Agent-based Error Handling in Spoken Dialogue Systems

Markku Turunen, Jaakko Hakulinen

Department of Computer and Information Sciences
University of Tampere, Finland
mturunen@cs.uta.fi, jh@cs.uta.fi

Abstract

In this paper, we introduce an agent-based error handling architecture for spoken dialogue systems. In this architecture, all the parts of the error-handling process on the different interaction levels (input, dialogue and output) are explicitly modeled. Error handling is divided into individual, preferably application independent components. The proposed architecture makes it possible to construct adaptive and reusable error handling components and entire error-handling toolkits. The architecture is especially suitable for multilingual applications. The architecture is implemented as part of the Jaspis speech application development environment and it uses Jaspis' agent-based interaction model.

1. Introduction

Errors are not uncommon in speech communication. Utterances are frequently misheard, misunderstood or just missed. It is often said that the major problem in speech applications is their inability to detect and correctly handle different types of errors. Thus, error management in spoken dialogue applications is crucial for successful interaction. However, most of the current tools for speech application development do not have decent support for error management and the task is left to the application developer.

Error management is often understood as error recovery only and is isolated from the dialogue with a user. Some error-handling solutions even try to hide the errors from the other parts of the system. The first one of the three important features which we see that an error handling architecture should have, is that error handling should be seen as a part of the dialogue flow and used as a part of it.

Error handling can also enormously increase the complexity of interaction, especially dialogue management, and lead to complex and problematic structures. Thus, the second feature of error handling architecture is that we must support modular components that take care of error handling independently from the main dialogue flow.

Error management is not dialogue level issue only. Instead, different parts of the error management process can be found on different levels of interaction processing. Duff et al. [1] have analyzed error management and found five different phases that take place in the input, dialogue and output parts of a speech system. Therefore, the third feature is that in order to successfully handle errors, a speech system must support all phases of the error handling process on the different interaction levels.

In this paper, we introduce an error handling architecture that supports the three features of error management as described above. Error management is divided into several phases that provide an explicit and modular view to the error management process. Software components called agents,

evaluators and managers are used to handle errors on input, dialogue and output levels. The architecture is realized as a part of the Jaspis speech application platform [2]. Jaspis makes it possible to build adaptive, modular and in many cases application independent and multilingual error handling components and even complete error handling toolkits for speech applications.

We begin by giving a brief overview of error management in speech systems. After that we introduce the most important features of the Jaspis framework, which should help understand the realization of the error-handling model. Then the realization of the error management architecture in Jaspis is explained step by step. After that we explain how the proposed architecture could be used in real world applications. Finally, we draw conclusions and make suggestions for future work.

2. Error handling in Speech Applications

Error management has usually been separated into error detection and error correction. Duff et al. have presented a more detailed four level model of error correction [1] in their 1996 work and later [3] added an additional step. The phases are *error detection*, *diagnosis of cause*, *planning a repair*, *executing the repair* and *closure and return to the primary dialogue*. We feel that their last phase is rather complex and instead suggest two separate phases, which are *informing the user* and *closure and return to the primary dialogue*. We have also included error prevention as an additional phase.

Based on the above discussion we suggest that the following seven phases must be included in the error management architecture:

1. *Error detection*. The system detects an error or detects that the user tried to correct an error that occurred earlier. Especially semantic errors need automatic error detection methods.
2. *Diagnosis of cause*. The system analyses the causes that led to the error and informs the rest of the system so that it can better correct the error and prevent further errors from occurring.
3. *Planning a correction*. The system decides how to handle the error, i.e., chooses the proper error correction strategy.
4. *Executing the correction*. The system uses an error correction method, such as selection from a list, to correct the error.
5. *Informing the user about the error*. The system informs the user about the error, if necessary, and about the reasons that led to the error. The user is also informed what is going to happen next.
6. *Closure and return to the primary dialogue*. After error correction the system chooses how to return to the primary dialogue.

7. *Preventing Errors*. In some cases, the system can modify its behavior to better match the user's actions and prevent further errors from occurring.

3. Jaspis and its Interaction Model

The Jaspis framework is a general speech application development platform. Here, we describe only the components needed to understand the realization of the error-handling model in Jaspis. A more comprehensive description of the general Jaspis architecture and its adaptive interaction model can be found in our previous paper [2].

