MARKKU TURUNEN

Jaspis - A Spoken Dialogue Architecture and its Applications

ACADEMIC DISSERTATION
To be presented, with the permission of
the Faculty of Information Sciences of the University of Tampere,
for public discussion in the Auditorium Pinni B 1096,
Kanslerinrinne 1, Tampere, on March 13th, 2004, at 12 o’clock.

Acta Electronica Universitatis Tamperensis 325
University of Tampere
Tampere 2004
Abstract

Speech can be an efficient and natural way for communication between humans and computers. Many practical applications have been constructed, but the full potential of speech applications has not been utilized. In addition to technological shortcomings, the development of speech applications lacks suitable techniques, methodology and development tools. For example, mobile and multilingual communication needs flexible and adaptive interaction methods which take into account the needs of different users and different environments. This dissertation addresses the following question: what kind of a system architecture do advanced speech applications require? The following challenges are specifically addressed: How could the system architecture support advanced interaction techniques? How could application development be supported by suitable models, methodology and tools?

This dissertation introduces the Jaspis speech application architecture that has been designed to support adaptive and flexible human-computer interaction techniques. This work also presents several applications constructed on the Jaspis architecture. Two multilingual e-mail applications, and two timetable applications are presented to provide concrete examples. The challenges of pervasive speech applications are introduced in the context of an ubiquitous computing application. The findings from the use of the Jaspis-based applications are reported.

Several Jaspis-based interaction models and tools are introduced to facilitate the development of practical applications. The focus is on human-computer interaction issues, and solutions are presented for tasks such as error management, Wizard of Oz experiments and corpora collection. Finally, an enhanced version of the Jaspis architecture is presented. The Jaspis2 architecture focuses on distributed and concurrent spoken dialogues, which are needed in pervasive applications, and in general to provide more natural interaction models. The main contribution of this work, the Jaspis architecture, will be released simultaneously with this dissertation as an open source software for speech application researchers and developers.
Preface

This dissertation has taken a long time – six years of constructive research carried out concurrently with running applied science projects. It has required a lot of compromises, practical software engineering work and administrative responsibilities. Still, I could not imagine any other way to accomplish this kind of work. Most importantly, I have not spent these years alone in front of my monitor. There are a lot of people who have supported me and contributed to this work. I would like to express my profound gratitude to them.

First of all, I thank my supervisor, Professor Kari-Jouko Räihä. He has made this research possible in the first place, and has provided unfailing support. In particular, he is the founder and head of the TAUCHI research unit (Tampere Unit for Computer-Human Interaction), which has been an extraordinary place to carry out constructive HCI research. It has been a privilege to be a part of its rise to an internationally recognized research center.

This work would not have been possible without the existence of the Speech-based and Pervasive Interaction Group. The collaborative efforts of the group have produced numerous application prototypes, which are indispensable in this research area. The contribution of the group members has helped me to find time to write this thesis during the hectic reality of everyday project work. I wish to express my deepest gratitude to all present and past members of the group, and especially to Jaakko Hakulinen, who has co-authored much of the work done during these years.

The Department of Computer Sciences has provided good facilities for this research. There are a lot of staff members who have provided the support needed to complete this work – thank you all. Jori Mäntysalo maintained the infrastructure, including the servers used to run our applications – his technical support during the last years has helped me to save precious time for actual research work.

I want to thank Professor Michael McTear for consenting to act as opponent in the public defense of this thesis, and the reviewers, Dr. Alexander I. Rudnicky
and Dr. Bernhard Suhm. In particular, I am very grateful to Dr. Suhm for his thorough review and constructive suggestions. Thanks also to Virginia Mattila for language checking.

This work has required close collaboration with industry and other research units. The work was carried out in the following research projects: DUMAS (European Union), New Methods and Applications of Speech Technology (National Technology Agency), EMFi-based User Interfaces (National Technology Agency), A Speech-based Bus Timetable Service (Ministry of Transport and Communications), USIX Interact (National Technology Agency), User Interfaces for Ubiquitous Computing (Academy of Finland), Mobile User Interfaces (Hewlett-Packard), Speech User Interfaces (National Technology Agency). I just hope that there would have been enough space to thank all industrial and academic partners individually.

Hundreds of days spent in conferences, project meetings and seminars have sometimes been exhausting, but always rewarding. During these trips I have met many colleagues who have influenced my work. Participation in research networks such as MUMIN (Nordic Network for MUltiModal Interfaces) and COST 278 (Spoken Language Interaction In Telecommunication) has provided an excellent way to disseminate the work and collaborate with other researchers. I have also participated in the Tampere Graduate School in Information Science and Engineering (TISE).

Finally, my loving thanks to Heli – this would never have been possible without her support, both at work and at home - who could ask for more?

Contents

1 INTRODUCTION...................................................................................................................1
  1.1 Research Questions and Contribution.................................................................2
  1.2 Thesis Overview........................................................................................................5

2 SPEECH-BASED HUMAN-COMPUTER INTERACTION........................................9
  2.1 Speech Application Technologies........................................................................10
    2.1.1 Levels Of Speech Processing........................................................................11
    2.1.2 Speech And Speaker Recognition Technologies.......................................12
    2.1.3 Speech Synthesis Technology......................................................................17
  2.2 Types of Speech Applications.............................................................................18
    2.2.1 Conventional Speech Applications...............................................................20
    2.2.2 Multilingual Speech Applications.................................................................22
    2.2.3 Multimodal Speech Applications...................................................................23
    2.2.4 Pervasive Speech Applications......................................................................26
    2.2.5 Adaptivity In Speech Applications.................................................................27
  2.3 Speech Interface Design.......................................................................................29
    2.3.1 Speech-based Communication.......................................................................30
    2.3.2 Conversation Techniques.............................................................................33
    2.3.3 Error Handling...............................................................................................35
    2.3.4 Confirmations..................................................................................................38
    2.3.5 Feedback, Help And Guidance........................................................................39
    2.3.6 Universal Commands.....................................................................................40
    2.3.7 Non-speech Audio..........................................................................................41
  2.4 Evaluation...............................................................................................................44
2.4.1 Technology Evaluation ................................................................. 44
2.4.2 Application Level Metrics ............................................................. 47
2.4.3 Empirical Experiments And Data Collection ................................. 50

3 DEVELOPMENT OF SPEECH APPLICATIONS .................................... 55

3.1 Speech System Components .............................................................. 55
  3.1.1 Speech Recognition Components .................................................. 56
  3.1.2 Speech Interpretation Components ............................................... 58
  3.1.3 Dialogue Management Components .............................................. 60
  3.1.4 Output Generation Components ................................................... 60
  3.1.5 Speech Synthesis Components ...................................................... 61
  3.1.6 Other Components ................................................................. 62

3.2 Speech System Architectures ............................................................. 63
  3.2.1 Software Architectures ............................................................... 63
  3.2.2 Architectural Styles ................................................................. 65
  3.2.3 Architecture Styles In Speech Systems .......................................... 72

3.3 Dialogue Management in Speech Application ................................... 73
  3.3.1 Dialogue Management Strategies And Initiative ............................ 75
  3.3.2 Dialogue Control Models ............................................................ 78
  3.3.3 Dialogue Management In Speech Applications .............................. 85

3.4 Multimodal and Multilingual Inputs and Outputs .............................. 86
  3.4.1 Multimodal Interaction In Speech Systems ..................................... 86
  3.4.2 Multilingual Speech Inputs And Outputs ....................................... 89

3.5 Example Systems ............................................................................. 90
  3.5.1 Dialogue Manager Architectures: MALIN ..................................... 91
  3.5.2 Client-server Architectures: Galaxy-II .......................................... 92
  3.5.3 General Agent Architectures: OAA .............................................. 94
  3.5.4 Dialogue Manager Oriented Agent Architectures: WITAS .......... 96
  3.5.5 Dialogue Manager As Controller: MITRE .................................. 98
4.5.3 Devices And Engines.................................................................152
4.5.4 Clients, Connections, Servers And Handlers.......................154
4.6 System Framework....................................................................155
4.6.1 General System Structure.........................................................156
4.6.2 XML In Jaspis.........................................................................158
4.6.3 Information Access Protocol....................................................159
4.6.4 Standards And Markup Languages..........................................166
4.7 Summary and Comparison with Other Architectures.................168

5 JASPIS-BASED APPLICATIONS..................................................175
5.1 E-mail applications: Mailman and AthosMail............................175
5.1.1 Mailman Functionality..............................................................177
5.1.2 Application Architecture..........................................................183
5.1.3 Speech Interface Issues.............................................................189
5.1.4 Multilinguality In The E-mail Domain......................................193
5.1.5 User Experiences....................................................................197
5.1.6 Summary.................................................................................202
5.2 Bus timetable Systems.................................................................203
5.2.1 The Busman System.................................................................204
5.2.2 Interact System........................................................................210
5.2.3 User Experiences.....................................................................216
5.2.4 Summary.................................................................................220
5.3 Pervasive speech applications....................................................221
5.3.1 Doorman System.....................................................................222
5.3.2 Awareness Information............................................................225
5.3.3 Additional Functionality............................................................227
5.3.4 Application Structure...............................................................228
5.3.5 Wizard Of Oz Experiment.........................................................231
5.3.6 Summary.................................................................................233
## 6 Interaction Techniques and Tools

6.1 Support for Error Handling in Speech Applications

6.1.1 Requirements For Error Handling

6.1.2 Jaspis Error Handling Model

6.1.3 Implementation Issues

6.2 A Model for Modular Output Generation

6.2.1 Implementation Issues

6.3 Application Visualization and Control Tools

6.3.1 Visualization Tools

6.3.2 Application Control Tools

6.4 Evaluation and Demonstration Tools

6.4.1 Wizard Of Oz Experiment Tools

6.4.2 Design Sessions And Demonstrations

6.5 Support for Speech Corpora Collection

6.5.1 Data Collection

6.5.2 Representation Of Interaction Data

## 7 JASPIS² Architecture

7.1 Challenges of New Application Areas

7.1.1 Distributed Dialogues

7.1.2 Concurrent Dialogues

7.1.3 Active Systems

7.1.4 Adaptation And Information Sharing

7.1.5 Jaspis And New Challenges

7.2 Jaspis² overview

7.3 Triggers and Transactions

7.4 Information Access

7.5 Collections
8 CONCLUSIONS .............................................................................................................................. 285
  8.1 Review ............................................................................................................................. 285
  8.2 Contribution .................................................................................................................. 287

REFERENCES .................................................................................................................................. 291
1 INTRODUCTION

The development of speech applications has been an active research area for long. During the last few years, fairly sophisticated research prototypes have been developed in numerous projects, such as in the DARPA Communicator project [Pellom et al., 2001; Rudnicky et al., 1999; Seneff & Polifroni, 2000]. This large-scale project focuses on travel information services. Similar applications have been developed in many smaller national and international projects.

Commercial systems are also available. For example, a suite of TellMe [TellMe, 2003] applications provides a wide selection of voice-controlled services such as financial information and news. Further examples of commercial applications are timetable services, such as train and flight information services.

Despite the numerous research prototypes and some commercial services, the development of speech applications is still not a mature field, and the full potential of speech-based interaction has not been utilized. Especially in the case of minority languages such as Finnish, speech applications are not common.

The reasons for the slow progress in the field include the lack of suitable frameworks, methodology and tools for the construction of practical applications. Speech application development is usually done in ad hoc ways and using in-house tools, although some commercial development environments are available, such as Nuance Voice Platform [Nuance, 2003] for VoiceXML development.

In this dissertation the focus is on development aspects of speech-based interaction. Speech technology still has many challenges, but practical
CHAPTER 1: INTRODUCTION

applications can be constructed with current technology. In order to do this, we need interaction techniques which take into account both technology limitations and the nature of speech-based communication. In the case of multilingual applications differences between languages must be taken into account.

A lot of work has been done in the field of human-computer interaction research, and issues such as prompting and evaluation methods have been covered in many publications. Similarly, techniques for dialogue management and fusion of multimodal information have been investigated extensively in recent years. Several development frameworks and tools have also been introduced.

Nevertheless, many research challenges exist. For example, error management, multilingual interaction and evaluation tools have still not been covered very well. Similarly, dialogue management techniques, for example, have still mostly focused on interaction models that originate from text-based dialogue systems, and do not take into account the nature of speech-based human-computer interaction. This same inflexibility can be seen in system architectures, which often consist of pipelined, monolithic black-box components.

The key issues in contemporary speech applications are adaptive and flexible interaction methods. These are needed in many application areas, such as in multilingual and multimodal applications. In order to facilitate the development of speech applications, we need advanced techniques, models, methodology, tools and system frameworks.

1.1 Research Questions And Contribution

In this dissertation I address the research challenges mentioned. The main research question is: what kind of a system architecture do advanced speech applications require? In particular, I address the challenge: how could the system architecture support advanced interaction techniques? In addition, the
following challenge is addressed: how could application development be supported by suitable models, methodology and tools?

To address the research questions, the Jaspis architecture is introduced. A set of design principles for an advanced speech application architecture based on a survey in speech-based human-computer interaction and speech application development is introduced. According to the design principles, the architecture should be **generic, extensible, adaptive and flexible, modular and distributed**, and it should support **collaborative application development, reusability and standards**. I will present the limitations of existing architectures and explain how the Jaspis architecture supports the principles mentioned.

An important perspective in this dissertation is multilinguality and minor languages. One of the motivations is to support the development of Finnish speech applications, and in general to support the techniques which are suitable for minority languages and multilingual applications. Because of technical and cultural reasons, the focus of development is often different than in the case of major languages. Multilinguality is addressed in various ways in this dissertation, as discussed in the following paragraphs.

The main contribution of this dissertation, the Jaspis architecture, is both a conceptual model and a concrete speech application development framework. The architecture provides in particular a flexible interaction coordination model, an explicit system level adaptation mechanism, a shared system knowledge and an efficient distribution model. The architecture introduces the agents – managers – evaluators paradigm. Compact agents support highly modular systems and reusability. The evaluators are a key concept in adaptation. Managers coordinate other components in a flexible way, allowing highly distributed systems to be constructed. Various Jaspis components, extensions and tools provide support for collaborative application development. The use of XML for information representation supports standards.

Compared to existing architectures, Jaspis supports adaptivity and alternative components in a novel way. It provides explicit system level adaptivity available
CHAPTER 1: INTRODUCTION

for all system components. The adaptation mechanism is general, distributed, and can easily be extended. In addition to the overall coordination of system modules, Jaspis provides a layered model for the internal coordination of modules. Interaction management is based on managers and agents. They make it possible to construct highly distributed systems in a flexible, but still coordinated manner. Together with the shared information management, indirect messaging between components is achieved. This overcomes the lack of coordination, which is a limitation in many blackboard architectures.

Most general system architectures support elemental tasks only, such as communication between components. Specialized development environments, on the other hand, may provide high-level components for tasks such as dialogue management, but they are usually limited to specific solutions (such as VoiceXML or finite-state based dialogue control models). Jaspis is a general architecture and it does not force developers to use predefined solutions for tasks such as dialogue management; instead it offers general models for interaction level tasks and contains extensions for speech technology integration.

For practical speech application development Jaspis provides a layered communication management subsystem with interfaces to several technology components, such as speech recognizers. The communication management subsystem supports virtual devices for high-level control of vendor specific engines and their timely-depending coordination, including a basic support for multimodal fusion. The communication subsystem, like other parts of the architecture, is highly distributed and multiple communication protocols can be used simultaneously.

In order to validate the usefulness of the Jaspis architecture I will present several Jaspis-based applications. These applications show how Jaspis can be used for application development in many domains, and how it addresses their development challenges, such as multilinguality. Likewise, it shows how matters arising from application development have influenced the development of the architecture. I will also present how the Jaspis architecture can be used to
1.1 - RESEARCH QUESTIONS AND CONTRIBUTION

implement general models for tasks such as error handling and modular multilingual output management in speech applications. In addition, several Jaspis tools supporting application development tasks are introduced, such as interactive visualization tools for testing, debugging and demonstrations, Wizard of Oz tools for usability tests and support for a general annotation format for corpora collection.

The Jaspis architecture is used for research prototypes in various national and international research projects. In addition, a Jaspis-based bus timetable system has been implemented for a local company and it is in everyday use in Tampere.

In order to facilitate development of advanced speech applications for both research and commercial purposes the Jaspis architecture is freely available and will be released at the same time as this thesis as an open-source software under GNU Lesser General Public License [LGPL, 2003].

Finally, the challenges of emerging application areas are addressed. Pervasive computing applications and the need for richer interaction methods create new challenges for speech application development. I will present the challenges of future speech applications and show how the second generation of the architecture, Jaspis², addresses them. In particular, I will explain how the Jaspis architecture has been extended to support concurrent interaction in a coordinated and synchronized way.

1.2 Thesis Overview

This dissertation consists of two parts. The first part contains Chapters 2 and 3. It introduces the field of speech applications and the most important issues of application development. Both human-computer interaction and software development perspectives are covered. The constructive part of the dissertation (Chapters 4-7) introduces the Jaspis architecture, several Jaspis-based speech applications, experiences on their use and various interaction models and development tools. The challenges of new application areas are discussed, and I
CHAPTER 1: INTRODUCTION

present how the Jaspis\textsuperscript{2} architecture addresses these challenges. The constructive part is motivated by the first part and takes into account both technical and human-computer interaction perspectives. Next, an overview of each chapter is presented.

Chapter 2 introduces the main aspects of speech-based human-computer interaction. This includes the constraints of the underlying speech technology, the nature of speech-based communication, and interaction design principles and evaluation methods for speech applications. These are presented from the application development viewpoint, and different types of speech applications and their needs are presented. The human-computer issues presented provide background for techniques developed in the constructive part of the dissertation.

While Chapter 2 focuses on the human-computer interaction aspects of speech applications, the viewpoint in Chapter 3 is more technical and oriented on software development. The issues covered include the identification of speech application components, and their relations and organization as a part of the system architecture. The main architectural models are presented, and their usage in speech applications is discussed. Approaches to dialogue management and the coordination of multimodal and multilingual inputs and outputs are presented. Chapter 3 ends with a description of speech application architectures and development tools.

Chapter 4 presents the Jaspis architecture. Its design principles, architectural basis and various components and modules are explained. Human-computer interaction and development aspects presented in the previous chapters are taken into account here. The most important implementation and software development issues, such as reusability, are presented. Chapter 4 ends with a comparison with other architectures.

Chapter 5 introduces several speech applications constructed using the Jaspis architecture. These applications include two multilingual e-mail applications, two bus timetable applications and an ubiquitous computing application. Chapter 5 also reports several user experiments that have influenced the development of
the interaction techniques and tools presented in Chapter 6. Specific areas covered include the handling of multilinguality and the presentation of structured speech outputs, among others.

Chapter 6 continues along the lines of the previous chapter, but focusing on general interaction and application development solutions which may be used across applications and domains. I present general models for error handling and modular output management in speech applications. Several Jaspis tools supporting application development tasks are introduced for testing, debugging, demonstrations and corpora collection. The motivation for this chapter comes from the previous chapters. Especially the human-computer interaction issues presented in Chapters 2 and 5 are addressed here.

In Chapter 5 and 6 experiences and examples of the various projects in which the Jaspis architecture has been used are presented. Several new challenges have arisen during these projects, and they are summarized in Chapter 7. The principles of the resulting Jaspis$^2$ architecture are presented, including the support for concurrent interaction. This work is still fairly preliminary, and years of work, and experiences from new generation speech applications will be needed to complete it. Other future work is outlined and conclusions are presented in Chapter 8.
2 SPEECH-BASED HUMAN-COMPUTER INTERACTION

The use of speech is expected to make the human-computer interaction more natural and efficient than it has been so far. Many successful speech applications have been constructed, but speech is still far from being the most common input and output modalities. In order to successfully construct speech applications, system designers and implementers must know the nature of speech, the characteristics of the underlying technology and interaction techniques and models that are suitable for speech-based human-computer communication. In this chapter I present the most important issues of speech applications from this viewpoint.

First, speech application technologies, i.e. speech recognition and speech synthesis are discussed from the application development perspective. Technical presentation is omitted and the focus is on those issues which have an impact on speech-based human-computer interaction. For a technical introduction, a guidebook such as [Huang et al., 2001] is recommended.

After the technology overview, an introduction to speech applications is presented. This includes conventional speech applications, such as telephony systems, and new application areas, such as multimodal and pervasive applications. Applications are presented from the viewpoint of application development, including interface issues and the underlying system architecture. The last two sections present application design principles and evaluation methods. Issues such as the nature of speech and speech-based communication, evaluation metrics and empirical experiments are also presented.
CHAPTER 2: SPEECH-BASED HUMAN-COMPUTER INTERACTION

The purpose of this chapter is to present an overall view of the factors which affect speech application development. This information is needed when speech-based interaction techniques are implemented and underlying system architectures are constructed. There are many issues which should be taken into account, and the diversity of speech-based communication emphasizes the need for flexible and adaptive interaction technologies, which are presented in the constructive parts of the dissertation.

2.1 Speech Application Technologies

Although speech is the most natural form of communication between humans, it has several limitations when used in interaction between humans and computers. Some of its advantages which are present in human-human communication cannot be used in human-computer interaction. There are multiple reasons for this, both technical shortcomings and those which are related to the nature of speech as a communication modality. In many cases, communication is not based on the best possible solutions, but instead the technology limits choices and even dictates the design [Mane et al., 1996].

In order to successfully use speech in the interface, we must know the state of the art and reality of speech technologies, as well as what kind of interface element speech is. First, it is important to realize that speech-based human-computer interaction is different from written language and spoken communication between humans. Secondly, speech technology has many technical limitations, and it is not realistic to believe that these will disappear in the near future. In this section I present those issues of speech technology which affect spoken interaction.
2.1 - SPEECH APPLICATION TECHNOLOGIES

2.1.1 Levels Of Speech Processing

Speech can be structured in many different ways depending on its use. One useful categorization is illustrated in Figure 1. The categorization is adapted from [Schmandt, 1994: 9]. It contains eight levels, which are organized into three layers. The core speech technologies (speech recognition, speech synthesis) belong mainly to the bottom layer (acoustic, articulator and phoneme levels), while applications deal mainly with the top layer (discourse, pragmatic and semantic levels). The middle layers (syntactical and lexical levels) are needed both in technology and application areas. When speech architectures are considered, we are dealing mainly with interaction methods, but many issues related to the core technologies should be taken into account as well.

