

Visualization of Spoken Dialogue Systems for Demonstration, Debugging and Tutoring

Jaakko Hakulinen, Markku Turunen and Esa-Pekka Salonen

Speech-based and Pervasive Interaction Group, TAUCHI, Department of Computer Sciences
University Tampere, Tampere, Finland
{Jaakko.Hakulinen, Markku.Turunen, Esa-Pekka.Salonen}@cs.uta.fi

Abstract

Graphical elements have been found very useful when spoken dialogue systems are developed and demonstrated. However, most of the spoken dialogue systems are designed for speech-only interaction and are very hard to extend to contain graphical elements. We introduce a general model to visualize speech interfaces. Based on the model we present an implemented visualization framework, and several example visualizations for demonstrations, debugging, and interactive tutoring of speech applications.

1. Introduction

Speech interfaces are efficient in many situations. They are available via telephone when other interfaces are not possible and suit well in situations where users' hands or eyes are busy. Furthermore, speech can be very efficient in certain tasks like selecting objects by multiple attributes. When graphics are available, it is best to combine benefits of speech and graphics in a multimodal interface.

There are some situations where traditional spoken dialogue system and graphics can work efficiently together. We present cases where visualizations are connected to telephone based applications. Graphics are not normally available when these applications are used. However, there are certain situations where graphics can be very useful. When users learn to use new systems, they are often presented with manuals that are graphical in nature, e.g., web pages. Visualizations can also be used when systems are demonstrated, for example, in educational purposes. Finally, developers can work more efficiently if they are provided with sufficient tools to visualize the internals of the system they are developing.

In this paper we present a general model to visualize existing speech-based applications. Based on the model, we have implemented a concrete visualization framework. Using the framework we have visualized several existing applications without any modification on their program code. The requirements for the visualization are discussed and our implementation is analyzed respectively.

We also present examples of the visualizations including a speech balloon visualization of spoken interaction for demonstration purposes and interactive software tutors that carefully guide new users when they learn to use a spoken dialogue system.

Next we study briefly the background and context of this work and consider the idea of visualizing spoken interaction. Technical details and experiences with the various uses of the visualizations follow. The paper ends with conclusions.

2. Visualization of spoken human-computer interaction

Various visualizations of speech and spoken communication have been developed. Speech signals, as well as entire dialogues [1, 2], have been visualized for research development and educational purposes. Many annotation tools also have visualizations to help analysis of dialogue annotations [3]. Finally, dialogue models have visual representations in graphical tools that are used to design and implement spoken dialogue systems, such as CSLU Toolkit [4].

The available visualizations are mostly aimed for developers and researcher. On the other hand, graphics are used in multimodal systems in many ways. The visual component can help users to better follow the interaction [5] and a mental model that the visual component can provide can transfer to speech only interaction [6].

Real time visualization of spoken dialogue systems aimed for end users have not been widely used so far. However, experiences from multimodal systems and static tutorials [7] suggest that they can be very effective.

Spoken interaction between a dialogue system and a user can be visualized in various ways using multiple dimensions. For example, the visualization can be static, dynamic or animated. Focus of visualization can be on the conveyed information, systems internal representation, turn taking, prosody etc. In this work we concentrate on dynamic visualization and discuss visualization of spoken utterances and interaction style. Speech is essentially temporal and serial in its nature and to convey the flow of the interaction to the users, the visualization must be real time. The nature of interaction easily changes if the interaction slows down significantly because of the visualization.

Target groups for visualizations include both developers and users. Developers are familiar with the nature of the interaction, and are interested in what the systems internal status is. They often need detailed technical visualizations about results of speech recognition and natural language understanding. Users, on the other hand, can benefit from the visualization when they familiarize themselves with a new system. They need to learn the interaction style, e.g., how turn-taking occurs in the dialogue. They also need to learn the system functionality and what kind of inputs the system understands. The dialogue structure can also be made more explicit when users can, e.g., see dialogue history.

As always with speech interfaces, error management is important. Users must learn to identify situations caused by speech recognition errors. When users see the results of speech recognition, they can notice the recognition errors immediately.

3. General visualization framework

A visualization framework for speech-based applications must fill certain requirements. Preferably it should be general so that there is no need to independently integrate the visualization to each new application. At the same time, its architecture should make it possible to add application specific visualizations as necessary and adapt the visualization to different needs.

The visualization and especially the connection between visualization and an application must be efficient. The visualization might become misleading, or even useless, if it slows down the interaction. We can avoid some of the efficiency bottlenecks by using an architecture that enables distribution on different computers. Distribution makes it also possible to visualize applications from different physical locations. For example, the interaction with a telephone based application could be visualized outside the premises where the application server is located. Preferably the architecture should also be robust in error situations. In particular, problems in the visualization should not have an effect on the application.