3.1. General Principles

The general idea of the Jaspis interaction model is based on distributing the different parts of the interaction process into independent units that are specialized in small tasks. There can, however, be several alternative implementations for the same tasks. These implementation units are called *agents*. To select the most suitable agent for each situation Jaspis uses *evaluators*. For coordinating the interaction it uses *managers*. This is illlustrated in Figure 1.
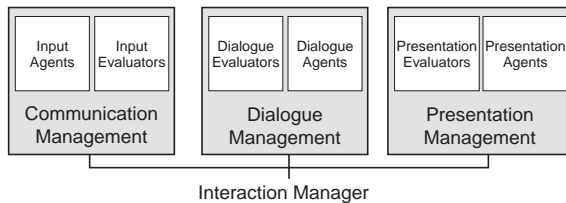

Figure 1: Main components of the Jaspis architecture.

Communication manager controls inputs, *presentation manager* controls outputs and *dialogue manager* controls dialogue flow. *Interaction manager* takes care of overall coordination of the different managers. All these components contribute to the error management and therefore their principal ideas are briefly described in the following sections.

All dynamic information is stored in *information storage* (blackboard type shared memory). Every component of the system has access to all of the information. Using shared information we can use contextual information to adapt interaction for different users and situations. This is needed, for example, in handling non-trivial error situations.

3.2. Communication Management

Communication management takes care of all input and output between the user and the system on the technical I/O level. It also uses *input agents* to interpret the inputs received from the input devices. Each agent is specialized in one part of the input processing and can use the other agents' output in their own processing. Input agents do not only process input after they have received the entire input from a device but also during the input processes. This way an agent can give feedback to a device while input is still being received. This can be used e.g. to control a speech recognizer.

Input evaluators are used to combine and further evaluate inputs processed by input agents. They also handle multimodal inputs i.e. combine inputs from different modalities and decide when to stop gathering more inputs.

Input agents and input evaluators can be used to take care of all input level error handling tasks and implement error detection and error diagnosis components.

3.3. Dialogue Management

Dialogue management is the component that controls the flow of the dialogue. It uses interpreted and conceptualized input messages produced by the communication manager as its inputs and produces conceptual output messages to the presentation component as its outputs. *Dialogue agents* do the actual work of the dialogue manager. The dialogue flow is split among the different agents. There are different agents that specialize in different dialogue situations. There can also be several agents that can take care of the same situation but in different ways. For example, some agents can be more system initiative while others may let the user take the initiative. In this way, we can also provide alternative error correction strategies in an adaptive manner.

Since we have both complementary and alternative agents, *dialogue evaluators* are used to evaluate how well each agent suits the current situation. There are several evaluators and each evaluates just one aspect of agents' suitability. Evaluators specialized in error handling know which error correction agents can be used in different error situations.

3.4. Presentation Management

Presentation management takes care of generating outputs to the user. Dialogue management produces output messages in a conceptual form. The final, lingual form of these messages is constructed in presentation management. This lingual form is generated to speech and played to the user by the communication manager.

The *presentation agents* do the actual language generation in the presentation component. Presentation management contains different agents for different kinds of output messages and there can also be several agents that can speak out the same message but in different ways. The agents can use all the information in the information storage. *Presentation evaluators* select the best presentation agent for each situation. In error handling we use specialized agents and evaluators, which handle error-related speech outputs.
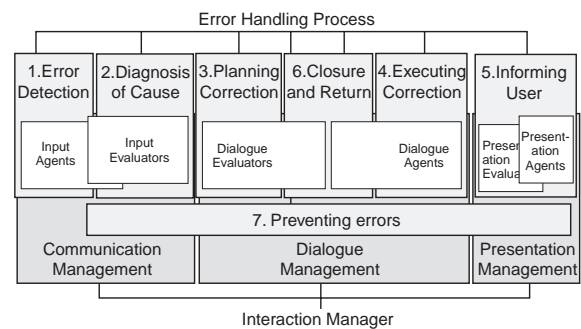

Figure 2: Error handling in Jaspis.

4. Error Handling in Jaspis

Next we describe how the error management architecture can be realized using the Jaspis framework. Figure 2 presents this realization. We show that it is possible to construct application independent and reusable error handling components. If necessary, application dependent components can also be built to maximize the quality of the system.

4.1. Error Detection

Error detection components can be implemented in Jaspis as input agents. We can write several small input agents for different kinds of errors and reuse these in different applications. It is also possible to write application dependent agents and keep their implementation separate from the application independent ones.

