Where do we go from here?
Research and commercial spoken dialog systems.

Roberto Pieraccini **Juan Huerta**

IBM T.J.Watson Research Center
1101 Kitchawan Road, Route 134
Yorktown Heights, NY 10598

Abstract

The spoken dialog industry has reached a maturity characterized 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 one hand commercial dialog systems are largely based on a pragmatic approach which aims at usability and task completion. On the other hand, spoken dialog research has been moving on a parallel path trying to attain naturalness and freedom of communication. However, the evolution of the commercial path shows that naturalness and freedom of expression are not necessarily a prerequisite for usability, given the constraints of the current technology. The difference between the two goals has been influencing a parallel evolution of the architectures and in particular of the dialog management abstractions. We believe it is the time to get a high level perspective on both lines of work, and aim to a synergistic convergence.

1 Introduction

There are different lines of research in the field of spoken dialog. Some researchers attempt at understanding, and possibly replicating, the mechanisms of human dialog through linguistically motivated studies on human-human 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 dialog system engineering (McTear, 2004), which aims at developing programming styles, models, engines and tools which can be used to build effective dialog 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 paper is concerned with the latter, namely the engineering of spoken dialog systems. However, every discussion on the engineering of dialog systems would be flawed if we did not take into consideration both the nature of human-human dialog—as this is the most efficient realization of spoken dialog available in nature—and the goal of usability.

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 approximate human behavior—from the linguistic point of view—is certainly more usable that 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 during the past decade was 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 those 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 given that limitation, we would find that, for a large number of useful applications, naturalness and freedom of expression may actually hinder usability (Oviatt, 1995; Williams and Witt, 2004). For instance, let's consider spoken lan-

[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 and translation, speech synthesis and text to speech.

guage understanding technology. In spite of the advances of the past decade, even in 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. Moreover, as of today, there are no viable error recovery dialog 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 some call centers 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 characterized 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 flight status domain model: 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. 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 dialog fashion, generally referred to as *directed dialog*. Directed dialog, even if seemingly more restrictive from the point of view of the user, can attain much higher usability and task completion rates that free form interaction does with 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 disfluencies can attain quite high speech recognition rates. On the other hand, a more open interaction would increase the space of possible user expressions at each turn, thus 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 dialog strategies may outperform natural language free-form interactions was realized by speech technology vendors during the early and mid 1990s. The development of a *spoken dialog market* during those years led to the rise, in the late 90's, of a well structured industry of speech engines, platforms, and tool vendors, application developers, and hosting companies, together with an increased attention to the industrial standards. 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]. The speech and the Web world started to merge, and the benefits of this standardization trend took a momentum amplified by the simultaneous emergence of Web standards (e.g. J2EE, JSP, etc.).

It is interesting to notice that the research community has often started from dialog approaches based on general principles (e.g. Grice, 1975) that once coded give machines a reasonable behavior for reacting to different dialog situations. Then, in order to cope with the limitations of the technology, research started falling back to more restrictive dialog 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 dialog applications, the commercial community is moving now towards more free form types of interactions. One example of that is with respect to those types of applications where *directed dialog* cannot be applied. Applications of this type are characterized by a domain model which is complex and

[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.

[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 Multi Modal Annotation: a language for the representation of semantic input in speech and multi-modal systems-- http://www.w3.org/TR/emma/.

unknown to the majority of users. Help desk applications, for instance, fall in this class. For example, a directed dialog system for routing callers to the appropriate computer support may prompt user with: *Is your problem related to hardware, software, or networking?* But users, most likely, would not know which of the three categories would apply. A solution would be providing a menu that includes all possible problems, but it would be too large to enumerate, and building a grammar that captures all the possible expressions that can be used to describe all the possible problems is impractical. In other words, the underlying domain model is largely unknown or vague at best with respect to users. 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 the predefined categories. 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 just a simplified 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, the dialog can still be structured in a directed dialog manner, because the output of the interaction is going to be one of a predefined number of categories.

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 dialog systems. We call this the *VUI-completeness* principle: the behavior of an application needs to be completely specified with respect to every possible situation that may arise during the interaction. No unpredictable user input should ever lead to unforeseeable behavior. Only two outcomes are acceptable, the user task is completed, or a fallback strategy is activated (e.g. escape to operator or an explicit failure statement is expressed).

