

# Agent-based Adaptive Interaction and Dialogue Management Architecture for Speech Applications

Markku Turunen and Jaakko Hakulinen

Human-Computer Interaction Group  
Department of Computer and Information Sciences,  
FIN-33014 University of Tampere, Finland  
{mturunen, jh}@cs.uta.fi

**Abstract.** In this paper we present an adaptive architecture for interaction and dialogue management in spoken dialogue applications. This architecture is targeted for applications that adapt to the situation and the user. We have implemented the architecture as part of our Jaspis speech application development framework. We also introduce some application issues discovered in applications built on top of Jaspis.

## 1 Introduction

Speech-based communication can differ greatly between individual users and situations. For example, some people prefer that the computer takes the initiative, but others may prefer a more user-initiated style of communication. Speech is also very language and culture dependent and differences between user groups can be large. In order to construct efficient speech applications we need interaction methods that adapt to the different users and situations.

In this paper we present an advanced architecture for speech-based interaction and dialogue management. We use dialogue agents and dialogue evaluators for adaptive dialogue handling. Together with input and presentation frameworks they form a foundation for adaptive interaction methods. Based on these components we can build adaptive and reusable interaction methods for speech applications. This architecture is included in the Jaspis speech application development framework which is described next.

### 1.1 Jaspis architecture

Jaspis is a general speech application development architecture. It is based on Java and XML. Jaspis is a freely available framework to support the development and research of spoken language applications. Jaspis is described in detail in [4]. Here we focus on the features that allow for flexible interaction and dialogue management. A brief explanation of the key components of Jaspis follows.

Jaspis contains two features that are needed in adaptive applications. First, all information is stored in a shared place. This blackboard-type shared memory is called information storage. A conceptual, language independent form to present information is used whenever possible. This is a key feature to adapt applications to different situations. All system components store all information in the information storage. In this way each component can utilize all the information that the system contains. Using shared information it is possible to utilize information such as dialogue history and user profiles in every single component.

The second main feature is the flexible interaction coordination model. This is realized using managers, agents and evaluators. The agents are system components that handle various interaction situations, such as speech output presentations and dialogue decisions. The evaluators attempt to choose the best possible agents to handle each situation and the managers coordinate these components.

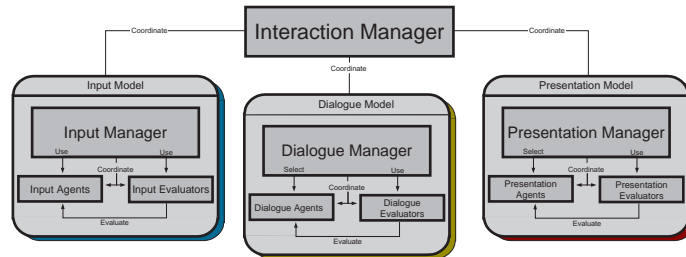
Next we will introduce the general interaction architecture followed by a more detailed description of the dialogue handling components. After that we present some applications issues discovered in the applications written using the Jaspis architecture. The paper ends with discussion and conclusions.

## 2 Adaptive Interaction Management

Speech based dialogue can be divided into three parts. In a speech application we need to receive inputs from a user, carry out actions according to the user's inputs and present the response back to the user. Therefore, we need three kinds of interaction handling components. We must use some kind of input handling component to deal with the user inputs, have a dialogue component to keep the conversation going on in a meaningful way and also have a component for output presentation.

In adaptive applications we can have competing interaction strategies for the same tasks and also complementing strategies for different tasks. Since the interaction is not necessarily based on sequential control, we must have a way to handle it in a more flexible way. All these needs yield some kind of coordination, selections and evaluations of different possibilities.

In Figure 1 we present our approach to adaptive interaction management. We use agents, evaluators and managers to represent the interaction techniques and their coordination. All three interaction sub-models consist of these components. The basic idea is similar for all models: interaction agents are used to present interaction techniques, evaluators are used to determine which agents are suitable for the different situations and managers are used for overall coordination. The only difference is that in the dialogue and presentation models the selection of an interaction agent takes place before the agent handles the situation, but in the input model input evaluators operate after the input agents have operated on and ranked the results. Next we introduce how Jaspis implements the interaction coordination described above.