Recognition of an error can be done by the user or by the system. There are several ways the input agents can try to recognize an error. In the simplest cases it is possible to look at the recognition result confidence scores and use certain threshold levels etc. We can also use several different speech recognizers and compare the results. For example, input agents can use several different speech recognizers, such as a combination of normal, restricted vocabulary recognizer and an unrestricted vocabulary phoneme recognizer [4]. Input agents are also utilized when the user recognizes errors and acts to correct them. These error detection agents can use parameters such as content and duration [5] or prosodic features [6] of the user responses.

4.2. Identifying the Cause of the Error

Input evaluators are used as error reasoning components. They try to reason the cause of an error. The cause of the error reported by the user can also be reasoned. We can write separate evaluators for this task. These processes can include complex combination of different information sources, such as user model and dialogue history. We can also use multimodal information in cases where such information is available.

4.3. Designing Error Correction

Since the error correction is basically about changing the route of the dialogue, the error repair design is a task for the dialogue evaluators. Special evaluators check the type of the error, dialogue state and information of past errors and repairs and choose the best possible dialogue agent available.

One important part of the error correction design is avoiding error spirals, where an error correction procedure leads to another error. If the current error is followed with the very same error correction procedure, we can end up in endless loops of similar error/error correction combinations. Therefore, one dialogue evaluator needs to see what kinds of error repairs have been done in this dialogue and with what results, to be able to terminate possible error cycles. This evaluator can be one of the application independent reusable components.

4.4. Error Correction

The error correction is usually a small dialogue itself. This can be a single confirmation, for example, a yes/no question or a selection from a short list. In some cases the dialogue agents can even completely ignore the error correction procedures. In these cases it is the task of the presentation agents to inform the user about the situation.

The dialogue agents allow us to implement the correction dialogues in separate dialogue agents. These can be independent of the agents that implement the main dialogue. This keeps the dialogue agents very simple and easy to write and maintain.

It is also possible to write error correction dialogues that are application independent. This naturally requires separate dialogue and concept models from where the conceptual output messages can be constructed. However, if such models are used, we can re-use the error correction dialogue agents.

4.5. Informing the User

Special presentation agents and evaluators handle all speech outputs related to the error management. First, we have specialized presentation agents for error handling output. This way we can construct application independent agents which present all error situations to the user (though some of the agents may be application dependant if needed).

We can also have separate presentation agents for the situations where error correction has finished and we need to inform the user about that. These agents can be easily implemented by extending the basic agents by adding the additional information to the behavior inherited from the original agent. A specialized evaluator then selects these agents when needed.

4.6. Return to the Primary Dialogue

When an error correction dialogue is finished, we need to return to the original dialogue. Since the situation may have changed dramatically we may need to adjust the original dialogue flow. The selection of the following dialogue step is up to the dialogue evaluators. They should select the next dialogue agent suitable to carry on from the current situation.

4.7. Preventing Errors

Error prevention can take place in all interaction levels. When dealing with speech inputs we can have specialized input agents and evaluators that try to reduce errors by controlling input devices and utilize information from previous error situations. For example, if we detect that we are having many recognition errors we can reduce the size of the recognition vocabulary or use alternative vocabularies.

On the dialogue level, we can provide alternative dialogue agents to be more fail-safe for users with a lot of errors. These dialogues are not usually very efficient, but in problematic situations they can be the only choice for successful interaction.

Special output agents can be designed to provide alternative information in situations where errors are likely to appear. It is the duty of presentation evaluators to check if the user needs these agents.

5. Handling Errors in Practical Applications

To make our ideas more concrete we present examples of how the proposed architecture can be used to handle errors in real-world applications. The architecture is available as a part of the Jaspis distribution. We have used Jaspis to construct several applications, including e-mail application *Mailman* [7], bus timetable information service *Busman* and speech-based ubiquitous computing environment *Doorman*. All of these applications have very different logic behind them and especially different dialogue control strategies (state-based and form-based). Still, the same error handling methods and components can be used in all of these applications.

Error detection agents have close connection to the speech recognition engine. Basic agents can operate without any domain knowledge by just using the recognition results

from recognizers. Different recognizers (word-spotting, dictation etc.) can have different agents.

More advanced error detection agents can utilize context knowledge (dialogue history, user model) to detect possible errors. For example, an agent can check if the result is meaningful in the current context. In an e-mail application, it can find out that there are only four e-mails in the second folder although the recognizer has produced "*Read fifth e-mail*". This is a possible error situation and should be noted. Similarly, if the user uses alternative input mechanisms (such as rings doorbell several times), the error detection agent can alert the system that the recognizer is not performing well.