In order to ensure that an application is *VUI-complete,* its behavior needs to be specified for each possible situation, or class of situations. Today, a complete VUI specification is standard procedure in commercial deployments and it is generally represented by a graph that describes all the possible dialog states, complemented by tables that describe al the details of each state. Transitions between dialog states are described with conditions based on the user inputs and other pieces of information (e.g. previous user inputs, backend response, personal user information, etc.). 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 subsequently implement the application using the platform of choice. 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 model available to the developer. This is the reason why, most of commercial dialog managers, follow the same abstraction utilized in the VUI specification.

2.1 Control and Expressiveness

In order to allow developers to implement detailed VUI specifications, the programming paradigm adopted by the dialog manager or authoring tools should allow a fine <u>control</u> of the system behavior. However, a too low-level development paradigm my result in prohibitive development costs for large and complex applications. Hence the programming paradigm needs also to be <u>expressive</u> enough to allow implementing complex behavior in a simple and cost effective way. These two features are often competing, since in order to guarantee more expressiveness the dialog manager has to allow for sophisticated built-in behavior, which may be hard to bypass if one wants to attain a detailed control of the interface. An effective dialog manager is thus the result of a trade-off between control and expressiveness. This can be summarized by the following principle: *simple things should be easy, complex things should be possible*.

3 Dialog Management

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

There is not an agreed upon definition of what a dialog manager is; different systems described in the literature attribute different functions to it. Some of these functions are, for instance: integrating new user input, resolving ambiguities, confirming and clarifying the current interpretation, managing contextual information, communicating with the backend, managing speech recognition grammars, generating system outputs, etc. In fact, the minimal functionality required by a dialog 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 behavior.

4 Reference Architectures

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

Figure 1 shows a general functional architecture of a dialog system, mostly used in research prototypes. Input speech is collected via a telephone[10] 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 dialog manager. The dialog 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 *dialog strategy*. The most obvious action performed by the system as a response to a user utterance is a system utterance, or prompt, which can be generated as text and transformed into speech by a *text-to-speech* engine, or selected from a set of pre-recoded samples[11]. Other types of action performed by the dialog manager include interactions with the backend system, or any other type of processing required by the application.

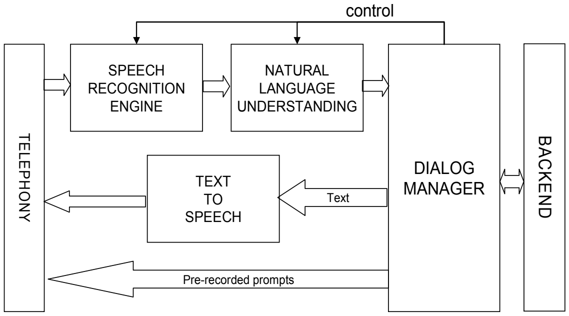

Figure 1: Functional architecture of a dialog system mostly used in research prototypes.

The above described architecture has been implemented in many different forms in research. Of particular interest is the Galaxy architecture (Seneff et al., 1999) which was used in the DARPA Communicator[12] project and allowed 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 dialog state or context. Thus in order to improve system performance, the dialog manager can change the parameters of the language model and language understanding depending on the current dialog context. Several systems did implement this feedback loop with resulting improved performance (Xu and Rudniky, 2000).

Commercial system architectures evolved in a different way. The basic assumption on which most of the commercial deployed systems were based, and still are, is that properly designed prompts can effectively control the space of user expressions. If that's true, at each turn, there is no need for the system to be able to understand all the possible expressions that users could say. Users are in fact *directed* (thus the term *directed dialog*) and enticed into speaking exactly what the system expects. It is clear how this assumption, if true, can potentially allow the attainment of very high task completion rates. Under this assumption, commercial dialog systems provide the speech recognizer with an appropriately designed grammar at each turn of the interaction. Each grammar—typically a SRGS standard 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 dialog system designers use more specific prompts such as *Are you interested in arrivals or departures?* or *From which city is the flight departing?*

The benefit of using restricted grammars in directed dialog 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 recognizer to produce

[10] We refer here to telephone-based systems. However, the concepts expressed in this paper can be generalized to other types of system that do not make use of telephone communication, such as embedded systems for mobile devices and for automobiles.

