A Talking Elevator: The Dialog Model

Jennifer Moore

Project Seminar
Wintersemester 2006
Universität des Saarlandes

Table of Contents

Chapter I

Introduction

As technology becomes more ubiquitous in our world, so the desire to provide more natural, intuitive interactions between man and machine mounts. Yet often, the intermediary between man and machine is an artificial mapping from key interfaces to procedural commands, a legacy of earlier attempts at automation. These mappings, such as the buttons in an elevator, limit the potential value of a system, placing an arbitrary bound on how the user can communicate his or her goals.

When a user steps into an elevator, the goal is not, for the most part, to go to a new floor level. The goal of the user is, quite often, to go to a particular destination on that floor level, whether it is a colleague's office, a meeting in the conference room, or the lab. Buttons for floor levels serve to provide a simplified mapping between the user's goal and an elevator command. Dialog systems, however, offer the means to rise above such considerations.

In the context of a spoken dialog system for an elevator application, our goal is two-fold: eliminate artificial button or key interfaces in favor of more intuitive forms of communication, namely spoken natural language; and secondly, develop a system that can intelligently handle user demands over perfunctory requests.

In the first case, it is clear that button-to-floor mappings are a straightforward approach for a functional elevator experience; this system, however, will provide the means to get to a floor level through multiple sources, whether it is the name of a colleague, a room, or an office number.

In the second case, button interfaces are useful for those who already know the mapping between their destination and the floor level in the building. In other words, guests in the building may need to look up on a table index (if one exists) to find the floor information. An intelligent dialog-driven system, however, could capitalize on this data to facilitate the flow of that information among its user-base. The elevator could know, for example, that experiments take place in the Eye-tracking lab, and that the lab is located on the fourth floor, down the hall to the right. In this way, it might also infer that anyone wishing to participate in the latest experiment needs to go to the fourth floor.

I present a spoken dialog system that enables such interactions in an elevator. Designed to operate as an intelligent receptionist, it is able to communicate in three different languages, recognize familiar speakers, interact with users through spoken dialog, and importantly, aid the user in arriving at his or her desired destination.

1 Background

The original Talking Elevator project was developed in previous years as a way to showcase the various language technologies being pursued by researchers in the Computational Linguistics department at Saarland University. The system was installed in the elevator of the department's main building and was in use for several weeks. It featured voice recognition and speech synthesis in German, and a very simple dialog model which could ask the user for a floor level and execute the user's request.

Major problems with the old system were often due to poor speech recognition quality. Many times it would recognize its own synthesized output as user input and act accordingly. So too would it often mistake user input of floor numbers as the 'basement', or would return to the initial 'which floor' prompt when the user replied in the negative. Moreover, clarification dialog was non-existent; if the elevator could not understand a request, it simply errored out with an apology and returned to the initial prompt.

In 2006, another attempt was made at building a usable, full-featured system which would be installed in the elevator of the newly completed research building located behind the Computational Linguistics department. This report describes a part of the work done on that system, namely an expanded version of a dialog model that aims to address some of the problems mentioned above, fulfill the required objectives of the system, and satisfy the goals introduced at the start of this report.

2 Goals of the Project

The objective of the Talking Elevator project was not simply to re-implement the system used in the elevator of the old building, but to enhance and improve it. Therefore, three features were strictly required for the final version of the system, and are as follows:

1. A system that would interact with the elevator to exectute floor requests.
2. A system that could speak in and understand German, French, and English.
3. A system that could recognize individual speakers by their voice.

The first can be seen as the basic requirement of any such system: functionality. The second arose from the desire to embrace the international flavor of the department, and in a sense, address the needs of its community. The third is a novelty which would set the stage for future interesting and useful enhancements, but remains limited to basic functions for the first version of our system.

In order to accomplish these goals, our task was to make use of various existing components and modify them for the application at hand. The main components of the system included speech recognition, speaker recognition, speech synthesis, database access, elevator hardware access, and a dialog model to integrate all components and handle user interactions.

