Integrated Development and on-the-Fly Simulation of Multimodal Dialogs

Silke Goronzzy, Nicole Beringer
3Soft GmbH, Speech Dialog Systems Group, Erlangen, Germany
\{silke.goronzy, nicole.beringer\}@3soft.de

Abstract

This paper presents a tool that allows the integrated development of multimodal dialog systems. We show how graphical/haptic and speech interface can be designed in an integrated way, which is in contrast to many state-of-the-art HMI design processes. Graphical/haptic and speech interface are often designed more or less independently of each other, which can result in inconsistencies of various kinds. The integrated design results in truly multimodal interfaces, which allow a change in modality at any point. A key feature is the capability to immediately simulate the complete HMI during the specification process, which allows evaluation of all modalities in combination at very early development stages. The tool can be used for rapid prototyping of multimodal dialog systems in the research area, but at the same time fulfills several requirements that are posed to the HMI development in a mass production process, e.g. in the automotive area.

1. Introduction

It is commonly accepted that the use of multiple modalities for human machine interfaces (HMIs) potentially facilitates the handling of complex systems for users. The most frequently used modalities are haptics (resulting from graphical user interfaces (GUIs)) and speech dialog. Also gesture and emotion (either extracted from speech, from facial expressions or even from bio-signals) have proven to be effective in dialog systems like SmartKom [1]. Apart from graphical interfaces that have been extensively used for various kinds of systems, especially speech technology has taken the hurdle from pure research systems to real products. Special solutions for speech recognition that can cope with the adverse environments in e.g. cars have been provided and optimised over the years. The use of microphone arrays, echo cancellation and noise reduction techniques and the training of acoustic models on car data results in satisfying speech recognition performances even at high speeds or with open windows.

Still the problem remains in many products using a speech interface in combination with a GUI, that the GUI is designed and developed rather independently from the speech dialog, not seldomly using different tools. The resulting speech enabled HMIs often do not fully exploit the potential inherent to such multimodal interfaces. The problems range from potentially ‘simple mistakes’ such as speech commands differing from what is displayed on the GUI to more severe inconsistencies such as different system behaviour depending on whether the user used a speech command or navigated via haptics. Also the multimodality is often rather restricted in a way that a dialog once started by speech cannot be continued by haptics. However, there are many situations where it would be beneficial if users could freely choose between the different modalities depending on personal preferences and/or the current situation.

Entering the destination in a navigation application could serve as an example here. After speaking the city name, usually a list of potentially matching cities is displayed on the GUI. Some users might intuitively want to use a built-in rotary knob for confirming the highlighted city name. This kind of interaction, however, is often not possible in current products. Only the selection/confirmation via voice is allowed, if a button is pressed while in a speech dialog, the speech dialog is aborted. Such behaviour is not intuitive and difficult to understand for naive users.

Inconsistencies between GUI and speech dialog are a result of parallel development processes which are independent of each other. In order to develop truly multimodal and consistent HMIs the use of one tool and one data model for all modalities involved is desirable. Using such a common data model allows automatic consistency checks and helps to develop sound interfaces with the modalities really supplementing each other.

Additional requirements to the HMI development that arise from production process are e.g. easy change management of the HMI. It is clear to see that changes in the GUI very often also result in changes in dialog which might be cumbersome, if different specifications and designs need to be maintained.

In order to fulfil all the above mentioned requirements, we developed a tool that allows the joint development of GUI and speech dialog for multimodal HMIs. Its architecture allows further extension if more modalities are to be added. The tool was originally developed as a design tool for GUIs and was recently extended for speech dialog. It can interface to various speech recognition and synthesis engines and can thus be flexibly employed for various applications.

This paper is structured as follows: the next section describes the state of the art of multimodal dialog systems followed by the section about the basic mechanisms of our tool for GUI as well as for speech dialog design together with the general requirements to such a tool. Section 3.2 gives details about how exactly the speech dialog is modelled. In Section 4 one of the key features, namely on-the-fly simulation of the complete HMI is explained. Then, we give details about the evaluation of HMIs using the tool in Section 5. We will talk about true multimodality and interfaces to other dialog engines in Sections 6 and 7 before we talk about future work in Section 8.

2. State of the art

In order to make human-computer interaction more natural, the use of spoken AND graphical/haptic information is desirable and also necessary, when considering the size of existing 3G mobile data devices or mobile usage situations, e.g. visual access to the display should be avoided when driving a car.

