

Agents for Integrated Tutoring in Spoken Dialogue Systems

Jaakko Hakulinen, Markku Turunen, Esa-Pekka Salonen

Speech-based and Pervasive Interaction Group, Tampere Unit for Computer-Human Interaction
Department of Computer and Information Sciences
33014 University of Tampere
FINLAND
{jh, mturunen, eps}@cs.uta.fi

Abstract

In this paper, we introduce the concept of integrated tutoring in speech applications. An integrated tutoring system teaches the use of a system to a user while he/she is using the system in a typical manner. Furthermore, we introduce the general principles of how to implement applications with integrated tutoring agents and present an example implementation for an existing e-mail system. The main innovation of the approach is that the tutoring agents are part of the application, but implemented in a way which makes it possible to plug them into the system without modifying it. This is possible due to a set of small, stateless agents and a shared Information Storage provided by our system architecture. Integrated tutoring agents are easily expandable and configurable, and general agents can be shared between applications. We have also received positive feedback about integrated tutoring in initial user tests conducted with the implementation.

Introduction

While some designers believe that a speech system should be so intuitive that one can immediately start using it, at least some teaching and guidance is usually necessary regardless of the type of the application. The main issue is that the user must know what to say to the system. Grammars in speech systems are always limited in some way. Especially, command based systems with a small vocabulary are powerful only after the user has been taught how to use the system. On the other hand, in systems with open grammars and sophisticated natural language understanding, it is necessary to teach the user the limits and capabilities of the system. This information is often on a higher conceptual level, but may be effectively taught in context. In general, the user must know what he/she can say to the system.

Prompt design is considered to be the key issue in user guidance when systems are targeted to a wide audience [1]. However, it has been found that the initial impression of a system affects the users' image of the system also later on. In a research by Kamm et al. [2] users studied a written tutorial on a web page before they started using the system. Another group learned to use the system by following its prompts and using its help features. The latter group had significantly lower satisfaction with the system, even though they reached comparable performance.

For all of the reasons mentioned it is important to give the users guidance on how to use a speech system. Speech systems often include some sort of manuals on web pages and in print. However, users in general are not very interested in reading manuals, but they would rather want to get into action as soon as possible. We introduce integrated tutoring in

speech systems as a solution for this problem. These systems have tutoring as part of their functionality, so users can start to use the applications right away and learn as they go. We show that it is possible to implement integrated tutoring in a way that makes it possible to plug the tutoring components into applications without modifying the original applications.

In this paper we focus on the implementation of the tutoring agents and present an example implementation for an existing speech-based e-mail application. We begin by introducing integrated tutoring systems, and especially the dialogue management part of them. Next, we describe the technical implementation of the integrated tutoring system on top of the Jaspis architecture [3]. This is followed by a concrete example with the Mailman application [4]. Finally, we discuss some issues we have confronted with our tutor system.

Integrated Tutoring Systems

By integrated tutoring we mean functionality where a system gives guidance to a user on the operation of the system while the user is using the system. From the user's point of view, we have implemented the tutoring as a separate dialogue participant, not as part of the system.

Integrated tutors are often used in computer games. They teach the user how to play the game by telling the user what to do next while the user is playing the game. This way the tutorial session is more meaningful and pleasant for the user. By introducing such a system to speech applications, we gain the same advantages. It can also be very efficient to teach some features of the system by letting the user use the features with the actual system. We can also give guidance on certain features at moments when the user can or should try them out, i.e. we can select meaningful context for tutoring. Tutoring is also an efficient way to tell about new features or modifications to an already existing system.

Comparing tutoring to using system prompts to teach the user, we would like to point out several differences. Instead of informing the user about different possibilities, the tutor usually gives instructions to the user on what to do next. The tutor can also check that the user is really doing what was asked of him and possibly discard incorrect inputs. This way we can, for example, build safe learning environments (see [5]) where the user cannot make any fatal operations. In speech systems this is often a welcome feature because the possibility of recognition errors increases the occurrence of mistakes.

