Chapter 1

WHERE DO WE GO FROM HERE?

Research and Commercial Spoken Dialogue Systems

Roberto Pieraccini
SpeechCycle Inc.
New York, NY, USA
roberto@speechcycle.com

Juan M. Huerta
IBM T.J.Watson Research Center
Yorktown Heights, NY, USA
huerta@us.ibm.com

Abstract The spoken dialogue industry has reached a maturity characterised by a vertical structure of technology vendors, platform integrators, application developers, and hosting companies. At the same time industrial standards are pervading the underlying technology and providing higher and higher levels of interoperability. On the one hand commercial dialogue systems are largely based on a pragmatic approach which aims at usability and task completion. On the other hand, spoken dialogue research has been moving on a parallel path trying to attain naturalness and freedom of communication. However, given the constraints of the current technology, the evolution of the commercial path shows that naturalness and freedom of expression are not necessarily a prerequisite for usability. The difference between the two goals has been influencing a parallel evolution of the architectures and in particular of the dialogue management abstractions. We believe it is the time to get a high level perspective on both lines of work, and aim at a synergistic convergence.

Keywords: Spoken dialogue system; voice user interface; dialogue manager

1. Introduction: Overview of Dialogue Systems

There are different lines of research in the field of spoken dialogue. Some researchers attempt at understanding, and possibly replicating, the mechanisms of human dialogue through linguistically motivated studies on human–human

L. Dybkjær and W. Minker (eds.), Recent Trends in Discourse and Dialogue, 1–24.

corpora. Others are interested in general design principles that, once applied, would result in usable human–machine user interfaces based on speech recognition and speech synthesis technology. Then, there is spoken dialogue system engineering (McTear, 2004), which aims at developing programming styles, models, engines and tools which can be used to build effective dialogue applications. The three lines of research are, in a way, orthogonal and complementary. The focus of the first is on understanding human communication, the second on designing the interface for usable machines, and the third on building those usable machines. The topic of this chapter is concerned with the latter, namely the engineering of spoken dialogue systems. However, every discussion on the engineering of dialogue systems would be flawed if we did not take into consideration both the nature of human–human dialogue – as this is the most efficient realization of spoken dialogue available in nature – and the goal of usability. Dialogue systems can be further studied in accordance to other dimensions, for example modality (e.g., speech only, audiovisual, multimodal), input sensors (e.g., telephone, microphone, keyboard, joystick, touch screen, gesture capture), target platform (e.g., embedded, server-based), application domains (e.g., pervasive help, personal assistance, transactional systems, command and control) and many others. We believe that these dimensions are complementary to the focus of this chapter.

The goal of usability – i.e. building machines that are usable by untrained users – is often confused with that of building human-like conversational systems. This is based on the underlying tacit assumption that a machine that approximates human behaviour, from the linguistic point of view is certainly more usable than one that does not. Although possibly true in the limit, this assumption is often misleading, especially if we consider that the performance of spoken language technology[1] today is still far from near-human performance. However, most of the research carried out during the past decade has been directed towards unconstrained natural language interactions based on the assumption that *naturalness* and *freedom* of expression are the essential goals to pursue, and usability would automatically follow from having reached these goals.

The limitation of current spoken language technology is a fact we have to live with. Thus, if we undertake the goal of building usable systems with that limitation, we would find that naturalness and freedom of expression may actually hinder usability (Oviatt, 1995; Williams and Witt, 2004) for a large number of useful applications. For instance let us consider spoken language understanding technology. In spite of the advances of the past decade, even in

[1]With the term *spoken language technology* we refer to all the technologies that attempt the replication of human spoken language skills by machines, including speech recognition, spoken language understanding, speech to speech translation, speech synthesis, and text to speech.

well-defined domains, unrestricted understanding of speech is still far to be on a par with humans. So, any spoken language system that encourages free and natural user interactions is bound to a non-insignificant level of understanding errors (Sagawa et al., 2004; Bohus and Rudnicky, 2007). Moreover, as of today, there are no effective error recovery dialogue strategies[2] available for unconstrained natural language interactions. Conversely there are several types of transactional applications that achieve high usability with interactions that are not *natural* and *free*. After all call centres often adopt scripts to be followed by their customer service representatives (CSR) which do not leave much freedom to callers.[3] Most of the applications in this category are characterised by a domain model that is well understood by the user population. For instance, the model for ordering pizzas is known to most of the users: a number of pies of a certain size (small, medium, or large) with a selection of toppings (mushroom, pepperoni, etc.) The same applies to the domain of flight status information: flights can be on time, late, or cancelled, they arrive and depart daily from airports which serve one or more cities and can be identified by a number or by their itinerary and time. All of this is generally well understood by most of the users of commercial flights. Banking, stock trading, prescription ordering, and many other services belong to the same category.

Generally, when the domain model is quite simple and known by the users, as in the above cases, applications can be implemented in a structured dialogue fashion, generally referred to as *directed dialogue*. Directed dialogue, even if seemingly more restrictive from the point of view of the user, can attain much higher usability and task completion rates than free form interaction does, given the limitations of the current technology. In fact, when users are prompted to provide specific pieces of information, the system can activate grammars designed to collect exactly that information. Moreover, as discussed in Oviatt (1995), user guidance reduces user disfluencies. Thus, the combination of user direction, strict grammars, and less disfluent speech can attain quite high recognition accuracy. On the other hand, a more open interaction would increase the space of possible user expressions at each turn, often causing a reduction of the recognition accuracy. Furthermore, without direct guidance, most users will be lost and would know neither what to say nor what the capabilities and limitations of the system are.

The concept that well-structured directed dialogue strategies may outperform natural language free-form interactions was realized by speech

[2]One of the problems arising when trying to implement error recovery in unconstrained speech is the automatic detection of recognition errors. In fact, today's speech recognition confidence measures are still highly unreliable, especially when one attempts to apply them to portions of an utterance. Without viable error correction, interaction with machines may be extremely frustrating for the user.

[3]As a matter of fact, human–human flight reservation generally follows a precise script that is dictated by the order of the entries in the CSR database.

technology vendors during the early and mid-1990s. The development of a *spoken dialogue market* during those years led to the rise, in the late 1990s, of a well-structured industry of speech engines, platform and tool vendors, application developers, and hosting companies, together with an increased attention to the industrial standards. In fact several standards are today governing the speech industry, such as VoiceXML 2.0,[4] MRCP,[5] SRGS,[6] SSML,[7] CCML,[8] and EMMA.[9] Mainly driven by the VoiceXML standards, the speech and the Web world started to merge, and the benefits of this standardisation trend took a momentum amplified by the simultaneous emergence of Web standards (e.g. J2EE[10] and JSP[11]).