In the last years much research was done in the area of multimodal dialog systems. But still, optimum usability of multimodal interfaces is difficult due to many open parameters. One
big problem is to provide all the information flow and dialog status of one modality to the other. It is difficult to administrate parallel logical flows of single modalities without blowing up the manager components in a dialog system (like dialog manager, intention recognition) and to coordinate multimedia output (e.g. speech synthesis and graphical output). There are also other open questions of how to consider multimodal inputs [2]. Are they synchronous or time different? Equivalent in describing the same intention even if not time synchronous? How to process ambiguous inputs? First match, multimodal verification (only the information which is common to all modalities) or additional match? State-of-the-art dialog systems like EM-BASSI [3] or SmartKom [1] develop the single modalities separately. Multimodality is achieved in an integration process by modelling render component types. These systems tend to be rather complex software systems which require more elaborate software designs to assemble heterogeneous components into an integrated and fully operational system [4]. System integration platforms, like MULTIPLATFORM [5] allow the integration of large-scale research prototypes due to an open component architecture which allows message-passing and interconnection of distributed software modules. Unfortunately, they do not offer a possibility to directly optimise the dialog system as a whole. The uni-modal components have to be updated each for itself and it can happen that different aspects of a multimodal usage cannot be realised. Also, in terms of mass-production these platforms are still too complex and slow for updating the dialog system in question.

Niklfeld et al. [6] propose several situation adaptation types, e.g. system-controlled, where the system autonomously recognises problems (e.g. high levels of ambient noise) and monitors the interaction in order to avoid problems. Their adaptation architecture uses mainstream technologies and standards, tests itself about capabilities and limitations and is general enough to scale from first small prototypes to bigger systems which are close to market-readiness and provides the basis for usability research. However, a multimodal integrator is the only part where information from more than one modality is processed. It does not provide an immediate information exchange between the modalities. To provide multimodal access to multimedia databases different modality specific processes can be considered as agents (see MIAMM [7]). Throughout the project a Multimodal Interface Language (MMIL) was developed to provide information aggregation and transportation between the different modules. In [8] a methodology is presented that allows to identify the task and to validate the spoken dialog models. The main issue of this methodology is that it starts to combine the production of the task model with internal and external field tests before the final dialog model is completed.

3. HMI Specification and Development

An HMI specification and design process e.g. for infotainment systems in the automotive domain typically involves graphical layout, the determination of menu logic, the speech dialog, and the administration and maintenance of different languages. The graphical layout determines how the different views of the GUI look like, which kind of widgets are used and how these behave (e.g. how a button looks like if it gets pressed). In order to describe the system behaviour the menu logic or application flow must be determined in a way that the user can later easily navigate in the menus to find the desired functions of the infotainment system. In addition to potential user actions, like selecting mp3 titles or programming the navigation system, also internal system events, like incoming telephone calls, have to be considered. Finally, since HMIs need to be available in multiple languages, all language specific GUI texts and particularly speech dialog prompts, vocabulary, grammars etc. need to be administered in such a way that the change of the HMI language becomes possible at the runtime of the system.

All these different tasks are usually carried out using different tools in different formats resulting in a HMI description that is spread among many different documents. As a result, it becomes particularly difficult to keep the HMI specification consistent.

3.1. A Tool for Generating GUIs

In order to solve this problem a tool that allows the model-based specification and design of HMIs was developed. The core of the system is a data pool in XML format, in which all information that is relevant for the HMI is stored. Whenever HMI related information changes, e.g. because the user selected a tile in the mp3 application or because the navigation application returned a list of matching destination cities, the corresponding information is written to the data pool and the affected HMI components are notified about the changes. The tool uses several editors for the different tasks at hand:

- View Editor: graphical layout design for each view
- Event Editor: Definition of all events that influence the HMI
- State-chart Editor: Definition of the application flow and specification of possible paths between states

Due to the data pool architecture and event mechanism chosen for this tool, a simulation of the HMI becomes immediately possible while designing the interface. This means that the look & feel of the HMI as well as the usability can be evaluated at very early design stages. Another advantage of the model-based specification is that it allows to automatically generate code for the target platform to extents of 60-80%. The tool furthermore allows multiple users to work on the same HMI.

