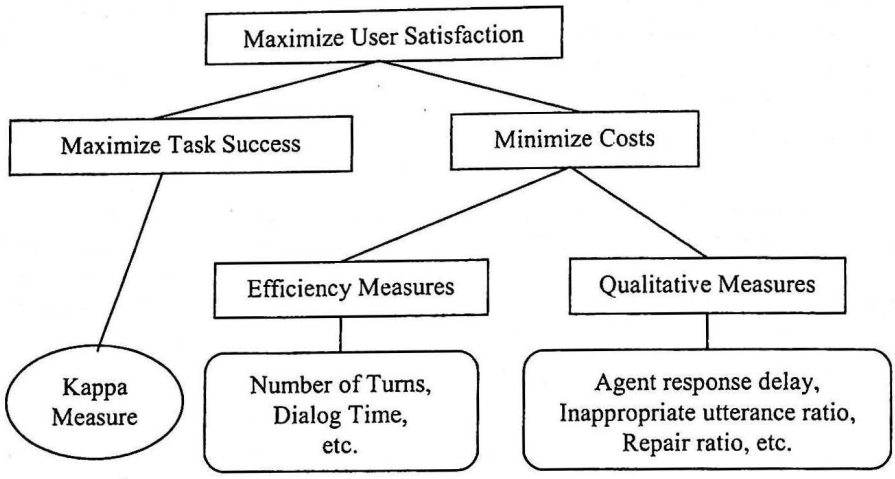


Figure 17.28 PARADISE's structure of objectives for spoken dialog performance [57].

For example, imagine an ATIS-like application that had the following information attributes, with the possible values listed in Table 17.5. An utterance such as "*I want to go from Torin to Milan*" communicates legal DC and AC attribute values from user to system. This is a limited-domain system by assumption, so confusions are assumed to occur within the possible values of the application. For example, if the system instantiates the *Depart-City* (DC) slot with *Trento* instead of *Torin* after processing the given sample utterance, it is a confusion that can be recorded in a confusability matrix over all dialog test sessions. A subsection of such a possible confusability matrix, covering only the DC and AC attributes, is shown in Table 17.6, which shows only confusion within an attribute type that covers a consistent vocabulary (city names, instantiating the DC and AC attributes). In practice, however, the full matrix might show confusions across attribute types, such as *morning* for *Milan*, etc.

Given a confusability matrix M over all possible attributes in the application, we can apply the Kappa coefficient [48] to measure the quality characterizing the task's success at meeting the information requirements of the application:

Table 17.5 Attribute-value table [57].

Attribute	Possible Values
Depart-City (DC)	Milan, Rome, Torin, Trento
Arrival-City (AC)	Milan, Rome, Torin, Trento
Depart-Range (DR)	Morning, evening
Depart-Time (DT)	6am, 8am, 6pm, 8pm

Amazon/VB Assets
Exhibit 1011
Page 930

Table 17.6 Confusability matrix for city identification [57].

	Depart-City				Arrival-City			
Data	Milan	Rome	Torin	Trento	Milan	Rome	Torin	Trento
Milan (depart)	**22**		1		3			
Rome (depart)		**29**						
Torin (depart)	4		**16**	4			1	
Trento (depart)	1	1	5	**11**			1	
Milan (arrive)	3				**20**			
Rome (arrive)						**22**		
Torin (arrive)			2		1	1	**20**	5
Trento (arrive)			1		1	2	8	**15**
sum	30	30	25	15	25	25	30	20

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)} \tag{17.8}$$

where $P(A)$ is the proportion of times that the AVMs for the actual set of dialog agree with the AVMs for the interpreted results, and $P(E)$ is the proportion of times that AVMs for the dialog and interpreted results are expected to agree by chance. $P(E)$ can be estimated by $P(E) = \sum_{i=1}^{n} \left(\frac{t_i}{T}\right)^2$, where t_i is the sum of the frequencies in column i of M and T is total frequencies $(t_1 + \cdots + t_n)$ in M. The measure of $P(A)$ (how well or poorly the application did in information extraction) is calculated simply by examining how much of the total count occurs on the diagonal: $P(A) = \sum_{i=1}^{n} M(i,i)/T$.

In addition to task success, system performance is also a function of several cost measures. Cost measures include efficiency measures, such as the number of dialog turns or task completion time; as well as qualitative measures, such as style of dialog or how good the repair mechanism is. If a set of test dialogs is available, with experimentally measured user satisfaction (the predicted categories), the kappa measure, and quantitative measures of cost (denoted as c_i, such as counts of repetitions, repairs etc.), linear regression can be used, over the z-score normalization of these predictor terms, to identify and weight the most important predictors of satisfaction for a given system. Thus, the performance can be defined as:

$$\text{Performance} = \alpha * \mathfrak{N}(\kappa) - \sum_{i=1}^{n} w_i * \mathfrak{N}(c_i) \tag{17.9}$$

where $\mathfrak{N}$ is the z-score normalization function $\mathfrak{N}(x) = \frac{x - \overline{x}}{\sigma_x}$.

Amazon/VB Assets
Exhibit 1011
Page 931

Evaluating a dialog system involves having a group of users perform tasks with ideal outcomes. Then the cost measures and task success kappa measure are estimated. These measures are used to derive the regression weights in Eq. (17.9). Once the regression weights are attained, one could possibly predict the user satisfaction when a subpart of the dialog system is improved.

17.9. CASE STUDY—DR. WHO

Dr. Who is a project at Microsoft Research on its multimodal dialog system development. It incorporates many of the dialog technologies described in this chapter. We use Dr. Who's SLU engine as an example to illustrate how to effectively create practical systems [22, 58–61]. It follows the mathematical framework illustrated in Eq. (17.1). The system architecture is shown in Figure 17.29. Since it intends to serve as a general architecture for multimodal dialog systems, it makes some simple assumptions at the architecture level. First, it replaces the *speech recognizer* and *sentence interpretation* modules with a *semantic parser* for each modality. The *response rendering* is merged into *dialog manager* with different XSL style sheets for each media output.

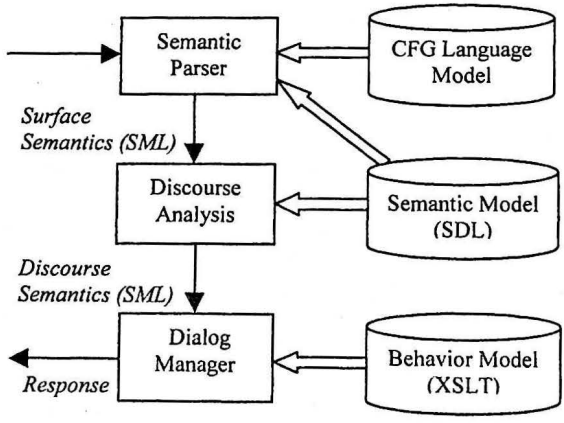


Figure 17.29 The Dr. Who system architecture [60].

17.9.1. Semantic Representation

Semantic representation is a critical part in Dr. Who's SLU engine design. Essentially, the semantic objects are an abstraction of the speech acts, the domain knowledge, and the application logic. They are designed to encapsulate the respective language models and dialog actions that govern their creation and behaviors. The system components communicate with one another through events surrounding the semantic objects. In this view, the dialog (including logic inferences) is an integral part of the discourse semantic evaluation process.

There are two types of semantic objects in Dr. Who. The first type is the *functional* semantic object that is used to represent linguistic expressions in the user's utterance. The

Amazon/VB Assets
Exhibit 1011
Page 932

second type is the *physical* semantic object that is used to represent real-world entities related to the application domain. Both types of semantic objects are represented by semantic frames and specified in the semantic markup language (SML), which is an extension of XML. Following the principles of the XML schema, Dr. Who defines the schema of SML in another XML called semantic definition language (SDL). SDL is designed to support many discourse and dialog features. In addition, SDL is suited to represent the domain knowledge via the application schema, the hierarchy of the semantic objects, and the semantic inference rules.

The format of various semantic classes follows SDL representations in Dr. Who. The terminal and nonterminal nodes on the parse are denoted in SDL with tags `<verbatim>` and `<class>`, respectively. These tags refer to the semantic objects and have the *name* and *type* attributes. The `type` attribute corresponds to the entity type the semantic object eventually would be converted to; it plays a key role in inheritance and polymorphism, as described in Section 17.3.1. When a semantic object is unique in its type, SDL can automatically assume its type as the name. In addition, SDL defines a `<cfg>` tag for the language model that governs the instantiation of a semantic object ,and the language model could be stored in another file. An `<expert>` tag can be defined for the system resource to physically convert a semantic object to a domain entity. Finally, the tag `<slot>` in SDL defines the descendant for a nonterminal node.

Take the semantic class for Microsoft employee directory as an example. The simple application answers queries on an employee's data such as office location, phone number, hiring date, etc. An item that can be asked is a semantic terminal `DirectoryItem` as defined in Figure 17.30. To allow users to ask more than one directory item at one dialog turn, a multiple semantic class `DirectoryItems` is also defined recursively, as shown in Figure 17.30.

```
<verbatim type="DirectoryItem" ...>
   <prod name="office"/>
   <prod name="phone"/>
   <prod name="hiring date"/>
   ...
</verbatim>
<class type="DirectoryItems" ...>
   <slot type="DirectoryItem"/>
   <slot type="DirectoryItems"/>
   <cfg ref="DirectoryItems.cfg"/>
</class>
```

Figure 17.30 The terminal semantic class `DirectoryItem` and nonterminal semantic class defined in Dr. Who using SDL. Note that the definition of `DirectoryItems` contains a recursive style, which can accommodate more than one `DirectoryItem` [59].

Amazon/VB Assets
Exhibit 1011
Page 933

The <prod> tags inside a terminal semantic object indicate that the terminal is of an enumeration type, and all the possible values are text normalized to the string values of the *name* attribute. The main speech act, the query, is modeled by the functional semantic class DirectoryQuery, as shown in Figure 17.31.

```
<class type="DirectoryQuery" ...>
    <slot type="Person"/>
    <slot type="DirectoryItems"/>
    <expert clsid="..."/>
    <cfg ref="Directory.cfg"/>
</class>
<include ref="PeopleGrammar.sdl"/>
```

Figure 17.31 The main semantic class DirectoryQuery defined in Dr. Who using SDL [59].

The semantic object can be instantiated following the language model in "Directory.cfg" and, once instantiated, is handled by a system object identified by its class id (clsid). The system object then formulates the query language that retrieves the data from the database. It is also possible to embed the XML version of the query language (e.g., XQL) within the <expert> tag. Semantic models can be nested and reused, as shown in the <include> tag in the above example, where the semantic model for people is referred.

17.9.2. Semantic Parser (Sentence Interpretation)

For speech modality, Dr. Who employs a speech recognizer with unified language models [62] that take advantage of both rule-based and data-driven approaches, as discussed in Chapter 11. Once we have text transcription of user's utterances, a robust chart parser [61] similar to the one described in Section 17.4.1 is used for sentence interpretation.

The emphasis of sentence interpretation is to annotate the user's utterance in a meaningful way to generate functional semantic entities. Essentially, the surface SML represents a semantic parse. Thus, after a successful parse, the corresponding surface semantic objects are instantiated based on the semantic classes whose CFG grammars are fired. While in SDL we use static tags such as <class> and <verbatim> for the semantic classes, the instances of a semantic object use the object name as the tag in SML. For example, the surface SML for an utterance *"What is the phone number for Kuansan"* is shown in Figure 17.32.

```
<DirectoryQuery ...>
   <PersonByName type="Person" parse="kuansan">
       Kuansan
   </PersonByName>
   <DirectoryItem type="DirectoryItem" parse="phone num-
ber">
       phone
   </DirectoryItem>
</DirectoryQuery>
```

Figure 17.32 The surface semantic object DirectoryQuery represented in SML after a successful parse [59].