From a different point of view it is interesting to notice that the research community has often adopted dialogue approaches based on general principles (e.g. Grice, 1975) that once coded give machines a reasonable behaviour for reacting to different dialogue situations. Then, in order to cope with the limitations of the technology, research started falling back to more restrictive dialogue strategies. In contrast, the commercial community started from a pragmatic approach, where each interaction is practically designed in the minimal details by *voice user interface* (VUI) experts (Barnard et al., 1999).

After mastering the crafting of directed dialogue applications, the commercial community is moving now towards more free-form types of interactions. One example of that is offered by a category of applications where *directed dialogue* cannot be applied. Applications of this type are characterised by a domain model which is complex and unknown to the majority of users. Help desk applications, for instance, fall in this class. For example, a directed dialogue system for routing callers to the appropriate computer support may prompt the user with: *Is your problem related to hardware, software, or networking?* Unfortunately the vast majority of users would not know which of the three categories would apply to their problem. A solution to that would be to provide a menu that includes all possible problems, but that menu would be too long to

[4]http://www.w3.org/TR/voicexml20/.

[5]Media Resource Control Protocol: a protocol for the low level control of conversational resources like speech recognition and speech synthesis engines: http://www.ietf.org/internet-drafts/draft-shanmugham-mrcp-06.txt.

[6]Speech Recognition Grammar Specification: a language for the specification of context-free grammars with semantic attachments: http://www.w3.org/TR/speech-grammar/.

[7]Speech Synthesis Markup Language: a language for the specification of synthetic speech: http://www.w3.org/TR/2004/REC-speech-synthesis-20040907/.

[8]Call Control Markup Language: a language for the control of the computer-telephony layer: http://www.w3.org/TR/ccxml/.

[9]Extensible MultiModal Annotation: a language for the representation of semantic input in speech and multimodal systems: http://www.w3.org/TR/emma.

[10]Java Platform, Enterprise Edition is the industry standard for developing portable, robust, scalable and secure server-side Java applications: http://java.sun.com/javaee/index.jsp.

[11]Java Server Pages technology provides an effective way to create dynamic web content: http://java.sun.com/products/jsp/.

be practical. In other words, the underlying domain model, unlike that of pizza orders and flight information, is largely unknown by the user population. The solution to this problem consists in letting callers express themselves freely, and back the system with a statistical classifier able to assign user utterances to one of several predefined categories. In other words the system is charged with the burden of mapping user expressions to the domain model, and not the users themselves. This technique, known as How May I Help You (Gorin et al., 1997), statistical call routing, or statistical natural language understanding (Chu-Carroll and Carpenter, 1999; Goel et al., 2005) is not more than a very simple form of language understanding which combines the robustness of a structured approach (a limited number of categories, or routes) with the flexibility of natural language (an open prompt leading to a large number of possible user expressions). In fact, using this technology, the dialogue can still be structured in a directed dialogue manner, because the output of the interaction is going to be one of a predefined number of categories.

The goal of this chapter is to give a high level view of the domain of dialogue systems both in the research as well as in the commercial domain, with a focus on the problem of dialogue management. The rest of this chapter is organised as follows: Section 2 describes the importance of dialogue *control* and authoring *expressiveness* in commercial dialogue applications and the principle of VUI completeness. Section 3 provides a working definition of dialogue management. Section 4 gives a detailed overview of the basic architectures used for building dialogue applications. The following sections (5, 6 and 7) describe the main approaches to dialogue management, namely programmatic, finite state, and inference based. Section 8 provides an overview of the latest trends in commercial dialogue systems, and finally Section 9 reports our closing conclusions.

2. VUI Completeness

The need for a detailed control of the VUI is thus an important factor driving the architectural and engineering choices in commercial dialogue systems. We call this the *VUI-completeness* principle: the behaviour of an application needs to be completely and explicitly specified with respect to every possible situation that may arise during the interaction. No unpredictable user input should ever lead to unforeseeable behaviour. Only two outcomes are acceptable, the user task is completed as specified in the design specification, or a fallback strategy is activated (e.g. escalation to an operator).

In order to ensure that an application is *VUI-complete*, its behaviour needs to be specified for each possible situation, or class of situations. Today, a complete VUI specification is standard procedure in commercial deployments and is generally represented by a graph that describes all the possible dialogue states, complemented by tables that describe all the details of each state. Transitions

between dialogue states are described with conditions predicated on the user inputs and other pieces of information (e.g. previous user inputs, back-end response and personal user information). The precise wording of system prompts is also specified in the design, along with an indication of the type of utterances accepted at each turn. The VUI specification document is then handed to a team of developers who implement the application using the platform of choice. In some situations the VUI designers use advanced authoring tools that lead to the complete development of the application without, or with limited, intervention of software engineers and developers. In order to reduce development costs, it is thus important to guarantee a direct mapping between the formalisms and abstractions used by the VUI designers and the programming models available to the developers. This is the reason why most commercial dialogue managers follow the same abstraction utilised in the VUI specification.

Research systems, on the other hand, are typically designed to manage dialogue in situations where the product space of inputs, dialogue states, and outputs makes an explicit and exhaustive enumeration of all the possibilities impossible or impractical at best. This is due in part to the aim that the research community has towards handling unrestricted natural language input and mixed-initiative[12] dialogue control. On the other hand, an explicit enumeration (e.g., expansion into a deterministic graph) of the possible inputs as well as the possible dialogue transition states is required in commercial systems to establish a complete VUI specification that can be signed off by the client who pays for the development of the system itself. It is in general uncommon to find research systems which present full VUI-completeness.[13]

2.1 Control and Expressiveness

In order to allow developers to implement detailed VUI specifications, the programming paradigm adopted by the dialogue manager or authoring tools should allow a fine control of the system behaviour. However, a too low-level development paradigm may result in prohibitive development costs for large and complex applications. Hence the programming paradigm needs also to be expressive enough to allow implementing complex behaviour in a simple and cost effective way. These two features are often competing, since in order to guarantee more expressiveness the dialogue manager has to allow for sophisticated built-in behaviour, which may be hard to bypass if one wants to attain

[12]The term mixed-initiative is generally used to refer to those dialogue systems that allow the user, as well as the system, to change the course of the interaction at any point in the dialogue.