Two teams were involved in building this system. I was involved in both teams, but the bulk of my contribution to the project consisted chiefly in extending and improving the dialog model initially developed by students from the first team. This report is devoted to the work that went into developing an expanded version of the initial dialog model. Although this version was not used in the final system due to time constraints, the ideas and design of the model provide the foundation for future improvements to the current implementation.

3 Outlook

This report is outlined as follows: the first chapter introduces the specific problem of modeling dialogs and the challenges that arise given our project-specific resources and constraints; the second chapter identifies the conceptual design of the model with sketches of user scenarios to illustrate specific functionality; chapter three describes the actual implementation of the model; a conclusion is given at the end.

Chapter II

Modeling Dialogs

Natural language dialogs are most often modeled as finite state automata. This is because the back-and-forth exchange of a dialog is easily characterized as a path through various user input and system output 'states', in which decisions about the next state are based on input from the current state. While this approach has the advantage of allowing, from an architectural point of view, fairly intuitive dialog design, it also introduces unique challenges in terms of flexibility in the overall management of a system.

This chapter aims to introduce the challenges and constraints of building a dialog model given our resources, as well as describe the architecture of the overall system.

1 Resources

We made use of software specially designed for the design and management of dialog systems. DialogOS is an application that introduces a graphical user interface for developing dialogs as a finite state machine. Each state in a dialog is visually displayed as a node in a graph, with edges spanning from node to node representing the possible paths along which to follow. The states in the model roughly correspond to user input and system output, and a dialog model is constructed as a path through various (possibly branching) input and output states.

In DialogOS, input and output nodes are not limited to handling speech between the user and the system, but can take in data from any external source, such as query results from a database, the current floor level from the elevator hardware, and so on, as well as send out data in the form of calls to an external client, such as a request to go to another floor. This effectively transforms the dialog model into the hub of the entire system, as it functions as the global controller for all external modules and components.

Our dialog system makes use of several kinds of components, including speech and speaker recognition, as well as speech synthesis. Each component is managed from within DialogOS via the use of input/output nodes in our graph.

2 Architecture of the System

The basic archictecture of the system includes various external components in addition to the dialog model. Speech and speaker recognition is provided by Nuance software, and speech synthesis in German, French, and English is provided by AT&T Natural Voices. In addition, interaction with the elevator hardware is accomplished using a serial port client, while dynamic textual data for speech output and other data is handled using mySQL databases.

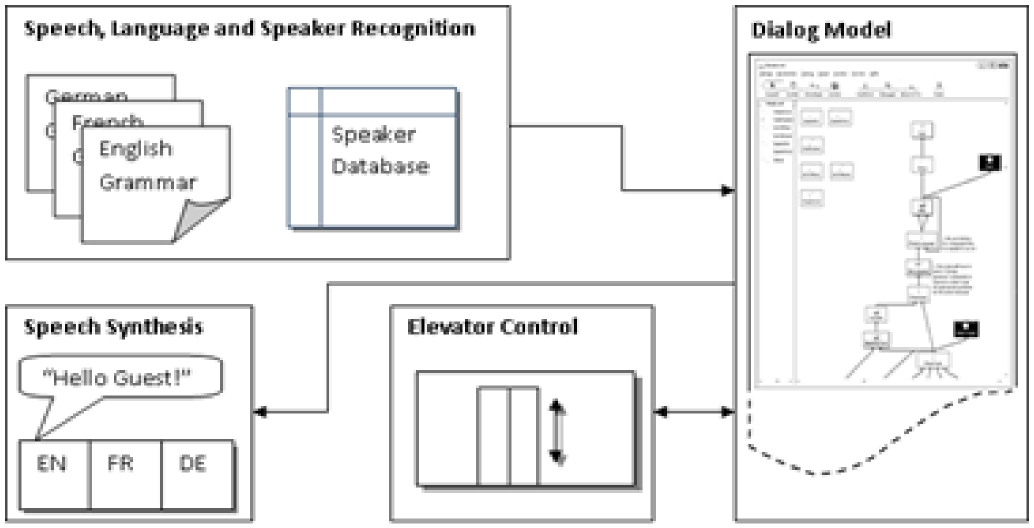

Figure 1: System Architecture

The graph above shows the interaction between the dialog model and the different external components. The model takes input from the database and the speech recognition component, sends data to the speech synthesis component, and both sends data to and receives data from the elevator client.