Amazon/VB Assets
Exhibit 1011
Page 934

17.9.3. Discourse Analysis

As mentioned in Section 17.5, the goal of discourse analysis is to resolve surface semantic objects to discourse semantic objects. For the surface semantic object in Figure 17.32, the discourse engine binds the three semantic objects (i.e., the person, the directory item, and the directory query itself) to real-word and functional entities represented in the SML example, as shown in Figure 17.33.

```
<DirectoryQuery ...>
   <Person id="kuansanw" parse="kuansan">
      <First>Kuansan</First>
      <Last>Wang</Last>
      ...
   </Person>
   <DirectoryItem parse="phone number">
      <phone>+1(425)703-8377</phone>
   </DirectoryItem>
</DirectoryQuery>
```

Figure 17.33 The discourse semantic objects for the surface semantic object illustrated in Figure 17.32 [59].

Note that the parse string from the user's original utterance is kept so that the rendering engine can choose to rephrase the response using the user's wording.

When an error occurs, the semantic engine inserts an `<error>` tag in the offending semantic objects with a code indicating the error condition. For example, if the query is for a person named *Derek*, the discourse SML might appear as shown in Figure 17.34.

```
<DirectoryQuery status="TBD" focus="Person" ...>
   <PersonByName type="Person" parse="Derek" status="TBD"...>
        <error scode="1" count="27"/>
        <Person id="derekba">
           <First>Derek</First>
           <Last>Baines</Last>
           ...
        </Person>
        <Person id="dbevan">
           <First>Derek</First>
           <Last>Bevan</Last>
           ...
         </Person>
           ...
    </PersonByName>
           ...
</DirectoryQuery>
```

Figure 17.34 A discourse semantic object in Dr. Who contains an `<error>` tag indicating the error condition [59].

Amazon/VB Assets
Exhibit 1011
Page 935

In Figure 17.34, semantic objects that cannot be converted (e.g., `DirectoryQuery` and `PersonByName`) are flagged with a status "TBD". Discourse SML also marks the dialog focus, as in the `DirectoryQuery`, that indicates the places where the semantic evaluation process fails to continue. These two cues assist the behavior model in deciding the appropriate error-repair responses.

Dr. Who uses three priority types of entity memory (discourse memory, explicit, and implicit turn memory) to resolve relative expressions. Anaphora and deixis are treated as common semantic classes, so they can be resolved according to the algorithm described in Section 17.5.1.1. Ellipsis is treated as an automatic inference. Unless marked as NO _INFER in the semantic class definition, every slot in a semantic class can be automatically inferred. The strategy to automatically resolve partially specified entities is as follows.

During the evaluation stage, a partially filled semantic object is first compared with the entities in the three-entity memory based on the type compatibility. If a candidate is found, the discourse analysis module then computes a goodness-of-fit score by consulting the knowledge base and considering the position of the entity in the memory list. The semantic object is converted immediately to the entity from the memory if the score exceeds the threshold. In the process, all the actions implied by the entities are carried out following the order in which the corresponding semantic objects are converted. For example, the second user's query in the dialog illustrated in Figure 17.35 contains an ellipsis reference to `DirectoryItem office`, which can be resolved using the discourse entity memory.

U: Where is his office?
S: The office is in building 31, room 1362.
U: How about Kuansan's?
S: The office is in building 31, room 1363.

Figure 17.35 A dialog example in the Dr. Who system. The second user's query contains an ellipsis reference to `DirectoryItem office` [59].

17.9.4. Dialog Manager

To support mixed-initiative multimodal dialogs, Dr. Who employs a plan-based approach instead of dialog grammars. The dialog manager that handles dialog events surrounding semantic objects is very similar to a GUI program that handles GUI events surrounding graphical objects. These events can be handled synchronously or asynchronously based on various implementation considerations. In addition, the design enables a seamlessly integrated GUI and speech interface for multimodal applications to embrace the same human-computer interaction model.

Dr. Who SLU engine can use XSL-transformations (XSLT) [59] for specifying the behavior of a plan-based dialog system. XSLT, a recent World Wide Web Consortium (W3C) standard, is a specialized XML intended for describing the rules of how a structured document in XML can be transformed into another, say in a text-to-speech markup language for speech rendering or the hypertext markup language (HTML) for visual rendering. Its core construct is a collection of predicate-action pairs: each predicate specifies a textual pattern in the source document, and the corresponding action will produce a text segment in the

Amazon/VB Assets
Exhibit 1011
Page 936

output whenever the pattern specified by the predicate is seen in the source document. The output segment is specified through a programmable, context-sensitive template. XSLT defines a rich set of logical controls for composing the templates. The basic programming paradigm bears close resemblance to a logical programming language, such as Prolog, which facilitates logic inference in plan-based systems. As a result, XSLT possesses sufficient expressive power for implementing crucial dialog components, ranging from defining dialog plans, realizing dialog strategies, and generating natural language, to manipulating prosodic markup for text-to-speech synthesis and creating dynamic HTML pages for multimodal applications.

Assuming TTS output, the planning rules that render the discourse SML of Figure 17.33 in text can be expressed in XSLT as shown in Figure 17.36.

```
<xsl:template match="DirectoryQuery[@not(status)]">
    For <xsl:apply-templates select="Person"/>, the
    <xsl:apply-templates select="DirectoryItem"/>.
</xsl:template>
<xsl:template match="Person">
    <xsl:value-of select="First"/>
    <xsl:value-of select="Last"/>
</xsl:template>
<xsl:template match="DirectoryItem">
    <xsl:apply-templates/>
</xsl:template>
<xsl:template match="phone">
    phone number is <xsl:value-of/>
</xsl:template>
```

Figure 17.36 A TTS response-rendering rule for discourse SML of Figure 17.33. This rule generates a text message "*For Kuansan Wang, the phone number is +1(425)703-8377*" [59].

This rule leads to a response *For Kuansan Wang, the phone number is +1(425)703-8377*. Elaborated functions, such as prosodic manipulations in text to speech markup, can be included accordingly. To change the output to Web presentation, the above XSLT style sheet can be slightly modified for rendering in HTML as a table, as shown in Figure 17.37.

The Dr. Who SLU engine has a concept called *logical container* as a dialog property to be encapsulated in a semantic class. Three types of logical containers can be accessed in the definition of semantic classes. A semantic class is an AND type container if all its attributes must be evaluated successfully. If this requirement is not met, the evaluation of the AND type semantic object is considered failed, which will prompt the system to post a dialog-repair event. An OR type container requires at least one attribute to be successfully evaluated. Similarly, for an exclusive or (XOR) type container, one and only one attribute must be successfully evaluated.

Amazon/VB Assets
Exhibit 1011
Page 937

```
<xsl:template match="DirectoryQuery[@not(status)]">
<TABLE border="1">
    <THEAD><TR>
          <TH>Properties</TH>
          <TH><xsl:apply-templates select="Person"/> </TH>
    </TR></THEAD>
    <TBODY><xsl:apply-templates select="DirectoryItem"/>
    </TBODY>
</TABLE>
</xsl:template>
<xsl:template match="phone">
    <TR> <TD>phone</TD> <TD> <xsl:value-of /> </TD> </TR>
</xsl:template>
```

Figure 17.37 An HTML response-rendering rule for discourse SML of Figure 17.33. It generates a visual table representation rather than a text message [59].

Figure 17.38 shows a semantic class hierarchy corresponding to the partial plan shown in Figure 17.20. The dialog goal—to gather information for booking a flight—corresponds to the highest-level semantic class *Book Flight*. Evaluating this semantic class drives the dialog system to traverse down the semantic class structure, eventually fulfilling all the steps necessary to achieve the dialog goal. This is achieved by recursively evaluating the attributes, instantiating semantic objects actively if necessary. The logical relation of each semantic class determines the rules of instantiation and dialog repair. For instance, if the user specifies the trip to be one way only, the evaluation of the *One Way Flag* semantic class becomes successful. As the *Inbound Trip* semantic class is an XOR container, the dialog system bypasses the evaluation of the *Itinerary* attribute in the *Inbound Trip* semantic class.

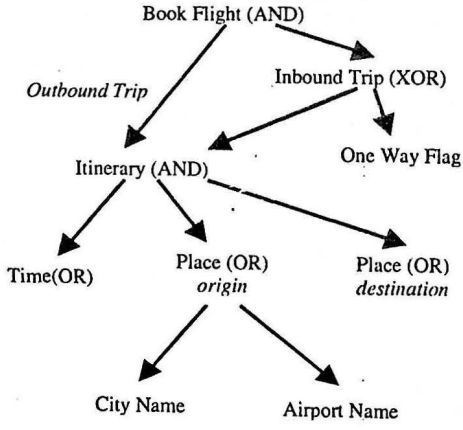


Figure 17.38 A semantic tree hierarchy corresponding to the partial plan shown in Figure 17.20 in an airline reservation application [58].

Amazon/VB Assets
Exhibit 1011
Page 938

The *Itinerary* semantic class encapsulates the basic elements to specify a one-way trip. Since it is designated as an AND type container, the dialog manager tries to acquire any missing information by actively instantiating the corresponding semantic classes it contains. The active instantiation event handlers for these classes solicit information from the user by implementing certain prompting strategy. On the other hand, the *Place* semantic class, which is used to denote both the origin and the destination, is implemented as an OR container. The user may specify the location by either the city name or the airport name.

17.10. HISTORICAL PERSPECTIVE AND FURTHER READING

Traditional natural language research has its roots in symbolic systems. Motivated by the desire to—understand cognitive processes, the underlying theories tend to be from linguistics and psychology. As a result, coverage of phenomena of theoretical interest (usually a rare occurrence) has traditionally been more important than developing systems with a broad coverage.

On the other hand, speech recognition research is driven to produce practical usable applications. Techniques motivated by knowledge of human processes have been less important than techniques that can be used for real applications. In recent decades, interest has grown in the use of engineering techniques in computational language processing, although the use of linguistic knowledge and techniques in engineering has lagged somewhat. The ATIS program sponsored by DARPA had a very significant influence upon the SLU research community [34]. For the first time, the research community started seriously evaluating SLU systems on a quantitative basis, which revealed that many traditional NL techniques designed for written language failed to deal with spoken language in practice.

For limited-domain SLU applications, vocabularies are typically about 2000 words. CMU's Phoenix SLU system [63] set the benchmark for domain-specific spoken language understanding in the DARPA ATIS programs. It is based on an island-driven semantic parsing approach. After years of engineering, the speech understanding error rate ranges from 6% to 41%. Since conversational repairs in human-human dialog can often be in the same range for these systems, the determining factor in these domain-specific SLU applications may not be the error rates but instead the ability of the system to manage and recover from errors. Many of these were described in detail in the *Proceedings of the DARPA Spoken Language Systems Technology Workshop* published by Morgan Kaufmann from 1991 to 1995. The special issue *of Speech Communication on Spoken Dialog* [45] also includes several state-of-the-art system descriptions.