[13]In the DARPA Communicator evaluations, participant sites implemented systems with common requirements on the travel planning problem. Had a VUI-complete specification been set forth by the community as a joint effort, part of the evaluation would had simply consisted in verifying application compliance against the specification document. Still the user experience and the usability of the interfaces would have played an important role in the differentiation and evaluation of the systems.

a detailed control of the interface. An effective programming and authoring paradigm for dialogue systems is thus the result of a trade-off between control and expressiveness. This can be summarised by the following principle: *simple things should be simple and complex things should be possible.*[14]

3. Dialogue Management

The design of a proper dialogue management mechanism is thus at the core of dialogue system engineering. The study of better dialogue managers and proper dialogue engineering aims mainly at reducing the application development costs. But it is also a way to move towards more sophisticated human machine interactions, since it is only with proper engineering of dialogue systems that we can raise the complexity threshold that separates what is practically realizable from what is not.

There is not an agreed upon definition of what a dialogue manager is; different systems described in the literature attribute different functions to it. Some of these functions are: integrating new user input, resolving ambiguities, confirming and clarifying the current interpretation, managing contextual information, communicating with the back-end, managing speech recognition grammars, generating system outputs, etc. In fact, the minimal functionality required by a dialogue manager covers two fundamental aspects of all interactive applications: keeping track of session states and deciding what the next action for the system to take is. Of course there are many ways of coding these two functions in order to achieve a desired interactive behaviour. The rest of this chapter will describe some of them.

4. Reference Architectures: Research and Commercial

In order to describe different approaches to dialogue management, it is important first to define, at a high level, the architecture of spoken dialogue systems.

Figure 1 shows a general functional architecture of a dialogue system, mostly used in research prototypes. We refer here and in the following to telephone-based systems. However, some of the concepts expressed in this chapter can be generalised to other types of system that do not make use of telephone communication, such as embedded systems for mobile devices and for automobiles. While the description of how some of these principles apply to different non-telephony systems is beyond the scope of this chapter, we should mention that commercial systems, especially in the embedded area, are

[14]This maxim is attributed to Alan Kay.

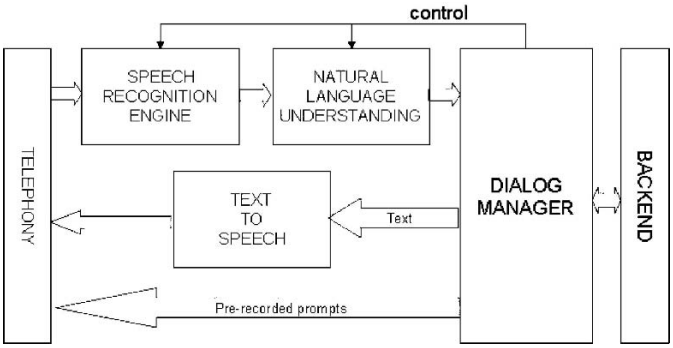

Figure 1. Functional architecture of a dialogue system mostly used in research prototypes.

moving towards multimodal applications. Applications where speech is not the only modality, but is integrated with haptic and visual interfaces, are becoming more and more common, and certainly need more sophisticated architectures than the ones described here.[15] A discussion on some of the issues related to multimodal systems can be found in Pieraccini et al. (2005).

In the most common configuration of a spoken dialogue system architecture, input speech is collected via a telephone interface and dispatched to the speech recognition engine which provides one or more recognition results (for instance the *n-best* recognition results). Each recognition result is then fed to a *natural language understanding* processor which extracts the semantics of the utterance. A formal representation of the semantics, generally a structured set of attribute-value pairs, is then passed on to the dialogue manager. The dialogue manager, based on the current utterance semantics, and on the stored contextual information derived from previous turns, decides the next *action* to take according to a *dialogue strategy*. The most obvious action performed by the system as a response to a user utterance is a system utterance, generally referred to as *prompt*, which can be generated as text and transformed into speech by a *text-to-speech* engine, or selected from a set of pre-recorded samples.[16] Other types of action performed by the dialogue manager include interactions with the back-end system, or any other type of processing required by the application.

The above described architecture has been implemented in many different forms in research. Of particular interest is the Galaxy architecture (Seneff et al.,

[15]Examples are SmartKom (http://smartkom.dfki.de/) and Embassi (http://www.embassi.de).

[16]High-quality prompts are today obtained by splicing pre-recorded phrases with TTS generated content, using concatenative speech synthesis.

1999) which was used in the DARPA Communicator[17] project and allowed the interchange of modules and plug-and-play across different research groups.

One thing to notice in the above described architecture is that the specific language models used by the speech recognition and natural language understanding engines are supposed to be constant throughout a whole session. In fact, one of the basic assumptions behind most research prototypes is that the system should be able to understand all the possible expressions defined by the language model at any point during the interaction. However it is clear that there is a correlation between the distribution of possible utterances and the dialogue state or context. Thus in order to improve system performance, the dialogue manager can change the parameters of the language model and language understanding depending on the current dialogue context. Several systems did implement this feedback loop with resulting improved performance (Xu and Rudnicky, 2000).

Commercial system architectures, instead, evolved in a different way. The basic assumption on which most of the commercial deployed systems were, and still are, based can be expressed by the following statement: properly designed prompts can effectively control the space of user expressions. Thus, based on this assumption, there is no need for the system to be able to understand, at each turn, all the possible expressions that users could say, since the user will mostly speak what is suggested by the prompts. Users are in fact *directed* (thus the term *directed dialogue*) and guided to speak exactly what the system expects. It is clear how this assumption, if true, can potentially allow the attainment of very high task completion rates by limiting the *unknowns*. Under this assumption, commercial dialogue systems provide the speech recogniser with grammars that are specifically designed for each turn of the interaction. Each grammar – typically a context-free grammar with semantic attachments – is specifically designed to accept the utterances that are expected to be possible user reactions to the specific prompt played at that particular turn. So, instead of a generic prompt like *Hello, this is XYZ flight status information line, how can I help you today?* commercial dialogue system designers use more specific prompts such as *Are you interested in arrivals or departures?* or *From which city is the flight departing?* Prompts and grammars, thus, need to be designed together.