3 Challenges

Although finite state machines lend themselves nicely to the representation of natural language dialogs, there are nevertheless certain challenges to face in terms of implementation and deployment of a dialog system. These challenges arise chiefly from the need to handle multiple kinds of input from multiple kinds of sources at multiple stages in a dialog.

In the first place, some aspects of the system may require a multi-threaded approach, such as handling background processes which may only affect the current dialog state once they have met a certain criteria or passed a certain threshold. These issues are specifically encountered while handling multiple user dialogs in the elevator, or handling different kinds of elevator requests, such as a spoken request versus a button interface request. Finite state automata, however, are inherently procedural, making it near impossible to handle such processes without a specific multi-threading mechanism.

Secondly, the procedural nature of the dialog model demands that conversations be well-planned and strictly executed. In other words, the dialog must follow particular scenarios in which user input and system output is more or less fixed. Although we make use of databases to add variation to spoken system output, the output nevertheless must conform to the exact meaning as required by the current dialog state. For example, a given output state may be the confirmation of a user request, as in "Did you say level four?" while variations of this may be "Level four, you said?" or "Level four, did you say?" with a rising intonation towards the end. This representation presumes, however, that the user will address the system with pat phrases, such as "Take me to level four, please" or "I'd like to go to level four". In real life, users may inadvertently hop from one conversational context to another, leaving the system stranded in a prescribed path.

4 Discussion

The design and implementation of the dialog model attempts to address some of the issues presented above, namely by introducing situation-specific graphs and multi-threading simulation, which will be explored in the following chapters.

Chapter III

Dialog Design

Once the traditional button interface, with the familiar mapping between buttons and floor levels, is removed from the equation, more complex interactions with the user are suddenly possible. The user is no longer limited in requesting the floor level as the desired destination. By making use of built-in knowledge, the system can find relations between floor levels, office numbers, offices belonging to specific individuals, and room names, allowing the user to make more precise requests. Each new possibility, however, introduces a new scenario which the dialog model must support.

The dialog model was initially designed to reflect basic scenarios for requesting to go to a new floor level and clarification routines. Later, speaker recognition, another basic requirement of the system, imposed scenarios for recognizing a known user and taking him to a familiar destination, such as his office. To support each scenario, the model consisted of a single top-level graph outlining the central flow of each scenario, with minor sub-graphs representing re-occurring procedures. Different 'tasks' were considered, such as going to a new floor level, waiting for user input, or clarifying a user request, and each task was handled through a single state.

This initial approach, however, became ever more inflexible as additional scenarios were considered. Indeed, once speaker recognition was factored in, our single-level graph suddenly became quite complex, prompting the need for a re-design of the fundamental model.

To achieve more flexibility, each scenario was then clustered together as a single unit, representing a conversational 'situation' which can be modified internally without affecting the rest of the model. Rather than relying on a single node to handle all types of tasks, from floor requests to clarification, each task is handled as a subgraph with its own set of internal variables and situation-specific dialog, and global variables were removed. The top-level graph is then transformed into a higher-level navigational model between between various situations, allowing new situations to be easily implemented or disabled.

This chapter presents some of the scenarios included in the initial model, describing the subsequent refactoring of these scenarios as conversational situations.

1 Basic Scenarios

Although there are many ways in which a user might request to be taken to a destination, the most basic and familiar request is that in which the user asks to go to a particular floor level. To handle this type of situation, we considered the following scenarios for requesting a floor level, handling misunderstandings, and addressing recognized speakers.

Request: Floor

This basic scenario illustrates several fundamental decisions with regards to the design of the dialog model.

- ➢ "Hello, elevator."
- ➢ "Hi, where can I take you today?"
- ➢ "Level four, please."
- ➢ "Alright, level four."