**Fig. 1.** The adaptive interaction management architecture.

## 2.1 Managers

The managers control all interaction tasks. They can have any number of individual tasks, but they all share two common tasks: they use evaluators to determine how each interaction technique, i.e. an agent, suits the current interaction situation and then select one particular agent to take care of the situation. It is up to each manager to decide how it uses the results it gets from the evaluators.

In the current Jaspis implementation managers also decide when they can handle the interaction on the top level (i.e., which manager should handle the situation). They use evaluators for this task too. Basically, managers check if their agents are able to handle the current situation and how successfully they are able to do it.

The interaction manager controls the interaction on the highest level. It should be aware of the other components in the system and control how they carry out the interaction. An application developer can specify exactly how the interaction management should be done. For example, we can use scripts to add domain specific behavior to the interaction manager. This kind of approach is utilized in the GALAXY-II architecture [3]. We prefer autonomous sub-components which determine themselves when they are able to handle the interaction. This allows for more flexibility and adaptability for the interaction. Jaspis includes a manager switching scheme where an opportunity to react to events is offered to managers based on their priorities.

## 2.2 Interaction Agents

The interaction agents implement the actual interaction techniques. Agents are often understood to be mobile, intelligent and autonomous. We do not use agents in that sense: they can be intelligent, and they are autonomous in a sense but also controlled in a way. Mobility is not an issue here.

Agents are specialized for specific tasks, such as error handling or presentation of tables. The fact that agents are specialized makes it possible to implement reusable and extendable interaction components that are also easy to write and maintain. For example, we can write general interaction techniques, such as error correction methods, to take care of error situations in speech applications. Very simple and generic

agents for error handling are ones that ask the user to repeat their input if it was not understood. This process actually requires three agents: one that determines that no input was recognized, one that decides that we should ask the user to repeat the input, and one that generates the actual output message.

Although agents can be specialized for different tasks, we can also have different agents for the same tasks. In this way we can support different interaction strategies inside an application in a modular way. Because of this, an application can adapt dynamically to the user and the situation. For example, we can have different agents to take care of speech outputs spoken in different languages. We can take the agents from the example in the previous paragraph and write a presentation agent that now asks the user to repeat the input using a different language. We have now the same error correction procedure in a new language.

The set of attributes the agents have indicate what they can do and how well. For example, we can have attributes for the languages that a presentation or an input agent can handle. Based on the agents' attributes and information about the current situation and user, we can determine if an agent is suitable to handle the given situation. Some of the interaction agents may be totally application independent and some are closely tied to the application and/or to the domain. Application independent behavior is usually preferred so that the same components can be used in different applications.

### 2.3 Evaluators

A manager uses evaluators to evaluate all available agents and chooses the most suitable one. The basic idea is that a manager uses evaluators to compare the different attributes of agents and chooses the one recommended by the evaluators. Evaluations are separated to small independent components (the evaluators) so that we can write reusable evaluators and keep the potentially complex evaluations manageable.

The selection of the most suitable agent is based on the overall situation which may include things such as current inputs, dialogue context and user preferences. The evaluation can depend on the interaction task at hand. Different evaluators are specialized for evaluating different aspects. For example, one evaluator can evaluate some specific aspects of agents, such as the ability to process certain input modality. Other evaluators can deal with more general issues, such as if a particular dialogue agent suits the overall dialogue history. Evaluators can also be application independent or tied to the application domain. By using application independent evaluators, we can write reusable components and when necessary, new application specific evaluators can also be written or existing evaluators can be extended.

When evaluators evaluate agents, each evaluator gives a score to each agent. Each agent therefore receives several scores. Multiplying the given scores together yields the final evaluation. As the scores are between 0 and 1, the final result also falls in this range. If an evaluator gives a "0" score to an agent it means that the agent is not capable of handling the situation and "1" means that the evaluator sees no reason against using the agent. This simple scheme can be extended to include scaling factors and more complex functions when needed.

### 3 Adaptive Dialogue Management

The dialogue manager takes care of the overall communication between the user and the system. The input and presentation managers handle interaction tasks in detail. In most of the current speech systems dialogue management is handled by a single component called dialogue manager. We believe that this monolithic approach is not ideal for speech-based or other applications that are highly context sensitive. In order to utilize reusable interaction components and to adapt to the user and the situation we need more flexible methods for coordinating the dialogue.