In this work the focus is on the top layer, while some of the middle layer aspects are also considered. The top layer processes speech depending on its meaning – on the word, sentence and discourse levels. These are used in most parts of speech systems, especially in natural language understanding, dialogue...
management and output generation modules. The middle layer consists of lexical and syntactic levels. Speech is represented here as words and sentences. These are used, for example, when synthetic speech outputs are generated (including the prosody control) and when speech recognition grammars are constructed.

2.1.2 Speech And Speaker Recognition Technologies

Speech recognition technology has been studied actively during the last three decades. Highly sophisticated recognizers have been constructed, but still inadequate speech recognition is considered as the main obstacle in speech systems. Two reasons for this are unrealistic assumptions and improper application design, which does not take erroneous recognition technology into account. Speech recognition technology has many limitations, and they will remain present in the near future, but with appropriate interaction techniques most limitations can be overcome and successful applications can be constructed.

Problems of speech recognition come from multiple sources. One of the main reasons is variations in the speech signal. Variations can be categorized differently, for example as linguistic, acoustic and speaker variability [Zue et al., 1996] or as context, style, speaker and environment variability [Huang et al., 2001]. Not all sources of variability can be controlled by a speech system, but some environmental and speaker variability, for example, may be controlled to some extent. Examples include the use of speaker and environment adapted recognition grammars and acoustic models.

Spontaneous spoken language contains a lot of ungrammatical elements. Hesitations, false starts, repairs and overlapping are common in everyday speech, likewise different non-speech elements. Although they may be a good source of information at the upper levels of speech processing (e.g. at discourse level), they cause complexity at the lower levels and make speech harder to understand. Most non-speech elements in spoken communication are ignored by
speech recognizers and in certain conditions, for example when speech-to-noise ratio is low, visual speech recognition can make a great impact on recognition accuracy [Massaro & Stork, 1998; Liu et al., 2002].

It is also noteworthy that people talk differently to computers than to other humans [Fraser & Gilbert, 1991]. People alter their speaking styles when they talk to computers, and it is reasonable to believe that we will develop new kinds of communication styles when speech applications become more common. Similarly, it is not realistic to believe that people will communicate with computers in the same ways as they communicate with other humans. Currently, we lack both knowledge and novel interaction methodologies for speech-based human-computer interaction. One of the main motivations of this dissertation is to make it possible to utilize new interaction techniques in speech applications.

Since speech recognition technology does not support unlimited recognition, the properties of speech recognizers are important factors when speech-based applications are designed and implemented. Recognizers have various capabilities, and they are suitable for different purposes. Even inside a single recognizer there may be numerous parameters which affect the performance. For application developers, it is crucial to be aware of the capabilities of the recognizers, their limitations and the choices available. The technology evaluation, which lasts the whole lifetime of the application development, is an essential part of the system development. This is presented in more detail in Section 2.4.

In Table 1 I have summarized main characteristics of speech systems from the point of view of the speech recognition technology. I have included those features which are most important when a speech recognizer’s suitability for a particular task is evaluated. The properties of the task and the recognizer should match as closely as possible. From the point of view of a speech recognition task, the most difficult properties are in the far right column. Not surprisingly, those are also the most desired properties in speech applications. Next, I present these properties in more detail.
CHAPTER 2: SPEECH-BASED HUMAN-COMPUTER INTERACTION

Vocabulary and language

<table>
<thead>
<tr>
<th>Vocabulary size</th>
<th>small</th>
<th>middle-size</th>
<th>(very) large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grammar (LM)</td>
<td>phrases</td>
<td>CFG</td>
<td>n-gram</td>
</tr>
<tr>
<td>Extensibility</td>
<td>fixed</td>
<td>changeable</td>
<td>dynamic</td>
</tr>
</tbody>
</table>

Communication style

<table>
<thead>
<tr>
<th>Speaker</th>
<th>dependent</th>
<th>adaptive</th>
<th>independent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speaking style</td>
<td>discrete</td>
<td>continuous</td>
<td>spontaneous</td>
</tr>
<tr>
<td>Overlap</td>
<td>half-duplex</td>
<td>barge-in</td>
<td>full-duplex</td>
</tr>
</tbody>
</table>

Usage conditions

<table>
<thead>
<tr>
<th>Environment</th>
<th>clean</th>
<th>normal</th>
<th>hostile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel quality</td>
<td>high-quality</td>
<td>normal-quality</td>
<td>low-quality</td>
</tr>
</tbody>
</table>

Table 1: Properties of speech recognition systems.

2.1.2.1 Vocabulary size and recognition grammars

Vocabulary size and recognition grammars (language models, LM) characterize the interaction possibly better than other properties. By using small vocabulary recognizers (around 10-200 words) it is possible to construct many spoken dialogue applications, even with conversational abilities, but some application areas are not possible. For example, it is possible to construct fairly sophisticated speech-only e-mail applications with a couple of dozen words in the vocabulary. However, if we want to build bus timetable systems we usually need at least a middle-sized vocabulary. Jaspis-based applications presented in Sections 5.1 and 5.2 offer examples of these areas.

Recognition grammars also have a great influence on the interaction model. If we can change or dynamically construct grammars, we can design the system to be context-sensitive and utilize dialogue knowledge, interaction history and user profiles to use personalized recognition grammars. In this way, we can use
2.1 - SPEECH APPLICATION TECHNOLOGIES

several small and robust vocabularies in different situations. On the other hand, if we cannot change grammars, we should design the grammar and the system interaction model to be suitable for all situations and to all users.

Some current speech systems offer both n-gram and context free grammar language models, and an option to use large or even very large vocabularies (over 50,000 words). N-gram models are most usable in dictation and other tasks which require very large vocabularies, and in which usage conditions are optimal and the recognizer can be adapted to the speaker. On the other hand, in hostile environments, where speakers are unknown, even small vocabularies and restricted grammars may cause problems.

In practice, most speech recognizers have serious problems with open grammars and large vocabularies. Furthermore, defining large vocabulary grammars or collecting relevant data for n-gram models can be a laborious task. Fortunately, most speech applications do not need large vocabularies and can be designed to work with small or medium sized vocabularies and limited grammars if they are properly designed.

2.1.2.2 Communication style

Communication style may vary from speaker dependent, discrete speech and half-duplex communication to speaker independent, continuous and full-duplex communication. Each of these parameters has an impact on the interface. For example, the barge-in capability allows users to interrupt system outputs, and this way the outputs can be designed to be longer and more informative than without the barge-in capability. In some applications it is possible to use speaker dependent or adaptive models if the recognizer supports them, but in public applications, where users are not known, speaker and gender independent models are the only possible choices. Speaker adaptive models may be very robust, especially if customized vocabularies are used, which makes their use justified in some application areas.

With current recognizers there is no need to speak in a discrete manner. However, even if the speaking style can be continuous, it usually helps if words
are pronounced clearly and properly. In practice, recognizers are not very robust for spontaneous speech. Word-spotting can be a good choice for some applications, but it is not supported efficiently by most recognizers.

Overlapping speech is not in fact a property of speech recognizers, but rather a property of a system. Barge-in affects prompt design, but also dialogue design. With a half-duplex interface the speech recognition grammar must be designed to support strictly turn-based communication. With full-duplex support the user and the computer may speak at the same time, regardless of each other. In a typical situation the user wears headphones and uses a close-talk microphone. Barge-in is typical for telephone-based communication and brings more flexibility to the interface than the half-duplex interface. It is important to note that even if the underlying technology supports these features, applications must also provide support.

2.1.2.3 Usage conditions

Usage conditions range from clean to hostile environments and from high-quality to low-quality channels. High-quality conditions may include close-talk microphones in single-person offices. In low-quality conditions, mobile phones may be used in noisy, public places. More typical conditions can be shared offices and homes in which basic desktop microphones are used. Even with state-of-the-art recognizers, the recognition accuracy may be dramatically impaired if usage conditions do not meet those used in the recognizer training phase. Recognizers may contain different acoustic models for different conditions. The lack of a proper acoustic model may make the recognizer unsuitable for certain tasks.

Finally, differences between recognizers for different languages may be considerable, for example there are highly sophisticated recognizers available for English users, but only a couple of limited recognizers are available for Finnish users. When applications for multiple languages are constructed, the limitations of different recognizers must be taken into account and the system interaction
must be adapted according to the resources available. This is one of the issues which should be addressed in the system architecture and interaction techniques levels.

Speaker recognition, i.e., speaker identification and speaker verification are not widely used in practical applications, but I expect them to become more important in the future. They are especially useful in pervasive applications, in which speakers must be identified in a transparent manner and the interaction should be adapted to each particular user. In Section 5.3 I present how speaker recognition is used in the ubiquitous computing system Doorman.

2.1.3 Speech Synthesis Technology

Unlike speech recognition technology, speech synthesis technology is fairly matured, and full-featured synthesizers are available for most languages. There are no major obstacles when synthesizers are used in speech applications. Some limitations exist, however. Although synthesized speech is easily intelligible, it is not very pleasant to listen to (English “natural voices” are an exception), which limits its use in certain application areas. Truly multilingual synthesizers are also rare, and in some application areas this causes problems. In some cases it is possible to use recorded speech instead of synthesis, but for most applications this option is too restrictive and inflexible.

Different synthesis technologies (articulatory, concatenative and formant synthesis) have their own characteristics. From the viewpoint of domain coverage we can divide synthesizers into two categories: general synthesizers and limited domain synthesizers. Limited domain synthesizer can be an appropriate solution for tasks where the best possible quality is required and the domain does not require free-form text to be synthesized. One example is the limited domain synthesis used in the Carnegie Mellon Communicator system [Black & Lenzo, 2000] and the USI project [Shriver et al., 2000]. The main drawback of this approach is that limited domain synthesizers require labor and data intensive customization work, and this is usually not affordable, especially
in non-commercial applications. For most situations, general synthesizers offer acceptable quality.

Prosody control is one aspect which affects speech quality. In general, speech synthesizers handle only basic prosody control, and natural sounding speech requires content knowledge and other information sources in order to be rendered efficiently. Proper use of different prosodic elements (pauses, pitch etc.) at application level can make speech outputs both more intelligible and pleasant to listen to. In fact, in some sense synthesizers can produce higher quality speech than average human speakers [Hakulinen et al., 1999].

Multilingual applications, where speech outputs may contain multiple languages (in some cases even intrasententially) can be hard to make natural or even intelligible. In practice, multilingual speech outputs appear in many application areas and their problems are very diverse. In Section 5.1 I introduce an example case in the form of a multilingual e-mail application Mailman.

In summary, speech technologies have many limitations, which can be compensated by an appropriate interface design. In this section I have presented most important characteristics of speech application technologies from this viewpoint. With all this technical variation, the key issue is the integration of components into a working system [McTear, 2002]. This is an important motivation for the constructive part of this dissertation.

2.2 Types Of Speech Applications

Successful speech applications have been constructed in many research projects and several commercial applications are also available. In this section I provide a summary of various speech application areas.

One way to approach speech applications is to divide them into two categories based on the modalities used. In the first category are applications which use speech as their main modality, while to the second category belong multimodal
applications where speech is not the dominating modality. An alternative approach is to classify systems into conventional speech applications (telephone applications, dictation system, voice portals) and new application areas (pervasive computing, mobile applications). Here I divide speech applications into conventional, multilingual, multimodal and pervasive applications, because each of these areas has different development and research questions. In addition, I present how adaptivity may be applied in these different application areas. In practice, it is not important how applications are classified, but it is important to realize that speech applications are much more than traditional spoken dialogue systems and there is a wide range of research problems associated with them beyond the speech recognition [Suhm, 2003].

As there are multiple application types, applications can also be constructed from many different perspectives by using different approaches. The most advanced systems are usually produced in multidisciplinary projects. A typical project consist of people working with core technologies (speech recognition and synthesis, natural language processing), while others work with interaction technologies (dialogue management, human-computer interaction). People with different backgrounds may have overlapping solutions for the same problems. In this work I consider various approaches to speech application development and take these into account in the constructive part of this dissertation. The diversity of speech applications especially provides many challenges for the underlying architecture and for the interaction techniques.

The terms speech application and spoken dialogue system are used through this dissertation. The latter is often used to refer to applications which emphasize the use of natural spoken language. Often these systems operate on a turn-by-turn basis [Fraser, 1997] and the interaction is modeled as a dialogue between the computer and the user. This term is widely used by authors with a background in speech technology or dialogue management research. Sometimes it is also used to distinguish conversational (natural language) systems from more command and control oriented systems. It may also be used to refer to techniques used in dialogue management. Bernsen et al. [1998] have categorized speech systems
and use the term “interactive speech system” to emphasize the language processing parts of speech applications in contrast to speech applications which do not use natural language processing techniques. In this dissertation I mainly use the term speech application to refer to all speech-based applications regardless of their focus or implementation details.

2.2.1 Conventional Speech Applications

The first speech applications were telephone-based interactive voice response (IVR) systems, which used speech outputs and telephone keys for interaction. These applications are still popular and possibly the most important examples of widely used commercial speech applications. Typically, these applications are designed to replace human operators (i.e. speech is used as a substitute, see Section 2.3). Problems with these systems include that the DTMF interface may be awkward and user satisfaction poor, if the service does not match the quality offered by human operators. On the other hand, these applications can be very useful for users if they introduce new services which are not possible or affordable with human operators (or in any other way). From the telephone operator's point of view these applications may offer huge savings in the long run. The current trend is that DTMF inputs are replaced by speech inputs. This may promote both greater user satisfaction and cost savings in some application areas, as presented for example by Suhm et al. [2002].

In addition to IVR applications, many other forms of telephony applications have dominated the field. These include information services, such as timetable, weather forecasting and banking services, e-mail applications and voice portals. Most of the well-known commercial applications and research prototypes fall into this category. Examples of timetable systems include the Philips Automatic Train Timetable System [Aust et al., 1995] and the different systems developed in the Communicator project [DARPA, 1998]. A well-known weather forecasting service is MIT-Jupiter [Zue et al., 2000a], while MailCall [Marx &
2.2 - TYPES OF SPEECH APPLICATIONS

Schmandt, 1996] and ELVIS [Kamm et al., 1998] are probably the best known speech-based e-mail systems. Voice portals which integrate several applications into an unified speech interface include SpeechActs [Yankelovich et al., 1995; Martin et al., 1996] and TellMe [TellMe, 2003]. An example of a graphically enhanced portal is WebGALAXY [Lau et al., 1998], which combines a www-browser with the telephony interface.

Most of the current speech application research prototypes are telephony applications. They may be fairly sophisticated and include state-of-the art recognizers, natural language understanding and response generation components and use advanced techniques for dialogue management. Still, the integration is a key issue for successful applications, as noted by McTear [2002]. In this dissertation I refer to many telephony applications and present four such applications in the constructive part of this dissertation. Furthermore, the Jaspis architecture, which is the main contribution of this work, was originally designed for telephony applications, although it has subsequently been extended to other application areas as well.

Desktop applications form another popular area of speech applications. Most desktop applications are either targeted at dictation or the control of existing graphical applications, such as word-processors or spreadsheet programs. Dictation applications make it possible to use speech instead of typing. Some dictation applications allow also to transcribe recorded speech. Many companies offer dictation applications, either for general use or customized for special domains, such as medicine. Dictation applications are relatively popular within special groups. From the application viewpoint dictation systems represent more technological advances in speech recognition than state-of-the-art in interactive system development, although usability is an important issue in dictation applications. There are still many interesting challenges related to these applications (e.g. error correction [Suhm et al., 1999]). One problem with current dictation systems is that they rely quite heavily on visual feedback and multimodal inputs. This limits their usefulness in speech only applications, even if the underlying speech technology makes it possible to dictate.
CHAPTER 2: SPEECH-BASED HUMAN-COMPUTER INTERACTION

Command and control applications make it possible to accomplish tasks which are usually done by using the mouse and the keyboard. For example, the user may start programs, control windows and menus using his/her voice. Nevertheless, they have not been widely used except for special purposes. Such applications may become more popular when computers become small and embedded in the environment (personal digital assistants, ubiquitous computing). Currently, they are mostly used in interfaces which are intended to replace existing graphical interfaces. This was a popular field of research in the late eighties and early nineties [Edwards, 1988; Mynatt & Edwards, 1992; Mynatt & Weber, 1994]. Today basic command and control capabilities are included in dictation systems.

I personally feel that command oriented applications are underrated. Most people working with spoken dialogue systems pay no attention to them. Currently the focus is on conversational systems, but these are not suitable for all tasks and situations, as Rosenfeld et al. [2001] state in their analysis. Especially when pervasive computing applications become more popular, we need different approaches, since in these applications conversational, turn-based communication is not a suitable solution. In this dissertation I consider all application types and present solutions which are suitable for as many types of applications as possible.

2.2.2 Multilingual Speech Applications

Translation systems are one example of multilingual speech applications. These systems translate spoken utterances between users. Typical applications include booking systems and other similar applications, where two human participants have a common goal, such as booking a hotel room or a trip, and the computer translates their spoken utterances between languages in real-time. Examples include JANUS [Waibel et al., 1991], ATR-MATRIX [Takezawa et al., 1998] and VERBMOBIL [Wahlster, 2000]. The current trend in these applications is
2.2 - TYPES OF SPEECH APPLICATIONS

translation between western and oriental languages, and the focus is on natural language translation technologies. This work does not have a special focus on translation systems, although the Jaspis architecture is suitable for them as well.

Another form of multilingual applications is systems allowing users to access information services using multiple languages. A typical example is MIT JUPITER [Zue et al., 2000a], a weather forecasting information system which has been ported to Japanese as MOKUSAI [Zue et al., 2000b] and to Mandarin Chinese as MUXING [Wang et al., 2000a]. The systems mentioned are monolingual in the sense that one system instance serves only users in one language. From the viewpoint of technology, they often use different technologies for different languages, such as different recognizers, synthesizers and natural language processing components. Some other components, such as the system architecture, information processing components and interaction techniques may be language independent and support multilinguality.

A third type of multilingual applications uses multiple languages with the same user. The need for multiple languages arises from the application domain. Typical examples of this approach are speech-based e-mail systems, such as the Mailman system [Turunen & Hakulinen, 2000b]. The use of multiple languages is essential in these applications. For example, it would be hard to imagine a monolingual e-mail system for Finnish users, since a typical mailbox contains messages written in multiple languages. Furthermore, single messages containing multiple languages are surprisingly common in Finnish. The proper handling of multilingual e-mail messages is far from trivial and such applications must be designed to take multilinguality into account on all system levels. The challenges of multilingual speech applications will be discussed in the following chapters and the Mailman system is presented in Section 5.1.

2.2.3 Multimodal Speech Applications

Multimodal speech systems have been a popular research subject, but still multimodal systems are rather rare in everyday use. From the historical
perspective, multimodality offers promising opportunities, as presented by the famous Put-That-There system [Bolt, 1980]. Combined pointing and speech inputs offered a natural way to communicate, and later authors added gaze direction tracking to disambiguate other modalities [Bolt & Herranz, 1992; Koons et al., 1993]. In the study by Weimer and Ganapathy [1989] speech inputs made a dramatic improvement in the interface which was based on hand gestures. Similar results have been reported in other studies as well [Hauptman, 1989; Hauptman & McAvinney, 1993].

The systems mentioned show many benefits of multimodal interfaces: common arguments favoring multimodality state that they prevent errors, bring robustness to the interface, help the user to correct errors or recover from them and add alternative communication methods to different situations and environments [Cohen & Oviatt, 1994]. Multimodal interfaces may also bring more bandwidth to the communication and it is possible to simulate other modalities, for example in the case of disabled users [Smith et al., 1996]. Multimodal interface may also help users to understand how the system works [Terken & te Riele, 2001]. It should be noted, however, that multiple modalities alone do not bring these benefits to the interface: currently there is too much hype in multimodal systems, and the use of multiple modalities may be ineffective or even disadvantageous. Oviatt [1999b] has presented common misconceptions (“myths”) of multimodal interfaces. Similar misconceptions are usually related to the use of speech as an input modality. Not surprisingly, many of the “myths” which Oviatt presents relate to speech.

Among the most studied multimodal speech applications are so-called “talking heads” or “speaking agents”. Good examples are systems built at the KTH, such as August [Gustafson et al., 1999a; Gustafson et al., 1999b] and Adapt [Gustafson et al., 2000]. In these systems the main interface element is audio-visual speech synthesis, which uses anthropomorphic figures to convey facial expressions and head-movements.
2.2 - TYPES OF SPEECH APPLICATIONS

In addition to talking heads, systems with animated interactive characters have been constructed. Good examples of this approach are systems built at DFKI [André & Rist, 2000] and SICS [Höök et al., 1999]. These systems are often focused on multimedia presentation techniques and agent technologies rather than on the use of speech. The way in which presentation techniques are used in these systems is noteworthy, however, and the presentation management of the Jaspis architecture is partially motivated by these systems.

Another active area of research in multimodal applications is information kiosks (intelligent kiosks). In these systems modalities such as speech and haptics (e.g., touch-screens) are used to provide interface for users in public places. An early example is the MASK project [Gauvain et al., 1996] and a more recent example is the SmartKom project [Wahlster et al., 2001]. A common argument favoring the use of speech in these applications is that speech is a useful modality for environments in which pointing devices are not appropriate. On the other hand, the public nature of speech and disturbances in the environment may limit its usefulness. As noted in [Christian & Avery, 2000], the use of speech in kiosk environment is not automatic success.