In the first place, dialog initiation is prompt-based, meaning that the user must address the elevator system with a familiar greeting of the language. Supported prompts are language-specific and include "Hi, elevator", "Salut, ascenseur", and "Hallo, Aufzug". This decision was strictly based on the limitations of the software. This allows the system to perform a quick language check before continuing the dialog in the language of address. Presumably, a multi-lingual speech recognition component would allow open-ended dialog initiation, with the user directly requesting to be taken to a new floor.

Secondly, once the user has made a request, the dialog system must in some way confirm the request before it is exectuted. Confirmation and affirmation is an important part of the dialog, not only for reassuring the user that the system is functioning properly, but also to avoid executing a misunderstood request. In many dialog systems, this confirmation process is achieved by directly prompting the user with, "You said level four. Is this correct?" and waiting for an unambiguous yes/no response.

Although this type of direct prompt allows the speech recognition component to receive clear input, it is relatively cumbersome and inefficient for the user. For this reason, the model makes use of implicit confirmation first by subtly repeating what the user has asked, and then waiting for a few moments for the user to contradict the statement. If no contradiction is made during the short pause, the elevator executes the request.

Clarification

When the user terminates the request with a "no" during the confirmation pause, the model then proceeds with a clarification routine to discover the intended request.

- "Hello, elevator."
- "Hi, where can I take you today?"
- "Level two, please."
- "Alright, level three."
- "No, I said level two!"
- "I'm sorry. Level two, you said?"
- "Yes, thanks."

Clarification is necessary to redress the problems of the speech recognition component, which in itself faces acoustic echo problems in the elevator chamber. It is also needed in case the user inadvertently asks for the wrong floor level, and other mistakes of these kinds.

Known Speaker

The final scenario supported by the basic system made use of the ability of the system to recognize familiar speakers. For those users who have enrolled with the elevator system, the elevator can make use of database information to find the user's office location, and support the following kind of scenario:

- "What's up, elevator?"
- "Hello John. Your office, today?"
- "Yes, thank you."

Note that, in the scenario above, the user has an individual prompt with which to address the elevator. This is due to the inadequacies of the speaker recognition component which is trained on a single prompt for all speakers, and unable to distinguish speakers among more than ten possibilities. Therefore, speakers are placed in 'groups' each with a specific prompt. In this way, the speaker recognition component distinguishes a given user among a group of less than ten possibilities based on the given prompt.

2 Conversational Situations

In order to enable our model to quickly and efficiently expand with support for new dialogs, user scenarios are modeled as single, contained units. These units are in fact subgraphs with dialog routines suitable for the given situation. Although this tactic introduces some redundancy, such as with clarification dialogs, it nevertheless allows more natural sounding dialog that was not possible when imposing a one-size-fits-all routine to certain situations. There is always a trade-off in terms of generality versus redundancy; here, naturalness was preferred over efficiency, at least in terms of reusing dialog graphs.

Other than the *Request Floor* scenario described in the previous section, there are at least three other topics which form the basis of a 'conversational situation'. Each situation is entered based on the response of the user to the general request posed at the beginning of a dialog, namely "Where would you like to go?" If the user's response fits none of these topics, the user is presented with a help dialog; otherwise the dialog will pass into one of these situations, along with all important information heretofore supplied by the user.

Each topic is illustrated in the dialog samples below, and represent several of the more immediately obvious destinations based on the people who work in the building in question, and the people who visit.

Request: Room

In this situation, the user asks to visit a particular room in the building, specifically those with special names attached. Examples of rooms include "the Bathroom" (alternatively, "the WC", "the Restroom", designated per language), conference or meeting rooms, such as "the Conference Room" simply, or "the Aquarium" as one room is commonly known.

These names are stored in a database and included in the speech recognition grammar. When a room name is mentioned, the room id is returned as a semantic tag, which is then used to find the floor level of that room in the database.

- ➢ "Hi, where can I take you today?"
- ➢ "Eye Tracking Lab, please."
- ➢ "The Eye Tracking Lab, on level three. Just a moment please."
- ➢ [At level three] "The Eye Tracking Lab is down the hall, to the left. Have a nice day!"

In the example above, brief directions are provided as an aid to unknown users once the requested floor level is reached. These directions are included in the database and can be found using the room id. Future modifications to the dialog model might extend this option to provide more detailed directions should the user request it.