The benefit of using restricted grammars in directed dialogue applications becomes evident when looking at the error control logic typically adopted by commercial systems. In fact even with very restricted grammars there is always a chance for the recogniser to produce erroneous interpretations, or for the user to speak utterances outside the domain. Thus in case of poor recognition

[17] http://communicator.sourceforge.net/.

scores, commercial dialogue systems *direct* users to correct presumably erroneous interpretations by using very strict prompts, such as: *I think you said Austin, is that correct? Please say yes or no*. And since the system cannot afford to confuse a yes with another phonetically similar word at this point in dialogue (misrecognitions in correction subdialogues would lead to enormous user frustration), the grammar associated with the confirmation prompt is typically restricted to yes/no utterances and a reasonable number of synonyms and command words (such as *help* and *operator*).

Early commercial dialogue systems were built using proprietary architectures based on IVR (Interactive Voice Response) platforms. Soon, the speech application development community realized the importance of industrial standards and started to create recommendations to guarantee interoperability of platforms and engines. After the introduction of VoiceXML 1.0[18] in year 2000, conversational systems started to conform to a general Web architecture, such as the one shown in Figure 2. The convergence of speech and Web technologies (the so called Voice Web) has allowed the speech industry to leverage existing Web skills and resources, and reduce the need for specialised developers.

The core of commercial dialogue systems exemplified by Figure 2 is the *voice browser* which accepts documents written in a markup language specific for speech applications, such as VoiceXML. The voice browser exchanges information with a Web server using the HTTP protocol in analogy with the browser and server in traditional visual Web applications. VoiceXML markup documents instruct the browser to activate the speech resources (speech recognition, TTS, prompt player, etc.) with a specific set of parameters, such as grammars for the speech recognition engine, prompts to be synthesised by the text-to-speech system, or audio recording to be played. Once the user's speech has been recognised, and the recognition results returned to the browser in the form of a structured set of variables, the browser sends them back to the Web

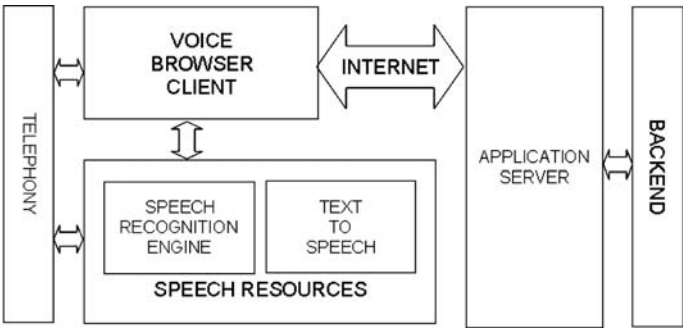

Figure 2. Typical architecture of commercial dialogue system.

[18] Voice eXtensible Markup Language.

server, together with the request for another VoiceXML document. The Web server then replies by sending the requested document to the browser, and the interaction continues in this fashion.

Using static VoiceXML documents, the dialogue manager function is actually distributed across the various VoiceXML pages, as in a static visual website, the navigation is distributed across the collection of HTML documents. In fact each document includes instructions for the browser to request the next document once the current one has been executed. All the VoiceXML documents and the corresponding resources (such as grammars, prompts, etc.) are typically stored statically on the Web server and *served*[19] to the browser upon request. However, as it happened in the visual Web world, developers found the mechanism of encoding the whole system in static VoiceXML pages quite limiting, and soon they started to write programs on the server for generating dynamic VoiceXML documents. In this case the application is actually managed by a program running on the application server, which acts as a dialogue manager and that generates dynamic VoiceXML documents upon requests by the browser. The introduction of the J2EE/JSP technology makes this process straightforward and in line with mainstream Web programming.

Generating VoiceXML dynamically on the server has the advantage of providing the developer with more powerful computational capabilities than those available on the voice browser client, and thus accommodating, in a more flexible way, the dynamic nature of sophisticated interactions and business logic. Moreover, there are security restrictions on the client browser that may prevent direct access to external resources, such as back-end databases. The evolution of server-based programming of applications brought the separation of the dialogue management functionality from the presentation (i.e. the activation of speech engines, playing of the prompts, etc.), and the realization of general purpose dialogue managers and programming models for developing speech applications on the server.

In spite of the different architectural evolution of research and commercial dialogue systems, the need for a powerful dialogue manager is felt by both communities. In the next few sections we will discuss some of the available models of dialogue manager which have been introduced in recent years.

5. Programmatic Dialogue Management

The simplest form of dialogue manager is a generic program implemented in a procedural programming language such as C++ or Java (or as a Java servlet in the case of Web-based architectures) that implements a dialogue application

[19] Voice browsers use caching strategies similar to those used by visual Web browsers. So, large grammars may be cached on the client so as to avoid significant resource provisioning latency.

without any underlying general interaction model. Early commercial dialogue applications were typically developed, on the deployment platform, as native code following a given VUI specification. Before the advent of VoiceXML and the Web programming paradigm for voice applications, IVR vendors integrated speech recognition engines directly in the platforms which had proprietary programming environments or proprietary APIs.[20]

However, building each application from scratch becomes soon an inefficient and repetitive activity. Like in all areas of software development, vendors tried to reduce the cost of application development by introducing libraries of reusable functions and interaction templates, often for internal consumption, but also as products that could be licensed to third parties. Libraries were also complemented by programming frameworks, generally in the form of sample code or templates, which could be reused and adapted to different applications.

Dialogue modules[21] developed by various speech recognition and tool providers, constitute one of the first forms of commercial reusable dialogue functions. Dialogue modules encapsulate all the low level activities required to collect one or more pieces of information from the user. That includes prompting, re-prompting in case of rejection and timeout, confirmation, disambiguation, etc. The collection procedure, including prompts, grammars, and logic for standard pieces of information, such as dates, times, social security number, credit card numbers, currency, etc., was thus encoded once and for all in pieces of reusable and configurable software. Developers could also build their own custom dialogue modules. Thus dialogue modules became, for many, the standard approach to directed dialogue. Applications were then implemented with the programming model available for the chosen platform. Each state of the dialogue flow was associated to a specific dialogue module, and the programming model of the platform was the glue used to implement the whole dialogue.

6. Finite State Control Management

The finite state control dialogue manager is an improvement on the programmatic dialogue manager. The finite state control dialogue manager implements a separation between the logic of directed dialogue and its actual specification. The logic is implemented by a finite state machine engine which is application

[20]Some platforms used GUI application development environments that were originally designed for touchtone (DTMF) applications, and then extended to handle speech recognition and TTS. Other commercial platforms allowed access to the functionality of the IVR and the speech recognition/TTS engines by exposing a proprietary API, and allowing it to be invoked by common programming languages such as C, Java, and Visual Basic.