Allen's *Natural Language Understanding* [1] is a good book on natural language understanding with a comprehensive coverage of syntactic processing, semantic processing, discourse analysis, and dialog agent. Knowledge and semantic representation comprise the most import fundamental issue for symbolic artificial intelligence. Several AI textbooks [33, 56, 65] contain comprehensive description of knowledge representation. The use of semantic frames can be traced back to case frames or structures proposed by Fillmore [16]. SAM [44] is among the first systems using semantic frames and template matcher for natural language

Amazon/VB Assets
Exhibit 1011
Page 939

processing. The description of semantic classes and frames in this book mostly follows the systematic treatment of semantic classes in the Dr. Who system [58-60].

Speech-act (sometimes called dialog-act) theory was first proposed by Austin [4] and further developed by Searle [42]. It is an important concept in dialog systems. You can acquire more information about speech-act theory and its application to dialog systems from [12, 40, 43]. Cohen [10] provides a good comparison of different approaches for dialog modeling, including dialog grammar (finite state), plan-based and agent-based (dialog as teamwork). We treat agent-based dialog modeling as an extension of plan-based dialog modeling, as described in Section 17.6.2. Agent-based approach is a very popular framework for multimodal user interface, and interested readers can refer to [11]. Hudson and Newell [23] incorporate probability into finite state dialog management to handle uncertainty in input modalities, such as pen-based interface, gesture recognition, and speech recognition. J. Allen's book [1] has a systematic description of plan-based dialog systems. De Mori's *Spoken Dialogs with Computers* [39] is another excellent book that contains dialog systems and related technologies.

Much of the content in this chapter follows the architecture and implementation of semantic frame based approaches. In particular, we use plenty of descriptions and examples of the Dr. Who SLU engine developed at Microsoft Research [22, 58-60]. The description of plan-based systems is based on semantic frame representation and pattern matching. There is no need for explicit dialog-act analysis and logic reasoning, since these important knowledge sources are encapsulated in the semantic frames.

In addition to the semantic frame-based approach, there other approaches that rely on formal NL parsing, logic form representation, speech acts, and logic inference [2, 41]. Message generation for telephone application is well studied and reported in [5, 6, 49], which provide experimental results for various prompting strategies. Most evaluation schemes for the SLU systems focus on the end-to-end system. Human factors are important in overall evaluation [7, 35, 52].

REFERENCES

[1] Allen, J., *Natural Language Understanding*, 2nd ed., 1995, Menlo Park CA, The Benjamin/Cummings Publishing Company.

[2] Allen, J.F., *et al.*, "Trains as an Embodied Natural Language System," *AAAI-95 Symposium on Embodied Language and Action*, 1995.

[3] Andernach, T., M. Poel, and E. Salomons, "Finding Classes of Dialogue Utterances with Kohonen Networks," *Proc. of the NLP Workshop of the European Conf. on Machine Learning (ECML)*, 1997, Prague, Czech Republic.

[4] Austin, J.L., *How to Do Things with Words*, 1962, Cambridge, MA, Harvard University Press.

[5] Basson, S., "Integrating Speech Recognition and Speech Synthesis in the Telephone Network," *Proc. of the Human Factors Society 36th Annual Meeting*, 1992.

[6] Basson, S., "Prompting The User in ASR Applications," *Proc. of COST232 (European Cooperation in Science and Technology) Workshop*, 1992.

Amazon/VB Assets
Exhibit 1011
Page 940

[7] Basson, S., *et al.*, "User Participation and Compliance in Speech Automated Telecommunications Applications," *Proc. of the Int. Conf. on Spoken Language Processing*, 1996, pp. 1680-1683.

[8] Biber, D., *Variation Across Speech and Writing*, 1988, Cambridge University Press.

[9] Clark, H.H. and S.E. Haviland, "Comprehension and the Given-New Contract" in *Discourse production and comprehension*, R.O. Freedle, Editor 1977, Norwood, NJ, Ablex Publishing Corporation, pp. 1-38.

[10] Cohen, P., "Models of Dialogue," *Proc. of the Fourth NEC Research Symposium*, 1994, SIAM Press.

[11] Cohen, P.R., "The Role of Natural Language in a Multimodal Interface," *Proc. of the ACM Symposium on User Interface Software and Technology*, 1992, pp. 143-149.

[12] Cohen, P.R. and C.R. Perrault, "Elements of a Plan-Based Theory of Speech Acts," *Cognitive Science*, 1979, 3(3), pp. 177-212.

[13] Constantinides, P.H., S. Tchou, C. Rudnicky, "A Schema Based Approach to Dialog Control," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, pp. 409-412.

[14] Core, M. and J. Allen, "Coding Dialogs with the DAMSL Annotation Scheme," *Proc. AAAI Fall Symposium on Communicative Action in Humans and Machines*, 1997.

[15] Davies, K., *et al.*, "The IBM Conversational Telephony System For Financial Applications," *EuroSpeech'99*, 1999, Budapest, Hungary, pp. 275-278.

[16] Fillmore, C.J., "The Case for Case" in *Universals in Linguistic Theory*, E. Bach and R. Harms, eds. 1968, New York, NY, Holt, Rinehart and Winston.

[17] Galitz, W.O., *The Essential Guide to User Interface Design: An Introduction to Gui Design Principles and Techniques*, 1996, John Wiley & Sons.

[18] Grosz, B., M. Pollack, and C. Sidner, eds. *Discourse*, in Foundations of Cognitive Science, ed. M. Posner, 1989, MIT Press.

[19] Heeman, P.A., *et al.*, "Beyond Structured Dialogues: Factoring Out Grounding," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[20] Hemphill, C.T., J.J. Godfrey, and G.R. Doddington, "The ATIS Spoken Language Systems Pilot Corpus," *Proc. of the Speech and Natural Language Workshop*, 1990 pp. 96-101.

[21] Hitzeman, J., *et al.*, "On the Use of Automatically Generated Discourse-Level Information in a Concept-to-Speech Synthesis System," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia, pp. 2763-2766.

[22] Huang, X., *et al.*, "MIPAD: A Next Generation PDA Prototype," *Int. Conf. on Spoken Language Processing*, 2000, Beijing, China.

[23] Hudson, S.E. and G.L. Newell, "Probabilistic State Machines: Dialog Management for Inputs with Uncertainty," *Proc. of the ACM Symposium on User Interface Software and Technology*, 1992, pp. 199-208.

Amazon/VB Assets
Exhibit 1011
Page 941

[24] Hulstijn, J. and A.V. Hessen, "Utterance Generation for Trans. Dialogues," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[25] Jackendoff, R.S., *X' Syntax: A Study of Phrase Structure*, 1977, Cambridge, MA, MIT Press.

[26] Jelinek, F., *et al.*, "Decision Tree Parsing using Hidden Derivational Model," *Proc. of the ARPA Human Language Technology Workshop*, 1994, pp. 272-277.

[27] Jurafsky, D., L. Shriberg, and D. Biasca, *Switchboard SWBD-DAMSL Shallow-Discourse-Function Annotation Coders Manual, Draft 13*, 1997, http://www.colorado.edu/linguistics/jurafsky/manual.august1.html.

[28] LDC, *Linguistic Data Consortium*, 2000, http://www.ldc.upenn.edu/ldc/noframe.html.

[29] Miller, S., *et al.*, "Recent Progress in Hidden Understanding Models," *Proc. of the ARPA Spoken Language Systems Technology Workshop*, 1995, Austin, Texas, Morgan Kaufmann, Los Altos, CA, pp. 22-25.

[30] Miller, S. and R. Bobrow, "Statistical Language Processing Using Hidden Understanding Models," *Proc. of the Spoken Language Technology Workshop*, 1994, Plainsboro, New Jersey, Morgan Kaufmann, Los Altos, CA, pp. 48-52.

[31] Minsky, M., "A Framework for Representing Knowledge" in *The Psychology for Computer Vision*, P.H. Winston, Editor 1975, New York, NY, McGraw-Hill.

[32] Nilsson, N.J., *Principles of Artificial Intelligence*, 1982, Berlin, Germany, Springer-Verlag.

[33] Nilsson, N.J., *Artificial Intelligence: A New Synthesis*, 1998, Academic Press/Morgan Kaufmann.

[34] Pallett, D.S., *et al.*, "1994 Benchmark Tests for the ARPA Spoken Language Program," *Proc. of the 1995 ARPA Human Language Technology Workshop*, 1995, pp. 5-36.

[35] Polifroni, J., *et al.*, "Evaluation Methodology for a Telephone-Based Conversational System," *The First Int. Conf. on Language Resources and Evaluation*, 1998, Granada, Spain, pp. 42-50.

[36] Prevost, S. and M. Steedman, "Specifying Intonation from Context for Speech Synthesis," *Speech Communication*, 1994, **15**, pp. 139-153.

[37] Reinhart, T., *Anaphora and Semantic Interpretation*, Croom Helm Linguistics Series, 1983, University of Chicago Press.

[38] Roberts, D., *The Existential Graphs of Charles S. Peirce*, 1973, Mouton and Co.

[39] Sadek, D. and R. De Mori, "Dialogue Systems" in *Spoken Dialogues with Computers*, R. De Mori, Editor 1998, London, UK, pp. 523-561, Academic Press.

[40] Sadek, M.D., "Dialogue Acts are Rational Plans," *Proc. of the ESCA/ETRW Workshop on the Structure of Multimodal Dialogue*, 1991, Maratea, Italy, pp. 1-29.

[41] Sadek, M.D., *et al.*, "Effective Human-Computer Cooperative Spoken Dialogue: The AGS Demonstrator," *Proc. of the Int. Conf. on Spoken Language Processing*, 1996, Philadelphia, Pennsylvania, pp. 546-549.

[42] Searle, J.R., *Speech Acts: An Essay in the Philosophy of Language*, 1969, UK, Cambridge University Press.

Amazon/VB Assets
Exhibit 1011
Page 942

[43] Searle, J.R. and D. Vanderveken, *Foundations of Illocutionary Logic*, 1985, Cambridge University Press.

[44] Shank, R., *Conceptual Information Processing*, 1975, North Holland, Amsterdam, The Netherlands.

[45] Shirai, K. and S. Furui, "Special Issue on Spoken Dialogue," *Speech Communication*, 1994, **15**.

[46] Shriberg, E.E., R. Bates, and A. Stolcke, "A Prosody-Only Decision-tree Model for Disfluency Detection," *Proc. Eurospeech*, 1997, Rhodes, Greece, pp. 2383-2386.

[47] Sidner, C., "Focusing in the Comprehension of Definite Anaphora" in *Computational Model of Discourse*, M. Brady, Berwick, R., eds., 1983, Cambridge, MA, pp. 267-330, The MIT Press.

[48] Sidney, S. and N.J. Castellan, *Nonparametric Statistics for the Behavioral Sciences*, 1988, McGraw Hill.

[49] Sorin, C. and R.D. Mori, "Sentence Generation" in *Spoken Dialogues with Computers*, R.D. Mori, Editor 1998, London, UK, Academic Press, pp. 563-582.

[50] Souvignier, B., *et al.*, "The Thoughtful Elephant: Strategies for Spoken Dialog Systems," *IEEE Trans. on Speech and Audio Processing*, 2000, **8**(1), pp. 51-62.

[51] Sowa, J.F., *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, 1999, Brooks Cole Publishing Co.

[52] Springer, S., S. Basson, and J. Spitz, "Identification of Principal Ergonomic Requirements for Interactive Spoken Language Systems," *Int. Conf. on Spoken Language Processing*, 1992, pp. 1395-1398.

[53] Standards, N.C.f.I.T., *Conceptual Graph Standard Information*, 1999, http://www.bestweb.net/~sowa/cg/cgdpansw.htm.

[54] Steedman, M., ed. *Parsing Spoken Language Using Combinatory Grammars*, in Current Issues in Parsing Technology, ed. M. Tomita, 1991, Kluwer Academic Publishers.