Request: Number

Users may likewise request a room by an office number or room number. This is especially useful, for example, when conference attendees or experiment subjects, who may only be given the room number, enter the building but are unfamiliar with its layout.

- ➢ "Room 3.02, please."
- ➢ "Room 3.02, the Eye Tracking lab. One moment, please. "
- ➢ [At level three] "Room 3.02 is at the end of the hall, on your left."

Here, care is taken in the grammar to ensure decimal numbers are properly identified. Possible variations include "three oh two", "three point oh two", and "three two". Also, room numbers may be paired with the names of its occupants or with a commonly known name, such as "the Hiwis Room". As users may tend to catch familiar names more quickly than spoken numbers, this feature allows indirect clarification between the elevator and the user before proceeding to the new floor level.

Request: Person

Finally, users may request a floor level by the name of an employee with an office in the building. Again, this feature is useful for visitors to the building wishing to see someone in particular, but not knowing where the individual's office is located.

- ➢ "I'm here to see Prof. Müller."
- ➢ "Prof. Müller's office is on the third floor. One moment, please. "
- ➢ [At level three] "Prof. Müller's office is the second door on the right."

In this scenario, users may refer to an individual by first name, last name, full name, or titled name. This feature is enabled by three factors, including the speech recognition component, the database, and the dialog model. First, the grammar must include all names which are to be recognized. This is achieved dynamically and automatically using a script to extract all names of individuals in the building from the database and inserting them into the grammar in a proper format. In this way, names are easily maintained and updated as the staff in the building changes.

Although it was not implemented in the final version due to time constaints, this feature should also include a disambiguation component for cases when two individuals have the same name. Currently, all names are unique, causing no ambiguity; names belonging to more than one invidual is not handled.

Future enhancements (not presently supported) might include mixed-situation handling. For example, a user who had asked for a particular room number and found the current occupant was not the intended, might clarify the request with a name, as in:

- "Room 3.11 please."
- "Room 3.11, Dr. Burnley and Emily Wilson's office. One moment, please. "
- "No, no! I wanted to see Prof. Müller."
- "Prof. Müller's office is (now) located in room 3.14. Would you like to go there now?"
- "Yes, thank you."

In the above example, the dialog model might make a quick check for news items such as office changes and include certain time expressions like "now", thus indirectly providing information updates for the user.

3 Discussion

The dialog design presented in this section is an attempt to model human-elevator interactions more fully, naturally, and efficiently. Fully, in the sense that it models interactions as if another human were installed in the elevator by abstracting the goal (floor level) into related objectives (someone's office, a special room, etc.) that can be understood by the system. Naturally, in the sense that one-size-fits-all dialog routines are avoided in favor of situation-specific conversations. Efficiently, in the sense that future enhancements can be quickly added without modifying too many existing structures.

While this dialog was not employed in the final version of the system, it nevertheless has promise for a more flexible, maintainable system, which would bring several value-added features to its user base. In the next section, we will see how these features were implemented in this version of the dialog model.

Chapter IV

Implementation

The implementation of the dialog model encompasses more than just user scenarios. The dialog communicates with any external component needed to enable the flow of a scenario, including speech recognition and speech synthesis, as well as databases and the elevator hardware itself. The following section describes implementation details for each of these components, as well as the overall flow of the dialog model from a software point of view.

1 Spoken Input

Spoken input is handled via the Nuance client for speech and speaker recognition. Nuance makes use of a grammar to determine what the user has said and converts it to textual input. The dialog model, however, does not receive the entire textual input. Rather, the Nuance client sends back semantic 'tags' representing the basic meaning of what was said. For example, if the user says, "Take me to level four please" or "The fourth floor please", the Nuance client will return a semantic tag of the type 'floor=4' which the dialog model can more easily handle. Additional semantic tags for request types include 'room', 'num', and 'person', which refer to special room names, specific office numbers, and individual person names with offices in the building. Other semantic tags for 'language', 'speaker', and 'confirm' return the language of input, the recognized speaker, or a yes/no answer from the user, respectively.