3.1. Visualization framework architecture

Our generic model for visualization framework consists of three components as seen in Figure 1. The visualized application has one or more connection components, which provide access to the internals of the system. The visualization itself is running as a standalone component. These two are connected via a “middleman” component, which is a server for both the visualization and the application acting like a proxy between them. The reason for a separate component in the middle is that this way neither of the other components needs to be a server. E.g. web applets, which are potential visualization containers, cannot be servers. The separate communication component makes it also easier to configure firewalls. Furthermore, the middleman component can increase reliability; problems with the visualization do not break down the application as middleman can tell the application to keep on running. There are also some efficiency gains, especially when the application uses a turn based interaction model. The middleman can also simplify the development of the two other components as it can take care of much of the complexity related to timing the interaction.

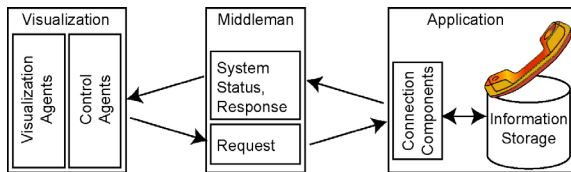


Figure 1: General model for the visualization framework.

The visualized application needs connection components that can provide information from the application. In order to control the systems behavior the connection components need also means to modify internal state of the system. This is important in debugging and some tutoring situations. For example, when tutoring new users removing incorrect inputs is an option. In general, the ability to modify the system state

offers numerous possibilities to make truly interactive visualizations and add-ons to existing applications.

3.1.1. Underlying application architecture

The general model presented can be implemented to many systems. For example, in Galaxy II [8] the connection components could be connected to the hub just like any other component. Our applications are implemented on top of Jaspis architecture [9]. One of the core features of Jaspis is centralized Information Storage. All components have access to the storage and store all information there. This makes it possible to visualize any information wanted without modifying the application. In addition, shared information makes it possible to modify the system state, and in this way control the system behavior.

Jaspis-based systems contain managers and agents, among other components. Managers in Jaspis represent high level task distribution of spoken dialogue system. Each manager has a set of agents that are compact software components to carry out actual tasks, such as dialogue decisions.

3.1.2. Connection components

We have implemented generic connection components that can be included into any Jaspis-based system. These components have access to the Information Storage and all information inside the application is available for the visualization. Connection components can also modify the Information Storage and thus control the system.

The generic connection components are built as managers and they connect to the middleman component when they receive a turn. They can halt the system by not giving the turn away in timely manner. Because of this the connection to the visualization components must be fast, in particular when real time visualization is required. To optimize the speed of the connection, the connection components send the most important information to the middleman by default, such as pending outputs, input results and a short description of application status. This removes the need for separate requests for frequently needed information.

We have also a different version of the connection component that is a standard Jaspis input/output (I/O) component. This component is run concurrently with other I/O components. It also produces I/O results that are processed by Jaspis input processing components. While the component enables more concurrency it requires more complicated configuration. Both types of connection components can be used in parallel to achieve maximum control and speed.

Adding the connection components to existing applications is straightforward. The components are added to the system configuration. The default components work with all Jaspis based dialogue applications. Because the components provide access to the information storage via a standard interface, components in visualization side can access application specific information. To further optimize speed and modify the conveyed data, small application specific connection components can be implemented.

3.1.3. Visualization components

The visualization components we have implemented are Jaspis applications. They have two managers; one runs the actual visualization agents and the other includes control

agents that decide when and what to visualize. Visualizations agents run concurrently with the rest of the system. This enables, for example, animations while the application is processing. A callback routine provides feedback from the visualizations agents to the control agents.

The control agents control the visualization agents and use the middleman component to communicate with the visualized application. In simple visualizations one control agent requests the visualized application to process normally and make the visualization agents display the information about the status of the application. In tutoring systems, control agents decide what kind of tutoring to provide to the user and control when the application is allowed to proceed. These decisions are based on the users' interaction with the tutoring and information found in the applications Information Storage.

3.1.4. Middleman

The middleman component is a server that serves the two other components. It stores internally the information that the application sent about its status, the requests sent by the visualization and their responses. The possible requests from visualization to the application are: *run*, *pause*, *IOResult* and *ISOperation*. The first asks the system to proceed normally. The second halts the application until new request is received. The *IOResult*-operation creates an I/O result and lets the application process it. The *ISOperation* is used to query and modify the Information Storage of the application.