In tutoring systems the teaching usually has some sort of structure, so that it is more than just a set of random hints. In addition to teaching the user the actual commands, the tutor can teach on a more conceptual level. The tutor can also check if the user has learned the taught subject by giving the user such tasks that the user has to use skills taught earlier. If

the user has problems, the tutor can adapt and teach the subject again. We may also have a tutor which allows the user to ask questions from it. In traditional tutoring systems this is an essential part of tutoring, but when the tutor is part of the tutored system, we can also build effective tutors which get all their feedback from the user by following how the user is using the system (for such functionality in traditional tutoring systems see e.g. [6]). Finally, when the tutor is a separate dialogue partner, there is no need to shorten the system prompts, as the user becomes more familiar with the system.

Dialogue Management

An integrated tutoring system brings up interesting design and technical questions. From the user's point of view the tutor may be seen as a separate person in the dialogue, or the system itself may act as the tutor. The same distinction can be made from the technical point of view as well. The tutoring functionality can be implemented as part of normal dialogue management or it can be a separate implementation working in parallel to normal dialogue management.

When the tutor is presented to the user as a separate dialogue partner, the dialogue becomes a three party one. This causes some changes to the dialogue structure as, for example, turn taking becomes more complicated and dialogue design must answer the question of when and how the tutor takes the turn. The tutor may comment on the user input and tell what will happen next or it can speak after the system and comment on what the user just heard. The tutor can also give advice on what can be said next.

We should also separate the dialogue initiative of the user-system dialogue from the initiative of the user-tutor dialogue. The tutor can take the initiative in user-tutor dialogue while the system may still work in user initiative mode. This way we can effectively teach users in user-initiative systems without altering the systems.

An interesting change to the dialogue structure occurs when a tutor modifies the user inputs. The tutor can, for example, discard an input and let the user try to speak again if the input was not what it was expected to be. In a case like this the tutor controls the user-system dialogue. Another step is when the tutor "speaks" to the system, i.e. gives inputs to the system on behalf of the user and acts as an example to the user. The user may also want to speak to the tutor directly. This may lead to complex situations which need to be taken into account in many components of a system, including the speech recognition grammars.

Implementation of Tutoring Agents

When tutoring functionality is implemented into a dialogue system, especially into an existing one, it is desirable that the tutor can be implemented without altering the existing base system. This way the base system can be available without the tutor and without any modifications. This also ensures that the tutored system does not differ from the base system. Such an implementation can be hard or even impossible with monolithic systems. Our tutoring implementation for the Mailman system and our model for tutoring agents are based on the Jaspis architecture [4]. The idea of several small agents controlling the dialogue in Jaspis proved to be very powerful when we implemented the tutor agents. No alterations were made to the actual Mailman program code. Instead, the tutoring implementation is just a set of new agents working in parallel to the agents of the Mailman system. Next, we intro-

duce the principles of the system architecture behind the tutorial agents.

System Architecture

Jaspis is an architecture for spoken dialogue systems [4]. The tutoring agents use three of the main features of Jaspis: the shared Information Storage, evaluators, and small, stateless agents. The shared Information Storage is a component that all the other components in Jaspis based systems have access to. This is the storage where each component places all the information that they need to store. Information Storage can be seen as a kind of blackboard. It contains, for example, the dialogue state and the user model. Because of this, all the other components can be stateless, i.e. they do not store any information about the dialogue state in their internal structures. This way we can always freely choose an appropriate component for each situation.

The other components in the Jaspis architecture are agents and evaluators. Agents are the components that actually implement the system functionality. Evaluators are used to select the best agent for each situation. Each agent and evaluator works under one of the several managers in a system. For example, dialogue agents and dialogue evaluators work under the Dialogue Manager. Every time Dialogue Manager receives a turn, i.e. is able to do anything, all the agents are evaluated to find which one, if any, is the most capable of handling the situation at hand. The idea is for the agents to be very compact so that in each dialogue turn there can be a different dialogue agent, or even several, to take the turn. For example, in the base Mailman system (not including the tutor features) there are 27 different dialogue agents. There is a separate agent for each system function and several small agents for different situations such as the different phases of logging in.