Based on the semantic tag returned, the dialog can determine the next action to take, such as which scenario to continue, which floor level to call, or the proper clarification routine to employ in cases of uncertainty or ambiguity.

2 Spoken Output

Spoken output is handled in a three-step process which includes querying a database for the appropriate textual output, replacing variables within the text string, and sending the modified string to a text-to-speech synthesis component for output to the user.

In the first step, the database is queried for an appropriate output string based on the current language and the current topic. Topics are categorized generally as one of 'ask', 'confirm', and 'clarify', and more specifically by 'floor_number', 'room_name', 'person_name' and so on. An example of an output string corresponding to the topic 'clarify_room_name' might be, "I'm sorry. Did you say the conference room?"

Secondly, output strings in the database may contain variables which act as placeholders for other strings. For example, the previous example output string would in fact be stored in the database as "I'm sorry. Did you say <room_name>?" wherein the placeholder for the room name would be replaced with the name of the room in the current dialog context. Moreover, once the dialog model receives an output string, the string is marked up with application-specific xml for voice and language information before being sent to the TTS system. This is especially useful for person names which are spelled identically for all languages, but are pronounced differently in each. In this way, the database stores only the bare output strings, and markup can be easily changed if the current TTS system is replaced.

In the final step, the output string is sent to the TTS component to be output as natural speech, before continuing on to the next dialog state.

3 The Top-Level Flow

The top-level graph of the dialog model represents the higher-level flow between various dialog sub-clusters, such as the decision paths between one conversational situation and another, between a dialog and a new floor level call, and so on.

Setup

The setup node is executed the first time the system is launched and is used for performing various setup tasks such as establishing a connection to the database, intializing variables with default values, and opening any other preliminary connections to external modules. Setup generally only occurs once and is not visited again until the system is rebooted.

Greeting

The initialization of a new dialog begins with an input node which waits indefinitely for new user speech input. Once the user greets the elevator, such as with "Hello, Elevator" or the equivalent in another language, the input is processed by the Nuance client for the utterance language and a recognized speaker.

If the language was improperly identified, the system returns to the 'wait for input' state until the user's input is recognized. If the language is correctly identified, then a call is made to the Nuance client to restart the speech recognition engine using the full grammar of the current language, and a language variable is set in the top-level graph which can be passed into all other subgraphs.

If the speaker was not recognized, then the dialog will pass to a general request floor dialog. If the speaker was recognized by the speaker recognition engine, then the dialog will pass to a user-specific floor request dialog, otherwise it will pass to a more general dialog for unknown users.

General Request

If the user was not recognized by the system from the user's greeting, the dialog will continue into a general request topic in which the system will ask a broad question, such as "Where would you like to go?". In the general request situation, the elevator can expect one of four kinds of answers. These answers include a floor level, a room name, a person name, or an office number, as described in Chapter II.

If the system does not receive a valid response, it will first fall into a clarification routine, or after several unsuccessful attempts, fall into a help routine. If the system receives no input at all from the user (i.e. speech recognition fails to recognize any input), it will first attempt the help routine, or after continued failure to recognize input, it will apologize to the user, reset the dialog, and wait for input from a new user.

Help

The Help routine was designed to help users find their desired destination through a series of questions. These questions are aimed at narrowing the field of possible destinations, until all destinations have been ruled out. An example help dialog would be:

- ➢ "Do you know the floor level of where you want to go?"
- ➢ "No."
- ➢ "Are you looking for a particular room?"
- ➢ "No."
- ➢ "Is there someone you would like to see?"
- ➢ "Yes."
- ➢ "What is the name of the person you would like to see?"
- ➢ "I think his name is Peter Miller.."

In this example, each question represents one of the four main topics as described in the previous chapter. Once a non-negative answer is found to any of these questions, the information from the user can be transfered to a more specific dialog on that topic in order to gain all necessary information for executing a goToFloor call.

Conversely, if no positive answer can be given, the system issues an apology to the user for being unable to help and resets for a new dialog.

Execute Request

Each topic request is designed to handle different kinds of input from the user. For example, users supply the particular office number for an office request, or the name of an individual for a person request. In order to execute a goToFloor call, however, only the integer floor level is needed to perform the task.