[55] Steedman, M., "Information Structure and the Syntax-Phonology Interface," *Linguistic Inquiry*, 2000.

[56] Tanimoto, S.L., *The Elements of Artificial Intelligence: An Introduction Using Lisp*, 1987, Computer Science Press, Inc.

[57] Walker, M., *et al.*, "PARADISE: A Framework for Evaluating Spoken Dialogue Agents.," *Proc. of the 35th Annual Meeting of the Association for Computational Linguistics (ACL-97)*, 1997, pp. 271-280.

[58] Wang, K., "An Event Driven Model for Dialogue Systems," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia pp. 393-396.

[59] Wang, K., "Implementation of Dr. Who Dialog System Using Extended Markup Languages," *Int. Conf. on Spoken Language Processing*, 2000, Beijing, China.

[60] Wang, K., "A Plan-Based Dialog System With Probabilistic Inferences," *ICSLP*, 2000, Beijing, China.

[61] Wang, Y., "A Robust Parser For Spoken Language Understanding," *Eurospeech*, 1999, Budapest, Hungary, pp. 2055-2058.

Amazon/VB Assets
Exhibit 1011
Page 943

[62] Wang, Y., M. Mahajan, and X. Huang, "A Unified Context-Free Grammar and N-Gram Model for Spoken Language Processing," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1639-1642.

[63] Ward, W., "Understanding Spontaneous Speech: The Phoenix System," *Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, Canada, pp. 365-367.

[64] Ward, W. and S. Issar, "The CMU ATIS System," *Proc. of the ARPA Spoken Language Systems Technology Workshop*, 1995, Austin, Texas, Morgan Kaufmann, Palo Alto, CA.

[65] Winston, P.H., *Artificial Intelligence*, 3rd ed, 1992, Reading, MA, Addison-Wesley.

[66] Young, S.R., *et al.*, "High Level Knowledge Sources in Usable Speech Recognition Systems," *Communications of the Association for Computing Machines*, 1989, **32**(2), pp. 183-194.

Amazon/VB Assets
Exhibit 1011
Page 944

CHAPTER 18

Applications and User Interfaces

The ultimate impact of spoken language technologies depends on whether you can fully integrate the enabling technologies with applications so that users find it easy to communicate with computers. How to effectively integrate speech into applications often depends on the nature of the user interface and application. This is why we group user interface and application together in this chapter. In discussing some general principles and guidelines in developing spoken language applications, we must look closely at designing the user interface.

A well-designed user interface entails carefully considering the particular user group of the application and delivering an application that works effectively and efficiently. As a general guideline, you need to make sure that the interface matches the way users want to accomplish a task. You also need to use the most appropriate modality at the appropriate time to assist users to achieve their goals. One unique challenge in spoken language applications is that neither speech recognition nor understanding is perfect. In addition, the spoken command can be ambiguous, so the dialog strategy described in Chapter 17 is necessary to clarify the goal of the speaker. There are always mistakes you have to deal with. It is critical

Amazon/VB Assets
Exhibit 1011
Page 945

that applications employ necessary interactive error-handling techniques to minimize the impact of these errors. Application developers should therefore fully understand the strengths and weaknesses of the underlying speech technologies and identify the appropriate place to use the spoken language technology effectively.

This chapter mirrors Chapter 2, in the sense that you need to incorporate all the needed components of speech communication to make a spoken language system work well. It is important also to have your applications developed based on some standard application programming interfaces (API), which ensures that multiple applications work well with a wide range of speech components provided by different speech technology providers.

18.1. APPLICATION ARCHITECTURE

A typical spoken language application has three key components. It needs an engine that can be either a speech recognizer or a spoken language understanding system. An *application programming interface* (API) is often used to facilitate the communication between the engine and application, as illustrated in Figure 18.1. Multiple applications can interact with a shared speech engine via the speech API. The speech engine may be a CSR engine, a TTS engine, or an SLU engine. The interface between the application and the engine can be distributed. For example, you can have a client-server model in which the engine is running remotely on the server.

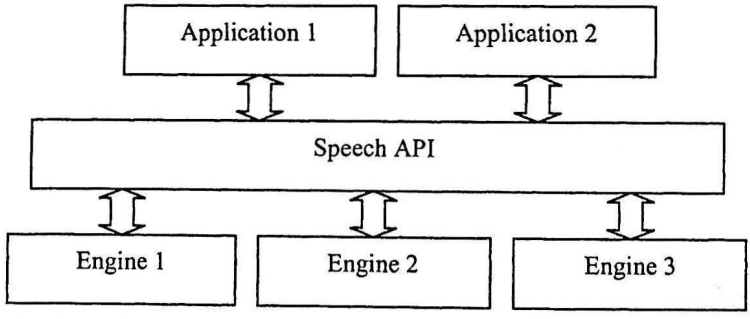


Figure 18.1 In a typical spoken language application architecture, multiple applications can interact with a shared speech engine via the speech API. The speech engine may be a speech recognizer, a TTS converter, or an SLU engine.

For a given API, there is typically an associated toolkit that provides a good development environment and the tools you need in order to build speech applications. You don't need to understand the underlying speech technologies to fully take advantage of state-of-the-art speech engines. Industry-standard based applications can draw upon support from many different speech engine vendors, thus significantly minimizing the cost of your applications development. For the widely used Microsoft Windows®, Microsoft's speech API

Amazon/VB Assets
Exhibit 1011
Page 946

(SAPI) brings both engine and application developers together.[1] Alternative standards are available, such as VoiceXML[2] and JSAPI[3].

18.2. TYPICAL APPLICATIONS

There are three broad classes of applications that require different UI design:

- *Office:* This includes the widely used desktop applications such as Microsoft Windows and Office.
- *Home*: TV and kitchen are the centers for home applications. Since home appliances and TV don't have a keyboard or mouse, the traditional GUI application can't be directly extended for this category.
- *Mobile*: Cell phone and car are the two most important mobile scenarios. Because of the physical size and the hands-busy and eyes-busy constraints, the traditional GUI application interaction model requires significant modification.

This section provides descriptions of typical spoken language applications in these three broad classes. Spoken language has the potential to provide a consistent and unified interaction model across these three classes, albeit for these different application scenarios you still need to apply different user interface design principles.

18.2.1. Computer Command and Control

One of the earliest prototypes for speech recognition is *command and control*, which is mainly used to navigate through operating system interfaces and applications running under them. For example, Microsoft Agent is a set of software services that supports the presentation of software agents as interactive personalities within the Microsoft Windows or the Internet Explorer interface. Its command-and-control speech interface is an extension and enhancement of the existing interactive modalities of the Windows interface. It has a character called Peedy, shown in Figure 18.2, which recognizes your speech and talks back to you using a Microsoft SAPI compliant command-and-control speech recognizer and text-to-speech synthesizer.

The speech recognizer used in these command-and-control systems is typically based on a context-free grammar (CFG) decoder. Either developers or users can define these grammars. Associated with each legal path in the grammar is a corresponding executable event that can map a user's command into appropriate control actions the user may want. They possess a built-in vocabulary for the menus and other components. The vocabulary can also be dynamically provided by the application. Command-and-control speech recognition allows the user to speak a word, phrase, or sentence from a list of phrases that the computer is expecting to hear. The number of different commands a user might speak at any time can

[1] http://www.microsoft.com/speech
[2] http://www.voicexml.org/
[3] http://java.sun.com/products/java-media/speech/

Amazon/VB Assets
Exhibit 1011
Page 947

be in the hundreds. Furthermore, the commands are not just limited to fixed ones but can also contain other open fields, such as "*Send mail to <Name>*" or "*Call <digits>*". With all of the possibilities, the user is able to speak thousands of different commands. As discussed in Chapter 17, a CFG-based recognizer is often very rigid, since it may reject the input utterance that contains a sentence slightly different from what the CFG defines, leading to an unfriendly user experience.



Figure 18.2 A talking character *Peedy*[4] as used in Microsoft Agent. Reprinted with permission from Microsoft Corporation.

Command-and-control recognition might be useful in some of the following situations:

- *Answering questions*. An application can easily be designed to accept voice responses to message boxes and wizard screens. Most speech recognition engines can easily identify *Yes*, *No*, and a few other short responses.
- *Accessing large lists*. In general, it's faster for a user to speak one of the names on a list, such as "*Start running calculator,*" than to scroll through the list to find it. It assumes that the user knows what is in the list. Laurila and Haavisto [23] summarized their usability study of inexperienced users on name dialing. Although the study is based on the telephone handset, it has a similar implication for computer desktop applications.
- *Activating macros*. Speech recognition lets a user speak a more natural word or phrase to activate a macro. For example, "*Spell check the second paragraph*" is easier for most users to remember than the CTRL+F5 key combination after selecting the second paragraph. But again, the user must know the command. This is where most simple speech applications fail. The competition is not CTRL+F5 itself, it is the memory of most users.
- *Facilitating dialog between the user and the computer*. As discussed in Chapter 17, speech recognition works well in situations where the computer essen-

[4] *Peedy* © 1993-1998 Microsoft Corporation.

Amazon/VB Assets
Exhibit 1011
Page 948

tially asks the user: "*What do you want to do*?" and branches according to the reply (somewhat like a wizard). For example, the user might reply, "*I want to book a flight from New York to Boston.*" After the computer analyzes the reply, it clarifies any ambiguous words (*Did you say New York*?). Finally, the computer asks for any information that the user did not supply, such as "*At what day and time do you want to leave*?"

- *Providing hands-free computing.* Speech recognition is an essential component of any application that requires hands-free operation; it also can provide an alternative to the keyboard for users who are unable to or prefer not to use one. Users with repetitive-stress injuries or those who cannot type may use speech recognition as the sole means of controlling the computer. As discussed in later sections, hands-free computing is important for accessibility and mobility.
- *Humanizing the computer.* Speech recognition can make the computer seem more like a person—that is, like someone whom the user talks to and who speaks back. This capability can make games more realistic and make educational or entertainment applications friendlier.

The specific use of command and control depends on the application. Here are some sample ideas and their uses:

- *Games and entertainment:* Software games are some of the early adopters of command-and-control speech recognition. One of the most compelling uses of speech recognition technology is in interactive verbal exchanges and conversation with the computer. With games such as flight simulators, for example, traditional computer-based characters can now evolve into characters the user can actually talk to. While speech recognition enhances the realism and fun in many computer games, it also provides a useful alternative to game control. Voice commands provide new freedom for the user.
- *Document editing*: Command and control is useful for document editing when you wish to keep your hands on the keyboard to type, or on the mouse to drag and select. This is especially true when you have to do a lot of editing that requires you to move to menus frequently. You can simultaneously speak commands for manipulating the data that you are working on. A word processor might provide commands like "*bold, italic*" and "*change font.*" A paint package might have "*select eraser*" or "*choose a wider brush.*" Of course, there are users who won't prefer speaking a command to using keyboard equivalents, as they have been using the latter for so long that the combinations have become for them a routine part of program control. But for many people, keyboard equivalents are a lot of hard-to-remember shortcuts. Voice commands provide these users with the means to execute a command directly.