Tutoring Agents

Our general tutorial implementation consists of a set of agents and evaluators. In dialogue management we have tutor agents for providing information to the user. Some give the user general hints and only take care that the hint comes in at an appropriate spot in the tutorial structure. Others give the user information about something concerning the current topic. When agents are evaluated, the state of the dialogue is examined to see if the situation is appropriate for each agent. The user model is also used to see if the information has already been taught to the user. If an agent receives a turn, it updates the user model to avoid receiving a turn again.

The turn taking decisions may include any kind of information available. However, excluding the user model, most of the decisions are based on four rules which all are based on information found in dialogue history. The base system and the tutor agents store their dialogue events in a common dialogue history. The tutor agent evaluation looks at the dialogue history for specific events. An agent may require the previous event to be of a specific type. Specific previous events may also prevent an agent from taking a turn. Furthermore, an agent may require a set of events to be found in the history and some events anywhere in history may also prevent an agent from taking a turn.

With these rules we have been able to make specific tutoring events happen at specific points in the dialogue, making sure that the structure of the dialogue and the tutorial stays coherent. We take care that the tutor does not teach something

which requires knowledge not yet taught, does not teach something that the user already knows and makes sure that the turn is given to the user or the system at an appropriate time. It is also possible to write special rules into single agents on a case-to-case basis. The tutoring agents also have different values that they return for their ability to take turn even if all their conditions match. This way we can make an agent to take turn before another if conditions for both are met.

Special agents manage the user inputs. They receive turn when another agent has the asked user to give a specific input. In such a case the agent has stored information about the request to the Information Storage. The agent managing input looks at the user input and if it matches the requested one, it does not alter the input but instead gives the user confirmation about successful recognition and possibly some description of what the system will do next. If the input differs from what was asked, the agents discard the input, inform the user about the recognition result and ask user to try to give the command again. Alternatively, an agent can simply tell the user that the input was not an anticipated one and give advice to the user to listen to what happens next. Input managing agents also take turn when no special input was requested, but the system receives only silence or a recognition error occurs. In such cases the tutor instructs the user to speak at a correct time. An agent could also modify or create inputs. This agent can be used when we want to give the user an example, i.e. the tutor uses the system.

In addition to the aforementioned agents, the dialogue module includes a special tutor agent called "RestoreIOResults". It modifies the input/output results (user inputs etc.) so that other tutorial agents can co-exist with the normal dialogue agents without conflicts, the base implementation does not see the tutor behavior at all. When a tutor agent takes turn, it stores all input/output results available into the shared Information Storage. When the tutor agent has done its actions, "RestoreIOResults" modifies the information storage so that the normal dialogue agents can continue. This way the base system implementation does not need to take the things that are happening within the tutorial into consideration.

There is also a special tutor mode evaluator under Dialogue Manager. This evaluator takes care of when the tutorial mode is selected that the tutoring agents have priority over the normal agents. It does not block the normal agents from working as the normal system is supposed to work all the time. This evaluator can however block the tutorial agents from taking turn when the tutorial mode is turned off.

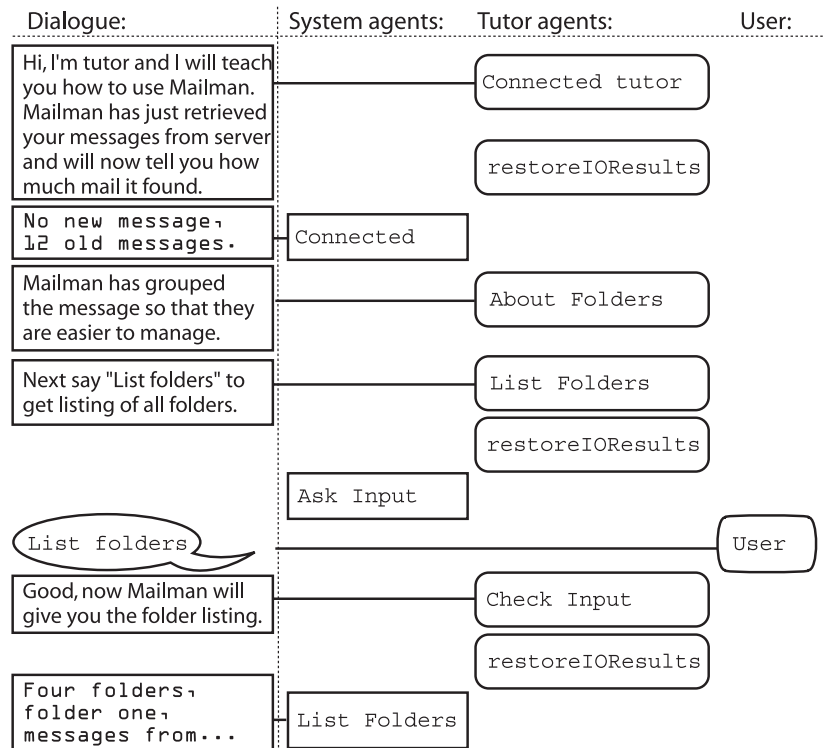
Tutorial Implementation in Mailman Application