Disambiguation of error-prone modalities using multimodal interfaces is the main motivation for the use of multiple modalities in many systems. Oviatt investigated multimodal speech systems from this perspective and she found that error-prone technologies can compensate each other, rather than bring redundancy to the interface and reduce the need for error correction [Oviatt et al., 1997; Oviatt, 1999a]. Various techniques for multimodal error handling have been studied in different contexts, for example in dictation applications [Suhm et al., 1999; Suhm et al., 2001b]. Error handling and technical aspects of multimodal interfaces, such as fusion mechanisms, are discussed in more detail in following chapters.
2.2.4 Pervasive Speech Applications

Pervasive applications, i.e. ubiquitous and mobile computing applications, introduce both new opportunities and challenges for speech as an interface element. Speech is a suitable element for pervasive applications because of the size of the devices, and the lack of large visual displays and familiar interaction devices, such as the mouse and the keyboard. By embedding systems into everyday environments (so-called “intelligent environments”) the focus of speech applications shifts from turn-based, user-initiative conversational systems in a more proactive and distributed direction.

An example of an intelligent environment is the MIT Virtual Room project [Coen, 1998]. The idea of its speech interface is to use context knowledge to allow multiple speech applications to be active at the same time with no need for central control. A technique called recognition forest is used for this purpose [Coen et al., 1999]. A recognition forest is a collection of specialized grammars for various interaction situations. Similarly, a more user-centric approach is the personalized dialogue management of the SesaME system [Pakucs, 2003]. The spoken interaction in the Virtual Room project is still fairly traditional. In the same project, gaze direction has been found to be a promising approach for activating and deactivating speech recognition in such environments [Oh et al., 2002].

Other well-known applications are Office Monitor and Nomadic Radio. Office Monitor [Yankelovich & McLain, 1996] is a system which serves visitors in an office while its occupant is away. Nomadic Radio [Sawhney & Schmandt, 2000] is a wearable speech system, which offers several information services, such as listening to e-mail and voice-mail messages, calendar events and news. In addition to synthetic speech, the system uses non-speech audio, speech recognition and tactile inputs. These systems motivated us to build the Doorman system, in which we investigate how speech and audio interface can be used to serve employees and visitors in an office environment. We use non-speech
2.2 - TYPES OF SPEECH APPLICATIONS

audio, such as walking sounds to convey awareness information to users in a natural and non-intrusive way [Mäkelä et al., 2003].

Systems involving robots are examples of applications in which the context plays an important role. These systems may include multiple participants, which makes the dialogue rather different from traditional single-user and single-system dialogues. A recent example is presented by Matsusaka et al. [2001].

Mobile speech applications have been a hot topic in current research, but not many practical systems have been introduced. Some work has been done in assistive technologies, which enable disabled users to access mobile devices by using speech [Manaris et al., 1999]. Speech-based telephone interface can be also used as an alternative for mobile devices [Tang et al., 2001]. In our ongoing mobile application project we investigate how mobile devices can be used to provide unified access to various speech systems.

As presented in this section, speech applications form a very diverse field and although the main focus is still on traditional applications, it is reasonable to expect that this situation will change. Multilingual, multimodal and pervasive applications will change the focus from the development of basic technologies to interface issues and new interaction methods and techniques are needed. The key issues are adaptiveness and flexible interaction. From the system viewpoint, this means that there should be either numerous very specific architectures or speech architectures must be suitable for a wide range of application areas and support adaptivity and distributed and concurrent dialogues. These are strong motivations for the Jaspis architecture. Next I discuss how adaptivity may be applied in speech applications.

2.2.5 Adaptivity In Speech Applications

Adaptivity may refer to various aspects in speech applications. For example, adaptive techniques are used in speech recognition to adapt acoustic models to different environments and users. The term adaptivity sometimes refers to the use of machine-learning techniques. These can be used in many places in the
CHAPTER 2: SPEECH-BASED HUMAN-COMPUTER INTERACTION

applications, such as in language processing and dialogue management components. The Interact bus timetable system is one example, and another is the AthosMail e-mail application. These Jaspis-based applications are presented in Chapter 5.

Yet another type of adaptive applications is systems which are targeted at special groups, such as disabled users. In this context adaptivity refers to ways which help different users to use applications by using customized interaction methods and techniques. Special groups constitute strong motivation for speech applications, and also for adaptive applications. In this dissertation the term adaptivity refers to all the ways which make the interaction suitable for different users and usage situations. It does not refer to any specific technique or user group.

In speech-based human-computer interaction users have more diverse ways of communication than in graphical user interfaces, although currently many speech applications are quite limited. Cultural differences are vast, likewise individual differences and preferences. Novice users and experienced users may want the interface to behave completely differently, for example to be system-initiative instead of mixed-initiative.

In speech interaction the characteristics of speech, such as its serial nature, bring limitations, which should be compensated using adaptive techniques which take different users into account. Similarly, technical problems can be solved by using adaptive interfaces which allow successful interaction even when the underlying technologies fail. Overall, adaptation means that the interaction changes due to user's characteristics, actions and the environment. An example of the benefits of interaction level adaptivity is reported by Litman and Pan [2002]. In their experiment an adaptive system outperformed a non-adaptive system with novice users.

When adaptivity is considered from the interaction level viewpoint, the handling of speech inputs and outputs, dialogue management and user modeling are the most important areas. Multilingual applications are examples of adaptive
2.2 - TYPES OF SPEECH APPLICATIONS

applications. Other examples are agent systems, which serve as assistants for the user. An example of such systems is the Circuit-Fix-It Shop system [Smith & Hipp, 1994], which assists the user in accomplishing electronic circuit repairing tasks using intelligent agents.

In this work I focus on the interaction level issues of adaption. The presented Jaspi architecture is designed for adaptive applications, which means that the architecture supports multiple similar components, which are selected dynamically. Selection is based on a distributed evaluation process. In this way, applications can be adaptive at the architectural level and they are not only collections of adaptive components. This is one of the main contributions of the architecture and this dissertation.

2.3 Speech Interface Design

Speech interface design may have a great impact on overall system quality. Many technical limitations can be compensated with properly designed speech interface [Kamm, 1994]. On the other hand, if the interface is poorly designed even the use of state-of-the-art technology may yield poor results. One problem of speech interface design is the lack of general guidelines and best practices to follow. Some work has been done, for example the guidelines used in the evaluation of the Danish Dialogue Project [Bernsen et al., 1998], the EAGLES handbook [Gibbon et al., 1997] and the framework-based approach presented in [Suhm, 2003], but in general more knowledge is needed.

Next, I present the most important interface design issues which should be considered during the system design, implementation and evaluation phases, and also supported by interaction techniques and the underlying system architecture. A more technically oriented view of speech applications, including dialogue management, response generation and natural language processing is presented in the next chapter.
2.3.1 Speech-based Communication

Speech has several characteristics which makes it an attractive interface element. At the same time, there are several aspects which should be taken into account. Next I present the main features of speech as an interface element. Based on this, I then present various motives for the use of speech.

One argument favoring speech in applications is its naturalness. Speech application developers hope to utilize all this when they use speech as an interface element [Kamm, 1994]. However, for various reasons we cannot use all human-human communication methods in speech interfaces. In fact, some common phenomena, such as overlapping speech, cause serious problems in human-computer interaction.

It must be remembered that speech communication is an acquired skill, and different principles may be more efficient or preferred when we communicate with computers using speech. In current applications, human-human communication methods are often imitated too directly. Corpus based systems especially, which often seek communication methods from human-human experiments, often suffer from this straightforward approach.

Speech can be a very efficient method of communication. A classic study by Chapanis [1975] showed that speech was the most efficient communication modality in interactive problem solving tasks between humans. Compared to other modalities speech was superior: it was more than twice as efficient as writing, the next most efficient modality. Chapanis also found that unimodal speech communication was only slightly slower than multimodal communication in which all other modalities were used. On the other hand, the nature of many human-computer tasks is different, and speech was found to be less effective than mouse and keyboard in typical office application tasks [Damper et al., 1996], although the results often suffer from dependency on the current state of technology, and contradictory results have been achieved [Poock, 1980; Martin, 1989]. One reason for this is that there is not clear separation between speech as
a modality and how speech technology performs in particular tasks. Therefore, results are often relevant only for a particular type of technology.

One advantage of speech is its ability to bring more bandwidth to the interaction when a multimodal interface is used. Especially in situations where multiple concurrent tasks are performed simultaneously more efficiency can be gained if multiple modalities can be used concurrently.

Spoken utterances can be very expressive. By using speech it is possible to perform operations which may be hard to do with other modalities. In particular, many tasks which are difficult to do with the direct manipulation paradigm can be easier to do with spoken commands. One example is the selection of similar objects from a large set of scattered objects. It could be hard to select objects on the basis of their features such as color and shape with direct manipulation interface, if objects are scattered around the display area, or they cannot be displayed at once. By using (spoken) language homogeneous objects can be selected with a single utterance. Speech can also be used for direct access, for example by using short-cuts instead of deep menu structures.

Efficiency should not be the only design factor. Speech is a popular modality and even preferred by some people over other modalities, although it may be less efficient. One reason for this can be that people use different methods for problem solving tasks. According to Bradford [1995], some people use visual-spatial methods to solve tasks, while others use an acoustic-verbal approach. Acoustic-verbally oriented users may prefer speech.

Speech has several limitations when used in human-computer interaction. These can be categorized for example to six categories presented in [Suhm, 2003]. One major problem is the memory limitations of users. When no visual information is available, users need to memorize all the meaningful information. Although this is often considered to be the case in command and control applications, natural language applications suffer from this problem as well. Users need to remember the dialogue state and all relevant information that the system has provided. This is because speech is a temporal medium. Users also need to know how to speak
CHAPTER 2: SPEECH-BASED HUMAN-COMPUTER INTERACTION

to the system: there is no spoken language interface which can understand unconstrained speech. For non-native speakers it may be easier to remember commands than to know how to speak “correctly” or “naturally” to the system. For these reasons, guidance is an important factor in speech applications.

When speech outputs are used, there are several factors which should be taken into account. Speech is a slow, temporal and serial medium, all of which make speech outputs hard to design and they should definitely be designed to be different from text outputs. On the other hand, this should not be exaggerated. For example, it is often assumed that long lists should be presented in small chunks, but recent studies [Suhm et al., 2001a] indicate that longer lists than usual may be more suitable in some situations, since they do not need memorizing and multi-layer menus are not needed. When used in non-private environments the public nature of speech should be taken into account. Pervasive applications are one application area where speech outputs should be adapted to the particular users and usage situations in order to safeguard the user's privacy.

People have many communication skills and habits which can be difficult to utilize in speech interfaces, but which may cause problems if they are not present in the communication. These include spontaneous, continuous style of speaking, overlapping and the use of pauses to indicate the structure of the conversation.

We can find six motives for speech as an interface modality. Speech may be the only possible modality, it may be the most efficient modality or it may be the most preferred modality. These are strong motives for using speech, since no other modality can outperform it. Speech can also be supportive, alternative or substitutive modality. The first two cases apply mostly to multimodal interfaces, in which speech can be used in various ways with other modalities. In a supportive role speech can be used to achieve more bandwidth and make the interface more robust, as for example presented by Oviatt [1999a]. As an alternative modality speech may be suitable for some users, while other users may prefer other modalities. In this role speech can be used in situations when no other modalities are currently present. If speech substitutes some other
modality (i.e. it is not the only possible modality for this situation), and it is not more efficient or preferred than the modality that it replaces, problems are likely and user satisfaction may decrease.

It is crucial to know the reasons why speech is used in the interface. If speech is not used properly, the interface may become unacceptable. For example, if the speech application replaces a human operator or if errors are serious, it is likely that people will demand low error rates, but in situations where speech is the only possible modality people may accept higher error rates. It is very different to design applications for visually impaired users, who usually have a strong motivation for using speech, than for average users, who may prefer graphical user interfaces and have low motivation.

2.3.2 Conversation Techniques

Speech interface constitutes conversation between the computer and the human, even if command and control style utterances are used. Because of this, we should always take conversational principles into account. Users have learned many communication skills in human-human interaction, and even if the communication between humans and computers is different in many ways, still many of those learned skills are present in speech-based human-computer interaction as well. And although we cannot utilize all the useful communication skills used in human-human communication, we should be aware of them and know the limitations arising in their absence. At the same time, we should utilize conversation techniques which are suitable for communication between human and computer. These issues are presented next.

*Turn-taking* techniques can be difficult to implement in speech-only interfaces. In natural conversation, people use non-verbal cues for turn-taking, such as gestures and eye movements, as well as acoustic cues, such as pitch variations. It has been argued that prosody is a potential cue for predicting dialogue structure [Hirschberg, 1999], but most of the current recognizers and speech systems do not support this functionality. Pauses, which are efficiently used for turn-taking
in human-human communication, may be ambiguous in speech interfaces, since there is no visual information to resolve their meaning. Turn-taking may be especially problematic in interfaces where barge-in is not supported, since it is natural for humans to interrupt the other speaker. If supported, this can be even more efficient in speech interfaces, since the user does not need to be polite to the computer.

**Prompting** is a key issue for successful interaction. People adapt to the way that the computer speaks and use both the same style and words which occur in the computer's turns. Prompts can guide the interaction in the desired direction and help both speech recognition, natural language understanding and dialogue management components to understand the user utterances better. On the other hand, even simple prompts, such as yes/no questions may cause misunderstanding if they are poorly constructed [Hockey et al., 1997].

Progressive prompting and tapering can be very useful techniques, because they allow the system to adapt to both experienced and novice users. In an optimal situation the system must support both system-initiative and mixed-initiative (user-initiative) dialogue strategies. Both strategies have their own advantages and disadvantages, and there are differences between application areas and users. In a study by Walker et al. [1998], the system-initiative strategy was preferred, although the mixed-initiative strategy was more efficient. With expert users, the mixed-initiative approach may be more suitable, while for novice and regular users system-initiative interface may perform better. If possible, the system should be able to use both approaches and alternate between them during the interaction when necessary.

**Barge-in** capability, or the ability to interrupt system prompts, allows the system to use longer utterances, for example in lists which are typically recommended to be short because of users' memory limitations. Even though barge-in can be efficient, the interface should also be suitable for situations in which barge-in is not supported. People may also prefer not to use barge-in capability [Schmandt, 1994], and in some situations, for example when a cellular phone is used,
recognition accuracy may be low with barge-in and interruptions are difficult because of the telephone layout.

*Navigation* and *menus* are important interface elements in many speech systems. Especially, in IVR systems navigation in menu structures is key to successful interaction. In these systems the success of interaction relies on the design of menus and other structured interface elements. Even if the functionality is not arranged into structured groups, many applications have structural elements in which the user navigates. For example, e-mail reading applications contain many different structural elements, such as lists and tables. When users read messages, they navigate inside the messages, and when they list and select messages, they navigate inside the mailbox. Navigation can take place in natural language applications as well as in command and control applications.

2.3.3 Error Handling

Error handling is a crucial feature in speech applications. Errors reduce both the usefulness of an application and user satisfaction. Successful error handling methods can make even applications with poor recognition accuracy usable. On the other hand, poorly constructed error handling may bring unwanted complexity to the system and cause users more annoyance than that caused by problems.

Error management is usually separated into error detection and error correction. Luperfoy and Duff [1996] have presented a more detailed four phase model, and later extended it to cover five phases [Luperfoy & Duff, 1997]. Their phases are *error detection, diagnosis of cause, planning a correction, executing the correction* and *closure and return to the primary dialogue*. Their last phase is rather complex and we have divided it into two phases, which are *informing the user* and *closure and return to the primary dialogue* [Turunen & Hakulinen, 2001a]. We also included *error prevention* as a seventh phase. A description of these phases follows.
1. **Error detection.** The system detects an error (recognition error or semantic error) or detects that the user tried to correct an error that has occurred earlier. In many applications, error detection is left to the user, but both the user and the system should be able to detect errors [Kamm, 1994]. Semantic errors especially need automatic error detection methods, since users do not usually notice them. Automatic error detection is not trivial, however, and recognition confidence scores alone, for example, may not be sufficient indicators.

2. **Diagnosis of cause.** The system analyzes the causes leading to the error so that it can better correct the error and prevent further errors. Without error diagnosis both the system and the user may repeat the same errors, thereby causing an error spiral. Unfortunately, error analysis is very hard to do.

3. **Planning a correction.** The system decides how to handle the error, i.e., it selects a suitable error correction strategy. In an ideal case both error costs and error correction costs are calculated and compared. If the error correction costs are lower than the costs that the error causes, error correction is justified. Otherwise, the error correction should be omitted. The initiation of an error correction dialogue should never be automatic, since this can lead to inefficient and unusable interface.

4. **Executing the correction.** The system initiates an error correction dialogue, such as selection from a list, to correct the error.

5. **Informing the user about the error correction.** The system informs the user about the error correction, if necessary, and about the reasons that led to it. The system also informs the user about 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. This can lead to a totally new dialogue state, since the error or its correction may have changed the context.

7. **Preventing errors.** In some cases, the system can modify its behavior to match the user's actions better and prevent further errors from occurring. Examples include the change from the mixed-initiative dialogue strategy to the system-initiative dialogue strategy, and the use of alternative recognition networks. Errors can also be prevented by offering the user more help. These are examples of dynamic error prevention. Error prevention can also take place beforehand, when the application is being designed. For example, the system vocabulary should be designed and tested so that recognition errors are minimized.

Errors can also be reduced by using other modalities. Oviatt investigated speech-based multimodal interfaces and found that a multimodal interface can reduce recognition errors significantly, for example to make recognition accuracy with accented speakers to be the same as with native speakers [Oviatt, 1999a]. Suhm et al. [2001b] have investigated multimodal error correction techniques for speech interfaces in the context of dictation applications and found novel techniques that do not require a keyboard.

Different error handling methods for speech applications have been examined and techniques have been presented. The usefulness of different methods has been evaluated and methods, frameworks and toolkits have been provided [Hudson & Newell, 1992; Suhm et al., 1996; Mankoff et al., 2000]. In Section 6.1 I present an error handling model which uses Jaspis agents and evaluators to implement the error handling phases mentioned.
2.3.4 Confirmations

Confirmations can be seen as part of error handling (for example, to aid error detection), but they can also be seen as interface elements of normal dialogue [O'Neill & McTear, 2002]. They are used to ensure that participants have a common understanding of the information. Schmandt [1994] distinguishes explicit and implicit confirmations. Explicit confirmations force the user to react, which may result in inefficient interaction. Nevertheless, when the consequences of an error are costly or impossible to undo, explicit confirmations are preferred. For example, if the user is going to give an order or delete objects, it is usually desirable that an explicit confirmation takes place if there is any reason to believe that an error has occurred.

When implicit confirmations only provide the user with feedback, they leave it to the user to take action if an error has occurred. Implicit confirmations are more lightweight and in some cases do not even disturb users. They are a better choice if unwanted actions can easily be undone. Especially in real-time applications explicit confirmations may cause more problems to the interaction than they solve. It is difficult to achieve real-time interaction with error correction in some applications, such as in virtual worlds [Turunen, 1998]. Although implicit confirmations are lighter than explicit confirmations they should not be overused. The information bandwidth is narrow, and especially with spoken outputs should be used economically.

One problem with implicit confirmations is that they may cause complexity in the interface. When explicit confirmations are used, the system can focus on error handling. When implicit confirmations are used, the system must monitor the normal dialogue for possible error corrections. Users may initiate error correction in many ways and this should be modeled in speech recognition and dialogue components. System utterances may also be lengthy if all possible misrecognized items are confirmed implicitly at the same time. Sometimes it is more efficient to do confirmations later in the dialogue flow.
2.3.5 Feedback, Help And Guidance

Navigation and feedback are key issues in speech interface design. One of the problems in speech user interfaces is that users may easily become disoriented and not know what to say or what the system is doing. The system should provide users with feedback and guidance when necessary. These functions are especially helpful for novice users [Mane et al., 1996]. It is reasonable to assume that speech applications are task oriented and dialogue participants are cooperative [Smith & Hipp, 1994; Bernsen et al., 1998]. Therefore, we can assume that system guidance is effective in helping the user to achieve his/her goal and provide the necessary information to the user.

Effective spoken feedback may be difficult to implement. Since speech is a slow output medium and may overload the user's cognitive memory easily, feedback should be brief enough but provide all the necessary information. The famous language philosopher Grice [1975] specified a set of maxims which are effective in cooperative conversations. These relate to the quality, quantity and manner of communication. One of Grice’s maxims states that a message should contain necessary information, but not less or more. Following this principle means that feedback should be used in an adaptive way (e.g. tapering), since experienced users may become annoyed, especially if barge-in is not supported. It is also meaningful to separate feedback, help and guidance and provide only those which are needed or requested in particular situations.

Help functionality in speech-only systems is a relatively little investigated area. Usually speech applications contain only initial prompts, which may be rather lengthy. User-initiative systems usually list available commands when requested. System-initiative applications may give all the help needed in system prompts. More natural language oriented systems can rely on their conversational capabilities. In practice, none of these approaches works very well and speech applications usually offer too little help. We have recently experimented with integrated tutors, which provide context-sensitive guidance for the user while he or she is using the application [Hakulinen et al., 2003].
CHAPTER 2: SPEECH-BASED HUMAN-COMPUTER INTERACTION

In multimodal applications help can be provided efficiently by using other, non-temporal modalities. This may also be an efficient approach when off-line help (i.e. tutorials and guides) is used [Terken & te Riele, 2001; Kamm et al., 1998]. In speech-only applications written instructions can be very useful.

2.3.6 Universal Commands

One solution to the “what should I say” problem is the concept of universal commands, i.e. unified commands which are available in multiple applications and are familiar to the user once learned. This approach is used in the Universal Speech Interface project [Rosenfeld et al., 2000], whose goal is to create a standardized interaction style for speech applications. The idea is that “interaction universals” provide more restricted user inputs than natural spoken language, but they are not so limited as typical application specific command and control interfaces. “Universals” include solutions for help, orientation, navigation, error correction and general system interaction [Shriver et al., 2001].