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

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

erroneous interpretations, or for the user to speak utterances outside the domain. Thus in case of poor recognition scores, commercial dialog systems *direct* users to correct a presumably erroneous interpretation 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 a no at this point in dialog (misrecognitions in correction sub-dialogs would lead to enormous user frustration), the grammar after this prompt is restricted to yes/no utterances and a reasonable number of synonyms

Early commercial dialog 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 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 specialized developers.

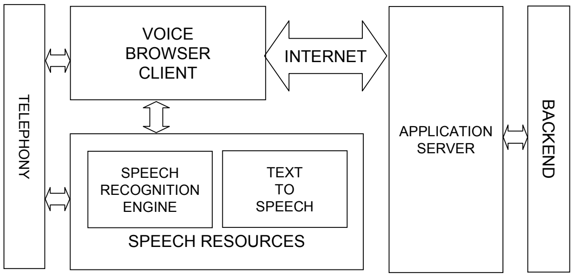

Figure 2. Typical architecture of commercial dialog system.

The core of commercial dialog 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 internet protocol (IP) 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 a particular grammar for the speech recognition engine, a prompt to be synthesized by the text-to-speech system, or an audio recording to be played. Once user's speech has been recognized, and the recognition results returned to the browser in the form of a structured set of variables, the browser sends them back to the to Web server, together with the request of another VoiceXML document. The Web server then replies by sending the requested document to the browser, and the interaction continues in this fashion.

Using plain vanilla VoiceXML, the dialog manager function is actually distributed across the various VoiceXML 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*[13] 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 dialog manager. 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 that may prevent direct access to external resources, such as backend databases. The evolution of server based programming of applications brought the separation of the dialog management functionality from the presentation (i.e. the activation of speech engines, playing of the prompts, etc.), and the realization of general purpose dialog managers and programming models for developing speech applications on the server.

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

5 Programmatic Dialog Management

The simplest form of dialog manager is a generic program implemented in C++ or Java (or as a Java servlet in the case of Web based architectures) implementing

[13] Voice browsers use caching strategies similar to those used by visual Web browser. So, large grammars may be cached on the client and thus avoid large resource provisioning latency.

the application without an underlying generic interaction model. Early commercial dialog 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 their platforms which had proprietary programming environments or proprietary APIs[14].

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.

Dialog modules, developed by various speech recognition and tool providers, constitute one of the first forms of commercial reusable dialog functions. Dialog 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 dialog modules. Thus dialog modules became, for many, the standard approach to directed dialog. Applications were then implemented with the programming model available for the chosen platform. Each state of the dialog flow was associated to a specific dialog module, and the programming model of the platform was the glue used to implement the whole dialog

6 Finite State Control Management

Finite state control dialog manager is an improvement on the programmatic dialog manager. The finite state control dialog manager implements a separation between the logic of directed dialog and its actual specification. The logic is implemented by a finite state machine engine which is application independent and thus reusable. Thus, rather than coding their own finite state machine mechanism, developers had to a description of the finite state machine topology in terms of a graph of nodes and arcs. Often the topology could be derived 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 dialog 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 dialog definition (e.g. dialogs and sub-dialogs). Finally, the engine itself can be harnessed to verify the overall topology, check for obvious design and implementation mistakes, such as unreachable nodes, loops, etc., and provide debugging and logging facilities. More sophisticated engines can have built-in behavior, like for instance handling specific navigation across dialog networks, recording usage information for personalized services, implementing functions such as *back-up* and *repeat*, etc. (Pieraccini et al., 2001).

The simplest form of finite state control dialog manager is built around the concept of *call-flo*w 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 (e.g. Figure 3). By navigating the call flow graph and 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 dialog systems since one has to explicitly enumerate all the possible choices at any node in the dialog.