3.2. Integrated Design of GUI and Speech Dialog

If speech dialog is to be added to the HMI, also this part needs to be appropriately specified. We need to describe, what kind of vocal interaction can be understood and how the system reaction should look like.

For the graphics part, consistency is ensured by using one globally visible data pool, that can be referenced by different parts of the GUI using specialised editors. Correspondingly, the speech dialog is integrated in exactly the same way, using yet another editor for specifying the speech dialog and its properties, which are stored in the global data pool. As the basis for the dialog flow the previously defined application flow can be used. Figure 1 shows an example dialog flow for a navigation task. In addition to the GUI events, also speech events are defined. A speech event is executed when a certain command (or corresponding synonyms/ phrases) was recognised. If there is a simple one-to-one mapping between a haptic event and a speech command, the same event can be used in order to keep the state-chart 'tidy'. Usually a separate view, displaying the information relevant to the speech dialog only, is used. How the

---

1 A state can be linked with a view to be displayed and the transitions between the states/views are triggered by events. The used state-charts are UML (Unified Modeling Language) compliant.
speech commands are specified is explained in the following section.

3.3. The Dialog Editor

Even though there is no necessity to design the GUI previous to speech or the other way round, existing designs of one or the other modality can and should be re-used. If for example the GUI is already available, it is important for the speech dialog designer to have access to the generated views, because the look & feel might influence the speech dialog.

Just like a view of the GUI is associated with a state in the state-chart, speech dialog and its properties are attached to states. With a special dialog editor, the allowed user utterances can be defined, meaning the allowed words, together with recognition grammars to define the possible phrases/sentences.

This includes the so called base vocabulary, which can be automatically generated from the corresponding view. This mechanism is supposed to ensure consistency between GUI and dialog such that all commands that are currently displayed on the GUI are allowed words in the speech recogniser’s vocabulary. Of course this base vocabulary can be extended by synonyms which can be manually added to the vocabulary to allow more freedom for the user. Also more complex grammars can be specified.

Dialog building blocks for recurring dialog turns (e.g. entering sequences of numbers, etc.) can be specified and re-used throughout the whole dialog. Special cases of such recurring building blocks are help messages and confirmations.

The dialog flow is generally held independently of the actual wording, so that both can be administered separately. This is important for maintaining several languages for both GUI and speech dialog. That means, when specifying the dialog, concepts like ’DIAL_TEL_NUM’ are used. Which wording can actually be used is defined separately.

The tool allows to specify dynamic vocabularies, to add new street names, music titles or the like. This is necessary to allow users to vocally reference information that is dynamically loaded by the different applications. Of course it depends on the recognition engine that is connected, whether it supports dynamic vocabularies or not.

Commands/phrases/sentences that are defined at parent states in the state-chart are inherited to the children states. Commands defined at children states are only valid at this particular state. Commands inherited from parent states can be deactivated explicitly if e.g. the vocabulary should be restricted to ‘yes/no’ answers in clarification dialogs.

3.4. Dialog Management Tasks

In order to configure the general dialog behaviour or appearance, several mechanisms are provided.

Help messages are standard dialog building blocks, that allow different help messages depending on the dialog history. In order not to overload the dialog state-chart, help sub-dialogs are by default not displayed. Help messages can be activated or deactivated for each state. When activated, corresponding (sub-) states are generated and can be visualised by double-clicking on the help symbol. Similarly, the confirmation strategy can be chosen just by activating the corresponding option. Currently, the modes ’explicit confirmation’, ’implicit confirmation’ and ’no confirmation’ are provided.

In order to ask the user for missing parameters, information can be attached to parent states in the state-chart (cross-checks), specifying what kind of information is obligatory.

Output Prompts can also be specified for each state. It can be distinguished if the transition to the current state was elicited by a haptic user action or by a speech command. It can thus be specified whether a speech prompt is always played when a specific state was entered or only if previously a speech command was used.

Multi-slot utterances/short cuts referring to one application (i.e. utterances containing more than one corresponding user action on the GUI) can be easily executed. Provided the recognition engine is configured appropriately, the recognised words are written to the data pool. For those slots/parameters that are missing for the current application the user can be asked back.

4. Simulation

For the simulation a recognition engine is connected via TCP/IP. Also real hardware control elements can be connected for the simulation via a serial interface.