Amazon/VB Assets
Exhibit 1011
Page 949

For most of the existing applications, before an application starts a command-and-control recognizer, it must first give the recognizer a *list* of commands to listen for. The list might include commands like "*minimize window*," "*make the font bold*," "*call extension <digit> <digit> <digit>*," or "*send mail to <name>*." If the user speaks the command as it is designed, he/she typically gets very good accuracy. However, if the user speaks the command differently, the system typically either does not recognize anything or erroneously recognizes something completely different. Applications can work around this problem by:

- Making sure the command names are intuitive to users. For many operations like minimizing a window, nine out of ten users will say *minimize* window without prompting.
- Showing the command on the screen. Sometimes an application displays a list of commands on the screen. Users naturally speak the same text they see.
- Using word spotting as discussed in Chapter 9. Many speech recognizers can be told to just listen for one keyword, like *mail*. This way the user can speak, "*Send mail,*" or "*Mail a letter,*" and the recognizer will get it. Of course, the user might say, "*I don't want to send any mail,*" and the computer will still end up sending mail.
- Employing spoken language understanding components as discussed in Chapter 17.
- Employing user studies to collect data on frequently spoken variations on commands so that the coverage is enhanced.

18.2.2. Telephony Applications

Speech is the only available modality for telephony applications besides the awkward-to-use DTMF interface. The earliest uses of speech technology in business were *interactive voice response* (IVR) systems. These systems include *infoline* services in the ad-supported local newspapers, offering everything from world news to school homework assignments at the touch of a few buttons. So what's the big deal with a speech telephony application? It offers greater breadth, ease of use, and interactivity. Navigating by voice rather than by keypad offers more options and quicker navigation. It also works better while you're driving.

To make a successful IVR application, you need to have speech input, output, and related dialog control. People have used IVR systems over the telephone to navigate the application based on the menu option to provide digit strings, such as the credit card numbers, to the application. Such system typically has a small to medium vocabulary. Today, you can use IVR to get stock quotes, people's telephone number, and other directory-related information. For example, you can call AT&T universal card services and the application asks you to speak your 16-digit card number. Most of these IVR systems use recorded messages instead of synthetic speech because the quality of TTS is still far from humanlike. Since speech output is a slow method to present information, it is important to be as brief as possible. Reducing the presentation of repetitive data can shorten the speech output significantly.

Amazon/VB Assets
Exhibit 1011
Page 950

Voice portals that let you talk your way to Web-based information from any phone are one class of compelling telephony applications. Linked to specially formatted Web sites and databases, the portals deliver what amounts to customized real-time news radio. You can tailor voice portals much as you do Web portals like *Yahoo!*[4], AOL[5], or MSN[6]. But surfing is restricted to the very limited subsets of information the portals choose to offer. These services typically avoid using synthesized speech. For options like news updates they rely on sound bites recorded by announcers. There are a number of free voice portals available, including TellMe, BeVocal, HeyAnita, Quack.com. Table 18.1 illustrates some of their features.

Table 18.1 Some free voice portal features. These portals are being developed and will roll out more features.

Category	Audiopoint[5]	Tell Me[6]
Traffic	Yes	Yes
Weather	U.S. and world cities	U.S.
News	Yes	Yes
Financial	Yes	Yes
Sports	Yes	Yes
Airline info	No	Yes
Restaurants	No	U.S.
Entertainment	Yes	Yes
Personalization	Yes	Yes

Digital wireless telephony applications could make full use of a client-server architecture because of limited computing resources of the client. The server performs most of the needed processing. The client can either send the speech waveform (as used in standard telephone) or the spectral parameters such as MFCC coefficients. Using a quantized MFCC (see Chapter 6) at 4.5 kbps, no loss of accuracy can be achieved [18]. The Aurora project tries to standardize the client server communication protocol based on the quantized MFCC coefficients[7] [6].

When people are engaged in a conversation, even if they have the graphical interface in front of them, they seldom use the vocabulary from the interface (unless prompted by TTS or the speaker). This has an important implication for the UI design. The use of a discourse segment pop cue such as *"What now?"* or *"Do you want to check messages?"* could reorient users (especially after a subdialog) and help them figure out what to say next. Wildfire[8] uses such pop cues extensively. The right feedback is essential, because speech recognition is not perfect. Designers should verify only those commands that might destroy data or trigger future events. People become frustrated very quickly if the error feedback is re-

[5] http://www.myaudiopoint.com or call 1-888-38-AUDIO.
[6] http://www.tellme.com or call 1-800-555-TELL.
[7] http://www.etsi.org/stq
[8] http://www.wildfire.com/

Amazon/VB Assets
Exhibit 1011
Page 951

petitive. It would be nice to design the error messages with different prompts according to recognition confidence measures and the dialog context.

Most current telephony systems do not have much intelligent dialog management beyond a simple state-based dialog model. Advanced dialog management as discussed in Chapter 17 is critical for error repairs, anaphora resolution, and context management. There are a number of institutions pursuing advanced telephony spoken language interfaces. You can find some of these excellent programs at Carnegie Mellon University[9] and Massachusetts Institute of Technology.[10]

18.2.3. Dictation

Dictation is attractive to many people, since speaking is generally faster than typing, especially in East Asian languages such as Chinese. There are a number of general-purpose dictation software products such as IBM's ViaVoice®, Dragon's NaturallySpeaking®, Lernout&Hauspie's Voice Xpress®, as well as the dictation capability built into Microsoft Office 2001. There are also vertical markets for dictation applications. Radiologists and lawyers are two examples of such niche markets. They are generally pressed for time. Radiologists are often hands- and eyes-busy when at work. Lawyers' work is often language intensive and they need to write long documents and reports. Both radiologists and lawyers have been using dictation devices, and they have a strong need to have customized language models for their specialized vocabulary. In fact, because of the constraints of the language they use, the perplexity is often much smaller than in the general business articles, leading to improved performance for both the medical and legal segments.

In dictation applications, it is important to convert speech into the text you would like to see in its written form. This implies that punctuation should be added automatically, dates and times should be converted into the form that is conventionally used, and appropriate capitalization and homonym disambiguation should all be seamless to users. Dictation is typically associated with word processing applications, which provide for a nice separation of input modes based upon the division of the primary task into subactivities:

- Text entry
- Command execution, such as *cross-out* to delete the previous word, or *capitalize-that*
- Direct manipulation activities such as cursor positioning and text selection

It is likely that separate single input modes for each activity can increase efficiency. In a study conducted by Morrison et al. [28], subjects switched modality from speech input to typed input while issuing text-editing commands and found the switch of modality *disruptive*. Karat et al. [19] compared the state-of-the-art dictation product with keyboard input.

[9] http://www.speech.cs.cmu.edu/speech/
[10] http://www.sls.lcs.mit.edu/sls/

Amazon/VB Assets
Exhibit 1011
Page 952

They reported that several of the initial-use subjects commented that keyboard entry seemed *much more natural*. This reflects the degree to which some people have become used to using keyboards and the fact that speech recognition still makes too many mistakes. Experienced users can enter transcription text at an average rate of 107 uncorrected words per minute. However, correction took them over three times as long as entry time on average, leading to an average input rate nearly twice as slow as keyboard (including correction). Thus, unless the accuracy of speech recognition and its error correction can be significantly improved for the mass market, text entry is best performed by keyboard.

The mouse is generally accepted as being well suited to direct manipulation activities. Further, based on human-factors studies, full typed input of commands, single keyboard presses, and accelerator keys is not as efficient as speech-activated commands [26, 33]. The case for the utility of speech input in word-processing applications relies upon its superiority over the mouse with respect to the activation of commands. Word-processing and text-entry applications are also naturally *hands-busy, eyes-busy* applications. It is inconvenient and time consuming for the user to have to interrupt the typing of text or to move his/her eyes from the work in order to execute word-processing commands. This is not necessary with speech activation of commands.

To improve dictation throughput, you should make it as easy as possible for users to correct mistakes. As illustrated in Figure 18.3, you can provide easy access to the *Correction Window* so the user can correct mistakes that the recognizer made. Thus the quality of *n*-best or word-lattice decoding discussed in Chapter 13 is critical. We use examples drawn from Microsoft Dictation[11] to illustrate these concepts.

You should also allow the user to train the system so it can adapt to different accent and environment conditions. A typical training wizard, illustrated in Figure 18.4, asks users to read a number of sentences. It is important to use a confidence measure, as discussed in Chapter 9, to filter out unwanted training data so that you can eliminate possible divergences of training the acoustic model. In Figure 18.4, the user is prompted with the sentence shown in the window. As the user reads the sentence, the black progress bar covers the corresponding word. If the user misreads the word, the progress bar typically stays where it is.

Another practical problem for most dictation products is that they have about 60,000 to 100,000 words in the vocabulary, which contains all the inflected forms, so *work* and *works* are considered as two different words. As the vocabulary is limited (for detailed discussion on vocabulary selection, see Chapter 11), you should provide a capability to dynamically add new words to the system. This capability is illustrated in Figure 18.5. You can listen to the new word you added to see if the computer guessed the pronunciation of the word correctly or not. If you want to pronounce the word differently, you can either type in the words that sound the way you want it to or type in the phonetic sequence.

[11] Microsoft Dictation was developed by the authors at Microsoft Research, and was included for free in the SDK4.0.

Amazon/VB Assets
Exhibit 1011
Page 953

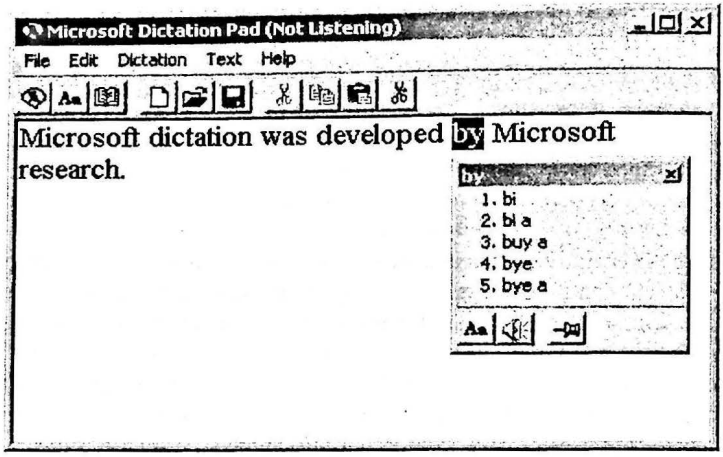


Figure 18.3 *Correction window* in Microsoft Dictation. By clicking the dictated word, the alternative words are listed in the correction window. If the dictated word is wrong, you can click the word in the correction window to easily replace the misrecognized word. Reprinted with permission from Microsoft Corporation.

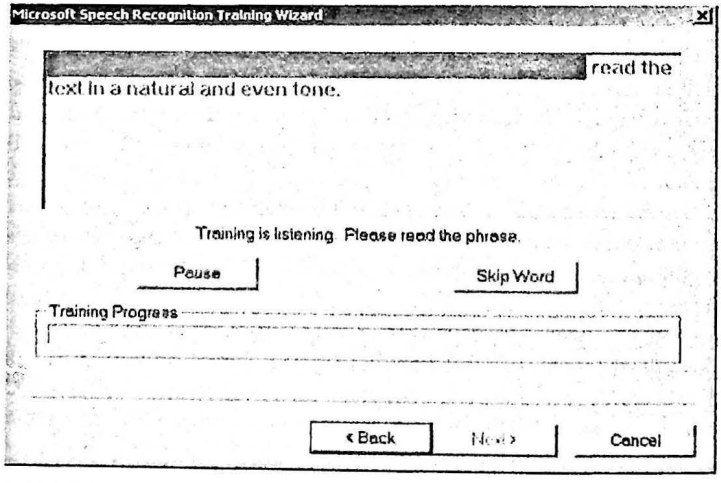


Figure 18.4 Training wizard in Microsoft Dictation. Reprinted with permission from Microsoft Corporation.