In fact the pure call-flow model is inadequate to represent even modest levels of mixed initiative, such as over-specification, i.e. more than on piece of information in a single utterance. For instance, if asked for the date of a flight[15] in a mixed initiative system that allows for over-specified requests, users may instead respond

[14] Some platforms used GUI application development environments that were originally designed for touch-tone (DTMF) applications, and then extended for handling speech recognition and TTS. Others allowed access to the functionality of the IVR and the speech recognition/TTS engines through a published proprietary API, that could be used in C, Java, Visual Basic, etc.,

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

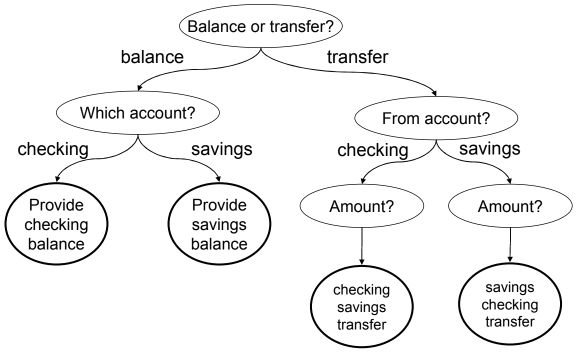

Figure 3. Example of call flow.

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.

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 dialog manager (we will refer to it as the *functional model*) has enough expressive power to represent sophisticated directed dialog and mixed initiative interactions. A full functional model of dialog management can also allow for recursion, i.e. full dialogs specified in a functional fashion can be, themselves, used as actions and associated to nodes of a higher level dialog, 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 (it will be explained later in this paper). More detailed descriptions of functional models of dialog management can be found in (Pieraccini et al, 1997; Pieraccini et al., 2001).

There are common misconceptions about the effective expressive and computational power of the finite state dialog model. In fact it is often attributed limited capabilities with respect to more sophisticated abstractions. This misconception derives from the confusion between a simple call flow model, which is completely described by a state machine with prompts on the nodes and choices on the arcs, and the richer functional model described above. In its simpler form the call flow model is indeed, computationally, a finite state model of dialog: i.e. the state of the dialog is univocally determined by the node of the call flow. In contrast, the functional model allows arbitrary functions at each node to manipulate arbitrary memory structures that can be shared across nodes. Thus the extended functional model is not, computationally, a *finite state model* of dialog; it just makes use of a *finite state representation* for the dialog control mechanism. In fact each *node* of the finite state machine describing the dialog control does not represent univocally the *state* of the dialog, because we need also to take in consideration the *state* of all the memory structures associated with the controller (e.g. the session state). A functional dialog 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 dialog is analogous to the definition of a function. However, a functional dialog manager specification is much easier to author and debug than a set of nested conditional or *case* statements[16].

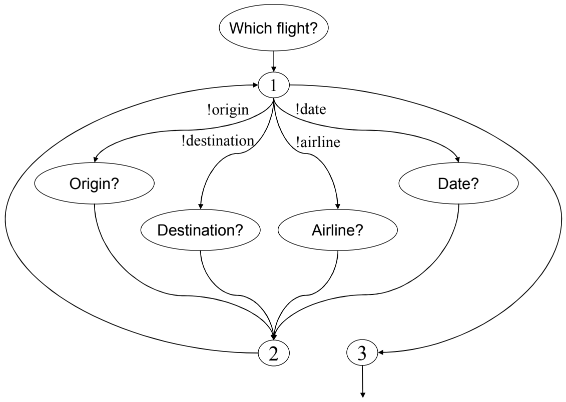

Figure 4: Graph representing a functional dialog controller implementing a FIA topology. The conditions on the arcs exiting a node are verified in the left-to-right fashion. Arcs without conditions are *else* arcs.

6.1 Handling Mixed Initiative in Functional Models

A clear limitation of functional models is in 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 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 dialog applications within the functional model paradigm consists in providing the controller engine with a behavior that corresponds to complex topologies,

[16] 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.

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 dialog. Powerful engines for functional dialog 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 dialog 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 dialog management with a fixed topology. The topology of the FIA controller is in fact shown by the example in Figure 4. The FIA topology is particularly suited for handling over-specified requests, allowing filling forms with multiple-field forms in any order. For instance, if after the initial question *Which flight?* the user specifies the destination and the airline, the arc `!origin` 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` is true. After the user has provided all the required pieces of information (origin, destination, airline, and date) the sub-dialog exits through node 3.