The use of universal commands requires learning. In the USI project tutorials are an essential part. Shriver et al. [2001] used two kinds of tutorials, general and application specific. They argue that application specific tutorials alone are not sufficient. The main issue, however, is to find suitable keywords for commands. They used Internet surveys to find keywords [Shriver & Rosenfeld, 2002]. Universal commands were also investigated in [Telephone Speech Standards Committee, 2000], where a unified set of commands is used for problem situations (clarification, navigation and leaving the system).

In a way, universals are similar to other “toolkit” based approaches, such as SpeechObjects [Burnett, 2000], which are reusable VoiceXML scripts. OOPS [Mankoff et al., 2000] is a toolkit for error handling and offers a set of ways to handle recognition errors in different applications. The difference with universals in comparison to many other approaches is that universals also include guidelines for the interaction and try to provide a coherent “look-and-feel” for
speech applications. Many voice portals, for example SpeechActs [Yankelovich et al., 1995; Martin et al., 1996] and TellMe [TellMe, 2003] provide common interfaces for several applications.

I believe that universal commands, in a general sense, will be an important issue for speech applications. The lack of standards in speech applications has led to a situation in which applications work in different ways and users need to learn the interface every time they switch applications. It is unrealistic to believe that unrestricted natural language understanding can be used in all speech applications. Universal commands are also one way to achieve universal access for all users. In the Jaspis architecture this challenge is addressed by using agents which can be reused between applications and provide universals for several situations, such as for error handling and other general interaction tasks.

2.3.7 Non-speech Audio

Non-speech audio can be an efficient way to convey information in speech applications, especially in auditory-only interfaces where it can be used as an additional output channel. Auditory information can provide feedback on the user actions and the state of the dialogue. Non-speech audio does not have all the limitations of speech, for example it is often faster than speech, since even a very short audio message can convey meaningful information in a small amount of time.

Non-speech audio can be categorized as auditory icons, earcons and music. Auditory icons are related to natural sounds which are familiar to users. They convey meaning which is derived from their origin rather than acoustic properties. Typical examples are sirens, sounds related to closing and opening of things etc. As with speech communication, their use is related to cultural conventions. If misused, they can easily convey misleading information. Auditory icons are often recorded sound samples, which limits their parameterization. Several techniques have been presented to manipulate and synthesize auditory icons [Gaver, 1993].
Music can be used in auditory icons. Music has several structural parameters, such as rhythm, timbre, pitch etc. These can be used to convey information in the same way as auditory icons. Music can be used to provide a non-obtrusive, coherent aural environment which is also aesthetically pleasant. Music is often used by playing recorded songs when the system is busy. This kind of use does not convey any other information except that the system is busy at the moment. More elaborate use of music can represent graphical entities, such as diagrams by using music [Alty & Rigas, 1998].

Earcons represent a different approach. Instead of conveying information on their origin, they represent information in their acoustic parameters. As earcons are usually not known to the users beforehand and they may be abstract, their use requires learning, but on the other hand they are free from the limitations the natural sounds often have and once learned, they can be easily recognized. Earcons can be used, for example, for navigation and representing structural information [Brewster, 1998]. In a way, earcons are similar to universal commands, and they may work with universal commands [Shriver et al., 2000]. Non-speech audio can be used to provide overview of large data sets, which are difficult to express using speech. In this approach, the data is presented aurally in the same way as in many visualization tools, i.e. information details are not presented, but the overall structure of the data is provided. For example, in [Hudson & Smith, 1996] a technique called an audio glance is used to provide an overview of users' e-mail messages. Non-speech audio has also been used to provide information on how programs work (program visualization and algorithm animation) and it could be interesting to use audio in speech application development tools.

Non-speech audio elements are used in many applications which are targeted at visually impaired users. One well-known application is EmacSpeak [Raman, 1997], which includes changeable auditory icon sets (themes) for common system functions. Non-speech audio is also used in many pervasive computing applications, such as in mobile personal digital assistants [Hindus et al., 1995;
del Galdo & Rose, 1999] and in ubiquitous computing systems [Mynatt et al., 1998; Sawhney & Schmandt, 2000]. Whatever form of auditory information is used, it should be used in a coherent way. Audio sounds should work together and create a meaningful auditory atmosphere. When used in several applications, they should be used in the same way. The use of standardized sound banks could help application developers to choose proper auditory elements for their applications.

From the viewpoint of system design, non-speech audio should be supported in speech applications better than it has been supported so far. Developers working with dialogue management or natural response generation usually do not take non-speech audio into account. While the use of non-speech audio in speech systems is not very advanced, I believe that it will be used more in the future (e.g. in pervasive applications) and better techniques for non-speech audio will be developed. In this dissertation I present how both non-speech audio and speech can be supported in the Jaspis architecture. A recent example of the effective use of non-speech audio are walking sounds [Mäkelä et al., 2003] that convey awareness information in a non-intrusive way.

Overall, speech-based communication is very diverse. Different user groups, environments and application domains require flexible interaction methods. While the key issue from the technology viewpoint is integration of components, adaptiveness and flexibility are needed to implement various interaction methods from the interface design viewpoint. They should be supported on all system levels. If the underlying architecture supports adaptive system components and interaction techniques, speech applications can be built to be more efficient in different usage conditions. Supporting for adaptivity and flexibility is one of my main motivations for the constructive part of this dissertation.
2.4 Evaluation

In this section I present various evaluation methods for speech applications. In general, most of the standard evaluation and testing methods and metrics from the speech recognition and usability communities can be applied to speech applications. Here I focus on speech-specific application level methods. Data collection and requirement analysis represent important parts of design and implementation processes. WOZ (Wizard of Oz) experiments are an important data collection and usability testing technique. They aid the development of applications and technical components, such as speech recognizers, to adapt to the task at hand. All of these aspects are relevant to speech application development, including the underlying architectures, and are discussed in this section. Later on in the constructive part of the dissertation I present various Jaspis tools that support the evaluation techniques described in this section. In the following chapters I also report several user experiments in which the Jaspis tools are used successfully.

2.4.1 Technology Evaluation

There are two main approaches to speech application evaluation. Technology evaluation measures how well technical components, such as speech recognizers and language understanding components perform in isolated tasks. Usability evaluation relates to human factors and interface issues. Next, technology evaluation is presented. Usability aspects are presented in following sections.

The most common recognition performance measure in speech systems is word error rate (WER), or the related metric word accuracy rate. These are calculated as the percentage of recognition errors and correct words, correspondingly, of the words spoken. WER is defined by the following formula:

\[ \text{WER} = \frac{\text{deleted} + \text{substituted} + \text{added words}}{\text{words in original message}} \cdot 100 \]
WER can be calculated by using a dynamic programming algorithm which aligns words according to the maximum substring matching problem. Most speech recognition toolkits, such as HTK [2003] include tools to calculate WER on the basis of the transcribed data and recognition results. *Sentence error rate* and *sentence accuracy rate* are related measurements, but evaluate at the level of whole sentences instead of individual words.

Lexical and syntactical level measurements are useful when speech recognizers are evaluated, but from the viewpoint of applications more higher-level technology evaluation measures are needed. These can be used to evaluate natural language understanding and dialogue management components. *Sentence understanding error rate* and *sentence understanding accuracy rate* correspond to the number of erroneously and correctly understood sentences. In practice this means that if a sentence is mapped to the same representation as a reference sentence, it is understood correctly. In this measurement the words that are not needed to extract the meaning from the sentence are not taken into account. Similarly, if two words refer to the same entity, they are treated as equal. *Concept error rate* and *concept accuracy rate* are related measures which indicate the rate of incorrectly and correctly, correspondingly, identified concepts from the sentence. Concept accuracy rate can be calculated in a similar way to word accuracy rate, but concepts are used instead of words. In general, the calculation of word error rate can be applied to other elementary units.

The quality of speech outputs can be measured by using two factors, intelligibility and pleasantness. Nowadays the intelligibility of synthetic speech is usually quite high, although listening to synthetic speech requires more cognitive resources than listening to recorded speech [Lai et al., 2000]. Intelligibility can be divided into segmental intelligibility, which indicates how accurately spoken sentences have been received by the user, and a comprehension, which measures how well the spoken sentences are understood. Segmental intelligibility can be measured in a similar way to WER by comparing transcriptions and reference messages. Comprehension can be measured by using
questions or tasks which require listeners to understand the meaning of the messages.

Pleasantness of speech can be measured by collecting user opinions. *Mean opinion score* can be used to measure the overall quality of speech. It is calculated by eliciting opinions from a large number of people. By using different scales and questions several aspects of pleasantness can be measured, as presented in our listening experiment [Hakulinen et al., 1999]. It should be noted that intelligibility and pleasantness are related but not directly correlated. Different parameters, such as the use of prosody affect these. They also have a significant effect on user acceptance: unpleasant speech output can lead to poor satisfaction with an otherwise sophisticated system.

Perplexity is the standard measurement to characterize the difficulty of a speech recognition task. Perplexity measures the degree of branching in a statistical language model, i.e., how different words can be combined. If the perplexity is high, i.e., words can be combined fairly freely, and thus it is more difficult for a recognizer to identify the right sequence of words. It should be noted that perplexity only concerns language models and does not take acoustic properties into account. Similar sounding words are more likely misrecognized than words with large acoustical differences.

Besides perplexity, there are no generally used task evaluation metrics. There are, however, many factors which make some tasks more difficult than others. For example, all the issues mentioned in Table 1 (page 14), such as communication style and usage conditions, influence recognition accuracy. In practice, comparisons are usually made on the basis of a corpus and common tasks. Such an approach is used in the DARPA Communicator program [DARPA, 1998].

When speech-based applications are being constructed, technology evaluation should be interpreted from the usability viewpoint. In many applications 20% WER may be acceptable, but in other applications 5% WER may be totally unacceptable. It is also important to realize that different errors have different
2.4 - EVALUATION

effects on the interaction. In some situations, one type of error may be very irritating and correction may have high cost factors, while in another situation another type of error may cause more severe problems. Error loops and other recurring errors can cause frustration to the user, even if the overall error rate is acceptable. Overall, mere error or accuracy rates do not tell much if they are not interpreted in the right context. Nevertheless, they are important factors which should be evaluated, since error rate is one of the most important factors which affect user satisfaction in speech applications.

2.4.2 Application Level Metrics

Both objective and subjective criteria can be used to evaluate speech applications. Task completion rates and times are common usability metrics. Nakatsu and Suzuki [1994] have adapted these standard usability metrics to speech applications:

- level of task completion
- time used to complete a task
- number of turns required to complete a task
- time and number of turns used for error corrections

In addition to the metrics mentioned, I consider the following to be equally meaningful (although confirmations can be seen as a part of the error corrections):

- time and turns used for confirmations
- degree and usage of different modalities

In the SUNDIAL project [Simpson & Fraser, 1993] several related measures have been used, for example transaction success, correction rate and contextual appropriates. The first two correspond respectively to the first and fourth criteria listed. Contextual appropriates indicates how meaningful answers the system gives the user.
Subjective criteria are as important as objective criteria. According to Nakatsu and Suzuki, subjective criteria are:

- user satisfaction
- cognitive load
- user preferences

In general, objective criteria are preferred in the evaluation of applications, and speech applications are no exception. Performance measures, such as task completion time and number of turns are used to fine tune speech systems. Subjective criteria should be taken into account as well, since they can have even greater impact on usability than performance criteria.

Research at MIT has developed dialogue-based metrics to evaluate their Communicator system [Glass et al., 2000]. The metrics used are query density and concept efficiency. These are used to evaluate the collective effectiveness of different system components over the course of a dialogue. Query density is used to measure how efficiently the user can introduce new concepts to the system. It is calculated as a mean of new concepts which a user gives in each turn. Concept efficiency measures how accurately the system can understand user queries, i.e., retrieve meaning from these. It is calculated as a mean of average number of turns required by the system to understand a concept.

Glass et al. [2000] claim that query density can be used to measure the performance of individual speech systems and even to compare different systems. Concept efficiency can be used for diagnosis purposes, since it can indicate recognition errors. These measures are easy to calculate, but the problem with them is that they favor user-initiative systems where the number of turns is minimized. As shown by Walker et al. [1998], system-initiative systems may be preferred to mixed-initiative systems, even if they are not so efficient. These metrics also favor systems which avoid confirmations.

In [Suhm & Peterson, 2001] a method to measure both the cost-effectiveness and objective usability of telephone voice user interfaces is presented. Cost effectiveness is measured by the human operator time saved, and the objective
usability is measured by task completion rates and times. The overall benefit of a voice system is the cost savings which it makes possible. According to the authors, this benefit covers both cost effectiveness and objective usability, since the time saved by the system indicates that the interaction proceeds successfully. The benefit measure can be calculated automatically from call recordings. This measure does not cover subjective usability, which must be measured by other methods.

PARADISE is a framework for evaluating spoken dialogue applications [Walker et al., 1997]. PARADISE allows system developers to use multiple evaluation measurements, such as tasks' completion time and user satisfaction, which can be combined into a single measure. PARADISE allows comparison of different versions of systems when they are iteratively developed. PARADISE can be used to evaluate similar systems, as it was used in the Communicator project [Walker et al., 2001]. Performance measurements can be calculated for whole dialogues or selected sub-dialogues. PARADISE allows comparison of different dialogue strategies. This is achieved by decoupling task requirements from how they are carried out. The authors claim that it provides a basis for comparing systems which perform different tasks, because the measurements normalize for task complexity.

Performance is modeled in PARADISE as a weighted function, which takes into account task success and dialogue costs. Weights are derived from the correlation between user satisfaction and task performance. The goal of the dialogue system is to maximize user satisfaction. This is achieved by maximizing task success and minimizing costs. Costs are calculated from efficiency measures, such as number of turns and repair ratio, and from qualitative measures, such as ease of task.

Evaluating speech systems using the PARADISE framework is not a simple task. PARADISE requires a number of steps to be performed, which involve laborious tasks such as user surveys. It has been used to evaluate real world applications by its authors [Walker et al., 1998; Walker et al., 2001; Litman & Pan, 2002], as well as by other researchers [Hjalmarsson, 2002; Smeele & Waals, 2003], but it
CHAPTER 2: SPEECH-BASED HUMAN-COMPUTER INTERACTION

has not been generally used for speech application evaluation, presumably due to its complexity and high cost. Another reason for this could be that most of the current speech application architectures do not support such evaluations. PARADISE also has some limitations, such as the fact that it does not measure the quality of solutions, i.e., which system answers are better than others.

2.4.3 Empirical Experiments And Data Collection

Empirical experiments and data collection are important parts of the speech application development. They are needed to collect relevant data to train and adapt technical components, collect task requirements, define grammars and vocabulary and perform usability studies. This empirical research can be divided into three categories: human-human studies, simulated human-computer studies and human-computer studies. The usefulness and limits of these approaches will be described next.

Human-human communication can inform the design of human-computer interaction, if properly applied. Communication between humans can be studied employing ethnographic techniques, such as observing people performing various tasks and recording their interactions. Typically natural conversions between two humans, such as the participants of a timetable guidance service, are recorded for later annotation and analysis. Transcribed recordings are needed for speech recognition engine configuration, for example, to train the acoustic models, or to develop language models. There has also been interest in training interaction components on the basis of human-human communication.

The main issue with human-human experiments is their applicability for human-computer interaction. As discussed earlier, speech-based human-computer communication differs in many ways from human-human communication. It is rarely possible to generalize from natural human-human dialogues to the human-computer interaction. The results of these studies can be misleading and result in unpractical or unusable systems. It is reasonable to believe that this difference
will be even greater when new ways of using speech in human-computer interaction emerge. For example, pervasive computing applications may need totally different communication methods from traditional conversational task-oriented applications. For these reasons, the applicability of results from human-human experiments should be verified before using them as the basis for designing human-computer interaction.

Simulated human-computer experiments are popular for the development of speech applications, especially if new applications exceed the capabilities of existing technologies. WOZ studies can be used in the design phase to ascertain the nature of the interaction and to extract suitable interaction patterns. In the later development phases it can be used to verify designed interaction techniques before they are implemented. Early WOZ studies can reveal problems in key application components, such as dialogue logic and prompts preceding any system implementation [Mane et al., 1996].

The degree of the simulation may range from a completely simulated system to an completely functional system, in which only one or more components are simulated. This approach is often called “a system in the loop”. Those parts of the system which are not fully operational can be simulated and the overall system can be tested and data can be collected. Simulation often applies to technical components, such as to the speech recognizer, which can be replaced by the human wizard. Alternatively, all technical components may be present, but the system logic, i.e. dialogue management may be performed by the wizard. The testing can be done in an iterative way in which the degree of simulation decreases as the system evolves. Simulated systems are also very useful when real-world usage data is needed before the system is implemented.

The problems of WOZ studies are twofold. First, they share the applicability problem with human-human experiments. Especially if the users know the real nature of the system, they may behave differently than with a real system. Secondly, it is not trivial to simulate computer applications in a coherent way and at the same time to respond accurately and fast enough. Furthermore, the simulation of errors and other technology-related limitations may be difficult. In
some cases, it may not be possible to simulate systems at all. For example, pervasive systems which are embedded in the everyday environment can be extremely problematic to simulate. Because of these limitations and problems, results of WOZ studies may be misleading. A representative example can be found from a bus travel information systems experiment, in which WOZ experiment results did not correspond to the studies conducted later with a working system [Johnsen et al., 2000].

The third form of empirical studies is real human-computer studies, where the interaction between the system and the user can be examined, the system can be tested, and data collected to further improve the system. Speech application development is often an iterative process, and the real usage may reveal several interaction level problems which do not occur in simulated studies or in human-human communication. Recognition-based technologies are not robust to variability, and even small changes in the environment and the overall system logic may lead to unexpected behavior.

In addition to data collection and system testing, human-computer studies are used to find suitable and unsuitable domain areas for speech applications. Various comparison studies have been made to either justify or assess the use of speech in applications. Speech is compared to the use of the mouse and keyboard, among other devices. The problem with many of these studies is that they often do not measure speech as a modality, and they can easily be affected by the current limitations of technology. On the other hand, many studies do not take technical limitations into account at all, i.e. they assume that technology will be perfect. Therefore, the results of these studies should be treated with caution. Another problem is that it is not usually possible to generalize their results to other application domains [Leatherby & Pausch, 1992].

To conclude, empirical experiments are useful and often indispensable parts of the speech application development process. Still, we must be aware of the differential limitations of the various methods and apply them judiciously. When human-human or WOZ studies are used, we must validate their results with
human-computer studies. The interesting question here is how various evaluation methods can be better supported on the system level. In the following chapters I present several features of the Jaspis architecture, along with tools and techniques, which can be used when speech applications are evaluated.
3 DEVELOPMENT OF SPEECH APPLICATIONS

In this chapter I present software issues of speech application development. Although this chapter focuses on application development, the human-computer interaction issues that were presented in Chapter 2 are taken into account. This chapter describes system architecture that support more natural and flexible interaction methods, while at the same time integrating the underlying components non-tightly. The focus here is on integration of components, which is identified as the key to successful dialogue systems [McTear, 2002]. The constructive part of this dissertation presents how the Jaspis architecture realizes non-tightly integrated speech applications.

This chapter first describes the various components of speech applications. The second section presents how these components can be organized, and what functionality the underlying architectures should support. Different approaches and their advantages and disadvantages are discussed. The third section explains dialogue management strategies and control models. The fourth section presents techniques for multimodal and multilingual applications. Finally, existing architectures, systems, development tools, description languages and standards are described.

3.1 Speech System Components

Speech systems consist of multiple software components which are usually organized into several modules. System modules handle tasks related to basic
CHAPTER 3: DEVELOPMENT OF SPEECH APPLICATIONS

technologies (e.g. a speech recognition module) and to interaction techniques (e.g. a dialogue management module). On a general level, a speech system consists of at least three main components: a speech output (synthesizer) module, a speech input (recognizer) module and a dialogue management module. Usually there are also output generation and input interpretation modules, which may or may not use natural language processing techniques. The majority of practical systems also contain a back-end module (such as a database module) and telephony speech applications must include a telephony module. Figure 2 illustrates this general setup. Next I describe each of these modules in more detail.

![Figure 2: Speech system components.](image)

3.1.1 Speech Recognition Components

A speech recognition module receives spoken utterances and transforms them into textual representation. Alternative results are usually represented by using n-best lists, which contain best recognition candidates ordered according to their confidence scores. Another representation form may be a graph representing alternative recognition results in a compact form, commonly called a word lattice. A word lattice can be transformed into an n-best list using algorithms that identify the top N paths through the lattice with the highest confidence scores.

Speech recognition results may contain recognized sentences, words, or even smaller recognition units, such as monophones or triphones which correspond to phonemes (allophones) in the speech signal. Confidence scores correspond to the acoustic properties of these units, and confidence scores on a higher conceptual
level are usually an average of the lower level confidence scores. In practice this means that sentence level confidence scores are calculated directly from the acoustic scores of words, which in turn are calculated from the phoneme level scores. A language model typically guides the recognition process, either during the scoring for the first best matching sequence of words, or by rescoring n-best lists after the recognition process.

The speech recognition module may be fairly independent of other system components. For example, it may use predefined recognition models, so it does not need any control from the application for selecting acoustic and language models. In its basic form the recognition module (server) accepts recognition requests and produces recognition results. In many applications, however, specific recognition models are employed dynamically to maximize accuracy in different situations. Usually this concerns language models which can be activated and deactivated depending on the dialogue state. Acoustic models can also be changed according to the user, environment and communication channel. Typical examples include specialized language models for situations such as confirmations, and acoustic models that are adapted to individual users. Context-sensitive language models may improve accuracy. If properly used, practical applications can be constructed despite limited recognition capabilities, since the task of the recognizer becomes less complex when the perplexity of the task decreases.