[21]Commercialised by SpeechWorks as *Dialogue Modules* and by Nuance as *Dialogue Objects*.

independent and therefore reusable. Thus, rather than coding their own finite state machine mechanism directly in computer code, as in the programmatic model, developers had to provide a description of the finite state machine topology in terms of a graph of nodes and arcs. This can be defined as a data driven approach. Often the topology could be derived directly from the VUI specification. Then developers had to complement that with a set of custom functions required by the application. Without a separation between the finite state machine mechanism and its topology, the implementation of the dialogue state machine logic was often left to the programming skills of developers, often resulting in an unmanageable spaghetti-like nest of *if-else* or *case* statements, with increased debugging and maintenance costs, and made it impossible to build applications above a certain level of complexity.

One of the obvious advantages of the finite state control management approach is that the topology of the finite state machine is generally easier to write, debug, and maintain than the finite state machine mechanism itself. Moreover, the finite state machine engine can allow for hierarchical and modular dialogue definition (e.g. dialogues and subdialogues). Finally, the engine itself can be harnessed to verify the overall topology, check for obvious design and implementation mistakes, such as unreachable nodes and loops, and provide debugging and logging facilities. More sophisticated engines can have built-in behaviour like for instance handling specific navigation across dialogue networks, recording usage information for personalised services, implementing functions such as *back-up* and *repeat*, etc. (Pieraccini et al., 2001).

The simplest form of finite state control dialogue manager is built around the concept of *call flow* developed initially for IVR systems. In its simplest realization a call flow is a graph where the *nodes* represent prompts, and the *arcs* represent transitions conditioned on the user choice at that particular node (e.g. Figure 3). By navigating the call flow graph and by selecting the right choices, the user can reach the desired goal and complete the task. The call flow model is quite limited and breaks for complex dialogue systems since one has to explicitly enumerate all the possible choices at any node in the dialogue.

In fact the pure call flow model is inadequate to represent even modest levels of mixed-initiative, such as over-specification, when more than one piece of information is given by the user in a single utterance. For instance, if asked for the date of a flight[22] in a mixed-initiative system that allows for over-specified requests, users may instead respond with any subset of date, origin, destination, and airline. In order to be able to handle this, the simple call flow model would need to represent explicitly all the possible subsets of user choices (e.g. date, date + time, date + origin, date + origin + destination) making the design and development impractical.

[22]It looks like the spoken dialogue community has a penchant for applications related to flights. We hope to see other domains of interest in the future.

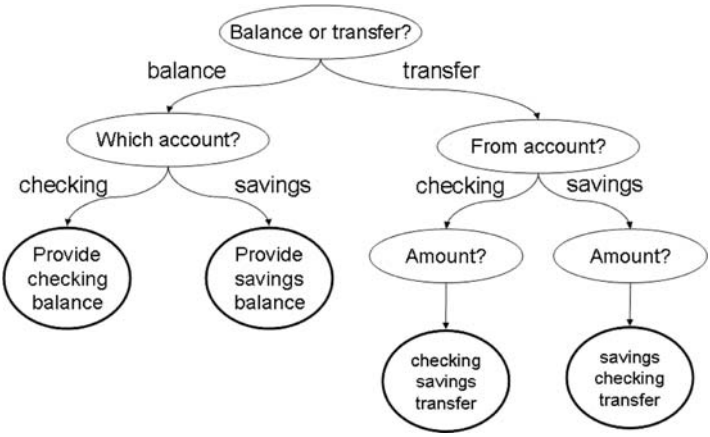

Figure 3. Example of call flow.

However, one can easily extend the concept of call flow and allow the state machine to assume any topology, to invoke any arbitrary function (action) at each node, and assume any arbitrarily complex condition on the arcs. Furthermore, one can allow any arbitrarily complex data structures (session state) to be writable and readable by the actions associated to the nodes. In this new extended form, the finite state control dialogue manager (we will refer to it as the *functional model*) has enough expressive power to represent sophisticated directed dialogue and mixed-initiative interactions. A full functional model of dialogue management can also allow for recursion, i.e. full dialogues specified in a functional fashion can be, themselves, used as actions and associated to nodes of a higher level dialogue, enabling thus hierarchical description of applications, and promoting modularity and reuse. An example of a control graph that handles over-specified utterances is shown in Figure 4 (explained later in this chapter). More detailed descriptions of functional models of dialogue management can be found in Pieraccini et al. (1997, 2001).

There are common misconceptions about the effective expressiveness and computational power of the finite state dialogue model. In fact limited capabilities with respect to more sophisticated abstractions are often wrongly attributed to finite state models of dialogue control. These misconceptions derive from the confusion that often exists between the functional model described above and the simplistic call flow model which is completely described by a state machine with prompts on the nodes and choices on the arcs. In its simpler form the call flow model is indeed, computationally, a *finite state model* of dialogue: i.e. the state of the dialogue is univocally determined by the node of the call flow. By contrast, the functional model allows arbitrary functions at each node to manipulate arbitrary memory structures that can be shared across nodes. Thus

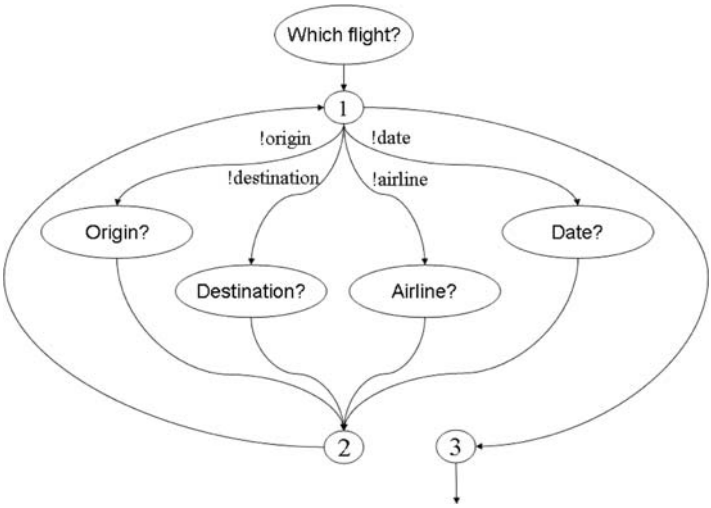

Figure 4. Graph representing a functional dialogue controller able to handle over-specified utterances. The conditions on the arcs exiting a node are verified in a left-to-right fashion. Arcs without conditions are to be considered as having an *else* condition.

the extended functional model is not, computationally, a finite state model of dialogue; it just makes use of a *finite state representation* – i.e. nodes and arcs – for the dialogue control mechanism. In fact each *node* of the finite state machine describing the dialogue control does not represent univocally the *state* of the dialogue because we need also to take into consideration the *state* of all the memory structures associated with the controller (e.g. the session state).