Therefore, once a request by the user has been successfully made, the information supplied is used to look up the corresponding floor level in the database, as well as any additional information, such as directions to the particular room. Once this information has been found, the goToFloor method is called.

Farewell

As an aid to the user, once a goToFloor call has been made, the system can also keep track of the current floor level, periodically checking until the elevator reaches the requested floor level. At this point, the elevator will issue a message to the user, stating the current destination, mention any pertinent directions, and finish off with a farewell message to the user. An example of a farewell might be:

- "Peter Miller's office, on the third floor. Peter's office is to the left, next to the Eye Tracking Lab. Have a nice day!"

This feature makes use of the getCurrentFloor method in conjunction with the elevator client which was not available during development. Therefore, although this feature has been implemented in the dialog model, it has not been fully tested.

Multi-User Handling

After the elevator executes a new goToFloor request, it will keep track of the current floor until it reaches the requested floor and can issue a farewell to the user. It is quite possible that a new user will have entered meanwhile, wishing to initiate a dialog. The dialog management system, DialogOS, however, does not currently support multiple processes which would enable the model to handle these two separate events simultaneously. A work-around was thus needed to accomplish this task.

In order to handle a new user dialog, while simultaneously keeping track of the current floor level requested by the previous user, the dialog model makes use of a series of jumps between different states of the current dialog. The model is broken up into mid-sized clusters, with each cluster having a unique label. In this way, once the dialog has passed through a given cluster, it immediately jumps to a process which checks the current floor level with the requested floor level. If the current floor is not the same as the requested floor, it will jump back to the same spot in the main dialog to continue with the next cluster. If, on the other hand, the elevator has reached the requested floor level, the elevator will suspend the dialog with the current user, issue the farewell message to the previous user, removing that user from the queue, before returning to the dialog with the current user.

There are many issues in handling multiple events in this way. The first is the added latency caused by jumping to another process in the middle of a user dialog. The second is the added complexity to the dialog graph when handling more than a single event. There is a balance between the size of the cluster and the number of jumps between processes. If a sub-graph is too big in itself, then the final jump may be made after the user has left the elevator entirely. With smaller clusters, however, more jumps would need to be made within subroutines, effectively violating the rules for structured programs. It is not possible to know how this would fare in actual practice, but it did no harm in single-user offline testing.

4 Discussion

The final version of this dialog model was never tested in the actual elevator, nor was it fully debugged. However, most of the features were simulated and tested to the extent possible without all components available and should be wholly usable after thorough testing of all user scenarios.

Conclusion

An elevator that can speak and understand speech, even speech of many languages, is a novelty that quickly loses its shine for practical purposes. An intelligent elevator, one that can provide information, whether in the form of inference between user-supplied data and user queries, or direct answers to where, when, and how questions, provides a true service on which the user can learn to rely. After all, pushing a button is inherently faster than communicating a floor request, confirmation, clarification, and execution.

Thus, the dialog model presented here was designed to best serve its user-base when practicality is not favored; in other words, this system is most valuable in those moments when the user does not already know the floor level of his or her intended destination. Whether it is the user from another building wishing to visit a colleague, a visitor to the department, or staff returning from a long absence, the user can afford to spend a few moments more in dialog with the elevator so as to arrive at the proper floor level the first time, rather than waste time at multiple floor levels, or even scanning multiple rooms down a hallway, in search of his or her desired destination.

This model achieves value-added service by supporting rich, full-featured dialogs in the elevator system. By modeling various request situation types, the system surpasses the basic functionalities of the elevator button panel, thus providing more possibilities in terms of usability and usefulness.

Finally, although this model was not used in the end, it has nevertheless furnished the developer with many interesting insights into both dialog design and dialog management systems. In terms of design, separating and containing conversations within the model go a long way in preserving maintainability and robustness. In terms of software to build dialogs, this project impressed upon me the need for multiprocess handlers, variable transparency (e.g. better support for non-global variables), and allowing subgraphs to have more of the same benefits as procedures, without having to be repeatedly called. At the least, the motivation for the system is solid, and future revisions can only benefit from all things learned.