We have an implementation of the integrated tutor in our Mailman application [3]. Mailman is a telephone operated speech interface to e-mail. It has a user-initiative speech interface that is used to

browse the mailbox. The telephone keypad can also be used to access most of the system functionality. Mailman has user profiles and users must login to the system when they call. The user models enable us to use the tutor only during the first calls. Furthermore, Mailman is a user-initiative system and therefore requires explicit training to learn the commands; it is an excellent system for an integrated tutor. Mailman has also some higher-level concepts, like automatic e-mail message grouping that our tutor describes to the user.

The tutor is implemented as a separate dialogue partner identified by a different voice. The tutor also refers to itself as "tutor" and the system as "system" and "Mailman". It gives user guidance and hints during dialogue. In the beginning of the tutoring it requires the user to follow its tutoring plan and discards unwanted inputs, thus controlling dialogue to some extent. The user is not able to speak to the tutor but the tutor monitors the dialogue and uses this to adapt its own behavior.

The implementation includes several agents. There are about 30 tutoring utterances that are generated by the presentation agents. As we implemented the Mailman tutor as a separate person and because the set of different tutor outputs is limited, we decided to use recorded voice as our tutor voice. Therefore, the task of the presentation agents is rather straightforward. Most of the tutorial utterances are simple canned utterances and some context sensitive outputs feature simple slot-filling. For instance, an actual example of currently used grouping is used when the tutor explains the concept of mail folder. Furthermore, a special presentation agent was implemented to inform the user that Mailman understood something else than the tutor asked the user to say.



Picture 1: Dialogue and corresponding agents

There are 25 dialogue agents in all. Most of them are basic information providing agents. Some provide information in specific situations such as right after login and during reading an e-mail message. Others just provide information whenever it is appropriate according to the state of the tutorial.

At the start of the tutoring, certain tutoring content is given in two sets of tutorial outputs. In a set, tutorial messages follow one another in a specified order. The first set describes to the user the concept of folders and how to browse them. The other set teaches browsing between e-mail messages and reading them. The first set is optional as is presented only if there are folders present. These sets include tutoring turns where the user is requested to give specific inputs. Each of these cases has a matching agent which checks if the given input indeed matches the requested one, gives the appropriate tutoring message and if necessary, discards the incorrect input before the Mailman system has a chance to process it.

Picture 1 contains an example where an excerpt of dialogue and corresponding agents are presented. The dialogue starts when a user has logged into the system and Mailman has retrieved e-mail messages from a mail server. This is also the point when the integrated tutoring comes into play. "ConnectedTutor" takes the turn and explains to the user the situation and what is going to happen. The next agent, "RestoreIOResults" takes the turn and removes the results of the tutor output messages and replaces them with whatever messages were present when the "ConnectedTutor" agent took turn. Next, the normal "Connected" agent of the Mailman system takes turn. It is followed by two tutor agents. The latter one asks the user to give a specific input. The tutor agent turn ends once again with "RestoreIOResults" agent and the system proceeds as usual, i.e. asking input from the user. When the user has given the input, the tutor checks that it was correct, gives approving output and once again "RestoreIOResults" receives turn, this time placing the user input back as the active input and the system reacts to the user's command in the usual way.