Language models can be divided into two categories: context-free grammars and statistical models. Context-free grammars are usually hand-coded and customized for particular tasks. The most popular statistical language model is the n-gram, which models probabilities of word sequences. To train n-gram models, a great deal of real data, such as from transcriptions of spoken dialogues is required. Context-free grammars are more suitable for compact, well-defined tasks, while n-grams are more suitable for larger, less constrained tasks. These approaches can be also combined by using n-grams for general interaction tasks and context-free grammars for well-defined situations, such as domain oriented
CHAPTER 3: DEVELOPMENT OF SPEECH APPLICATIONS

tasks, or using a unified context-free grammar and n-gram model [Wang et al., 2000b]. In practice, however, most recognizers use only one approach at a time.

There are no generally used representation formats for speech recognition grammars and recognition results, although VoiceXML defines a format for context-free grammars and statistical language models. Another format is JSGF [Hunt, 2000a]. Still, virtually all engine specific formats are variants of Backus-Naur Form. Most of the proposed formats are different from other linguistic information representation, and as a result of this, speech systems contain different formats to represent recognition grammars, recognition results, dialogue data and speech outputs. This becomes a problem when systems are evaluated and data is further processed, for example by manually annotating. There is a need for a general information representation model which includes speech inputs. Recognition module and the system architecture must represent information in a way that facilitates information exchange between system modules, as well as between applications. Section 6.5 describes how the Jaspis architecture supports the general, XML-based Annotation Graphs [Bird & Liberman, 2001] format as a comprehensive representation for all linguistic information inside speech applications.

3.1.2 Speech Interpretation Components

In speech systems the input interpretation module receives text sentences from the recognizer and extracts the meaning of these sentences. The language understanding components interpret the recognition results and transform them into a suitable semantic form. This process may involve several sub-processes, depending on the techniques used. This high-level process may include syntactic analysis, semantic analysis and discourse analysis tasks. These tasks correspond to the middle and top layers in the speech processing model, as illustrated in Figure 1 (page 11).
3.1 - SPEECH SYSTEM COMPONENTS

Since spoken language differs from written language and recognition introduces errors, robust parsing is needed. Robust parsing methods try to extract the essential meaning from the utterances rather than to understand them completely. For example, the system may try to extract topic and focus information from a sentence rather than trying to understand every word. It may, however, not be able to capture all nuances from the utterance. Robust parsing and complete linguistic analysis can be combined in several ways. For example, first by trying to interpret the complete message and if that fails then use robust methods. The use of multiple parsing components can make the overall parsing more robust, and in general it makes sense for a system architecture to support multiple parsing components.

The integration of speech recognizer and language processing components is often done in a trivial way: the speech processing module takes the recognition results and extracts conceptual meaning from these. More elaborate methods could be used, however. There are at least two good reasons why these components should be integrated more closely. First, the natural language processing components can help the recognizer to increase the recognition accuracy, although results have not been very encouraging so far. Second, the communication between the human and the computer could benefit from a less rigid turn-taking protocol, because certain dynamic and concurrent communicative actions contribute significantly to the robustness and ease of human-computer conversation. Examples include non-verbal communication and back-channel utterances, which do not always follow rigid turn-taking. According to Allen et al. [2001b], natural interaction requires incremental understanding and generation with flexible turn-taking. The Jaspis architecture supports flexible turn-taking by allowing speech recognition, language understanding and dialogue management to communicate in other ways than the traditional pipeline approach.
CHAPTER 3: DEVELOPMENT OF SPEECH APPLICATIONS

3.1.3 Dialogue Management Components

The dialogue management module is often seen as the central component in speech systems, and in some cases it controls both the interaction between the system and the user, and the interaction between the system components. The responsibilities of the dialogue management module are defined differently in different systems. The most fundamental task of the dialogue manager is to transfer the dialogue state – in an abstract sense – from one state to another. In practice, many dialogue managers also initiate and some even perform input interpretation, output generation and domain specific tasks themselves. This approach may easily lead to situations in which the dialogue manager is a monolithic component which controls everything in the system and makes it hard to build modular, distributed systems and reusable components [O'Neill & McTear, 2000]. In more advanced systems dialogue management is divided into smaller components. The Jaspis architecture supports adaptive, distributed and reusable agents to implement a modularized dialogue management.

3.1.4 Output Generation Components

The task of the output generation module is to produce system outputs, i.e. natural language sentences or fragments, which the system uses to communicate with the user. This is done on the basis of conceptual messages and/or the dialogue state. The response generation module decides which words are included in the sentences and which modalities, languages, voices and prosodic elements are used in the messages. It can also decide if the user needs additional information, feedback and confirmations. Human-factor issues (such as prompt design) play a crucial role in this highly application dependent process [Huang et al., 2001].

The response generation process can be divided into two major phases: content determination and message rendering. In the first phase the content of the response is determined and in the second phase the message is rendered. In the
first phase the system decides what should be presented to the user on the basis of the current dialogue state. The main concepts are chosen, and the overall structure of outputs is defined. In this phase the interaction history and the user model can be valuable information sources. For example, the system may omit information which is already known to the user (e.g. to taper lists). In the second phase actual words, and presentation modalities (such as speech, non-speech audio, and graphics) are chosen, and prosodic elements and synthesizer control codes are selected.

Some applications use other modalities, such as graphics, gestures and haptics in addition to the speech inputs and outputs. From the viewpoint of speech input interpretation and speech output generation modules, this requires techniques for multimodal fusion and fission. Techniques for multimodal fusion and fission are discussed in Section 3.4.

Today sophisticated response generation systems are limited to research prototypes [McTear, 2002]. Although the response generation part can have great impact on the interface and user satisfaction, it is sometimes neglected even in otherwise advanced speech systems. In the constructive part of this dissertation I present several methods for the efficient implementation of adaptive response generation. Section 6.2 presents a model which realizes modular output generation, including multilinguality and multimodality, using Jaspis agents, evaluators and managers.

3.1.5 Speech Synthesis Components

While speech recognizers have many parameters that must be configured properly, a spoken output is commonly generated using either prerecorded sentences or general-purpose synthesizers. Speech synthesis modules are typically used “out of the box”. The use of multiple synthesizers, to be used for example in multilingual applications, makes this module a little more complex, since it must control multiple synthesizers and in some cases also to decide which ones to use in which situations.
CHAPTER 3: DEVELOPMENT OF SPEECH APPLICATIONS

In its basic form the synthesizer module receives a text string which is transformed into a speech signal. The text may contain control codes, which suggest that the synthesizer should alter its output. The most typical control codes control prosody, change of speaking voices and their parameters. As with recognizers, there is no single agreed markup language for synthesis control, and in practice different synthesizers use different markup languages. Because of this, a general purpose synthesizer module should provide services which perform translations between markup languages. In the Jaspis architecture JSML [Hunt, 2000b] is used as the default markup language, but the architecture provides a mechanism for translations between markup languages.

3.1.6 Other Components

Real-world speech systems frequently contain other modules, such as application back-ends and telephony modules. The application back-end module works either with the dialogue management or task management module. In the ideal case, the dialogue management module operates on general, application independent level, while the task manager contains application and domain specific knowledge. In practice, most dialogue management components are to some extent application dependent, and there is no separation between domain independent and domain dependent task knowledge.

Typically, back-end modules interact with the databases and other information sources. Examples include travel information databases and mailboxes of users. The telephony module, which is necessary to implement telephony applications, communicates between the telephone hardware and the system. In practice, telephony, synthesis and recognition modules are integrated to some extent to make processing more efficient.

Speech systems are not limited to the modules mentioned above. Some systems contain additional modules, such as user and context modules, which are needed in emerging application areas, such as speech-based ubiquitous computing.
3.1 - SPEECH SYSTEM COMPONENTS

applications. Multimodal applications, such as information kiosks, contain modules related to the different modalities, such as graphics, haptics and computer vision. If a speech application contains such modules, they must be supported and coordinated by the system architecture. Jaspis supports any number or modules, and offers the agent – evaluator – manager paradigm for all system modules. The communication management subsystems provides basic support for coordination of multimodal inputs and outputs.

3.2 Speech System Architectures

The previous section discussed the various modules that need to be integrated to implement speech applications. These modules usually contain multiple components, which have their own internal organization and relations. This section describes how speech system components can be organized and what functionality the underlying system architecture should offer. This includes principles for interaction and information management between the system components, as well as software architectures that address specific speech application issues. I also present which features of different architectural styles are used in the Jaspis architecture.

3.2.1 Software Architectures

A software architecture defines the system in terms of components and interactions between them, while connectors are used to mediate interaction between the components [Garlan & Shaw, 1993]. Several views can be used to describe different aspects of software architectures, such as design or runtime views and module, logical, process, control, class or data views [Land, 2002].

The existing work done in the software engineering community and the emerging field of software architectures are seldom referred to in the publications on speech systems. One attempt to make speech application specific
theory has been reported by Bernsen et al. [1998]. Often the focus is on practical issues and individual modules, rather than on architectural principles. From this viewpoint, frameworks that provide the necessary building blocks (e.g. components) for implementation, and development environments, which contain tools to construct applications are more interesting for the application developer.

A framework suitable for practical software engineering must include a set of components to implement applications. Such a framework should provide building blocks for interaction between system components, i.e. the core infrastructure. Examples for architectures that provide the core infrastructure include Galaxy-II and the Open Agent Architecture [Martin et al., 1999]. Complete speech application frameworks provide components for all application needs, including dialogue management, speech recognition and natural language processing. This allows the application developers to focus on the implementation of system functionality.

Domain specific architectures targeted at different domains, such as speech applications, are called *reference architectures*. One example is the Galaxy-II architecture [Seneff et al., 1998]. In these architectures the structure of the components can be tailored to the needs of specific applications.

Development environments offer tools for application development, such as graphical debuggers. They can also support other activities of the application development process, such as corpus collection, dialogue design and development of recognition grammars and language models. An example is the CSLU toolkit [Sutton et al., 1998]. The main limitation of these systems is that their components are not easy to change.

While frameworks include interfaces and components for certain tasks in a particular domain, and development environments contain tools for various development tasks, they also support the organizational aspects of systems, and in this way are related to architectural models. This dissertation covers all of these areas, and uses the term *architecture* to refer to them. In this section I focus
more on the architecture styles, while other sections in this chapter cover specific frameworks and development environments.

The Jaspis architecture is both a general architecture and a concrete application framework. It also provides tools for application development. In this sense it is similar to the GATE architecture [Cunningham et al., 2002] (although the domain is different).

When we consider speech architectures from the human-computer interaction point of view, the most interesting issue is how the system could support more intuitive and natural speech-based interaction methods. This is seen as one of the key aspects in spoken dialogue applications [Allen et al., 2001b; McTear, 2002].

This dissertation focuses on how a software architecture can support better speech-based human computer interaction, in addition to making the process of building speech-based systems easier.

3.2.2 Architectural Styles

An architectural style refers to systems with similar features, i.e., it defines a family of systems in terms of patterns of structural organization [Garlan & Shaw, 1993]. They help us to characterize systems in abstract levels. The following sections review the well-known architectural styles and architecture models. In practice, most real systems employ more than one of the architectural styles. For example, the Jaspis architecture applies many of these styles in its various organizational models.

3.2.2.1 The Pipe-and-filter model

The pipe-and-filter model is a data-flow oriented model. Each component retrieves data from the input stream and produces results for the output stream. Unix commands and pipes are typical examples of this architectural style. Components are able to produce results incrementally, i.e. they can read part of the input stream and produce results from that, then read the next part of the input stream and so on. This property makes components act like filters. Filters
are independent, in particular from the state of other components. In a strict sense, components are not supposed to know the identity of other components in the pipeline, and the correctness of their results should not depend on the execution order.

Several variations of the pipe-and-filter model exist. *Pipeline systems* restrict the components to a linear and sequential arrangement, while in *batch sequential systems* components do not begin their processing before the input stream has been fully read. Taking this into account, many software architectures that are called pipeline systems are in fact batch sequential systems. Text-based natural language systems are typical examples. Other variations include *bounded pipes* and *typed pipes*, which limit the amount and type of data processed by the pipes.

The pipe-and-filter model has many desirable properties. For example, it supports the reuse of components. The main drawback, however, is that they are not particularly suitable for interactive systems [Garlan & Shaw, 1993]. In speech systems the sequential nature of pipe-and-filter architectures compels a simple and rigid turn-based interaction model, because it does not support the exploration of more flexible interaction models needed in more natural speech communication [Allen et al., 2001b]. The pipe-and-filter model still has a lot to offer, because its principles can be applied to sequential subsystems within complex hybrid systems. For example, it can be used to organize language processing components.

### 3.2.2.2 The Object-oriented model

The object-oriented model is the dominating style in current software architectures, and especially in graphical user interfaces. The main focus is on objects which encapsulate the data and provide functionality for manipulating it. Objects can be called *managers*, since they maintain the integrity of resources. Variations include for example whether or not components can be concurrent and whether they have multiple interfaces or not [Garlan & Shaw, 1993].
The advantages of the object-oriented model include the ability to change the implementation of components without altering their interface or other components. In practice this means that application programming interfaces, such as SAPI [Microsoft, 2003] and JSAPI [Sun, 2003] assume key roles in object-oriented speech systems. Agent-based systems, which are an emerging technology in speech applications, are based on the principles of object-oriented systems.

The disadvantage of object-oriented systems is that direct references to other objects cause changes when components or their structure change. It is not uncommon that software architectures have changes. For example, the SAPI interface changed almost completely between versions 4 and 5, and a similar change has occurred more recently to the Philips Speech API [Philips, 2003]. The Jaspis architecture solves this problem by minimizing interfaces between system components and using dynamic selection of objects.

3.2.2.3 The Layered model

The best known layered architecture, the ISO OSI model, is a good example of the layered model. Each component layer knows only its lower and upper layers. In practice, some parts of the layers may be opaque, so that they can be accessed more freely. This model is suitable for situations where well defined protocols between layers can be used and components can be unambiguously assigned to layers. Typical examples include operating and database systems.

The benefits of layered systems include support for multiple hierarchical abstractions and reuse of components. The layered model also offers fairly flexible interaction between components and is thus suitable for interactive systems. The main drawback is that it may be difficult to arrange components into layers, and even when this can be done, the resulting system may be inefficient if components arranged for non-proceeding layers do not have direct access between them.

The Jaspis architecture uses layers to arrange components for different purposes. It has a layered structure for the whole architecture (agents, evaluators and
managers) and separate layers for some parts of the architecture, such as information management and input and output processing. In addition to that, it provides direct method calls for several system components to ensure efficiency when needed. Linguistic information is also represented by using layered models.

3.2.2.4 The Event-based model

The event-based approach to software systems is popular among graphical user interfaces. Instead of directly referring to other components these systems include events which are broadcast to the registered components. In this way, components are invoked implicitly rather than explicitly. Usually components can also be called by using explicit method calls similarly as in object-oriented systems.

The main advantage of the event-based model is that components can be changed dynamically, i.e. they can be registered and unregistered dynamically. This facilitates reuse and simplifies modifications. The event-based model also supports complex interaction patterns, because the interaction flow among the system components can be flexible and complex. On the other hand, this can be seen as a disadvantage, because there is no direct control of how components are invoked. Components can not, moreover, make sure that the necessary functionality takes place when they broadcast messages. The data exchange can also cause inefficiency, if broadcasted messages are copied between multiple components.

In speech applications the event-based model is not widely used, except in those systems which are originally based on existing graphical user interfaces. However, the event-based model makes it possible to build distributed, complex and maintainable systems, which allows speech applications to be more natural and flexible. This is especially true in the case of pervasive applications. The Jaspis$^2$ architecture uses events in a way which makes it possible to control them in different layers. It also provides shared information storage to minimize
3.2 - SPEECH SYSTEM ARCHITECTURES

broadcasting bandwidth and provides blackboard style method calls for efficient interaction.

3.2.2.5 The Blackboard (repository) model

The blackboard model is a special case of the repository model. The central aspect of these systems is centralized data storage. Other components in the system, which may be called knowledge sources, agents and so on, operate on the basis of this data storage. The variation in repository systems comes from their control mechanisms: operations can be driven by the shared data or other system components. In the latter case the system is more like a database than a blackboard.

The blackboard model is especially suitable for artificial intelligence and signal processing applications. One well known example is the HEARSAY-II speech understanding system [Erman et al., 1980]. In this case, the shared data is organized into layered form similarly to the levels of speech processing presented in Figure 1 (page 11). An agent-based system can also benefit from this kind of shared and layered information.

The Jaspis architecture is based on the notation of shared information storage, but it is not a true blackboard system. In the first version of Jaspis the shared information storage acts more like a database. The Jaspis2 architecture supports more blackboard oriented features.

3.2.2.6 The Client-server (distributed processes) model

The client-server model, or more generally the distributed processes model, has become extremely popular with the Internet and other distributed service systems. The focus is on services provided by servers and used by clients. In a way, the services represent shared data in a similar way as shared data storages in blackboard systems. There may be different topologies to arrange the network between components. Many systems have one central component (a broker, a facilitator, a hub) which guides the traffic between services. An example of speech systems is the Galaxy-II architecture [Seneff et al., 1998], in which a hub
controls the traffic between servers. Agent architectures also use similar models, for example the Open Agent Architecture [Martin et al., 1999] has a facilitator which coordinates services and agents. The idea of having a central component is to reduce direct references between components. Resources can be also balanced more efficiently in this way.

Services are a useful way to represent distributed and shared resources. There exist mature, high-level protocols to exchange information between services. Modern examples are SOAP and XML-RPC. This makes the communication between components well-defined and the system can be extended by introducing new services and clients. In complicated and large systems it is important to share and distribute resources, such as speech recognizers, between multiple computers and dialogues to maximize computation resources. The Jaspis architecture uses a client-server architecture to represent component relations in a transparent way. Thus, highly distributed systems can be implemented in Jaspis.

3.2.2.7 The Finite-state machine model

The finite-state machine model focuses on states and transitions between them. It is a well known and widely used model, and it has been used in many speech applications. For example, the CSLU toolkit [Sutton et al., 1998] supports a finite-state dialogue model [McTear, 1998]. The state machine model has a strong theoretical background, good properties from the computational viewpoint, it is easy to understand and straightforward to use. The main disadvantage is that it is best suited for small systems, because the complexity characteristic of large systems results in excessively complex state machines.

One reason why state machines are used in many applications is that they can be visualized efficiently. In this way, it is possible to create rapid application development (RAD) environments, in which speech applications can be created without programming knowledge or need for specialized markup languages.
These RAD environments can be very attractive, as for example the experiences of the CSLU toolkit show [McTear, 1999].

Although the Jaspis architecture is not based on a finite-state model, finite-state based applications can be implemented efficiently by using Jaspis. For example, the Doorman application uses a finite-state model in its dialogue agents and evaluators. The benefit of the agent-based approach is that the finite-state model can be extended in several ways, for example to include global states and combine other models with it.

### 3.2.2.8 The Process control model

The process control model is useful in environmental systems which have a set of sensors and a control unit. Sensors produce information, which is processed by the control unit. The output of the control unit changes the state of the environment in which the sensors are located. This kind of approach can be used in pervasive computing applications, such as in the Oxygen project [Oxygen, 2003]. They make it possible to produce highly distributed applications which can serve multiple users in various environments.

### 3.2.2.9 Hybrid models and heterogeneous systems

As stated earlier, most real systems employ more than one of the architectural styles presented in this section. One example of these hybrid models (or heterogeneous systems) is a batch sequential system with a shared knowledge source. This kind of systems combine the pipe-and-filter and repository models. Systems can be also viewed from different perspectives and levels of abstraction. For example, a system can be modeled by using the object-oriented model to describe its components, while its functionality can be defined by using the state machine model, and its modular structure can be presented using the client-server model.

Bass et al. [1997] have defined three different approaches to heterogeneous architectures. In locationally heterogeneous systems runtime components use different architectural models. In hierarchically heterogeneous systems different
system components may have local internal architectural models which differ from the parent level models. In simultaneously heterogeneous systems multiple architectural styles can be used to describe the system.

Jaspis is a heterogeneous architecture. For example, its input/output subsystem differs from other parts of the system in many ways and uses a different architectural model internally, but is part of the overall system architecture model and not independent from it. This is presented in more detail in the next chapter.

3.2.3 Architecture Styles In Speech Systems

In most speech systems components are structured either in a pipeline fashion or using the client-server model with a central component, which facilitates the interaction between other components. An example of this is the Galaxy-II architecture [Seneff et al., 1998], which is used in the DARPA Communicator program and numerous other applications. Another well known architecture is the Open Agent architecture [Martin et al., 1999], which has been reused in more distributed systems including the MITRE architecture [Luperfoy et al., 1998] and WITAS [Lemon et al., 2001].

The state machine model is also popular among speech applications, especially as local model for dialogue management components. A representative example is the CSLU toolkit [Sutton et al., 1998], which integrates the state machine model with a graphical user interface for rapid prototyping of speech applications. Often the state machine model is combined with the pipes-and-filters model. In many state machine based systems a batch processing approach is used for interaction control.

Although the pipes-and-filters model is considered suboptimal for interactive systems [Garlan & Shaw, 1993], many speech systems are based on variations of the pipes-and-filters model. The reasons for the popularity of the pipes-and-
filters model include: familiarity with the pipeline model from the Unix environments and from text-based systems, and simplicity.

To generalize, architecture is considered to be only a practical issue in speech systems and it is often not modeled explicitly. This has recently been noticed [Allen et al., 2001b; McTear, 2002] and when speech systems become more complicated, more focus needs to be on system architectures. This is important from many viewpoints: for example, systems can be more efficient and easier to build and maintain if proper architectural models are used. From the human-computer interaction point of view, natural interaction requires distributed, flexible interaction models which need support from the system architecture. By using the traditional batch sequence model, for example, it is hard to implement functionality which takes into account issues such as back-channeling and multiple concurrent users.