#### 3.1 Dialogue Agents

The dialogue agents form the basic communication units between the user and the application. Each time the dialogue manager takes control, a single dialogue agent is activated and it decides what action the system is to take. The next time the dialogue manager is activated the selection of an agent is repeated. Another agent can continue the dialogue in the next turn, but the idea of agents is that a single agent is specialized in a specific situation in dialogues. Therefore, a dialogue flow is a sequence of dialogue units formed by the dialogue agents. As with agents in general, different dialogue agents are supposed to be suited for different situations. Each of them has a set of capabilities that reflects what this particular agent is suitable for. In a very simple case there is only one agent suitable for a given situation. In order to provide more flexibility and choices in the interaction we can add alternative agents with varying behaviors for the same dialogue situation.

Dialogue management strategies have been widely studied and different kinds of dialogue management approaches have been proposed and evaluated. For example, according to Walker & al. [6] mixed-initiative dialogues are more efficient but not as preferred as system-initiative dialogues. They argue that this is mainly because of the low learning curve and predictability of system-initiative interfaces. However, system-initiative interfaces are more inefficient and could frustrate more experienced users. Thus, both kinds of dialogue handling strategies are needed and should be used. Attempts to utilize both mixed-initiative and system initiative approaches have been carried out. For example, in [1] short-cuts are used to provide the benefits of system-initiative and mixed-initiative dialogue control strategies in a single interface.

From the viewpoint of an application developer, dialogue agents can also be used to implement different dialogue control models. Different dialogue control models (such as state-machines and forms) have different capabilities and benefits. In complex applications we may need several control models. Attempts to utilize several dialogue controlling models have been introduced for example to combine state-based and form-based dialogue control models. The dialogue management architecture of Jaspis provides explicit support for alternative dialogue strategies, dialogue control models and reusable dialogue components.

### 3.2 Dialogue Evaluators

When the dialogue manager is selecting an appropriate agent for a situation, it uses dialogue evaluators to carry out the actual evaluation. Dialogue evaluators are specialized for different kinds of tasks. The simplest evaluator, the *capability evaluator*, just checks which dialogue agents can handle the current input. Often only one dialogue agent is found and the selection is trivial. However, in some cases there are many competing agents available. This is especially the case when input agents have indicated that an error has occurred and an error handling agent is needed. Since there exist agents with different kinds of error handling strategies we must find the most suitable one to solve the error situation and let it control the situation.

Another example of a basic evaluator is the *consistency evaluator*. Since it is important to keep some kind of consistency in the dialogue flow, we must check that the chosen dialogue agents are coherent over time. This prevents the dialogue manager from switching to different styles of agents at every dialogue turn. The consistency evaluator uses the dialogue history to determine how well each agent suits the overall dialogue flow. As changing the dialogue style is sometimes useful, the consistency evaluator should allow complete changes in dialogue style, but it should not allow such changes to occur all the time. However, this behavior can be application and domain dependent and an application developer should inherit this evaluator and add application specific behavior to it.

## 4 Dialogue Management in Practical Applications

We have used Jaspis to construct several spoken dialogue applications in different domains. These applications include an e-mail application Mailman (Postimies) [5], a local transportation guide Busman (Bussimies) and a ubiquitous computing application Doorman (Ovimies). Next, we introduce some experiences based on the implementations of mentioned applications.

In form-based dialogue management for database retrieval tasks, multiple agents give several advantages. Many of the special situations can be implemented in their own dialogue agents. This makes the main dialogue manager agent simple and maintainable. Basic form-based dialogue management can be implemented in a domain independent manner, so that all domain specific information is in configuration files. The basic algorithms are reasonably simple. When dialogue management for special cases is implemented in several special agents, most of these agents can be also domain independent. In fact, even if the main dialogue agent is domain specific, the other agents can still be domain independent. Still, it is possible to develop domain specific agents for the special situations when domain knowledge is needed.