A functional dialogue manager is equivalent to a procedural program with a fixed structure based on nested conditional or *case* statements. The nodes are equivalent to function calls, while the conditions are equivalent to the conditional statements, and a whole dialogue is analogous to the definition of a function. However, a functional dialogue manager specification is much easier to author and debug than a set of nested conditional or *case* statements.[23]

6.1 Handling Mixed-Initiative in Functional Models

A clear limitation of functional models is that they often require a complete topological definition of the task that may be rather complex for certain types of applications. For instance, the implementation of mixed-initiative interactions

[23] As a proof of this, we leave to the reader the exercise of rewriting the controller in Figure 4 as a series of nested if/else-if/else statements.

may result in a control graph with a large, unmanageable number of arcs. One way to reduce the cost of designing and developing mixed-initiative dialogue applications within the functional model paradigm consists in providing the controller engine with a behaviour that corresponds to complex topologies, without the need for the developer to specify those in term of nodes and arcs. For example, in Pieraccini et al. (2001), the concept of state transition was extended to include special GOTO and GOSUB arcs to easily implement topic changes and digressions at any node of the dialogue. Powerful engines for functional dialogue models can also allow for effective authoring of *global* transitions that apply to whole sets of nodes.

6.2 Fixed Topology Models

One can implement functional dialogue managers that allow the developer to specify the control graph topology (Carpenter et al., 2002). On the other hand one could restrict the control graph to assume a fixed topology and allow developers to specify only a limited number of parameters.

The Form Interpretation Algorithm (FIA), the basis for the VoiceXML standard, is an example of a functional model of dialogue management with a fixed topology. The topology of the FIA controller is in fact the one shown in the example of Figure 4. The FIA topology is particularly suited for handling overspecified requests, allowing filling forms with multiple fields in any order. For instance, if after the initial question *Which flight?* the user specifies the destination and the airline, the arc *!origin* (i.e. *NOT origin*, meaning that the origin slot has not been filled) is traversed and the node *origin?* is executed next. As a result the user is asked to provide the origin of the flight. Then, the *date?* node is executed next since the condition *!date* proves to be true (i.e. a specific date is not available yet). After the user has provided all the required pieces of information (origin, destination, airline, and date) the subdialogue exits through node 3.

Another example of functional model with a fixed topology controller is the MIT dialogue management system (Seneff and Polifroni, 2000). In this case the control is defined by a sequence of functions that are activated when the associated conditions fire. Each function can modify a session state (i.e. a *frame*-like, or attribute-value memory structure) by adding additional information, including a flag which instructs the controller on what to do next. Possible flags are: CONTINUE, causing the execution of the next rule in the sequence, RETURN, causing the controller to return to the initial rule, or STOP the execution. Again, as in the VoiceXML case, developing a dialogue does not require a topological description of the control graph, which is fixed and has the functional form described by Figure 5, but the specification of the functions associated to the

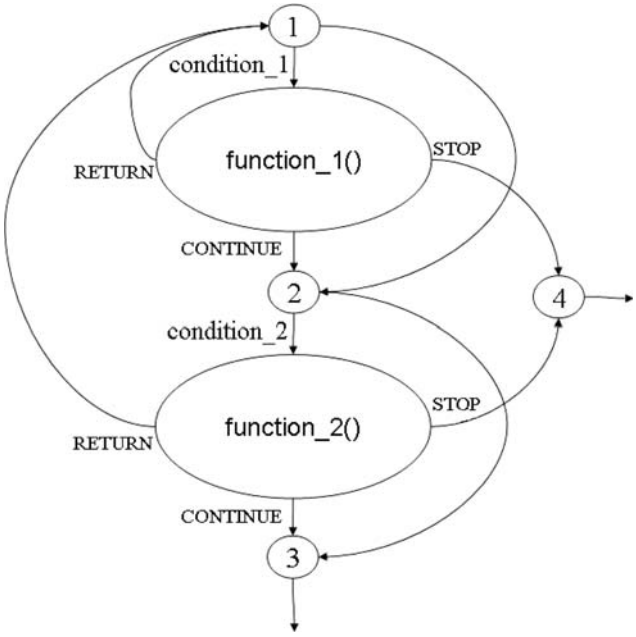

Figure 5. Functional control graph representing a rule based system.

nodes, and the conditions. The following is an example of a set of rules that implement the same subdialogue as the one in Figure 4.

$$!origin \rightarrow prompt\_origin()$$
$$!destination \rightarrow prompt\_destination()$$
$$!airline \rightarrow prompt\_airline()$$
$$!date \rightarrow prompt\_date()$$

7. Inference-Based Dialogue Managers

We have shown in the previous section how several types of dialogue manager can be reduced to a unique underlying model: the functional finite-state dialogue controller. The difference between them is whether developers are allowed to change the topology of the controller, and in the way they can author an application (e.g. by specifying a graph or a set of rules). However, there are classes of applications for which a specification through a finite state controller may appear impractical. As we discussed earlier, transactional applications with a well defined goal (e.g. giving information to the user, selling or buying) can often be effectively implemented with a finite state controller. On the contrary, some applications of the problem solving type (Allen et al., 2000)

with a high complexity require a higher degree of planning, for which the finite state controller can be inappropriate. Although we start seeing commercial technical support applications, which belong to the problem solving category, being successfully adopted by certain industries, most of the more complex problem solving applications have not yet found a channel to the market of spoken dialogue systems. This is probably because the research prototypes have not yet demonstrated the level of usability needed for commercial use. For instance, the deployment of some sophisticated research systems would require highly specialised development teams that may be prohibitively expensive in a commercial setting. Moreover the performance of the systems for the most complex problem solving tasks is not yet at the level required for commercial exploitation.