From the viewpoint of speech applications, there are two especially important issues in the system architecture. The first is interaction management between components. The second is information management. When speech applications are more sophisticated, for example when they allow multiple people to interact with the system concurrently, there is a need for flexible interaction management and efficient information sharing between system components. These two perspectives are the focus of the constructive part of this dissertation. In addition to the system specific view, interaction management between the system and the user is part of this work. The next two chapters focus on interaction management between the system and the user, starting with dialogue management and continuing with the coordination of multimodal and multilingual speech inputs and outputs.

### 3.3 Dialogue Management In Speech Application

The dialogue manager controls the overall interaction between the system and the user. The main task of the dialogue management component is to find a
suitable system action which corresponds to the user input, or more generally which corresponds to the current system state. This process can be seen as a mapping from a user action to a system action, or from one system state to another system state. Many dialogue managers, especially those used in text-based systems, tend to extend their functionality to natural language understanding and generation as well. The communication with the data source such as the database (or the application's functional core in terms of Nigay and Coutaz [1995]) is often one of the tasks of the dialogue manager.

Dialogue management is a fairly mature research area, and many sophisticated text-based systems have been constructed. However, in general they have not been proven to be very successful. So called natural language interfaces especially have not been used widely. Many of the principles found in these systems can be used in speech systems, although speech is a very different communication modality from text. It is not realistic to assume that text-based dialogue systems can be converted into speech-based dialogue systems trivially, e.g. by adding speech recognition and synthesizer components. The focus of these systems is different, and some of the research questions, especially those dealing with the nuances of written text, are not particularly relevant in speech systems. Nevertheless, speech systems can utilize knowledge from text-based dialogue systems.

In speech applications, dialogue management is part of the interaction flow between the system and the user, or multiple users. Beside the dialogue manager there are other components which take part to this interaction flow. Next, two main perspectives of dialogue management are presented. These are different dialogue management strategies and dialogue control models which can be used to control the interaction.
3.3 - DIALOGUE MANAGEMENT IN SPEECH APPLICATION

3.3.1 Dialogue Management Strategies And Initiative

One of the key aspects in dialogue management is how the initiative is handled. The dialogue management strategy used may be system-initiative, user-initiative or mixed-initiative.

3.3.1.1 System-initiative dialogue strategy

In a system-initiative dialogue system the computer asks questions from the user, and when the necessary information has been received, a solution is computed and a response is produced. System-initiative dialogue strategy can be highly efficient since the paths which the dialogue flow can take are limited and predictable. The most challenging issue for dialogue management is to handle errors successfully and ask relevant questions from the user. None of these tasks are trivial, however.

One of the key advantages of system-initiative dialogue strategy from the speech interface perspective is the predictable nature of the dialogue flow. This makes it possible to use context-sensitive recognition grammars, for example. Every dialogue state can have a tailored recognition grammar, which helps the recognizer to achieve more robust recognition results. In non-optimal situations, such as in telephone applications or public information kiosks this can make the application usable even if the recognizer cannot use other than simple recognition grammars.

The other advantage of system-initiative dialogue strategy is that the system guides the user to help the user to reach his/her goal. Since the system asks questions, the user can be sure that all necessary steps will be performed. This makes the user feel comfortable with the system and prevents disorientation. This approach is particularly suitable for novice users who do not know how the system works.

The disadvantages of system-initiative dialogue strategy include the clumsiness of interaction with experienced users. Especially if the system assumes that only single pieces of information are exchanged in every dialogue turn, the dialogue
CHAPTER 3: DEVELOPMENT OF SPEECH APPLICATIONS

may be long and advance slowly. This can be reduced by letting the system accept multiple pieces of information with a single utterance. In this way experienced users may pass certain dialogue turns by using more complicated expressions. Obviously, this does not only make the dialogue management more complicated, but also the recognition grammars more complex.

System-initiative dialogue strategy is not suitable for all tasks. It is most suitable for well-defined, sequential tasks where the system needs to know certain pieces of information in order to perform a database query or similar information retrieval tasks. Typical examples include database queries for bus timetables or flights. More open-ended tasks, such as e-mail applications cannot be modeled using sequential tasks without the interface becoming inefficient and inflexible. It is noteworthy that there are different tasks in many applications, and although one dialogue strategy may not be suitable for the overall dialogue flow, it may be suitable in some parts of the dialogue.

3.3.1.2 User-initiative dialogue strategy

The user-initiative dialogue strategy assumes that the user knows what to do and how to interact with the system. In its general form, the system waits for user inputs and reacts to these by performing corresponding operations. User initiative systems are often called command and control systems, although the language used may be rather sophisticated. In any case, the user is the active participant in these systems regarding the dialogue initiative. The advantage of user-initiative systems is that experienced users are able to use the system freely and perform operations any way they like without the system getting in their way. This is also natural in open-ended tasks which have many independent subtasks.

The main weakness of many user-initiative systems is that they require that users are familiar with the system and know how to speak. The common argument favoring user-initiative systems is that if the natural language understanding capabilities of the system are advanced, the system can understand freely spoken
natural language utterances. This is seldom realistic, since the use of unrestricted language leads to very open language models, which most commercial speech recognizers cannot handle. Furthermore, even if the computer could understand freely expressed sentences, the user would have to know the task structure in order to give all the necessary information to the computer. This loads the cognitive capabilities of the user.

Since both system-initiative and user-initiative dialogue strategies have their advantages and disadvantages, there is no single dialogue management strategy which is suitable for all situations. Different users and application domains have different needs, and the accuracy of the speech recognizer affects as well the selection of dialogue strategy. To summarize, different dialogue strategies are needed for different situations. For example, Walker et al. [1998] found that mixed-initiative dialogues are more efficient but not as preferred as system-initiative dialogues in the e-mail domain. They argue that this is mainly because of the low learning curve and predictability of system-initiative interfaces. System-initiative interfaces, on the other hand, are more inefficient and could frustrate more experienced users. This supports the view that different dialogue handling strategies are needed even inside single applications.

### 3.3.1.3 Mixed-initiative dialogue strategy

The so-called mixed-initiative dialogue management strategy assumes that the initiative can be taken either by the user or the system. The user has freedom to take the initiative, but when there are problems in the communication, or the task requires it, the system takes the initiative and guides the interaction. Applications can use mixed-initiative strategy in different ways. For example, tasks may form a hierarchy in which different subtasks can use different dialogue strategies. Alternatively the system can adapt the style of the interaction to suit particular users or situations based on the success of the interaction. This can be done, for example, by using the system-initiative strategy at the beginning and letting the user take more initiative when she or he learns how to interact with the system.
Similarly, if the user has problems with the user-initiative strategy, the system can take the lead if the interaction is not proceeding as well as expected.

If properly constructed, a mixed-initiative system can help the user by employing system-initiative strategy while still preserving the freedom and efficiency of user-initiative strategy. In practice, the mixed-initiative strategy is often a synonym for user-initiative strategy with system-initiated error handling. If the dialogue is modeled using the user-initiative strategy with addition of several system-initiative sub-dialogues, the support for system-initiative dialogues may be rather limited. On the other hand, if a predominantly system-initiative system allows the user to take the lead, the system may suffer from the problems of user-initiative strategy without gaining any real advantage for the interaction. This is especially true with speech systems, since the language model for a mixed-initiative system can be extremely complex.

The mixed-initiative strategy may be complex to model and use. One solution is to allow independent but related dialogue strategies to co-exist and take place when needed. This can make the modeling, use and maintenance of dialogue strategies more straightforward. This is the approach that the Jaspis architecture takes by supporting both complementing and competing interaction strategies by using the dialogue agents for implementing different dialogue strategies. In this approach, dialogue evaluators are used to choose between the dialogue agents, and so between dialogue strategies. This can make the interaction dynamically adaptive, and it still remains manageable. The details of this approach are presented in Section 4.4.1.

3.3.2 Dialogue Control Models

In this dissertation the dialogue control model refers to the ways in which the dialogue is implemented from the point of view of the system. This may correspond to the overall structure of the dialogue flow, as perceived by the user, or to the task structure of the dialogue. On the other hand, the task structure may
be different from the structure of the dialogue components. Similarly, the implementation of the dialogue model may differ from the mental model that the user has. As one of the main focuses of this dissertation is on architectural issues, it is meaningful to define the dialogue control model to cover all those ways which are used to represent the organization of dialogue components and interaction between these components.

3.3.2.1 Finite-state machines

Finite-state machines are logical choices for dialogue management in many speech applications. Many, if not most of the current commercial speech applications use finite-state machines for dialogue control. There are many reasons for the popularity of state machines. First of all, they are well known and straightforward to use. From the software engineering viewpoint, they are well-defined and supported. The architectural properties of finite-state machines were presented in Section 3.2.

In its most basic form a finite-state machine consists of a set of nodes representing dialogue states and a set of arcs between the nodes. Arcs represent transitions between states. The resulting network represents the whole dialogue structure, and paths through the network represent all the possible dialogues which the system is able to produce. Typically, nodes represent computer responses and arcs represent user inputs, which move the dialogue from one state to another.

The finite-state model represents dialogues explicitly and in an easily computable way. In this way it is possible to automate many tasks, such as finding a minimum sequence of turns (i.e., the shortest path) to produce a certain dialogue. States can also be used to model the task structures and context knowledge. For example, there can be a specific recognition grammar associated with every state.

Extensions to the basic model include sub-dialogues, or in a more general form different hierarchically organized finite-state machines. In this way universal commands and complete sub-dialogues, such as error correction sequences, can
be represented by using an independent finite-state machine, which can be embedded into other finite-state machines. In order to reduce connections between states, sub-dialogues can be global states, which means that there are default transitions from all other states to these states. These techniques are also used in graphical user interfaces [Green, 1986]. The structure of the state-machine can also be modeled during the runtime by removing and adding states and transitions between them. As previously mentioned, we have used Jaspis agents and evaluators for these purposes.

Finite-state machines are most suitable for well-structured and compact tasks. If there are numerous states, and more importantly, a lot of transitions between states, the complexity of the dialogue model increases rapidly. Common operations which can take place in most situations, such as error correction procedures, increase this complexity enormously. For these reasons, it is commonly argued that state-machines are mostly suitable for small-scale applications.

It is fairly obvious that finite-state models are not the best possible solution when the task structure is complex or it does not correspond to the dialogue structure. When the number of different possibilities, i.e., the number of connections between states increases, the dialogue model becomes unmanageable even if divided into subtasks. This is clearly the case when graphical tools, such as the CSLU toolkit [Sutton et al., 1998] are used: the complexity of graphical representation increases rapidly and makes it hard to maintain the system [McTear, 1998].

3.3.2.2 Frame-based systems

Frame-based systems use templates, i.e., collections of information as a basis for dialogue management. The purpose of the dialogue is to fill necessary information slots, i.e., to find values for the required variables and then perform a query or similar operation on the basis of the frame. Variations of the template approach include schemas, e-forms, task-structure graphs and type hierarchies
Frame-based systems are popular these days with the success of the VoiceXML markup language. They are, for example, used in many timetable and e-commerce applications.

In contrast to the state-machine approach, frame-based systems are more open, since there is no predefined dialogue flow, but instead the required information is fixed. In a frame-based system the dialogue can take any form to fill the necessary slots, at least in theory. Multiple slots can be filled by using a single utterance, and the order of filling the slots is free. Of course, there are practical limitations, as well as dependencies between slots which make these systems a little more complicated and the possible dialogue paths more restricted than in theory. The frame-based dialogue control model is a more natural choice for implementing mixed-initiative dialogue strategy than the finite-state model, since the computer may take the initiative by simply asking for the required fields.

The heart of form-based dialogues is the implementation of the dialogue control algorithm, i.e., the algorithm which chooses how to reach the user inputs. In the state-based approach there is always a path from the current state to another state, so the dialogue has an explicit interaction flow model. In form-based systems there are no such explicit relations between the system states (e.g. following turns). Various control algorithms have been presented to control dialogue in form-based systems [Chu-Carroll, 1999]. The common phenomenon with these systems is that the dialogue control is centralized and specified with a single algorithm. The mixed-initiative mode means in practice that the computer takes the initiative if there is something to clarify, something missing or ambiguity within a single task.

From an architectural viewpoint, the frame-based approach does not have any direct counterpart. It can be implemented by using the object-oriented, agent-based or event-based architecture models, for example. Jaspis agents and evaluators can be used in various ways to implement frame-based applications. For example, agents can be used to represent actions which correspond to slots,
CHAPTER 3: DEVELOPMENT OF SPEECH APPLICATIONS

while evaluators represent relations between the slots. Bus timetable systems, as presented in Section 5.2, are examples of frame-based Jaspis applications.

3.3.2.3 Event-based systems

In a way, event-based systems are similar to frame-based systems. The computer determines what to do on the basis of the interpreted user utterance and its own state. An example is presented in [Wang, 1998]. The event-based approach can be seen as a generalized model for information state based dialogue control models. Similar approaches include plan-based systems and collaborative agents.

3.3.2.4 Plan-based systems

Plan-based systems focus on the interpretation of the utterance and on the intention that it conveys. This information is then matched to the plan which models the dialogue. Plans may be related to the domain, or to the discourse in general [Litman & Allen, 1987]. Dialogue acts play an important role in this approach. Originating from speech act theory [Austin, 1962; Searle, 1969], dialogue acts are used in various meanings in practical dialogue systems. This approach sees both user and computer utterances as communicative acts, which are chained together to achieve the goal which the dialogue has. Since dialogue acts determine the success of the dialogue, it is important to recognize dialogue acts correctly. One of the current trends is to do this using machine learning techniques [Black et al., 2003].

3.3.2.5 Collaborative agents

In the collaborative agents approach the dialogue is seen as a collaborative process between partners. Partners work together to achieve a mutual understanding, i.e., a shared belief state. Participants have their own belief states, which they try to match by conveying application specific and independent dialogue acts. Various models for collaborative agents have been introduced, one of the most well-known being the BDI model [Bratman et al., 1988] and its variants [Traum, 1996]. In one variation only the system state is modeled, not the
3.3 - DIALOGUE MANAGEMENT IN SPEECH APPLICATION

user state [McGlashan et al., 1990]. Although it has many principles in common with the plan-based approach, the collaborative agent approach focuses more on the meaning of the dialogue rather than on the structure of the dialogue. As in the case of the frame-based approach, the dialogue control model can be described by using a single algorithm [Traum & Allen, 1994].

3.3.2.6 Theorem-proving systems

Theorem-proving systems are traditional artificial intelligence applications. They use the principles of logic programming to solve problems. The purpose of the dialogue control model is to extract axioms from the user inputs and find solutions for theorems by using the extracted axioms. Theorems can form trees, which represent goals and sub-goals. When all the sub-goals have been satisfied, the original goal is achieved. The system creates new sub-goals when necessary, for example to ask the user for missing information. In other words, the system selects which actions it should take to satisfy the current situation.

3.3.2.7 Dialogue description languages

Dialogue description languages, or dialogue oriented programming languages can be seen as a separate group, although they often use other dialogue control models, such as forms. The most popular dialogue description language for speech applications is VoiceXML, which has recently been widely used in speech applications. VoiceXML uses both state-machines and the frame-based model in its underlying dialogue control model. Dialogues are described using XML, which makes it possible to utilize the huge amount of existing XML tools. From the architectural viewpoint, VoiceXML browsers consist of an interpreter and a set of VoiceXML documents. The building blocks of the dialogue are forms and menus. VoiceXML supports sub-dialogues and embedded grammars. While the frame-approach is used for standard dialogue control, an event-based approach is used for other situations, such as for handling errors.

Dialogue descriptions languages are also used in the Philips speech products. Their HDDL dialogue description language [Aust & Oerder, 1995] is a
declarative language, which is based on event-driven and frame-based dialogue control models. It supports sub-dialogues and allows the manipulation of the recognizer and natural language processing objects. The DDL language is similar in some ways. It is part of the Generic Dialogue System Platform [Baekgaard, 1996]. It is a formal, visual description language based on the frame and event models.

3.3.2.8 Agent-based systems

Agent-based systems are popular these days and various approaches are included this category, see for example [McTear, 2002]. The term agent is very overloaded and used in various meanings in different contexts. In dialogue systems and in communication theory agents may refer to participants in a conversation. The term agent is also used in conjunction with anthropomorphic interface figures. Software agents have many definitions in computer science. There has been a lot of debate about various styles of software agents, such as intelligent agents, autonomous agents and interface agents. In general, this overloaded term is used to describe very heterogeneous systems, and sometimes system without any explicit dialogue control model. Thus, the term agent-based system does not represent any single dialogue control strategy. In particular, it does not characterize a system if the meaning of the term agent is not specified in that particular situation.

There are different definitions of what features agent-based systems should support, but none of these is representative for general use. The requirement of all desired features of software agents, for example, would lead to a situation in which there are not many agent-based speech systems. Similarly, it would not be practical to limit the term agent to cover only systems which utilize artificial intelligence technologies. From an architectural point of view, agent-based systems do not have any specific organizational structure of components.
3.3 - DIALOGUE MANAGEMENT IN SPEECH APPLICATION

3.3.3 Dialogue Management In Speech Applications

Various dialogue control models can be used to construct different kinds of dialogue management strategies, although some dialogue control models are particularly suitable for specific dialogue management strategies. For example, state machine systems are usually more suitable for system-initiative dialogues, while agent-based systems tend to be mixed-initiative [McTear, 2002].

The combination of multiple dialogue strategies and control models is useful and even necessary in complex applications. No single dialogue strategy is suitable for all different purposes. Similarly, different control models offer different benefits. The problem with many mixed-initiative systems is that they do not offer real system-initiative strategy, but instead they take the initiative in some specific situations only. There is a need for a dialogue strategy which can be adapted for individual situations and users. Similarly, applications which use certain control models are usually targeted at specific domains only, and it could be hard to port dialogue managers to other domains. More flexible approaches, which combine multiple dialogue strategies and control models are needed.

One often overlooked question in speech systems is how different dialogue strategies and control models can support reusability and portability. There have been several attempts to create general dialogue systems and reusable components, but in most cases dialogue components are heavily application specific. This is not surprising, since dialogues themselves are very domain dependent. Some parts of the dialogues, however, are fairly general. One obvious example is error handling.

Several attempts to increase reusability and portability have been introduced [Burnett, 2000; Polifroni & Chung, 2002; Bohus & Rudnicky, 2003; Pakucs, 2003; O'Neill et al., 2003]. These are further discussed in Section 3.5.7 and compared with the Jaspis architecture in Section 4.7. For example, separate discourse and task models can be used to separate general conversational tasks from domain dependent and independent tasks. Another approach is the use of modular dialogue components, which can be combined to form complete
dialogues. Dialogue description languages, especially VoiceXML, can be used in a such way. One interesting future direction is the automatic generation of dialogue systems [Polifroni et al., 2003].

Questions of adaptation and reusability are among the most important issues in the constructive part of this dissertation. The Jaspis architecture is designed to take these into account, and I also describe how this is realized in various Jaspis-based applications and interaction techniques.

### 3.4 Multimodal And Multilingual Inputs And Outputs

In addition to the dialogue management, there are other interaction management tasks between the system and the user. Although these are often tied to dialogue management, it is beneficial to model these as separate components, since this makes the systems more modular, adaptive and extensible. Two aspects are especially important: the handling of multimodal and multilingual inputs and outputs, which are addressed next.

#### 3.4.1 Multimodal Interaction In Speech Systems

Multimodal human-computer interaction can be viewed from different perspectives. First, we can make a distinction between the human and the computer sides. Research focusing on the human side tries to find out how different senses are used as modalities between the computer and the human. Computer science oriented research takes the opposite perspective and focuses on the use of multiple communication channels via different system input and output devices.

Multimodality can be limited to concern inputs only, and is defined often in this way. The use of multiple output channels is referred to as multimedia. Since outputs and inputs are not distinct, these two are considered together here and the term multimodality refers to both multimodal inputs and outputs.
Multimodality in its general form is beyond the scope of this dissertation, and the focus is on those multimodal techniques, which are suitable for speech applications.

### 3.4.1.1 Levels of multimodality

One can differentiate four different uses of multimodal inputs and outputs, as presented by Nigay and Coutaz [1993]. The two main dimensions are fusion and the use of modalities. Fusion may be combined or independent. Modalities may be used in sequential or parallel ways. These two dimensions form four main categories for multimodal inputs. They are alternate (combined, sequential), exclusive (independent, sequential), synergistic (combined, parallel) and concurrent (independent, parallel). This categorization is presented in Table 2.

<table>
<thead>
<tr>
<th>Fusion</th>
<th>Temporal use of modalities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequential</td>
</tr>
<tr>
<td>Combined</td>
<td>ALTERNATE</td>
</tr>
<tr>
<td>Independent</td>
<td>EXCLUSIVE</td>
</tr>
</tbody>
</table>

Table 2: Levels of multimodality [Nigay & Coutaz, 1993].

The exclusive use of multiple modalities is the most straightforward way to use multiple modalities. Independent modalities can be used at different times, and they are not combined in any way. From the system viewpoint, the selection and interpretation of modalities in different situations is the main support for multimodality.

More advanced ways to support multiple modalities are alternate and concurrent fusion techniques. If the system supports independent modalities at the same time in a parallel manner, modalities are used concurrently, but their results are not combined in any way. They can be used for different tasks, for example. Conversely, modalities are alternative when they are used at different times, but their results are combined.

The most sophisticated use of multimodality is the synergistic way. Here modalities are combined and can be used at the same time. This puts heavy
demands on the fusion mechanisms, since the fusion of concurrent devices requires that the fusion algorithm is applied every time when some of the devices return results. For example, in speech systems recognition results should be checked continuously to see if they make sense in the current context. In practice systems seldom use such methods.

The synergistic use of multiple modalities does not necessarily mean better or natural interaction. In fact, synergistic use can be hard, since the user must focus on the control of multiple devices, and as a result of this, cognitive load increases. For example, speaking while using a fine-control device may be hard, as well as listening to speech outputs while tracking the visual display. In most speech systems, multiple modalities are used exclusively, alternately or concurrently.