Another example of functional model with a fixed topology controller is the MIT dialog management system (Seneff and Polifroni, 2000). In this case the control is defined by a sequence of functions that are activated when the conditions associated to them fire. Each function can modify a session state (i.e. a *frame* 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 dialog does not require the description of the control graph, which has the functional form described by Figure 5, but the specification of the functions associated to the nodes, and the conditions. The following is an example of a set of rules that implement the same sub-dialog as the one in Figure 4.

```
!origin        → prompt_origin()
!destination   → prompt_destination()
!airline       → prompt_airline()
!date          → prompt_date()
```

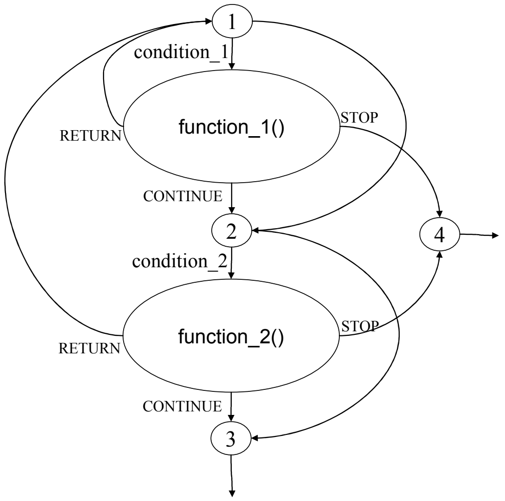

Figure 5. Functional control graph representing a rule based system

7 Inference Based Dialog Managers

We have shown in the previous section how several forms of dialog manager can be reduced to a unique underlying model: the functional finite-state dialog controller. The difference between them is in whether developers are allowed to change the topology of the controller, and in the type of authoring (e.g. graph or rules). However, there are classes of applications for which a specification through a finite state controller may result impractical. As we discussed earlier, transactional applications with a well defined goal (e.g. giving information to the user, selling or buying, etc.) can often be effectively implemented with a finite state controller. On the contrary, applications of the problem solving type (Allen et al., 2000) require a higher degree of planning, for which the finite state controller can be quite inappropriate. These types of applications, as of today, have not yet found a channel to the market of spoken dialog systems, partially because they are not yet at a level to demonstrate commercially viability. In fact their deployment still requires specialized development teams and is thus quite expensive. Moreover the performance of the resulting systems is not yet at the level required for a commercial exploitation.

In spite of its difficulty, the research community has been actively pushing the technology towards the solution of the dialog management problem for complex systems, especially under the auspices of the DARPA Communicator program. Successful prototypes have been demonstrated and tested based on sophisticated dialog 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 to 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 sub-goals. Thus, in order to develop an application, 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 dialog 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 conditions for their execution, are associated to individual goals. Internal nodes represent conditional controls on the execution of the underlying nodes. A dialog manager based on task ontology and a hierarchy of nodes is described in (Pellom et al., 2000). The dialog manager described in (Wei and Rudnicky, 2000) constructs a dynamic structure, called *agenda*, which is practically a list of sub-goals, where each sub-goal 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 the task to accomplish at any point in time. The product tree is dynamically created since the nature of the task may be dynamic as well (e.g. the number of legs in a flight is determined during the interaction and not known beforehand). In the form based dialog manager described in (Papineni, 1999) the inference mechanism is driven by a numerical function computed on a set of partially completed forms (i.e. sets of task-relevant slots), based on how close each individual hypothesized form is to the goal (i.e. the retrieval of information from the database)[17].

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

It is not yet clear whether any of the sophisticated inference dialog managers developed in research could be effectively used for mass production of commercial systems. One of the problem is that their behavior 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 dialog manager itself. Thus, in order to succeed in the commercial arena, inference engines have to produce systems with usability comparable or superior to that of an equivalent directed dialog for the same task, or provide services (e.g. problem solving applications) that cannot be provided with directed dialog, still with usability as the main goal. VUI completeness is an essential requirement which should be seriously taken into proper consideration for the more sophisticated dialog manager models.