In spite of its difficulty, the research community has been actively pushing the technology towards the solution of the dialogue management problem for complex systems, especially under the auspices of the DARPA Communicator program. Successful prototypes have been demonstrated and tested based on sophisticated dialogue managers that deviate from the finite-state controller model, and include some degrees of inference. A distinguishing feature of the inference based systems is that they refrain from attempting at a more or less explicit description of the relationship between states and actions, as in the finite state controllers, but rather resort to engines that draw decisions on the next action to perform based on a general strategy and on a formal description of the domain, typically in terms of goals and subgoals. Thus, in order to develop an application, rather than describing the VUI, one starts from a formal description of the domain model in such a way to allow the inference engine to drive the system to a cooperative solution.

In Stallard (2001) the dialogue control model is described by a tree representing the goal/sub-goal structure, with the leaves of the tree being the actions. Actions, which include pre-conditions for their execution, are associated to individual goals. Internal nodes represent conditional controls on the execution of the underlying nodes. A dialogue manager based on task ontology and a hierarchy of nodes is described in Pellom et al. (2000). The dialogue manager described in Wei and Rudnicky (2000) constructs a dynamic structure, called *agenda*, which is a list of subgoals, where each subgoal corresponds to the collection of some piece of information. A task is completed when all the items in the agenda are completed. The agenda is created, dynamically, by traversing a tree (i.e. the *product tree*) that describes, at any point in time, the current task to be completed. The product tree has to be created dynamically since the nature of the task may be dynamic as well (e.g. the number of legs of a flight is determined during the interaction and not known beforehand). In the form based dialogue manager described in Papineni et al. (1999) the inference mechanism is driven by a numeric function computed on a set of

partially completed forms (i.e. sets of task-relevant slots), based on how close to the goal (i.e. the retrieval of information from the database) each individual hypothesis is.[24]

Another line of research is based on statistical learning of the dialogue strategy using mathematical models derived from statistical machine learning, such as Markov Decision Process (Levin et al., 2000) or Bayesian network frameworks (Meng et al., 2003). It is still too early to be able to understand whether automated design of dialogue can allow building usable systems with a quality comparable to that of those designed by VUI expert designers.

It is not yet clear whether any of the sophisticated inference dialogue managers developed in research could be effectively used for mass production of commercial systems. One of the problem is that their behaviour is quite complex, and it may be difficult to predict all possible situations that could arise during the interaction. Thus VUI completeness may be hard to achieve. Research prototypes, so far, have been built by researchers with an intimate knowledge of the quirks of the dialogue manager itself, and often by those who built it. Thus, in order to succeed in the commercial arena, inference engines have to produce systems with usability and robustness comparable or superior to that of an equivalent directed dialogue for the same task, or implement applications that are so complex that they cannot be approached with directed dialogue, still with usability as their main goal. VUI completeness is an essential requirement which should be seriously taken into proper consideration for the more sophisticated dialogue manager models.

8. Current Industrial Trends

Reusable components (Huerta et al., 2005) and pre-packaged applications are the main trends of the industry of spoken dialogue systems today. Componentization and reuse effectively allow reducing deployment costs and risks and, at the same time, simplifying the design and development of more sophisticated applications. Thus the commercial world is approaching the creation of more complex applications through more and more sophisticated building blocks which allow reuse and interplay.

Additionally, the need for language flexibility and robustness has motivated the use of Natural Language Understanding (NLU) technology. This requirement has allowed NLU technology to move from just call routing (Gorin et al., 1997) to a more sophisticated use, like for instance understanding and categorising symptoms in technical support applications.[25]

[24] A commercial version of this dialogue manager was implemented by IBM and used in a financial application (T. Rowe Price).

[25] http://www.speechcycle.com.

9. Conclusions

The way applications are authored, what capabilities the systems have, and the overall usability that is eventually perceived by users reflect the different goals that research and industry have in the field of spoken dialogue systems. Whereas usability and cost effectiveness are the primary goals of the commercial community, research has traditionally aimed at naturalness of interactions and freedom of expression. However, often the latter does not necessarily lead to the former. The actual form assumed by dialogue managers in both communities is the consequence of those different goals. In fact, in order to achieve high usability, commercial deployments aim at having completely definable interfaces (control and VUI completeness), using efficient languages and architectures (expressiveness and simple-things-should-be-easy) while keeping the ability to achieve complex levels of interaction (complex-things-should-be-possible). At the same time, the focus of research is towards abstracting, validating and achieving complex levels of natural interaction. While at first glance both sets of goals might seem in conflict, we believe that an evolution towards more complex levels of interaction, while using an effective development framework and implementing a "controllable" (VUI complete) interface is possible.

We have shown that most commercial dialogue management abstractions fall into the functional finite-state controller mechanism, as well as some of the dialogue managers developed in research. The difference is in the constraints applied to the topology of the controller and in the type of authoring (graphs vs rules). We have also shown that there is a second category of dialogue managers, inference based, which is devoted to handling more complex interactions, such as problem solving applications. VUI completeness and economy of development are required for them to become viable and reach the level of usability needed to succeed in the commercial arena.

We believe that the authoring of applications should be aligned with the model used at design time, and possibly to the runtime environment. In this way efficiency can be achieved at all levels: design, development, and deployment. The framework should allow for the encapsulation of dialogue mechanisms into templates, components, and subroutines that abstract behaviours. Beyond allowing for a reduction of development costs, this is also the first step towards the implementation of more complex interaction mechanisms. Finally, the framework should have strict "directed" and thus controllable default behaviour, but at the same time should allow for more complex interactions to be triggered if and when these dialogue mechanisms would benefit the interaction (e.g. expert and power users).

An important consideration that needs to be made when talking about the use and usability of commercial dialogue systems is that their success has to

take into account the willingness of the user to cooperate. Even the best designed application based on the most advanced architecture fails when callers refuse to use it. The problem is that the general public, the population of users of commercial dialogue systems, is at best annoyed when they are faced with a computer rather than a live agent. This phenomenon can be attested by the *cheat-sheets*[26] published on the Web that suggest words and sequences of touch tone keys for callers of commercial customer care applications to get a human operator right away. We can make an analogy with many other automated systems that are massively used today, and that provoked a similar reaction when they were introduced first a few years ago, such as ATMs (or cash machines) and answering machines. ATMs and answering machines are ubiquitous today and nobody would ever think of them as *replacements* for bank tellers and receptionists, but useful tools that improve our way of life. Similarly dialogue systems are to be considered as tools that, if used properly, can provide faster and better service for the most common situations and problems.