### 3.4.1.2 Fusion techniques

When the results of multiple modalities are combined, fusion techniques are needed. Fusion techniques can be considered at different levels. Nigay and Coutaz [1993] define three levels: *lexical fusion*, *syntactic fusion* and *semantic fusion*. Lexical fusion is the lowest level fusion, and is used when hardware primitives are mapped to application events. Syntactic fusion synchronizes different modalities and forms a complete representation of these. Semantic fusion represents functional aspects of the interface. It defines how interaction tasks are represented using different modalities.

Nigay and Coutaz have presented a general model for multimodal fusion. Their PAC-Amodeus model [1995], which has been used in multimodal speech systems, uses agents for combining different modalities. In addition to the general approach presented by Nigay and Coutaz, there are various fusion mechanisms, some of them utilizing neural networks and other machine learning methods, such as in the IM2.MDM project [IM2.MDM, 2003]. In practice, however, the fusion is usually done in an ad hoc way in practical applications.
3.4 - MULTIMODAL AND MULTILINGUAL INPUTS AND OUTPUTS

3.4.1.3 Fission techniques

Fission is a process in which modalities are selected for outputs. For example, in multimodal speech systems outputs can be expressed by using synthesized speech, non-speech audio, text or graphics. Fission techniques have not gained as much attention as fusion techniques, and this is often thought of as a simple practical issue. Work has been done mainly in the context of multimedia systems, for example in the area of automated multimedia systems [André & Rist, 2000]. The focus in these systems is often more on the rendering of the information for different medias than in the selection of the media for different elements.

In this dissertation I will not concentrate on fusion and fission techniques as such, but instead I take these into account in the Jaspis architecture and its applications. The system architecture should support multimodality by providing an infrastructure in which various methods for modality fusion and fission can be implemented efficiently. In this way actual fusion and fission methods, which are often application specific, can be implemented in a coherent and transparent way. The Jaspis architecture provides a mechanism for applying several fusion techniques in a timely manner as part of its I/O subsystem. For fission (and in general for the mapping of abstract I/O requests into modalities and devices) it also includes a general model.

3.4.2 Multilingual Speech Inputs And Outputs

Multilingual inputs are used in many kinds of speech applications, as presented in Chapter 2. From the technical viewpoint, the first issue in multilingual systems is the selection of multilingual components. In principle, there is a choice between a single multilingual and multiple monolingual components. In practice, multilingual speech recognizers and synthesizers consist of monolingual engines, which share some of their resources or code. For example, a lot of work has been done in the area of multilingual speech corpora. But in
most cases it is up to the application to use monolingual recognizers and synthesizers for different languages.

Most of the research in multilingual applications focuses on translation techniques. They are beyond the scope of this dissertation, and here I omit techniques for such systems. More relevant issues for this work include the contextual, language independent presentation of information and generation of recognition networks and speech outputs for multiple languages. These are investigated in the context of the Jaspis architecture and its applications.

The support for multilingual systems should be twofold. First, the system should support language independent interaction techniques. For example, dialogue management and output generation components should use language independent concepts rather than language dependent concepts whenever possible. Secondly, the system should support language dependent interaction techniques whenever necessary. The second issue is not as trivial as it first sounds. There should be a coordinated way to incorporate language dependent behavior in the interaction, and if possible, implement this in as modular, automatic and transparent a way as possible. In this work I describe how the Jaspis architecture supports both of these ways in its various subsystems. I also present examples of multilingual Jaspis-based applications.

### 3.5 Example Systems

In this section I present speech systems and architectures which are concrete examples of various architectural models, dialogue control models and multimodal fusion techniques. These systems are fairly heterogeneous, but the common feature is that they are all either targeted at speech applications or are used successfully in this domain. In addition to the general presentation of the systems, I also present similarities and differences to the Jaspis architecture, while Chapter 5 provides examples of Jaspis-based speech systems.
3.5 - EXAMPLE SYSTEMS

3.5.1 Dialogue Manager Architectures: MALIN

The MALIN architecture [Dahlbäck & Jönsson, 1999; Flycht-Eriksson & Jönsson, 2000] represents typical dialogue management oriented speech system architectures. The main component is the dialogue manager. All other components are connected to this central component. The MALIN architecture is depicted in Figure 3.

![Figure 3: MALIN architecture](image)

The dialogue manager is mainly responsible for interaction management between the system and the user. As seen on the right of Figure 3, the MALIN architecture contains several (application specific) knowledge sources. The information sources (domain agents) are controlled by the Domain Knowledge Manager. The Dialogue Manager and the Domain Knowledge Manager communicate using messages. The dialogue control is based on dialogue grammars. Dialogue data is represented in a dialogue tree.

The MALIN architecture is a very typical speech system architecture. Similar architectures can be found in many speech systems. There are interpretation and generator components for speech inputs and outputs, the dialogue management component and various information sources. Additional modules are domain specific and related more to the information management than to interaction management.
CHAPTER 3: DEVELOPMENT OF SPEECH APPLICATIONS

3.5.2 Client-server Architectures: Galaxy-II

The Galaxy-II architecture [Seneff et al., 1998] is well known speech application architecture, or more precisely, a platform. The origins of the architecture are in the MIT Spoken Language Systems laboratory. Galaxy-II is used in several MIT applications, including various information service systems for weather forecast and flight information domains [Zue et al., 2000a; Seneff & Polifroni, 2000]. It is also a reference architecture in the DARPA Communicator Program [DARPA, 1998], which involves many research laboratories in the United States. The architecture is released as open source software, and it is maintained by MITRE. In addition to MIT and other Communicator program partners (e.g. CMU, CU, SRI), Galaxy-II is used in many other speech systems.

The central component in the Galaxy-II architecture is the HUB. This is used to coordinate the interaction between other components. Other components in the system are servers, which also act as clients for other servers. Servers are organized by using star topology. In principle this means that all information between components is routed via the HUB, but in practice the system allows direct connections between components for reasons of efficiency. Direct connections are still initialized via the HUB. The overall structure of the architecture is depicted in Figure 4.

Galaxy-II does not limit the number of servers connected to the HUB. The servers present in Figure 4 are examples, and not part of the core system architecture. Galaxy-II distribution contains the system infrastructure, i.e., the HUB and common libraries for the construction of components. Galaxy-II comes with documentation and interfaces for several programming languages. The architecture does not contain additional modules, such as speech recognizers or language generation modules. Some modules, however, are released as part of the accompanying Open Source Toolkit [MITRE, 2003]. In addition, Carnegie Mellon University offers their versions of the Communicator system [Rudnicky et al., 1999] as open source.
The basic units for communication in the Galaxy-II architecture are messages and frames. A frame is a collection of attribute-value structures. Messages between servers and the HUB consist of frames. The semantics of these frames are not specified in any way. The HUB contains scripts for routing the messages between components. These scripts form the top level of the interaction management in Galaxy-II based systems and are used by the HUB to determine which server they should contact in which cases. In addition to the routing of messages, the HUB keeps logs and performs other administrative tasks. The logging is XML-based [MITRE, 2003], and timestamps can be used for coordination of multimodal inputs.

The MITRE Open Source Toolkit [MITRE, 2003] provides several ready made servers, which include audio services and wrappers to the Festival synthesizer [Black & Taylor, 1997] and to the PostGreSQL database program [2003]. It does not contain any speech technology components as such, and the number of interfaces for existing speech technology engines is still rather limited. Besides technical servers, several general servers and extensions have been introduced for tasks such as context tracking [Filisko, 2002], XML-based messaging for multimodal applications [Kuo, 2002], rapid configuration of the dialogue
management [Crockett, 2002] and enhancing the development with graphical interface [Glass & Weinstein, 2001].

To conclude, the Galaxy-II provides the basic architectural principles and software infrastructure for the development of speech applications. In this sense it is similar to the Jaspis architecture. Galaxy-II, however, is a very general architecture, which means that it does not limit applications to follow any specific paradigms, but on the other hand it does not support application development in other ways than providing the very general framework. It does not, for example, provide any information management services or ways to make the interaction adaptive. These are among the issues which I address in the constructive parts of this dissertation.

3.5.3 General Agent Architectures: OAA

Open Agent Architecture [Martin et al., 1999] is not a speech-specific architecture, but a general purpose framework for the construction of agent-based systems. It has been used in many speech systems as a base infrastructure, which makes it especially interesting in this context. Here I introduce the main principles of OAA.

The heart of Open Agent Architecture is the Facilitator. This is similar to the HUB in the Galaxy-II architecture, and other central components in distributed systems. Although a system usually contains a single facilitator, OAA makes it possible to use multiple facilitator agents inside a single system. Other agents in an OAA system are meta agents, service agents and requesting agents. This is depicted in Figure 5.
3.5 - EXAMPLE SYSTEMS

Requesters specify goals to the Facilitator, also giving advice on how these goals should be achieved. Service providers offer services such as timetables or speech recognition capabilities for the Facilitator. They might be wrappers for underlying legacy components. Meta-agents contain domain or goal-specific information which is used by the Facilitator. The interaction between agents is coordinated by the Facilitator. It receives requests, finds suitable services and delivers results back to the requesters. It also provides functionality for broadcasting messages and establishing direct connections. This is called a delegation model. The Facilitator delegates requests for other agents, which in turn may present further requests in order to solve their tasks. The Facilitator is used as a coordinator, not a controller, as central components are viewed in many systems.

Interagent Communication Language (ICL) is used for communication between agents. ICL includes conversation and content layers. The conversational layer contains event types and their associated parameters, while the content layer contains Prolog-style expressions for specifying goals, triggers and data elements. Every agent specifies a set of solvables, which form high-level interfaces. There are two types of solvables: procedure solvables and data solvables. The latter can be used to implement shared databases.

OAA can be used in several ways in speech applications. Systems like the MITRE system [Luperfoy et al., 1998] uses it mainly to provide an infrastructure for communication between system components. In the MITRE system, the

![Figure 5: OAA agents](image)

*Figure 5: OAA agents [Martin et al., 1999].*
dialogue manager is still the main component, and it is used to coordinate the interaction between the system components. OOA has been used for more distributed processing models as well, for example in the WITAS system (presented in the next section).

There is no special support for speech applications in the standard OAA distribution, although it has been used successfully in speech applications and multimodal systems [Moran et al., 1997]. Several agents have been implemented in the DIPPER (“Dialogue Prototyping Equipment & Resources”) project [Bos et al., 2003]. DIPPER is a set of speech extensions to be used together with the OAA architecture. It mainly includes wrappers for speech technology engines, and for several dialogue and natural language processing components. Nevertheless, it can be concluded that OAA generally lacks available speech specific agents.

To conclude, OAA provides a strong infrastructure for distributed applications. Its future development includes the organization of facilitators for large systems, transaction management and modularization of the facilitator functionality. OAA2 adds also support for generalized data management. In the Jaspis architecture these questions are also addressed. The organization and selection of agents are central issues, which not only help to make systems more modular and manageable, but also makes it possible to choose between agents when multiple suitable agents are present.

3.5.4 Dialogue Manager Oriented Agent Architectures: WITAS

The WITAS system [Lemon et al., 2001] is a multimodal conversational system with autonomous mobile robots. Concretely, the system deals with robot helicopters. In such settings, dialogues are asynchronous, mixed-initiative, open ended and take place in dynamic environments. In order to support such dialogues, the WITAS system uses distributed agents of the OAA architecture.
3.5 - EXAMPLE SYSTEMS

There are six agents in total, the main component being the dialogue manager. This is depicted in Figure 6.

![Figure 6: The WITAS system [Lemon et al., 2001].](image)

The heart of the WITAS system is the dialogue manager, which coordinates dialogues and multimodal inputs. It maintains an *information state* corresponding to the dialogue state. The information state contains structures such as a stack of issues raised, a robot agenda, a list of salient issues, a modality buffer and various databases. In the second version of the system, the dialogue stack has been expanded into a tree. It also contains a dynamic task tree.

The WITAS system represents a typical approach to the use of agent-based architectures in speech systems. The underlying OAA architecture supports distributed interaction management, but the dialogue manager is still the main component which coordinates interaction and problem solving. All the complexity of interaction management is encapsulated in the dialogue manager. System information is also embedded in the dialogue manager. The dialogue is not, however, controlled by the dialogue manager, but instead it responds to events introduced by other agents. In this way, the dialogue manager is a kind of blackboard.

The WITAS system introduces many interesting questions, such as how the monolithic dialogue manager could be decomposed into more modular agents, how the selection and coordination of these agents can be done, and how
information sharing should be done. In the Jaspis architecture these questions are addressed and solutions are presented. The authors of the WITAS system have also addressed these questions recently [Lemon & Cavedon, 2003].

3.5.5 Dialogue Manager As Controller: MITRE

Luperfey et al. present an interesting variation in dialogue centered systems in their 1998 paper. The focus of the system is on discourse processing, which covers dialogue management, context tracking and pragmatic adaptation aspects. The architecture is targeted at near-future applications in which multiple participants take part in the dialogues.

Figure 7: MITRE architecture [Luperfey et al., 1998].

The central component in the MITRE architecture is the dialogue manager, which controls the interaction between the computer and the human. The interaction between system components is also controlled by the dialogue
3.5 - EXAMPLE SYSTEMS

manager. In this way, it “moves away from the standard NL pipeline” [Luperfoy et al., 1998]. In practice, this means that the MITRE system is a combination of batch-processing and centrally coordinated architectures. There is a default order of execution between components, but the dialogue manager may take the turn whenever feasible. The need may come from dialogue disfluencies, such as interruptions, repairs and back-channeling. The dialogue manager also has direct access to all other components. The architecture is depicted in Figure 7.

The implementation of the MITRE architecture uses Open Agent Architecture as its base infrastructure. Each system component is an OAA agent. The dialogue manager is in a special position and it is called frequently so that it can monitor the interaction between other components and to react whenever necessary. The dialogue manager is a kind of blackboard structure which drives the interaction in certain situations. Otherwise the default interaction processing order is used for agents. The architecture has been used for several internal applications, but not released for public use.

The MITRE architecture shows how there can be different levels for interaction processing and how the default processing can be changed dynamically. It uses one centralized component, which must continuously poll system interaction sources. Other components, although implemented as OAA agents, cannot react differently. Furthermore, there is no further modularization, and the adaptation is applied for a few phases only. The Jaspis architecture uses similar features, except that all components can react to the interaction in a controlled way and the system level adaptation takes place in all phases of the interaction.

3.5.6 Distributed Agents: TRIPS

The work done at the University of Rochester includes the dialogue systems TRAINS [Allen et al., 1996] and TRIPS [Ferguson & Allen, 1998]. In their 2001 paper Allen et al. [2001b] present several current topics for spoken dialogue systems, including the need for incremental generation and interpretation of outputs and inputs and flexible turn-taking to achieve more natural interaction
style. They present strong arguments for favoring systems in which the interaction management between system components is flexible. They also consider portability as an important issue. These are also key motivations for this dissertation.

The basic idea of the TRIPS architecture is the notation of loosely coupled components which exchange information by passing messages. The system architecture is modular, and, for example, there is no dialogue manager as such, but instead the functionality of traditional, monolithic dialogue managers is divided into smaller components. The TRIPS system architecture is depicted in Figure 8.

**Figure 8: TRIPS architecture [Allen et al., 2001a].**

The Interpretation Manager detects and interprets indirect speech acts, coordinates reference resolution, computes system discourse obligations and performs conversions from linguistic knowledge representation to domain specific knowledge representation. It uses several agents in this process (Reference Manager and Discourse Context). The Task Manager recognizes intentions, performs mappings between generic problem solving acts and domain
specific actions, and answers queries about specific objects and their role in the
task and domain. The Behavioral Agent does not have domain specific
knowledge, and deals only with generic problem solving acts.

The underlying infrastructure contains the Facilitator, which routes messages
between other system components. In a way, the Facilitator acts more like a
HUB in the Galaxy-II architecture than the Facilitator in the Open Agent
Architecture, especially since it does not contain any control logic. On the other
hand, it provides similar services as OAA, such as the sending of messages.
From the architectural viewpoint, the Facilitator is a HUB, and components are
organized into star topology. The architecture also contains shared knowledge
sources, such as the Discourse Context.

The TRIPS architecture contains many features which I consider essential for the
next generation of speech systems. The modularity of the system is a key feature,
and together with shared information sources it makes it possible to build
systems which do not suffer from the limitations of the central dialogue control
approach. While the MITRE system is a move away from centralized control, the
TRIPS architecture offers more flexible interaction management between system
components and in this way opens us new possibilities for interaction between
the human and the computer.

There are several issues, however, which I consider as potential bottlenecks for
flexible interaction. First, the TRIPS architecture still uses rather monolithic
components, and there are no architecture level sub-organizations or ways to
handle multiple components for the same purposes. Also, the architecture does
not provide any special support for shared information management. The
message passing paradigm can be problematic, since it easily leads to direct
connections between components. These issues are addressed in the Jaspis
architecture and are discussed in the next chapter.
CHAPTER 3: DEVELOPMENT OF SPEECH APPLICATIONS

3.5.7 Dialogue Management: CMU Agenda, Queen's Communicator And SesaME

Next I review three dialogue management approaches. These are the *agenda-based dialogue management architecture* [Rudnicky & Xu, 1999], the *Queen's Communicator* [O'Neill et al., 2003] and *SesaME* [Pakucs, 2003]. The purpose of these approaches is not to provide a complete speech architecture, but instead a model for dialogue management. These three models are particularly interesting approaches for dialogue management. First, they all use highly modular components (“agents”) to perform compact (well-defined) dialogue tasks. I consider this a very useful and efficient feature, since it supports separation of domain independent and domain specific tasks and it also supports reusability. Secondly, the dialogue control is not predefined, but instead evolves dynamically. This allows the system to be dynamic and adaptive. These features are addressed in the Jaspis architecture in a general way. In fact, it would be possible to implement these three dialogue models efficiently using the Jaspis architecture.

The *agenda-based dialogue model* has three key concepts: *handler*, *product* and *agenda*. Handlers are used to manage interaction on a small scale, while the product refers to agreed information and the agenda represents the structure of conversation topics. The architecture uses task-oriented dialogue control model.

In contrast to predefined tasks, the agenda model introduces dynamic dialogue tasks. A product is a tree which contains all information needed to complete a task. It also determines the organization and order of the information. A product tree is formed from a library of sub-trees, and it can be modified during the dialogue. The nodes in the tree are handlers which contain functionality related to particular information. They perform transformations for inputs which they receive via their receptors, i.e., they take user inputs and generate responses. Handlers deal only with sub-dialogues, which are focused on a particular information.
An agenda is an ordered list of handlers in a product tree. In other words, an agenda corresponds to a depth-first generated path of tasks. The handler at the top of the agenda (i.e. the leaf node of the topic path) is the focus of the dialogue. Nevertheless, it is possible to activate any of the other handlers. Bottom handlers are general handlers for topics such as requests for help. There are also dependencies between nodes, which are used to notify dependent nodes about changes in information states.

The control of interaction is performed in two phases. The input phase starts from the topmost handler of the agenda. When a handler processes information, it notes that the information (e.g. part of the user input) has been consumed so that other handlers do not try to process that particular piece of information. The second phase is the output phase, in which each handler is able to produce outputs. Handlers can also produce outputs during the input phase, and produce return codes, which are used to switch to the output phase, exit, promote the current handler and so on.

**RavenClaw** [Bohus & Rudnicky, 2003] extends the agenda-model using a two-tier architecture, which introduces Dialog Task Specifications. These form the upper layer of the dialogue management, while the agenda-model is the low-level dialogue engine. In the RavenClaw architecture compact dialog agents handle specific dialogue tasks. Dialog agents form the Dialog Task Specification tree. The Dialog Engine executes the tree in a dynamic manner using a stack, as presented above.

The aim of the RavenClaw system is to provide a clear separation of discourse and task specifications, as represented by the dialog stack and the task tree, correspondingly. The dialog engine uses domain-independent sub-trees for common tasks, such as confirmations. In this way, the system development focuses on domain-dependent task specification, while the dialog engine selects general “conversational strategies” when feasible. According to the authors, this aids rapid development of spoken dialogue systems.
The original agenda architecture is implemented as a part of the CMU implementation of the DARPA Communicator [Rudnicky et al., 1999]. As in other Communicator systems, Galaxy-II architecture is used as the underlying infrastructure. The RavenClaw architecture is used in several applications, most of them utilizing the Galaxy-II infrastructure, and one utilizing the Open Agent Architecture.

The Queen's Communicator [O'Neill et al., 2003] is based on the object-oriented dialogue management approach [O'Neill & McTear, 2000]. In this approach dialogue components use inheritance to separate generic functionality from domain-specific functionality. The system contains a DialogManager, which uses a set of domain experts to handle domain-dependent tasks. Generic dialogue tasks (e.g. confirmations) are implemented by the DiscourseManager. Since domain experts (agents) extend the DiscourseManager, the experts contain all the common functionality implemented by the DiscourseManager. The DomainSpotter is used for the selection of suitable domain experts.

The general behavior of the dialogue management focuses on a pragmatic confirmation mechanism [O'Neill & McTear, 2002]. In this model the attributes of concepts (e.g. “departure-place”) used in the dialogue management are confirmed either implicitly or explicitly by the DiscourseManager before they are used by the domain experts. Each attribute has two variables: the confirmation status (e.g. “new for system”, “repeated by user”) and the discourse leg. The discourse leg indicates the confirmation level of the attribute. It is incremented by one each time the value of the attribute is confirmed, and set to zero when the value of the attribute is modified. Only attributes greater than zero are used by the domain experts.

The domain specific dialogue management performed by the domain experts is represented using heuristic rules. The rules are represented in a declarative manner. Two kinds of rules are used: user-focused rules and database-focused rules. User-focused rules request information from the user, such as missing attributes. In addition, they initiate database operations. Database-focused rules
are used when database queries fail, for example to relax constraints used in queries.