The error reasoning components, input evaluators, are used to reason the consequences that have caused the error. For example, if an error detection agent has spotted a possible error but the recognition confidence scores are good we can conclude that the error is semantic, i.e., the user is disoriented or has simply used the wrong words. This should be handled on the dialogue level.

When choosing a dialogue strategy to handle an error, domain independent components can be used to choose the proper error-handling agent. For example, "yes/no" confirmations can be used when we have only two possible choices. Different agents, for example those implementing selections from a list can be used when we have several choices etc. These kind of basic strategies can handle most situations in a domain independent way. Other dialogue agents, implementing the main dialogue management does not need to handle any error issues. However, as in all cases, agents with domain knowledge can be added. For example, if in the bus time table system we want to clarify, which district the user wants to go to, we can use a special agent for this purpose, instead of presenting a list of bus-numbers from the recognizer vocabulary.

When using the agent-based error handling in dialogue management, the error correction procedure is performed like a normal dialogue and if more errors occur during the correction dialogue, they can be processed the same way as normal errors. This makes the dialogue agents simpler and easier to implement. Furthermore, different dialogue management strategies, such as user-initiative and system-initiative strategies can be used for different users. This makes it possible to utilize the benefits of different dialogue management strategies [8].

In multilingual applications, language dependant parts should be modularized and separated from language independent components. Most importantly, the handling of inputs and outputs should not be processed in dialogue management. To support multiple languages efficiently, architecture should contain two kinds of components. First, it should support components for different languages. Second, it should support shared components between languages. The proposed error handling architecture is designed and implemented with these in mind.

6. Conclusions and Future Work

We have presented an advanced error-handling architecture for spoken dialogue applications. In this model the error handling process is divided into modular components, which deal with different the phases of error handling in a coordinated way. The proposed architecture operates on input, dialogue and output handling levels. Most of the error handling components can be application independent, but application dependent components can be easily added. We also introduced

how the proposed model is realized in the Jaspis speech application development environment and gave examples on how the architecture can be used in practical applications.

The presented model does not define what kind of techniques we should use in error handling. In this sense it complements other error handling models and techniques, such as those presented in [9]. Future work includes the implementation of a number of error handling components for different situations. We would also like to evaluate the usefulness of this approach using real-world user tests. Since the usefulness of error handling is very hard to measure, we will try different dialogue based metrics, such as those suggested in [10]. We hope to be able to release a common, reusable and extendable error-handling toolbox for the development of speech applications.

7. Acknowledgement

8. References

[1] Duff, D., Gates B., and Luperfoy S., "An Architecture for Spoken Dialogue Management", 6th International Conference on Speech and Language Processing, ICSLP 1996, pp. 1025-1028.

[2] Turunen, M. and Hakulinen, J. "Jaspis - A Framework for Multilingual Adaptive Speech Applications", 6th International Conference on Speech and Language Processing, ICSLP 2000.

[3] Luperfoy S. and Duff D., "Questions Regarding Repair Dialogs". CHI'97 Speech User Interface Design Challenges Workshop Position Paper.

[4] Gustafson, J., Lundeberg, M., and Liljencrants, J., "Experiences from the development of August - a multi-modal spoken dialogue system", ESCA tutorial and research workshop on Interactive Dialogue in Multi-Modal Systems, IDS'99.

[5] Hirasawa, J.-I., Miyazaki, N., Nakano, M. and Aikawa, K., "New Feature Parameters For Detecting Misunderstandings in a Spoken Dialogue System", 6th International Conference of Spoken Language Processing, ICSLP 2000.

[6] Litman, D., Hirschberg, J. and Swerts, M., "Predicting automatic speech recognition performance using prosodic cues", 1st Conference of the North American Chapter of the Association for Computational Linguistics, NAACL 2000.

[7] Turunen, M. and Hakulinen, J., "Mailman - a Multilingual Speech-only E-mail Client Based on an Adaptive Speech Application Framework", Workshop on Multi-Lingual Speech Communication, 2000.

[8] Walker, M., Fromer, J., Fabbrizio, G., Mestel, C. and Hindle, D., "What can I say?: Evaluating a spoken language interface to Email". ACM CHI '98. pp. 582–589.

[9] Suhm, B., Myers, B. and Waibel, A. "Interactive Recovery From Speech Recognition Errors", 4th International Conference on Speech and Language Processing, ICSLP 1996, pp. 865-868.

[10] Glass, J., Polifroni, J., Seneff, S. and Zue, V., "Data Collection and Performance Evaluation of Spoken Dialogue Systems: The MIT Experience", 6th international Conference of Spoken Language Processing, ICSLP 2000.