Amazon/VB Assets
Exhibit 1011
Page 954

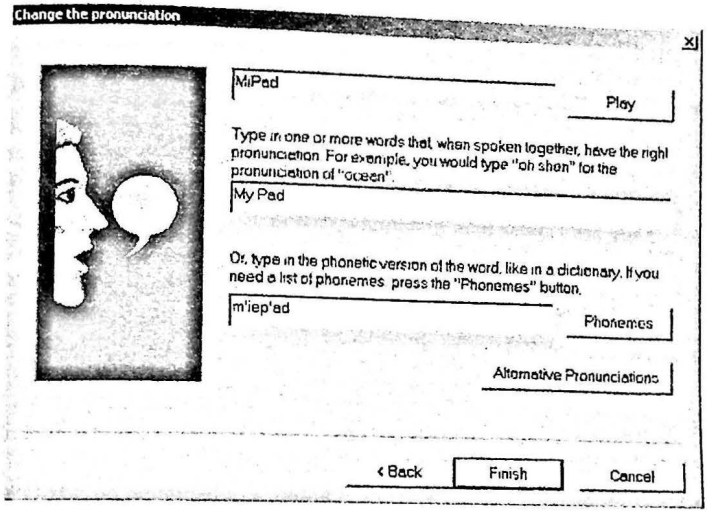


Figure 18.5 New word addition in Microsoft Dictation. Reprinted with permission from Microsoft Corporation.

18.2.4. Accessibility

Approximately 10–15 percent of the population has an impairment that affects their ability to use computers efficiently. Accessibility is about designing applications that everyone can use. The GUI actually makes accessibility a more critical issue. People with disabilities face more usability issues than those without. There are a number of Web sites dedicated to accessibility.[12]

People who cannot effectively use a keyboard or a mouse can use speech commands to control the computer and use the dictation software to input text. For example, Dragon's dictation software lets you control everything by voice, making it extremely useful for people with repetitive stress injury (RSI). A screen reader is a program that uses TTS to help people who have reading problems or are vision impaired. Windows® 2000 includes one such screen reader, called Narrator, which reads what is displayed on the screen: the contents of the active window, menu options, and text that the user types. Users can customize the way these screen elements are read. They can also configure Narrator to make the mouse pointer follow the active item on the screen and can adjust the speed, volume, and pitch of the narrator voice.

[12] http://www.el.net/CAT/index.html (The Center for Accessible Technology).
http://www.lighthouse.org/print_leg.htm (Lighthouse International).
http://www.w3c.org/wai (World Wide Web Consortium's Accessibility Page).
http://www.microsoft.com/enable (Microsoft Accessibility Home Page).

Amazon/VB Assets
Exhibit 1011
Page 955

18.2.5. Handheld Devices

For handheld devices such as *smart phones* or *Personal Digital Assistants* (PDA), speech, pen, and display can be used effectively to improve usability. The small form-factor implies that most of the functions have to be hidden. Speech is particularly suitable to access information that can't be seen on the display. Current input methods are generally pen- or soft keyboard-based. They are slower than dictation-based text input. For form filling, speech is also particularly suitable to combine multiple parameters in a single phrase. The increasing popularity of cell phones is likely to be the major thrust for using handheld devices to access information.

Potential usability problems of handheld devices are the lack of privacy and the problem of environmental noise. It is hard to imagine that we can have 100 people in a conference room talking to their handheld devices simultaneously. Since these devices are also often used in a noisy environment, you need to have a very robust speech recognition system to deal with signals around 15 dB. In Section 18.5 we illustrate the process of designing and developing an effective speech interface and application for a handheld device.

18.2.6. Automobile Applications

People already use cell phones extensively when driving. Safety is an important issue whenever a driver takes his/her eyes off the road or his/her hands off the steering wheel. In many countries it is illegal to use a cellular phone while driving. Speech-based interaction is particularly suitable for this eyes-busy and hands-busy environment. However, it is a challenge to use speech recognition in the car due to the relative high noise environment. Environment robustness algorithms, discussed in Chapter 10, are necessary to make automobile applications compelling.



Figure 18.6 Clarion in-dash AutoPC.

One such system is Clarion's AutoPC, shown in Figure 18.6. It enables drivers to access information by speech while driving. You can tell your car what to do by speech. AutoPC has a vocabulary of more than 1200 words. It understands commands to change radio stations, provide turn-by-turn directions based on its Global Positioning System (GPS) navigation unit, or read e-mail subject lines using TTS. Running on Microsoft's Windows CE operating system, AutoPC's control module fits in a single dashboard slot, with the computer and six-disc CD changer stored in the trunk.

Amazon/VB Assets
Exhibit 1011
Page 956

18.2.7. Speaker Recognition

The HMM-based speech recognition algorithms discussed in earlier chapters can be modified for speaker recognition, where a speaker model is needed for each speaker to be recognized. The term speaker recognition is an umbrella that includes speaker identification, speaker detection, and speaker verification. Speaker identification consists in deciding who the person is, what group the person is a member of, or that the person is unknown. Speaker detection consists in detecting if there is a speaker change. This is often needed in speech recognition, as a better speaker model can be used for improved accuracy. Speaker verification consists in verifying a person's claimed identify from his/her voice, which is also called speaker authentication, talker verification, or voice authentication.

In text-dependent approaches, the system knows the phrase, which can be either fixed or prompted to the user. The speaker speaks the required phrase to be identified or verified. The speaker typically has to enroll in the system by presenting a small number of speech samples to build the corresponding speaker model. Since the amount of training data is small, the parameter tying techniques discussed in Chapter 9 are very important.

Speaker recognition is a performance biometric, i.e., you perform a task to be recognized. It can be made fairly robust against noise and channel variations, ordinary human changes, and mimicry of humans and tape recorders. Campbell [3] and Furui [9] provided two excellent tutorial papers on recent advances in speaker recognition.

In general, accuracy is much higher for speaker recognition than for speech recognition. The best speaker recognition systems typically use the same algorithms, such as the ways to deal with environment noises and hidden Markov modeling, found in speech recognition systems. Applicable speaker recognition services include voice dialing, banking over a telephone network, telephone shopping, database access services, information and reservation services, forensic application, and voice login. There are a number of commercial speaker verification systems, including Apple's voice login for its iMac® and Sprint's Voice FONCARD®.

18.3. SPEECH INTERFACE DESIGN

No single unified interaction theory has emerged so far to guide user interface design. Study of the interaction has been largely associated with developing practical applications, having its roots in human factors and user studies. Practitioners have postulated general principles that we discuss in this section.

18.3.1. General Principles

There are a number of books on graphical user interface (GUI) design [10, 11, 17, 29, 41]. Usability research also generated a large body of data that is useful to UI designers. The Human Factors and Ergonomic Society (HFES) is developing the HFES-200 standard—

Amazon/VB Assets
Exhibit 1011
Page 957

Ergonomics of Software User Interface, which has many insightful implications for interface design. The most important criteria, in order of importance, are:

- Effectiveness or usefulness
- Efficiency or usability
- User satisfaction or desirability

These three general points mean that, first and foremost, a user interface must work, so the user can get the job done. After that, it is important that the user interface accomplishes its task as productively as possible. Last, but not least, the user interface can be further improved by focusing on user satisfaction as measured by means of questionnaires and user feedback.

The major difference between GUI and spoken language interface is that speech recognition or understanding can never be perfect. Imagine you are designing for a GUI and your input method is a keyboard and a mouse where the error rate is about 5% for the key or mouse presses. These 5% errors make it much more difficult to have an effective user interface. In GUI applications, when you mistype or misclick, you regard these errors as your own mistakes. In spoken language applications, all the misrecognition errors are regarded as system errors. This psychological effort is a very important factor in the UI design. Thus it is not stretching to say that most of the design work for spoken language applications should be on how to effectively manage what to do when something goes wrong. This should be the most important design principle to remember.

We also want to stress the importance of iteratively testing the UI design. In practice, users, their tasks, and the work environment are often too varied to be able to have a good design without going through a detailed and thorough test program, especially for new areas like spoken language applications.

18.3.1.1. Human Limitations

Applications make constant demands on human capabilities. When we design the user interface, we must understand that humans have cognitive, auditory, visual, tactile, and motor limits. As a general principle, we should make the user interface easy to use, and not overload any of these limits.

For example, if the font is too small, the speech volume is too low, or the application requires people to constantly switch between a mouse and a keyboard, we have a poor interface, leading to lowered productivity. People have trouble remembering long complex messages spoken to them in a continuous fashion. If these messages are delivered with synthesized speech, the problem is worsened, because of the greater attention required to understand the synthesized speech. Effective user interface design requires a balance of easy access to information and an awareness of human limitations in processing this information. In general, people tend to handle well 7 (+ or – 2), pieces of information at a time.

Amazon/VB Assets
Exhibit 1011
Page 958

18.3.1.2. User Accommodation

The interface and application should accommodate users as much as possible. This means that the design needs to match the way users want to do their work, not the other way around. People always have, or quickly create, a mental model of how the interface operates. You need to design and communicate a conceptual model that works well with the users' mental model. Their mental model should match the actual interface and the task, which is very important for the designer to consider consciously.

To accommodate users' needs, you must ask whether the interface you are designing helps people get their work done efficiently. Will it be easy to learn and use? Will it have an attractive and appropriate design? Will the interface fit individual tasks with a suitable modality? Will the interface allow the user to perceive that they are in control? Will the interface be flexible enough to allow the user to adjust the design for custom use? Does the interface have an appropriate tempo? Notice that speech is temporal. Does the interface contain sufficient user support if the user needs or requests it? Does the interface respond to the use in a timely fashion? Does the interface provide appropriate feedback to the user about the results of the actions and application status?

18.3.1.3. Modes of Interaction

The interface is unimodal if only one mode is used as both input and output modality. By contrast, multimodal systems use additional modalities in exchanging information with their users. When we interact with each other, we often rely on multimodal interaction, which is normal for us.

When speech is combined with other modalities of information presentation and exchange, it is generally more effective than a unimodal interface, especially for dealing with errors from speech recognition and understanding.

As an input modality, speech is generally not as precise as mouse or pen to perform position-related operations. Speech interaction can also be adversely affected by the ambient noise. When privacy is of concern, speech is also disadvantageous, since others can overhear the conversation. Despite these disadvantages, speech communication is not only natural but also provides a powerful complementary modality to enhance the existing keyboard- and mouse-based graphical user interface. As an input-output modality, speech has a number of unique characteristics in comparison with the traditional modalities:

- Speech does not require any screen real estate in the interface. For ubiquitous computing, this is particularly suitable for devices that are as small as our watches.
- Speech can be used at a distance, which makes it ideal for hands-busy and eyes-busy situations such as driving.

Amazon/VB Assets
Exhibit 1011
Page 959

- Speech is expressive and very suitable to bring information that is hidden to the forefront. This is particularly suitable in the GUI environment or with handheld devices as the monitor space is always limited.
- Speech is omnidirectional and can communicate information to multiple users easily. However, this also has implications relating to privacy.
- Speech is temporal and sequential. Once uttered, auditory information is no longer available. This can place extra memory burden on the user and severely limit the ability to scan, review, and cross-reference information.
- For social interfaces, speech plays a very important role. For example, if we need to pose a question, make a remark, and delegate a task, there is no more natural way than speech itself.

Because of these unique features, we need to leverage the strengths and overcome the technology limitations that are associated with the speech modality. The conventional GUI relies on the visual display of objects of interest, the selection by pointing, and continuous feedback. Direct manipulation is generally inadequate for supporting fundamental transactions in applications such as word processing and database queries, as there are only limited means to identify objects, not to mention that ambiguity in the meanings of icons and limitations in screen display space further complicate the usability of the interface.