Each request has also one of three possible synchrony modes. The request can be *asynchronous*, i.e., the call returns immediately and the middleman stores the request until the application connects to it. If mode is *synchronous* the call returns when the application has received the request. If mode is *waitForResponse*, the call returns when the application has replied to the request, for example, returned a result for an information storage query. Different modes can be used to optimize the speed and responsiveness.

4. Example visualizations

We have experimented with different uses of visualization framework. In demonstrations a basic visualization has been used and we have experimented with software tutors based on the visualization technique.

4.1. Visualizations for demonstrations

There are endless possibilities to visualize dialogues. We have used simple speech balloon visualizations in demonstrations. This set of components visualizes dialogues in real time with balloons as seen in Figure 2.

The visualization is used in demonstrations where pre-written slides with a sample dialogue have been used previously. The visualization is an important part of demonstrations. It provides spectators the basic view of the interaction; they can easily see what was said and how turns are taken. Spectators have something to look at and can easily follow the interaction. Visualization can also help if spectators have troubles hearing. More importantly, a visual representation of the spoken dialogue provides a persistent view. While speech disappears after it has been spoken, visualization remains on the screen. The demonstrator can refer to the dialogue to em-

phasize features of the system and spectators can read the dialogue back and forth.

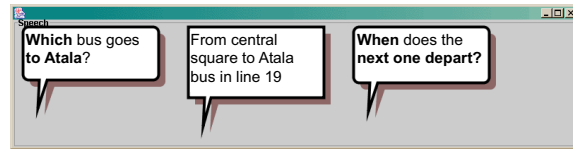


Figure 2: Balloon visualization of spoken interaction.

It is possible to develop more application specific visualizations to demonstrate some particular features, for example, recognition confidences or a user model. In Figure 2 the keywords of user input are highlighted by bold facing them. The application specific visualizations do not require any modifications to middleman component or the application's connection components because these provide access to systems information storage by default. We have also implemented a component that makes it possible to stop the visualized application temporarily by clicking a pause button in the visualization. This is a handy feature in demonstration situations when you need to pause the application to explain something and is needed in particular when debugging applications.

4.2. Debugging tool

When applications are debugged, tools for monitoring the system are required. In Jaspis-based systems, where all persistent data is held in Information Storage, a view to the storage is vital. Traditionally we have used a simple passive view of the information storage. This component is included into system configuration and it opens a window in the screen when the application is run. Developer must manually update the view and can then view the information storage content as a tree. The visualization system can provide access to the same information with benefits of distribution. The visualization can now run on a separate computer when it previously had to run on the same server as the application. The visualization can also be an active one, i.e., it can automatically update when something happens in the system. When a developer learns to read such animation type visualization, it is possible to instantly see what is happening in the system. The developers of speech based systems can benefit from such tools especially since testing applications is often quite challenging because of the probabilistic nature of speech recognition. The modular structure of the visualization tool makes it easy to add application specific visualizations to different applications. These can later develop into visualizations that can be used to demonstrate the application.

4.3. Software tutoring

The most advanced use of the visualization framework that we have experimented with is software tutoring. In these systems, a software component teaches the users how to use a spoken dialogue application. We have developed several versions of a software tutor to an e-mail system AthosMail [10], and a bus timetable system Busman. In the e-mail system a speech-only tutor was used successfully [11]. In the Busman application we used a multimodal tutoring. The tutor is implemented as a visualization component and it controls the application by stopping it while instructions are presented to the user. The tutor monitors the system status. For example, it

looks at input results to see if there have been speech recognition errors and uses this information to provide appropriate guidance. The tutoring asks a user to give some specific inputs to the system so that it can unambiguously recognize error situations and make sure that the user can give inputs to the system successfully.

The tutors include real time visualizations of the spoken interaction. The visualizations enable users to see speech recognition results and see when there are errors. This helps the users to understand the system behavior. The tutors are intended to be used when a system is used for the first time. A tutoring session last 10 to 20 minutes depending on how fast a user proceeds and if there are problems in the interaction.

4.3.1. Experiences with tutoring

We have developed the multimodal tutoring iteratively. From the numerous possibilities we selected four different concepts to be implemented. One used basic speech balloons for visualization and text for actual tutoring. Another one added visualization of forms used by dialogue management to provide more insight into the system. Third version had a graphical user interface that had the same functionality as the Busman system. By using the GUI, users could see what kind of queries they can make to Busman with speech. The fourth version, as seen in Figure 3, had a more graphical presentation; the tutoring was presented by a cartoon character and system activity was visualized with animated icons and balloons.