Basic form-based dialogue management could be split into following dialogue agents. The main dialogue agent handles the situations when the system is responding to a legal user input that defines a query. This agent simply checks if a resulting form is a legal query. If so, it makes the query, otherwise it asks the user to give more in-

formation. Another agent can take care of situations when we have received a result from the database. This agent informs the user of the results. If the query result set is too large to be presented to the user, a special agent takes control and forms a question to the user to restrict the query. If the database result set is empty, we can have a special agent to modify the query. Ambiguity resolution can also be seen as a special agent. When user input is ambiguous, this agent steps in. The last three agents mentioned require some domain specific information, namely information about the query keys and concepts of the system. If one is developing an information retrieval system, the domain specific program code could be restricted to these three components. This way porting the system to a new domain would require updating only these pieces of program code.

Other, more special cases, suitable for specific dialogue agents include different error handling situations and requests for the users to repeat information etc. If the user asks the system to repeat something that was just uttered, there can be a completely different dialogue agent to handle this. All we need is some kind of dialogue history where we can find what the state of the dialogue was before, so that we can reproduce our output. If the dialogue history is stored in a sufficiently generic form, the agent handling repeat requests can be completely independent of the other agents. Error handling situations can also be specific agents. For example, cases where the user informs the system that something went wrong, e.g. speech recognition made an error, specific agents can take care of the possible correction sub-dialogues. As far as it is possible to recognize user utterances that are out of the applications domain, they could be handled in their own agent, or a set of agents.

When a state-based dialogue is implemented using multiple agents, there are several possibilities on how to split the task between them. One way is to implement a single agent for each state of the state machine. This way we can include complex processing into all of the states. Another way is to build a single generic state based dialogue agent that just reads the state transition definitions from a configuration file. In yet another solution we can implement special states as separate agents and include the ones that do not require any complex processing into the single generic agent. In principle, we can also use multiple copies of a single agent but with different configurations, so that each state has its own instance of the same agent. This is a special way to produce the generic state based dialogue agent and the implementation could include very elegant program code. If we want, we can also split a single state into multiple agents, so that different agents would handle the different types of inputs in each state. This would cause us to have very compact agents, but obviously a lot of them.

No matter which one of the previous solutions we pursue, it is possible to separate special cases, such as those previously described (error handling, repetitions etc.). If these situations were handled in the traditional, state-based manner, they would multiply the number of states needed.

It is also possible to construct completely domain dependent dialogue management. When carefully designed, separation to different agents eases the development significantly. Even if these components can seldom be reused without any modifications, the agents themselves stay much smaller and therefore easier to develop and maintain. The main advantage is that when a well-designed application is further developed,

each agent can be developed without changing the other ones. Expanding the system can also be easier since new functionality can be added by just adding new agents.

## 5 Conclusions and future work

In order to build adaptive speech applications we need adaptive interaction and dialogue handling methods. We have here presented an advanced model for interaction and dialogue management to support adaptive speech applications. Our architecture includes adaptive interaction and dialogue management in the form of interaction and dialogue agents and evaluators. They make it possible to construct reusable interaction management components. Our solution has some common features with the work presented in [2]. In some sense our dialogue agents can be compared to handlers in Rudnicky's and Xu's architecture. However, our solution focuses more on the adaptive and alternative issues. It is also noteworthy that our architecture does not make assumptions on how the actual dialogue control strategies should be implemented.

The presented interaction and dialogue models are implemented in the Jaspis architecture and used successfully in various practical spoken dialogue applications which use different dialogue management strategies and control models. In the future we will extend our work to cover new applications areas dealing with mobile and pervasive computing. In particular we will extend our framework to cover issues such as multiple simultaneous dialogues with multiple participants and the use of context and locations awareness.

## References

1. Larsen, L.: A Strategy for Mixed-Initiative Dialogue Control. Proceedings of Eurospeech97 (1997) 1331–1334
2. Rudnicky, A., Xu W.: An agenda-based dialog management architecture for spoken language systems. Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (1999) 1–337
3. Seneff, S., Hurley, E., Lau, R., Pao C., Schmid, P., Zue, V.: Galaxy-II: a Reference Architecture for Conversational System Development. Proceedings of ICSLP98 (1998)
4. Turunen, M., Hakulinen, J.: Jaspis - A Framework for Multilingual Adaptive Speech Applications. Proceedings of 6th International Conference of Spoken Language Processing (2000)
5. Turunen, M., 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 (2000) 7-12
6. Walker, M., Fromer, J., Fabbriozio, G., Mestel, C., Hindle, D.: What can I say?: Evaluating a spoken language interface to Email. Proceedings of ACM CHI '98 (1998) 582–589