So, what is the main difference between research and commercial dialogue systems? We can certainly say that while some research was originally inspired by the dream of moving towards human-like interfaces characterised by fully unconstrained interactions (the dream of a HAL 9000-like machine from the celebrated 1968 *2001, A Space Odyssey* movie), commercial dialogue systems have to have more practical goals such as robustness, usability, testability, and ease of design and maintenance. Because of that, the commercial community took the approach of very controlled interfaces with constrained input and limited initiative on the part of the user. In other words they assumed that compliant users will *learn* how to use non-natural and highly limited interfaces, when compared with a human–human analogy, in order to get their task done. While the goal of research is certainly more ambitious, it has not had yet the opportunity to bring to life truly natural language mixed-initiative dialogue systems as a viable alternative to the constrained commercial directed dialogues. This is due, in part, to the lack of exposure that research systems have to a high volume usage that helps drive their improvement, the lack of research funds and the interest of the funding agencies, and also in part to the choice of research applications that often can be easily outperformed by analogous directed dialogue commercial applications. Should research concentrate on applications that are so complex that they *could not* be approached with equal or higher effectiveness by commercial systems, that would help establish a research goal way beyond the current technology.

[26]http://gethuman.com.

Finally we believe that a consolidation of the goal priorities (i.e. usability and naturalness of interaction) between research and the commercial world will foster further maturation of the technology. For this to happen, though, the *dialogue* needs to start.

References

Allen, J. F., Ferguson, G., and Stent, A. (2000). Dialogue Systems: From Theory to Practice in TRAINS-96. In Dale, R., Moisl, H., and Somers, H., editors, *Handbook of Natural Language Processing*, pages 347–376. Marcel Dekker, New York.

Barnard, E., Halberstadt, A., C., K., and Phillips, M. (1999). A Consistent Approach to Designing Spoken-Dialog Systems. In *Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 363–366, Keystone.

Bohus, D. and Rudnicky, A. (2007). Sorry, I Didn't Catch That! An Investigation of Non-Understanding Errors and Recovery Strategies. In Dybkjær, L. and Minker, W., editors, *Recent Trends in Discourse and Dialogue*. Springer. (This volume).

Carpenter, B., Caskey, S., Dayanidhi, K., Drouin, C., and Pieraccini, R. (2002). A Portable, Server-Side Dialog Framework for VoiceXML. In *Proceedings of International Conference on Spoken Language Processing (ICSLP)*, pages 2705–2708, Denver.

Chu-Carroll, J. and Carpenter, B. (1999). Vector-Based Natural Language Call Routing. *Computational Linguistics*, 25(3):361–388.

Goel, V., Kuo, H.-K., Deligne, S., and Wu, S. (2005). Language Model Estimation for Optimizing End-to-End Performance of a Natural Language Call Routing System. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 565–568, Philadelphia.

Gorin, A. L., Riccardi, G., and Wright, J. H. (1997). How May I Help You? *Speech Communication*, 23:113–127.

Grice, H. P. (1975). Logic and Conversation. Syntax and Semantics. In Cole, P. and Morgan, J. L., editors, *Speech Acts*, volume 3, pages 41–58. Academic, New York.

Huerta, J., Akolkar, R., Faruquie, T., Kankar, P., Rajput, N., Raman, T., Udupa, R., and Verma, A. (2005). Reusable Dialog Component Framework for Rapid Voice Application Development. In *Proceedings of International SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*, pages 306–321, St. Louis.

Levin, E., Pieraccini, R., and Eckert, W. (2000). A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies. *IEEE Transactions on Speech and Audio Processing*, 8(1):11–23.

McTear, M. (2004). *Spoken Language Technology: Toward the Conversational User Interface*. Springer, London.

Meng, H. M., Wai, C., and Pieraccini, R. (2003). The Use of Belief Networks for Mixed-Initiative Dialog Modeling. *IEEE Transactions on Speech and Audio Processing*, 1(6):757–773.

Oviatt, S. L. (1995). Predicting Spoken Disfluencies during Human-Computer Interaction. *Computer Speech and Language*, 9:19–35.

Papineni, K., Roukos, S., and Ward, R. (1999). Free-Flow Dialog Management Using Forms. In *Proceedings of European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 1411–1414, Budapest.

Pellom, B., Ward, W., and Pradhan, S. (2000). The CU Communicator: An Architecture for Dialog Systems. In *Proceedings of International Conference on Spoken Language Processing (ICSLP)*, pages 723–726, Beijing.

Pieraccini, R., Carpenter, B., Woudenberg, E., Caskey, S., Springer, S., Bloom, J., and Phillips, M. (2005). Multimodal Spoken Dialogue with Wireless Devices. In Minker, W., Bühler, D., and Dybkjær, L., editors, *Spoken Multimodal Human-Computer Dialogue in Mobile Environments*, pages 169–184. Springer.

Pieraccini, R., Caskey, S., Dayanidhi, K., Carpenter, B., and Phillips, M. (2001). ETUDE, a Recursive Dialog Manager with Embedded User Interface Patterns. In *Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 244–247, Madonna di Campiglio, Trento.

Pieraccini, R., Levin, E., and Eckert, W. (1997). AMICA: The AT&T Mixed Initiative Conversational Architecture. In *Proceedings of European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 1875–1878, Rhodes.

Sagawa, H., Mitamura, T., and Nyberg, E. (2004). Correction Grammars for Error Handling in a Speech Dialog System. In *Proceedings of Annual Meeting of the Human Language Technology Conference/North American chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 61–64, Boston.

Seneff, S., Lau, R., and Polifroni, J. (1999). Organization, Communication, and Control in the Galaxy-II Conversational System. In *Proceedings of European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 1271–1274, Budapest.

Seneff, S. and Polifroni, J. (2000). Dialogue Management in the MERCURY Flight Reservation System. In *Proceedings of ANLP/NAACL*, pages 11–16, Seattle.

Stallard, D. (2001). Dialogue Management in the Talk'n Travel System. In *Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 235–239, Madonna di Campiglio, Trento, Italy.

Wei, X. and Rudnicky, A. (2000). Task-Based Dialog Management using an Agenda. In *Proceedings of ANLP/NAACL*, pages 42–47, Seattle.

Williams, J. and Witt, S. (2004). A Comparison of Dialog Strategies for Call Routing. *International Journal of Speech Technology*, 7(1):9–24.

Xu, W. and Rudnicky, A. (2000). Language Modeling for Dialog System. In *Proceedings of International Conference on Spoken Language Processing (ICSLP)*, volume 1, pages 118–121, Beijing.