Just like in the graphics-only case, the complete HMI can now be simulated at any stage in the design process. It is important to mention that the simulation comprises both, the combination of GUI AND speech. As a result, the overall HMI can be experienced at very early stages in the design process.

Figure 2 shows an example for a simulation of a navigation task in an infotainment system. In the background on the left hand side, we can see the state-chart (the same as the one used in Figure 1), with the state the system is currently in being highlighted. On the right hand side we see a view that shows the later screen as used in the car together with a button panel, that simulates the buttons that are actually existing as physical buttons/rotary knob, etc. in the car cockpit. The buttons on the left side of the panel are the so-called hard keys (navigation, radio, tuner, setup), that always - no matter where the user is in the menu/dialog - cause a transition to the corresponding application if being pressed. The right side of the panel shows the soft keys. The function of the soft keys changes, depending on the current state. If one of the buttons is pressed, the appropriate action is directly executed and displayed on the screen. Correspondingly, speech commands can be defined that are always valid commands (reflecting the hard keys) and commands that are only valid in this state. Recognised commands are also executed immediately.
5. Evaluation as Part of the Design Process

The overall evaluation of a prototype is usually conducted at a very late stage of the system development where the design process is already completed and all the applications (such as navigation system, telephone, media player, etc.) are available. Even if usability studies have been carried out for the single components of a multimodal dialog system, e.g. ergonomic design of the GUI and applicability of the spoken dialog system, problems and inconsistencies of the combined modalities cannot be seen before all components (GUI and speech in our case) have been finalised. At this stage it is very difficult to give aid to the developers (see [2]) to optimise the system without redesigning the essential multimodal work flows to a great extent.

With our simulation capabilities it is possible to do the simulation of the overall HMI without the necessity to have the different applications (such as navigation system, telephone, media player, etc.) already available. As a result, problems and inconsistencies between the single modalities can be detected and redesigned at a very early production stage. The single modalities can also be improved with regard to multimodal usage in the development process. And last but not least our simulation model. Also it can be easily observed whether the behaviour of the modalities ‘fits together’.

With this the resulting HMI is truly multimodal in the sense that the user can now freely chose the modality at ANY point in the dialog. It is now no problem to mix speech and haptics input, since all user actions are immediately reflected in the data pool, which is visible for all components.

6. True Multimodality

During the simulation of the complete HMI it is now possible to realise if the modalities process the same tasks in different ways. Deviating references can easily be discovered and adapted to the other modality, since all modalities operate on the same data model. Also it can be easily observed whether the behaviour of the modalities ‘fits together’.

With this the resulting HMI is truly multimodal in the sense that the user can now freely chose the modality at ANY point in the dialog. It is now no problem to mix speech and haptics input, since all user actions are immediately reflected in the data pool, which is visible for all components.

7. Interfacing to Other Dialog Engines

After all components of the HMI have been fully specified and evaluated, it has to be ported to the target platform which is used in the later product. Depending on the actual requirements different recognition, synthesis and dialog engine might be chosen. Our tool is supposed to be independent from what applications, dialog and speech target engines are integrated. In order to do so, it is important that the data necessary for specifying speech dialogs can be extracted from the data pool to interface to the corresponding engines. We provide a generic interface, from which recognition grammars, dialog flow, vocabulary, and recogniser configuration can be converted.

8. Future Work

In order to be able to develop multimodal interfaces with really natural language capabilities some extensions are necessary. Vocal shortcuts which allow not only to give several commands referring to one application as is currently possible, but to several applications, are a feature that helps to fully exploit the potential of speech enabled HMIs. Of course for really natural dialog recognition engines that can cope with correspondingly large vocabularies and artifacts occurring in natural speech are a prerequisite. Another often requested feature is the possibility to also specify speech dialogs that are independent from the GUI so that if desired different applications can be controlled by different modalities at the same time. This requires the use of multiple, parallel state-charts, that can be synchronised at different points.

9. Conclusion

In this paper we presented a tool for the integrated design of multimodal dialog systems. In contrast to parallel design processes, graphical/haptic interfaces can be easily combined with speech dialog and potentially other modalities. By using one globally visible data pool that contains all information that is relevant to the HMI, we can ensure both, consistency and true multimodality. The simulation capability allows to experience the whole HMI including speech at very early design stages, which allows to conduct usability studies at a time in the development process where the results can still be integrated into the HMI. For interfacing to dialog, recognition and synthesis engines we provide a generic interface.

10. References