Generally, multimodality can dramatically enhance the usability of speech, because GUI and speech have complementary strengths and weaknesses. The strengths of one can be used to offset the weaknesses of others. For example, pen and speech can be complementary and they can be used very effectively for handheld devices. You can use *Tap and Talk* as discussed in Section 18.4 to activate microphone and select appropriate context for speech recognition. The respective strengths and weaknesses are summarized in Table 18.2. The advantage of pen is typically the weakness of speech and vice versa. This implies that user interface performance and acceptance could increase by combining both. Thus, visible, limited, and simple actions can be enhanced by nonvisible, unlimited, and complex actions. The multimodal interface is particularly suitable to people for completing contrastive functions or when task attributes are perceived as separable. For example, speech and pen can be used in different ways to designate a shift in context or functionality. Although most people may not prefer to use speech and pen simultaneously, they like to use speech to enter data and pen for corrections. Other contrastive tasks may include data versus commands and digits versus text. When inputs are perceived as integral, unimodal interfaces are efficient and work best.

Table 18.2 Complementary strengths of pen and speech for multimodal user interface.

Pen	Speech
Direct manipulation, requires hands and eyes	Hands/eyes-free manipulation
Good at performing simple actions	Good at performing complex actions
Visual feedback	Visual and location independent references
No reference ambiguity on the display	Multiple ways to refer to objects

Amazon/VB Assets
Exhibit 1011
Page 960

18.3.1.4. Technology Considerations

Speech recognition applications can be divided into many classes, as illustrated in Figure 18.7. The easiest category is the system that can only deal with isolated speech with limited vocabulary for a command-and-control application, where typically a context-free grammar is used to constrain the search space, resulting in a limited number of ways users can issue a command. The hardest is the one that can support channel- and speaker-independent continuous speech recognition with both dictation and robust application-specific understanding capability. This is also the one that can really add value to a wide range of speech applications. In Figure 18.7, a *command-and-control* application implies application-specific restrictive understanding capability—typically with less than 1000-word vocabulary and with a lower perplexity context-free grammar. A *dictation* application implies general-purpose dictation capability—typically with more than 60,000 words in the lexicon and a stochastic *n*-gram language model. A *dictation and SLU* application implies both general-purpose dictation and less restrictive application-specific spoken language understanding capability—typically with 60,000 words in the lexicon with both *n*-gram and semantic grammar language models for both recognition and understanding.

Although a number of parameters can be used to characterize the capability of speech recognition systems, user-centered design often dictates that you need to use the most challenging technology solutions. The speaking style (from isolated speech to continuous speech) can dramatically change the performance of a speech recognition system. Natural, spontaneous, continuous speech remains the most difficult to recognize; but it is more user friendly. Small vocabulary (less than 50 words) is generally more accurate than large vocabulary (greater than 20,000 words); but the small vocabulary limits the options the user has on the task. The lower-perplexity language model generally has a great recognition accuracy; but it imposes a considerable constraint on how the user can issue a command. Most systems may work in the high-SNR (>30 dB) environment; but it is desirable that the system should still work when the SNR is below 10 dB.

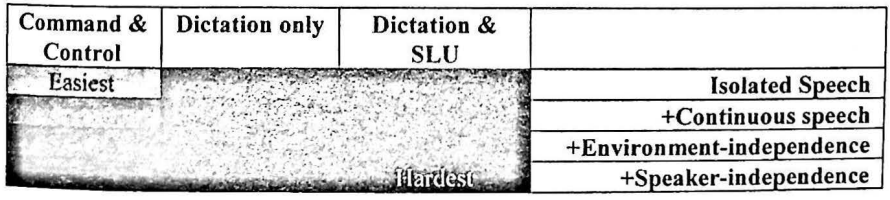


Figure 18.7 Illustrative degrees of difficulty for different classes of speech recognition (gray level from easy to hard).

Amazon/VB Assets
Exhibit 1011
Page 961

Associated with the input modality is the speech output modality for many speech applications. Speech as an output medium taxes user memory and may even cause loss of context and distraction from the task if it is too verbose. Most of the text-to-speech systems have a problem generating a prosody pattern that is dynamic and accurate enough to convey the underlying semantic message. Recorded speech still has the best quality; but it is generally inflexible in comparison to TTS. There are many ways to provide additional information to the TTS so that the final synthesized quality can be improved dramatically. Examples include the use of recorded speech for constant messages, transplanted prosody, and concept-to-speech synthesis as discussed in Chapters 16 and 17.

Improved quality and ease of use often need expensive authoring. This is true for speech recognition, synthesis, and understanding. For example, to support channel independence, you need not only additional channel normalization algorithms to deal with both additive and convolutional noises but also realistic training data to estimate model parameters. To support robust understanding, you need to author semantic grammars that have broad coverage. This often requires tedious authoring and labor-intensive data collection and annotation to augment the grammar. Ultimately, you need a modular and reusable system that can warrant the development costs. To make effective use of dialog and error-repair strategies, you need to adopt tools for preventing dialog design problems during the early stages, ones that guide the choice of words in dialog and feedback and ensure usability and correctness in the context, etc. In addition, as discussed in Chapter 17, whether you have a system-initiative dialog, user-initiative dialog, or mixed-initiative dialog will have significant ramifications on the authoring cost.

A spoken language application requires certain hardware and software on the user's computer in order to run.[13] Most new sound cards work well for speech recognition and text-to-speech, including Sound Blaster™. Some old sound cards are only *half duplex* (as opposed to *full duplex*). If a sound card is half duplex, it cannot record and play audio at the same time. Thus it cannot be used for barge-in and echo cancellation. The user can choose among different kinds of microphones: a close-talk headset microphone that is held close to the mouth, a medium-distance microphone, or a microphone array device that rests on the computer 30 to 60 centimeters away from the speaker. A headset microphone is often needed for noisy environments, although microphone array or blind separation techniques have the potential to close the gap in the future (see Chapter 10). Most speech products also need to calibrate the microphone gain by speaking one or two utterances, as illustrated in Figure 18.8. It adjusts the gain of the amplifier to make sure there is an appropriate gain without clipping the signal. This can be done in the background without distracting the user.

[13] Speech recognition engines currently on the market typically require a Pentium 200 or faster processor. Speech recognition for command and control consumes 1 to 4 MB RAM, and dictation requires an additional 16-32 MB. Text-to-speech engines use about 1-3 MB, but high-quality TTS can require more than 64 MB.

Amazon/VB Assets
Exhibit 1011
Page 962

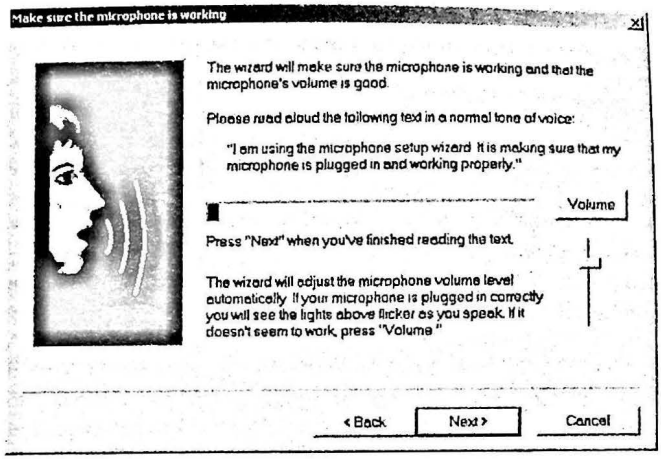


Figure 18.8 Microphone setup wizard used in Microsoft Speech SDK 4.0. Reprinted with permission from Microsoft Corporation.

18.3.2. Handling Errors

Speech recognition and understanding can never be perfect. Current spoken language processing work is very much a matter of minimizing errors. Together with the underlying speech-engine technologies, the role of the interface is to help reduce the errors or minimize their severity and consequences. As discussed in earlier chapters, there are errors associated with word deletion, word insertion, and word substitution. The recognizer may be unable to reject noises and classify them as regular words. The speaker may also have new words that are not in the lexicon, or may have less clear articulation due to accent, stress, sickness, and excitement.

To repair errors in the system, we need to take each individual application into consideration, since the strategy is often application-dependent. To have a compelling application, we must design the application with both the limitation of the engine and the interaction of the engine and UI in mind. For example, multimodal applications are generally more compelling and useful than speech-only solutions as far as error handling is concerned. A multimodal application can overcome many limitations of today's SLU technologies, but it may be inappropriate for hands-busy or eyes-busy scenarios.

Amazon/VB Assets
Exhibit 1011
Page 963

18.3.2.1. Error Detection and Correction

We need to detect errors before we can repair them. This is a particularly difficult problem that often requires high-level context and discourse information to decipher whether there are errors. We can certainly use confidence measures, as discussed in Chapter 9, to detect potential errors, although confidence measures alone are not particularly reliable.

There are many ways to correct the errors. The choice of which one to use depends on the types of the applications you are developing. We want to emphasize that correction based on another modality is often desirable, since it is unusual that two independent modalities make the same mistake. For example, speech recognition errors are often different from handwriting recognition errors. In addition to the complementary nature of these different errors, we can also leverage other modalities for improved error-correction efficacy. For example, a pen is particularly suitable for pointing, and we can use that to locate misrecognized words. A camera can be used to track whether the speaker is speaking in order to eliminate background noise generated from other sound sources.

Let's consider a concrete example. If you have a handheld device, you may be able to see the errors and respeak the desired commands or use a different modality such as pen to help task completion. When you want to create an email using speech, and a dictation error occurs, you can click the misrecognized word or phrases to display a list of *n*-best alternatives. You can select the correct word or phrase from the *n*-best list with the pen. Alternatively, you can respeak the highlighted word or phrase. Since the correct answer should not be the previous misrecognized word or phrase, you can exclude them in the second trial. If it is an isolated word in the correction stage, you can also use the correct silence context triphone models for the word boundary as the correct acoustic model. You can also use the correct *n*-gram language model from the left context with the assumption that the left words are all correct. With these correct acoustic and language models, it is expected that the second trial would have a better chance to get the right answer.

If you have telephony-based applications, you can count on keywords such as *no, I meant* to identify possible recognition or understanding errors. When errors occur, you should provide specific error-handling messages so that the user knows what response the system wants. An adequate error message should tell the user what is wrong, how to correct it, and where to get help if needed. Let's consider an example of the following directory assistant application:

Computer: Please speak the name

User: Mike

Computer: Please speak the name

User: Mike

Amazon/VB Assets
Exhibit 1011
Page 964

Clearly, the user does not know what responses the computer wants. An improved dialog can be:

Computer: Please speak the first and last name

User: Mike Miller

Computer: Which division is Mike Miller working?

User: Research group

You can use alternative guesses intelligently for telephony applications, too. For example, if the user says "*Mike Miller*" and the computer is unsure of what was said, it can prompt the user to repeat with a constrained grammar. In the example above, if the computer is unsure whether the user said Mike or Mark, it may want to respond with "*Did you say Mike Miller or Mark Miller*?" before it proceeds to clarify which division Miller is working in.

When an error is detected, the system should be adapted to learn from the error. The best error-repair strategy is to use an adaptive acoustic and language model so that the system can learn from the errors and not repeat the same mistake again. This is why adaptive acoustic and language modeling, as discussed in Chapters 9 and 11, is critical for real acceptance of speech applications. When an error occurs and the computer understands the error via user correction, you can use the corrected words or phrases to adapt related components in the system.