8 Current Industrial Trends

Reusable components (Huerta et al, 2005) and prepackaged applications are the main trends of the industry of spoken dialog 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.

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 dialog 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 dialog 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 level of interaction while using an effective development

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

framework and implementing a “controllable” (VUI complete) interface is possible.

We have shown that most commercial dialog management abstractions fall into the functional finite-state controller mechanism, as well as some of the dialog 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 dialog managers, inference based, which is devoted to handle more complex interactions, such as problem solving applications. VUI-completeness is 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 dialog mechanisms into templates, components, and subroutines that abstract behaviors. 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 behavior, but at the same time should allow for more complex interactions to be triggered if and when these dialog mechanisms would benefit the interaction (e.g., power users).

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 *dialog* needs to start.

References

Allen, J. F., Ferguson, G., Stent, A., 2000. Dialog systems: From theory to practice in TRAINS-96. in Dale R., Moisl H., Somers H. eds., *Handbook of Natural Language Processing*. Marcel Dekker, New York, 347-376.

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

Carpenter, B., Caskey, S, Dayanidhi, K., Drouin, C., Pieraccini, R., 2002. “A Portable, Server-Side Dialog Framework for VoiceXML,” *Proc of ICSLP 2002*, Denver (CO), September 2002.

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

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

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

Grice, H. P., 1975. “Logic and Conversation,” in Cole P. and Morgan J. L., eds, *Speech Acts*, New York, Academic Press, 41-58.

Huerta, J., Akolkar, R., Faruquie, T., Kankar, P., Rajput, Raman, T. V., Udupa, R., Verma, A., 2005., “Reusable Dialog Component Framework for Rapid Voice Application Development,” 8th International SIGSOFT Symposium on Component-based Software Engineering (CBSE 2005)

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

McTear M., 2004. “Spoken Language Technology,” Springer, 2004.

Meng, H.M., Wai, C., Pieraccini, R., “The Use of Belief Networks for Mixed-Initiative Dialog Modeling,” IEEE Transactions on Speech and Audio Processing, Vol. 11, N. 6, pp. 757-773, November 2003.

Oviatt, S. L., 1995. “Predicting spoken disfluencies during human-computer interaction.” *Computer Speech and Language, 1995, 9:19--35.*

Papineni, K., Roukos, S., Ward, R., 1999. “Free-flow Dialog Management Using Forms, “ *Proc. of Eurospeech*, 1999

Pellom, B., Ward, W., Pradhan, S., 2000. “The CU Communicator: an Architecture for Dialog Systems,” *ICSLP 2000.*

Pieraccini, R., Levin, E. and Eckert, W., 1997. “AMICA: the AT&T Mixed Initiative Conversational Architecture,” *Proc. of Eurospeech 97*, Rhodes, Greece, Sept. 1997.

Pieraccini, R., Caskey, S., Dayanidhi, K., Carpenter, B., Phillips, M., 2001. “ETUDE, a Recursive Dialog Manager with Embedded User Interface Patterns,” *Proc. of ASRU01 – IEEE Workshop,* Madonna di Campiglio, Italy, Dec. 2001.

Seneff, S., Lau, R., Polifroni, J, 1999. “Organization, Communication, And Control In The Galaxy-Ii Conversational System, “ Eurospeech 1999.

Seneff S., and Polifroni, J., 2000. "Dialogue management in the MERCURY flight reservation system," in *Satellite Dialogue Workshop*, ANLP-NAACL, Seattle, April, 2000.

Stallard, D., 2001. “Dialogue Management in the Talk’n Travel System,“ *Proc. of ASRU01 – IEEE Workshop, Madonna di Campiglio, Italy, Dec. 2001.*

Wei, X., Rudnicky, A., 2000. “Task-based management using an agenda,” ANLP/NAACL 2000

Williams J. D., Witt S., M., 2004. “A Comparison of Dialog Strategies for Call Routing.” *International Journal of Speech Technology* (Vol 7, No 1). January 2004, pp 9-24

Xu, W. Rudnicky, A. 2000 “Language modeling for dialog system?” Proceedings of ICSLP 2000 (Beijing, China). Paper B1-06