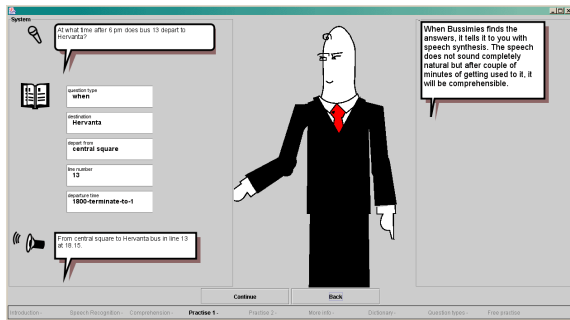


Figure 3: A Multimodal tutor for a spoken dialogue system.

Nineteen first year students without prior experiences on spoken dialogue systems tested the tutors. The balloon type visualization worked very well; users liked the fact that they saw the recognition results and also the systems outputs. When recognition errors were visible users noticed errors and often adjusted their speaking style, if necessary, to get better results. Users were also able to see what kind of errors the speech recognition makes. The other visualizations, especially the forms representing the systems internal logic, were not received so favorably and were often considered to contain unnecessary technical detail.

The technical structure of the visualization framework was very efficient for tutoring. We could monitor all the necessary information in the system. It is possible even to modify the system status, for example, discard misrecognized input. We made a design decision not to do this so that users would see how the system responds when there is a recognition error. The synchrony that the visualization framework provided was timely enough for the tutoring.

Based on the results we have further developed the basic balloon tutor and the GUI based tutor. The next step is to compare the tutors to a guidance that is not connected to the application to see, if the visualization indeed improves the learning results.

5. Conclusions

We have developed a visualization framework for spoken dialogue applications and found efficient uses for the visualizations. The framework has enabled us to use visualizations when demonstrating existing systems and it provides means to build advanced debugging tools. We have also experimented with rather complex software tutors implemented as visualization components. The architecture has efficiently provided all the functionality we have needed in these systems.

6. References

- [1] Boda, P. "Visualisation of spoken dialogues", *Proceedings of ICSLP-2000*, vol.1, 146-149, 2000.
- [2] Yang, L.-c. "Visualizing spoken discourse: Prosodic form and discourse functions of interruptions." *Proceedings of 2nd SIGdial Workshop on Discourse and Dialogue*. 2001
- [3] Dybkjær, L. & Bernsen, N. O. "Towards General-Purpose Annotation Tools - How far are we today?" *Proceedings of the Fourth International Conference on Language Resources and Evaluation LREC'2004*, Vol. 1, 197-200, 2004.
- [4] Cole, R. "Tools for research and education in speech science", *Proceedings of the International Conference of Phonetic Sciences*, San Francisco, CA, 1999.
- [5] Sturm, J., Bakx, I., Cranen, B., Terken, J. & Wang, F. "Usability evaluation of a Dutch multimodal system for Train Timetable Information", *Proceedings of LREC*, 2002.
- [6] Terken, J. & te Riele S. "Supporting the construction of a user model in speech-only interfaces by adding multimodality", *Proceedings of Eurospeech 2001 Scandinavia*, Vol. 3, 2177-2180, 2001.
- [7] Kamm, C., Litman, D. & Walker, M. "From Novice to Expert: the Effect of Tutorials on User Expertise with Spoken Dialogue Systems", *Proceedings of ICSLP-1998*, 1998.
- [8] Seneff, S., Hurley, E., Lau, R., Pao, C., Schmid, P. & Zue, V. "Galaxy-II: A Reference Architecture for Conversational System Development", *Proceedings of ICSLP 1998*, 1998.
- [9] Turunen, M. & Hakulinen, J. "Jaspis - A Framework for Multilingual Adaptive Speech Applications", *Proceedings of ICSLP 2000*, 2000.
- [10] Turunen, M., Salonen, E-P., Hartikainen, M., Hakulinen, J., Black, W.J., Ramsay, A., Funk, A., Conroy, A., Thompson, P., Stairmand, M., Jokinen, K., Rissanen, J., Kanto, K., Kerminen, A., Gambäck, B., Cheadle, M., Olsson, F. & Sahlgren, M. "AthosMail - a multilingual Adaptive Spoken Dialogue System for E-mail Domain" *COLING Workshop Robust and Adaptive Information Processing for Mobile Speech Interfaces*, 2004.
- [11] Hakulinen, J., Turunen, M., Salonen, E.-P. & Rähkä, K.-J. "Tutor Design for Speech-Based Interfaces", *Proceedings of DIS2004*, 155-164, 2004.