18.3.2.2. Feedback and Confirmation

Users need feedback from the system so that they understand the status of the interaction. When a command is issued, the user expects the system to acknowledge that the command is heard. It is also important to tell the users if the system is busy so that they know when to wait. Otherwise, the users may interpret the lack of response or feedback as errors of recognition or understanding.

If the user has requested that an action be taken, the best response is to carry out the action as requested, but if the action cannot be carried out, the user must be notified. Sometimes, the user may need this feedback to know whether it is her turn to speak again. If the interface is multimodal, you can use visual feedback to indicate the status of the interaction. For example, in Microsoft's MiPad research prototype, when the microphone is activated with a pen pointing to a specific field, the microphone levels are indicated in the feedback rectangle in strip-chart fashion as shown in Figure 18.9 for the corresponding field. Initially this wipes from left to right, but when the right edge is reached, the contents scroll to the left as each new sample is added. The height of each sample indicates the audio level; there is some indication given for the redline level for a possible saturation. It also shows the total number of frames as well as the number of remaining frames to be processed. Then, as the

Amazon/VB Assets
Exhibit 1011
Page 965



Figure 18.9 MiPad's dynamic volume meter. Reprinted with permission from Microsoft Corporation.

recognizer processes, the black bar wipes from left to right to indicate its progress. When the recognizer completes (or is canceled by the user), the dynamic volume meter is removed. The volume meter feedback described addresses a number of issues that are critical to most users: if the computer is listening or not; if the volume is too high or not, and when the computer finishes processing the speech data.

If the interface has no visual component, you can use auditory cues or speech feedback. If there is more than a 3-second delay after the user issues a command and the system responds, an auditory icon during the delay, such as music, may be used. A delay of less than 2-3 seconds is acceptable without using any feedback.

Since there are errors in spoken language systems, you may have to use confirmation questions to assure that the computer heard the correct message. You can associate risks with different commands, so you can have a different level of strategy for confirmation. For example, *format disk* should have a higher threshold or may not be allowed by voice at all. You can have an explicit confirmation before executing the command. You can also have implicit confirmation based on the level of the risks in the corresponding confidence scores from the recognizer. For example, if the user's response has more than one possible meaning (X and Y) or the computer is not confident about the recognized result (X or Y), the computer could ask, "*Do you want to do X or Y?*" You want to be specific about what the system needs. A prompt like "*Do you mean X or Y?*" is always better than *Please repeat.* You have to balance the cost of making an error with the extra time and annoyance in requiring the user to confirm a number of statements.

In a telephony-based application, you need to tell the users specifically that they are talking to a computer, not a real person. When AT&T used a long greeting for its telephone customer service, their prompts were as follows:

"AT&T Automated Customer Service. This service listens to your speech and sends your call to the appropriate operator. How may I help you?"

The longer version resulted in shorter utterances from the people calling in. The short automated version did not help shorten the utterances, because people did not seem to catch the automated connotation.

You can also design the prompt in such a way that you can constrain how people respond or use their words. Let's consider an example of the following directory assistant application:

Computer: Welcome to Directory Assistant. We look forward to serving you. How can I help you?

User: I am interested in knowing if you can tell me the office number of Mr. Derek Smith?

Amazon/VB Assets
Exhibit 1011
Page 966

To have the user follow the lead of the computer, an improved dialog can be:

> **Computer: Welcome to Directory Assistant. Please say the full name of the person.**
>
> **User: Derek Smith**

When the system expects the user to respond with both the first name and last name, you should still design the grammar so that the system can recognize a partial name with lower probabilities. This makes the system far more robust—a good example of combining engine technology and interface seamlessly for improved user satisfaction.

18.3.3. Other Considerations

When speech is used as a modality, you should remember the following general principles:

18.3.3.1. Don't Use Speech as an Add-On Feature

It's poor design to just bolt speech recognition onto an application that is designed for a mouse and keyboard. Such applications get little benefit from speech recognition. Speech recognition is not a replacement for the keyboard and mouse, but in some circumstances it is a better input device than both. Speech recognition makes a terrible pointing device, just as the mouse makes a terrible text entry device, or the keyboard is bad for drawing. When speech recognition systems were first bolted onto the PC, it was thought that speaking menu names would be really useful. As it turns out, very few users use speech recognition to access a window menu.

18.3.3.2. Give Users Options

Every feature in an application should be accessible from all input devices: keyboard, mouse, and speech recognition. Users naturally use whichever input mechanism provides them the quickest or easiest access to the feature. The ideal input device for a given feature may vary from user to user.

18.3.3.3. Respect Technology Limitations

There are a number of examples of poor use of speech recognition. Having a user spell out words is a bad idea for most recognizers, because they are too inaccurate unless you constrain the recognizer for spelling. An engine generally has a hard time detecting multiple speakers talking over each other in the same digital-audio stream. This means that a dictation system used to transcribe a meeting will not perform accurately during times when two or more people are talking at once. An engine cannot hear a new word and guess its spelling unless the word is specified in the vocabulary. Speakers with accents or those speaking in

Amazon/VB Assets
Exhibit 1011
Page 967

nonstandard dialects can expect more errors until they train the engine to recognize their speech. Even then, the engine accuracy will not be as high as it would be for someone with the expected accent or dialect. An engine can be designed to recognize different accents or dialects, but this requires almost as much effort as porting the engine to a new language.

18.3.3.4. Manage User Expectations

When you design a speech recognition application, it is important to communicate to the user that your application is speech aware and to provide him or her with the commands it understands. It is also important to provide command sets that are consistent and complete. When users hear that they can speak to their computers, they instantly think of *Star Trek* and *2001: A Space Odyssey*, expecting that the computer will correctly transcribe every word that they speak, understand it, and then act upon it in an intelligent manner. You should also convey as clearly as possible exactly what an application can and cannot do and emphasize that the user should speak clearly, using words the application understands.

18.3.4. Dialog Flow

Dialog flow is as important to speech interface as screen design is to GUI. A robust dialog system is a prerequisite to the success of the speech interface. The dialog flow design and the prompting strategy can dramatically improve user experience and reduce the task complexity of the spoken language system.

18.3.4.1. Spoken Menus

Many speech-only systems rely on tree-structure spoken menus as the main navigation vehicle when no sophisticated dialog system is used. The design of these menus is, therefore, critical to the usability of the system. If there are only two options, you can have "*Say 1 for yes, or 2 for no.*" However, if a menu has more than two options, you should describe the result before you specify the menu option. For example, you want to have the system say "*For email say 1. For address book say 2...*" instead of "*Say 1 for email. Say 2 for address book....*" This is because the user may forget which choice he wants before hearing all the menu options.

You should not use more than five options at each level. If there are more than five, you should consider submenus. It is important to place the most frequently used options at the top of the menu hierarchy.

You should also design the menu so that commands sound different. As a rule of thumb, the more different phonemes you have between two commands, the more different they sound to the computer. The two commands, *go* and *no,* only differ by one phoneme, so when the user says, "*Go*" the computer is likely to recognize "*No.*" However, if the commands were "*Go there*" and "*No way*" instead, recognition would be much easier.

Amazon/VB Assets
Exhibit 1011
Page 968

18.3.4.2. Prompting Strategy

When you design the prompt, use as few words as possible. You should also avoid using a personal pronoun when asking the user to respond to a question. This gives the user less to remember, as speech output is slow and one-dimensional. Typical users are able to remember three to four menu or prompt options at a time. You should also allow users to interrupt the computer with *barge-in* techniques, as discussed in Chapter 10.

It is often a good idea to use small steps to query the user at each step if application has a speech-only interface progressively. You can start with short high-level prompts such as "*How can I help you?*" If the user does not respond appropriately, the system can provide a more detailed prompt such as "*You can check your email, calendar, address book, and your home page.*" If the user still does not respond correctly, you can give a more detailed response to tell the user how to speak the appropriate commands that the computer can understand.

If the application contains users' personal information, you may want to treat novice users and experienced users differently. Since the novice users are unfamiliar with the system and dialog flow, you can use more detailed instructions and then reduce the number of words used the next time when the user is going through the same scenario.

18.3.4.3. Prosody

For TTS, prosodic features, including pitch, volume, speed, and pause, are very important for the overall user experience. You can vary pitch and volume to introduce new topics and emphasize important sections. Increased pitch dynamic range also makes the system sound lively.

Users tend to mimic the speed of the system. If the system is reacting and speaking slowly, the user may tend to slow down. If the computer is speaking quickly, the user may speed up. An appropriate speech rate for natural English speech is about 150 to 170 words per minute.

Pauses are important in conversation, but they can be ambiguous, too. We generally use pauses to suggest that it is the user's turn to act. On the other hand, if the system has requested a response from the user and the user has not responded, you should use the time-out period (typically 10 seconds) to take appropriate actions, which may be a repeat of a more detailed prompt or may be a connection to an operator. People are generally uncomfortable with long pauses in conversation. If there are long pauses in the dialog flow, people tend to talk more and use less meaningful words, resulting in more potential errors.

18.4. INTERNATIONALIZATION

To meet the demands of international markets you need to support a number of languages. For example, Microsoft Windows 2000 supports more than 40 languages. Software interna-

Amazon/VB Assets
Exhibit 1011
Page 969

tionalization typically involves both globalization and localization phases. Globalization is the process of defining and developing a product such that its core features and code design do not make assumptions about a single language. To adapt the application for a specific market, you need to modify the interface, resize the dialog boxes, and translate text to a specific market. This process is called localization. The globalization phase of software internationalization typically focuses on the design and development of the product. The product designers plan for locale-neutral features to support multicultural conventions such as address formats and monetary units. Development of the source code is typically based on a single code base to have the capability to turn locale-specific features on and off without modifying the source code, and the ability to correctly process different character encoding methods. Products developed with the single worldwide binary model typically include complete global functionality, such as support for Input Method Editors used in Asian languages and bidirectional support for Arabic and Hebrew languages.

In addition to the most noticeable change in the translated UI, spoken language engines such as speech recognition or text-to-speech synthesis require significant undertakings. You need to have not only the lexicon defined for the specific language but also a large amount of speech and text data to build your acoustic and language model. There are many exceptional rules for different languages. Let's take dictation application as an example. When you have inverse text normalization, whole numbers may be grouped differently from country to country, as shown in Table 18.3. In the United States, we group three digits separated by a comma. However, many European locales use a period as the number separator, and Chinese conventions typically do not group numbers at all.

Table 18.3 Examples of how numbers are grouped by different locales for inverse text normalization.

123,456,789.0	United States and most locales
123456789.0	Chinese
12,34,56,789,0	Hindi

For the speech recognition engine localization, there are a number of excellent studies on both Asian [5, 15, 16, 21, 24, 32, 34, 37] and European languages [7, 8, 20, 22, 27, 31, 40]. Many spoken language processing components, such as the speech signal processing module, the speech decoder, and the parser, are language independent. Major changes are in the *content* of the engine, such as the lexicon and its associated processing algorithm, the grammar, and the acoustic model or language model parameter.

In general, it is easier to localize speech recognition than either text-to-speech or spoken language understanding. This is because a speech recognition engine can be mostly automatically derived from a large amount of speech and text data, as discussed in Chapters 9 through 13. For Chinese, you need to have special processing for tones, and the signal-processing component needs to be modified accordingly [4]. For both Chinese and Japanese the lexicon also needs to be carefully selected, since there is no space between words in the lexicon [12, 46]. As discussed in Chapter 17, the parser can be used without much modifica-

Amazon/VB Assets
Exhibit 1011
Page 970