The existing Mailman agents featured in this example are the agents of the original Mailman system. This is the case for the entire system; the original Mailman agents have not been modified in any way.

Discussion

Integrated tutoring as a concept seems to be a promising one. Our initial tests with the first version of the tutor provided positive comments. However, some problems were spotted as well. Mainly these concerned the users' view of the state of the tutoring. We had to update the tutor to be more informative about the status of the tutorial. The latest version explicitly informs the user when the guided part of the tutoring is over and the user can start using the system freely. This was also a good spot for a summarization of covered subjects. Our plans for future work include usability testing of the Mailman tutoring functionality to find out how well the concept and implementation work in practice.

In the implementation we have encountered one problematic issue. The only signs about the tutoring agents that are left in Information Storage are the entries in the dialogue history. In our current implementation these entries are placed in the same dialogue history as the normal system events. Using the same dialogue history has caused us slight problems. These occur when Mailman uses the dialogue history and the user gives the "repeat", "what next" or "tell more" command. We

had to override the corresponding agents with special tutor agents to avoid repeating the tutor messages. This practice is based on the idea that the user perceives the tutor as separate from the system and that the user can talk only to the system. Therefore, for example, the repeat command should repeat the last system output, never a tutor output. We would like to avoid any alterations to existing systems and therefore these cases are a bit problematic. Overriding existing agents could be avoided by maintaining a separate, yet corresponding dialogue history for the tutor events. Separate dialogue histories should be very handy especially in systems where the user can speak to a tutor and we have two parallel dialogues.

Conclusions

We have developed an integrated tutoring feature into spoken dialogue systems that teaches users how to use the system. Tutoring as a way to teach the system to the users has certain benefits. Users can start using the system right away without a need for reading manuals or similar external activities. We can also provide teaching in a real life context and we are able to check that the user really learns what we teach and adapt the teaching when problems occur.

In this paper we have described the implementation issues of integrated tutoring. The agent based approach of the Jaspis architecture proved extremely efficient from this point of view. We did not have to alter our original system at all. Also, the shared Information Storage played an important role. In all, the Jaspis architecture was a very efficient and easy platform to develop such new functionality into our existing system. This extendibility will enable us to continue our work with the tutoring and make it easy to extend our current tutor. For example, providing the user a possibility to talk to the tutor would require adding some new agents and another recognizer to the system. Once again, the existing components would not need to be altered in any way. The explicit evaluator system of the Jaspis architecture was also used, which makes it easy to turn the tutor on and off, for example.

References

- [1] Yankelovich, N., "How Do Users Know What to Say?" *Interactions*, 3(6), 32-43, 1996.
- [2] Kamm, C., Litman, D. and Walker, M.A., "From Novice to Expert: The Effect of Tutorials on User Expertise with Spoken Dialogue Systems", *Proceedings of the International Conference on Spoken Language Processing*, (ICSLP98), 1998.
- [3] Turunen, M. and Hakulinen, J., "Mailman - a Multilingual Speech-only E-mail Client based on an Adaptive Speech Application Framework", *Proceedings of Workshop on Multi-Lingual Speech Communication* (MSC 2000), 7-12, 2000.
- [4] Turunen, M. and Hakulinen, J., "Jaspis - A Framework for Multilingual Adaptive Speech Applications", *Proceedings of 6th International Conference of Spoken Language Processing* (ICSLP 2000), 2000.
- [5] Nakatani, L.H., Egan, D.E., Ruedisueli, L.W., Hawley, P.M. and Lewart, D.K., "TNT: A Talking Tutor 'N' Trainer for Teaching the Use of Interactive Computer Systems", *Proceedings of CHI '86*, 29-34, 1986
- [6] Rickel, J., Ganeshan, R., Rich, C., Sidner, C.L. and Lesh, N., "Task-Oriented Tutorial Dialogue: Issues and Agents" *Working Notes of AAAI Fall Symposium on Building Dialogue Systems for Tutorial Applications*, 2000.