The dialogue manager of the Queen's Communicator is implemented using the Java programming language. The underlying infrastructure is Galaxy-II, and other modules of the system are provided by the CU Communicator implementation [Pellom et al., 2001].

SesaME [Pakucs, 2003] is an agent-based dialogue management architecture targeted for dialogue management in multiple domains. The key feature is the dynamic handling of distributed dialogues. The central component in the architecture is the Interaction Manager. It handles generic dialogues tasks, such as handling of errors and requesting of missing information. In addition, coordination tasks, such as access to the shared blackboard is performed by the Interaction Manager.

Domain specific dialogue management is performed by a set of autonomous agents. SesaME agents are compact and specialized for atomic tasks. They are organized into layers and executed on the basis of triggers. Since multiple agents may contribute to the same problem, special decision agents are used to choose between the results of agents.

The dynamic approach for handling of multiple domains is modeled using the Dialogue Description Collection. Dialogue descriptions are extended VoiceXML scripts. These are transformed into Java objects and further processed by The Dialogue Interpreter. A frame-based dialogue control model is used by the Dialogue Interpreter. The Dialogue Description Collection is updated from the base of the services available in the environment.

The SesaME architecture is based on the Java programming language and uses the ATLAS platform [Melin, 2001] for speech technology integration.
3.5.8 Multimodal Fusion: PAC-Amodeus

The coordination of multimodal inputs and outputs is attracting more attention in speech systems. The PAC-Amodeus model [Nigay & Coutaz, 1995], which is depicted in Figure 9, is one of the first models for managing multimodal inputs. Next I present the overall architecture of the PAC-Amodeus model and the main principles of its fusion mechanism.

Figure 9: PAC-Amodeus model [Nigay & Coutaz, 1995].

The PAC-Amodeus model derives from the ancestors of user interface architectures, especially from the ARCH [UIMS, 1992] and PAC (presentation – abstraction – control) [Coutaz, 1987] meta-models. The model consists of five main components, which are Dialogue Controller, Interface with the Functional Core, Functional Core, Presentation Techniques Component and Low Level Interaction Component. The Dialogue Controller contains a fusion mechanism. Conceptual Objects are exchanged between the Dialogue Controller and Interface with the Functional Core. Similarly, Presentation Objects are exchanged between the Dialogue Controller and the Presentation Techniques Component. Domain Objects and Interaction Objects are exchanged between the Interface with Functional Core and the Functional Core, and between the Presentation Techniques Component and the Low Level Interaction Component respectively.
The PAC-Amodeus model provides an abstract model for all user interface software development. It uses several levels of abstraction for exchanging information between different levels. It also takes domain independency and dependency into account. The overall principle of this model is also part of the Jaspis architecture, although the actual system structure of Jaspis is more complex.

The PAC-Amodeus model uses *melting pots* as a metaphor for hierarchical fusion of modalities. A melting pot is a two-dimensional structure, which represents *tasks objects* (vertical axis) in a *timely manner* (horizontal axis). When events are received from the user, they are abstracted and assigned to melting pots. Mapping is done on the basis of their meaning to the dialogue and from the time when they were received. In a way, melting pots are similar to frames. Melting pots form application specific structures, and they are embedded into *fusion engines*. This model is illustrated in Figure 10.

![Figure 10: Fusion of two melting pots in the PAC-Amodeus model [Nigay & Coutaz, 1995].](image)

The fusion engine tries to perform three types of fusion in the following order: *microtemporal fusion*, *macrotemporal fusion* and *contextual fusion*. Microtemporal fusion takes place when the complementary structural parts (task objects) of two melting pots overlap in time. In other words, microtemporal fusion combines *parallel* inputs. Macrotemporal fusion combines melting pots...
which do not overlap chronologically, but are close in time. It takes place when two pots belong to the same temporal window, i.e., they are sequential. Contextual fusion does not take time into account, but instead it combines inputs according to the overall system context. This is a highly application specific process.

Nigay and Coutaz present in their 1995 paper actual methods for fusion. The model is realized by using PAC agents and a set of fusion rules. Fusion rules take into account issues such as the handling of redundancy and erroneous fusions. Redundancy occurs when the same information is received from multiple sources during microtemporal fusion. Erroneous fusions can take place since the fusion mechanism continuously combines inputs without taking different time scales into account. This concerns speech inputs especially, since the processing of speech signals takes more time than the processing of direct manipulation input sources, such as mouse events or keystrokes. In the Jaspis architecture multimodal fusion is performed as part of input processing. The actual method for fusion is not predefined, since this would be a limitation for a general architecture.

### 3.6 Speech Application Development Tools

One of the key problems in the construction of speech applications is the lack of development tools. For example, only two of the architectures presented in the previous section are freely available to application developers. These frameworks, namely Galaxy-II and Open Agent Architecture offer only general infrastructure components, but no speech technology components or other development tools. Commercial development environments exist, but due to costs they are mostly unsuitable for small-scale development and research. In this section I present several speech application tools focusing on generally available tools. The tools presented include complete development environments as well as focused Wizard of Oz tools.
3.6.1 CSLU Toolkit

The CSLU toolkit [Sutton et al., 1998] is a platform for spoken dialogue application development. The development of the toolkit started already in 1992. The purpose of the toolkit is to provide a complete platform for research, language training, education, corpus collection and corporate use. The toolkit contains components for basic technologies (speech recognition, audio visual speech synthesis and natural language processing), tools for audio-visual speech processing, as well as tutorials and examples. The toolkit is freely available for non-commercial use. There is a separate license for commercial use.

The CSLU toolkit provides four levels of interface for application developers. At the lowest level developers can use C code to access the underlying technology components, such as the speech recognizer. There is no support for general speech application interfaces, such as Microsoft SAPI or Java Speech API. The package level is an intermediate layer between the script-language level and the low-level C libraries. The script layer is most useful for those application developers who want to extend the functionality of the applications created with the uppermost GUI layer.

The GUI layer (the RAD layer) provides a graphical interface for the direct manipulation of speech and dialogue objects. Users can create applications by connecting objects together and modifying their properties, such as recognition parameters. The graphical interface is depicted in Figure 11. As with RAD tools in general, the main focus is on the ease of use. The dialogue control is based on the finite-state model and offers features such as sub-dialogues and repair sub-dialogues. As discussed earlier, the state-machine dialogue control model is especially suitable for small-scale applications with well-defined task structure.

One of the main target uses of the CSLU toolkit is education. We have used it for teaching spoken dialogue systems in several courses. We found it suitable for the development of small-scale student projects, and students have implemented working applications using the toolkit. However, the toolkit is not suitable for all kinds of projects (e.g. large-scale applications and multilingual applications).
The main reasons are the lack of support for languages other than English and Spanish, the unavailability of common programming interfaces and the unfamiliarity of the Tcl scripting language (although relatively popular, it can still be considered to be a minor language if compared to languages such as Java). The state-based dialogue control model is also a limitation in some application areas. Nevertheless, basic speech application features such as menus, system vocabularies and error handling procedures can be implemented fairly easily. Similar experiences are reported in [McTear, 1999].

Figure 11: Dialogue design in the CSLU Toolkit [Sutton et al., 1998].
The popularity of the CSLU toolkit supports the assumption that there is a need for rapid application development environments. Speech application development is a highly iterative process, and the curve from the design phase to the first prototypes must be shortened. The Jaspis architecture does not yet offer a complete development environment for the construction of speech applications, but some tools and support for developers is included in the Jaspis architecture. Some of these tools are presented in Chapter 6.

3.6.2 SUEDE

SUEDE [Klemmer et al., 2000; Sinha et al., 2002] is another GUI-oriented speech application tool. Like the CSLU toolkit, it is freely available. The main focus is on rapid prototyping and Wizard of Oz tests, and the tool does not offer any other support for the building of actual applications. The tool is targeted mainly at interface designers.

Figure 12 depicts a typical design session with SUEDE. In addition to the design mode, SUEDE contains two other modes for the testing and analysis of dialogues. In the design phase dialogue scripts are created. In this context scripts (the topmost part of Figure 12) are dialogue examples, which the designer constructs by recording the computer's prompts and typical user responses from his/her own voice. The main part of the display consists of a design graph. The design graph is created from the script elements or new nodes.

In the test mode the design graph (state-machine) is transformed into a set of HTML documents. These documents correspond to nodes in the graph. The wizard has options for choosing actions which are based on the transformations between nodes. The wizard operates the GUI and the user hears the computer responses. There is a simulated error mode and the user responses are recorded for later analysis. In the analysis mode the designer sees the actual recorded scripts along with an annotated design graph.
SUEDE is a practical tool for the early design phases of the application. Its ability to allow quick prototypes with a testing facility is appealing. The main drawback is that the results cannot be used directly for the actual application development. It also does not allow the use of any speech technology components, which makes it less useful for many purposes. The design principles of SUEDE are still noteworthy and should be part of any speech application development toolkit.

The main problem with tools such as the CSLU toolkit and SUEDE is that when the development reaches the limits of the toolkit, the development must be redone all over with the real tools. This is time consuming, a lot of effort is wasted and re-engineering is harder to accomplish. Instead of testing separate tools for design and implementation we should find ways which help us to integrate these tools and make them accessible to designers and developers alike.
In the constructive part of this dissertation I present how the Jaspis architecture provides support for the design and prototyping of applications without limiting the capabilities of the system.

3.6.3 Microsoft Speech API

The Microsoft Speech Application Development toolkit [Microsoft, 2003] has matured over the years to be a useful environment for speech application development. The heart of the system is the speech application development interface (SAPI) and core classes for building speech applications with the unified API. The toolkit also provides all the necessary speech technology components for speech recognition and speech synthesis, along with several basic tools. High-level interfaces and advanced tools on top of SAPI are also available, such as SpeechStudio Suite [SpeechStudio, 2003] and Chant SpeechKit [Chant, 2003].

SAPI uses object-oriented and layered architecture models. It provides several layers of programming interfaces, which differ in complexity. On all levels the key elements are speech objects. SAPI offers objects for recognition engines, recognition grammars and speech synthesis. The architecture does not follow any particular dialogue control model, but instead assumes that developers implement the dialogue control.

Another attempt to create an unified application programming interface is Sun Java Speech API (JSAPI) [Sun, 2003]. It provides similar interfaces and speech objects as SAPI, but it lacks speech technology engines. Although the JSAPI specification and core class implementation are not new (it has been available since 1998), the current list of JSAPI compatible speech technology components is rather short.

From the development point of view it is important that the base architecture should support as many speech technology components as possible. Although the core architecture and all interaction components should use as high-level interfaces as possible, there should be concrete interfaces for various technology
components as well. The Jaspis architecture uses three approaches to provide both general and concrete interfaces. First, it encourages the use of general formats such as Annotation Graphs [Bird & Liberman, 2001] and JSML [Hunt, 2000b]. Second, it contains converters, which transform representations between formats, for example from JSML to SAPI. Third, it provides virtual engines, which provide interfaces for common speech technology interfaces such as SAPI.

3.6.4 VoiceXML

VoiceXML is an interface markup language, or dialogue description language, as presented in Section 3.3.2.7. There are a number of tools available for VoiceXML application development. One such is Nuance Voice Platform [Nuance, 2003], which provides a development environment for VoiceXML application development. Using the environment users can develop applications fairly easily. Nuance also provides servers for deploying these applications. Similar tools are available from other vendors as well. There are also freeware and open source VoiceXML tools, such as OpenVXI interpreter [2003].

VoiceXML provides an interesting option for rapid application development and prototyping. Many applications can also be done with this alone [Bennett et al., 2002]. An interesting question is how it can be used as a part of more complex dialogue systems. This approach is taken, for example, in the SesaME system [Pakucs, 2003], in which personalized VoiceXML dialogues are used according to the context. SALT [Wang, 2002] is a promising competitor for VoiceXML. It is more multimodal oriented than the rather speech-specific VoiceXML.

To summarize, several tools for speech application development exist, but the presented systems and tools are limited in various ways. Pipeline systems are limited to strict turn-based communication. It is quite hard to extend these systems for other domains, such as pervasive applications. The same applies for specialized architectures and tools, such as the CSLU toolkit and most...
commercial applications. In addition, these are restricted to certain types of
dialogue control models. General system architectures or application
programming interfaces provide extensible infrastructures for a variety of
application areas, but they do not offer concrete middleware components for
practical application development. Furthermore, they do not offer methods for
detailed coordination of applications, such as for internal coordination of
modules and system level adaptation.

When both human-computer issues and speech application development aspects
are taken into account, a set of design principles for an advanced speech
application architecture can be formed. According to these, the architecture
should be *generic, extensible, adaptive and flexible, modular and distributed,*
and it should support *collaborative application development, reusability and
standards.* In the next chapter I present how the Jaspis architecture addresses
these.
4 JASPIS ARCHITECTURE

Jaspis is a general speech application architecture designed for the challenges of advanced speech applications, especially multilingual and adaptive speech applications. While Jaspis is a general conceptual architecture, it is also a concrete framework which provides components for application development. This chapter focuses on general architectural principles, but also presents framework issues, such as what kind of components the implementation contains. The architecture was originally presented in [Turunen & Hakulinen, 2000a].

The Jaspis architecture is designed to provide support for human-computer interaction tasks, such as error handling, Wizard of Oz studies and corpora collection. These issues are discussed further in Chapter 6, which presents the development environment oriented aspects of the architecture. Application issues, including concrete examples from several Jaspis-based applications are presented in Chapter 5. A new version of the architecture, which supports concurrent and distributed dialogues, is introduced in Chapter 7.

The first section gives an overview of the architecture including its design principles. The following two sections present information and interaction management components. The next section introduces dialogue, presentation and input management modules. Communication, i.e. a low-level input and output module is presented after that. Next, the most important implementation issues are presented. Finally, a discussion and a comparison to other architectures are provided.
CHAPTER 4: JASPIS ARCHITECTURE

4.1 Architecture Overview

The main purpose of the Jaspis architecture is to provide a general and flexible framework for speech application development. Next I present those principles which are used in the design of the Jaspis architecture. According to these, an architecture should be generic, extensible, adaptive, modular and distributed, and it should support collaborative application development, reusability and standards. These principles are motivated by the technology, human-computer interaction and application development viewpoints of the previous chapters. After the presentation of design aspects, I introduce the main principles of the Jaspis architecture.

4.1.1 Design Principles

As presented in the previous chapters, speech applications need adaptive interaction methods in all system modules. For example, adaptive interaction methods are needed in speech outputs and speech inputs, which should be tailored to the language of the users. Similarly, dialogue management components should adapt the interaction flow to the situation at hand, for example by using different dialogue management strategies. Interaction management relates to adaptivity. For example, the pipeline model, which is used in many speech systems, is not particularly suitable for interactive applications. In order to support more natural interaction, more flexible interaction models are needed.

The second main design principle of the Jaspis architecture is modularity. Modular components support reusability and they are easier to maintain and extend than monolithic components. Modularity is also a key element in adaptation. Modular systems can be distributed efficiently. In practical speech applications different components, such as speech recognizers, speech synthesizers and natural language processing components often operate on different platforms, and a heterogeneous system setup is needed. In pervasive
applications, concurrent dialogues can be widely distributed to multiple computers.

The third design principle concerns collaborative and iterative application design and development. Speech applications are often developed by multidisciplinary groups, and solutions to the same problems are offered by using different approaches. For example, rule-based and statistical methods may be used alternatively and complementarity to interpret spoken utterances.

A speech architecture should provide an extensible and practical infrastructure for application development. Support for extensible functionality is important in practical development work, since application developers have existing solutions which should be compatible with the architecture necessitating a minimum amount of integration work. Standards and general solutions are key aspects in modern application development, and the system architecture should support these whenever possible.

4.1.2 System Principles

In order to support the design principles mentioned, the Jaspis architecture uses a modular and distributed system structure, an adaptive interaction coordination model and a shared system context. These form the basic infrastructure on which other features and components of the system are based.

Jaspis uses several architecture models in its various system modules. The overall structure is illustrated in Figure 13, which is a typical Jaspis-based system setup with several modules present. The top-level structure of the system is based on managers, which are connected to the central manager. This central component, the Interaction Manager, is similar to the facilitator and hub components in some architectures. Managers are connected using a star topology structure. Communication between components is organized as a client-server relationship paradigm.
CHAPTER 4: JASPIS ARCHITECTURE

Inside the system modules are local subsystems. The interaction coordination model of the Jaspis architecture is based on managers, agents and evaluators. Agents are interaction components which implement different interaction techniques such as speech output presentations and dialogue decisions. Evaluators are used to evaluate different aspects of the agents, i.e. to determine how suitable they are for different tasks. Managers are used to coordinate these components.

All information in Jaspis-based systems is stored in the shared Information Storage. All components of the system may access the content of the Information Storage using the Information Manager. Physically the Information Storage can be distributed similarly as other components.

The Jaspis core infrastructure provides conceptual models and reference implementations for presentation, input, dialogue and communication management modules. For example, the communication management module contains methods for coordination of devices, such as speech recognizers, and for low-level distribution of system components. The Information Storage and Information Manager are also part of the core infrastructure. These are presented

Figure 13: Top-level organization of modules.
4.1 - ARCHITECTURE OVERVIEW

in more detail in the following sections of this chapter. Beside these modules, the architecture contains several other modules, e.g. for development tasks. These are described in Chapter 6.

4.2 Information Management

Information management is a crucial aspect in adaptive, modular and distributed applications. There are three main methods to exchange information between system components: parameter passing, message routing and shared blackboards. Parameter passing is mainly used in component oriented systems, and especially in batch-sequence systems, where each component passes the required information to the next component in the pipeline. In the message passing approach components do not talk directly to each other, but instead a router component is used to deliver messages to recipients. In the blackboard approach all system information is stored in the shared knowledge base, which is used by all system components.

Each of these approaches has its advantages and disadvantages. For example, message passing is a straightforward and efficient way to exchange binary data, but hard to use for distribution of information for multiple recipients. This is where the message routing approach is particularly suitable. Message routing, on the other hand, may cause efficiency bottlenecks if messages are multiplied for each recipient. The blackboard approach provides several advantages for adaptive and distributed applications. Most importantly, it allows the utilization of shared information by all system components. The main drawback of this approach is the lack of control [Martin et al., 1999].

Jaspis has a shared Information Storage, which is a kind of blackboard, or more precisely, a database, since the system is controlled by the Interaction Manager. The architecture allows direct connections between certain components to ensure efficiency for binary data transfer (e.g. transfer of speech data). In the Jaspis² architecture a more blackboard and message routing based approach is used.
CHAPTER 4: JASPIS ARCHITECTURE

Nevertheless, the coordination and control are performed by the Interaction Manager to achieve architecture level coordination.

4.2.1 Information Management Overview

Jaspis contains a layered architecture for information management, as illustrated in Figure 14. The architecture consists of four layers. The bottom layer is the database layer, which contains the Information Storage, which holds all the shared system information. Other system components do not access the Information Storage directly. Instead, they use the Information Manager for this purpose. It forms the top layer, which provides an application interface to the Information Storage. The two middle layers are communication layers. The Information Access Protocol defines the Information Storage operations. Standard communication protocols (such as XML-RPC) may be used for communication between the Information Manager and the Information Storage.

Layered information management has several advantages. First, by using a layered architecture additional functionality may be implemented in the Information Manager while keeping the Information Storage and the Information Access Protocol constant. Similarly, the Information Manager may be ported for...
4.2 - INFORMATION MANAGEMENT

new programming languages without the need to port the Information Storage, which may be shared between languages.

The second advantage is that new versions of the Information Storage may be implemented without need for other modifications. For example, domain related information may be stored in a relational database, while system information may use hierarchical XML representation, and speech data may be stored using a methods optimized for audio recordings.

The third advantage of the layered information management model is that because information access protocol and communication protocol are separate, various communication methods may be used. The reference implementation provides both ad hoc socket and XML-RPC implementations. Furthermore, these can be used simultaneously, as demonstrated in Figure 15. This communication model is not specific to the information management, but instead is general for the whole system. It is presented in more detail in Section 4.5.

Multiple Information Managers may be present inside a single system. For example, there can be implementations of the Interaction Manager in different programming languages. In distributed systems, which run on multiple computers, each computer may have its local instance of the Information

Figure 15: Information management components.
CHAPTER 4: JASPIS ARCHITECTURE

Manager. This is one way to make systems more modular and distributed and prevent efficiency bottlenecks with a single Information Storage. The right side of Figure 14 illustrates how multiple alternative instances of components may be inside information management layers. As illustrated in Figure 15, the Information Access Protocol uses XML for information exchange. This is presented in more detail in Section 4.6.3. Next, the information management components are presented in detail.

4.2.2 Information Storage

The Information Storage holds all the shared system data, i.e. the shared system context. The Jaspis architecture assumes that individual components do not store any high-level information inside them, but instead they use the Information Storage for that purpose. This makes the interaction components stateless and the system is able to adapt to interaction by choosing proper components for each interaction situation. To make this possible, the system assumes that every component updates its knowledge from the Information Storage when activated and writes modified information back to the Information Storage when deactivated.

Since real-world systems may contain a large amount of data, it is not practical to store all low-level information in the shared storage. System components should store only necessary high-level information inside the Information Storage. In the ideal case, the shared data should be represented on such a conceptual (abstract) level that it can be used by other components as well. As an example, information should not be stored in a programming language specific or technology specific format. General representations, such as Annotation Graphs [Bird & Liberman, 2001], may be used to represent all linguistic information. Section 4.6.4 presents how Jaspis supports various standards, and Section 6.5 presents its support for Annotation Graphs.
The programming interface for the Information Storage is straightforward: it provides one method call. This *process* method is shared among all Jaspis components (see Section 4.6.1 for more information). It takes a single parameter, an XML document, which defines all operations which the Information Storage is able to handle. In a nutshell, the Information Storage takes XML requests and produces XML results. The *Information Access Protocol* defines the format of the Information Storage requests and results. This will be presented in detail in Section 4.6.3.

The Information Storage offers only a minimal set of operations needed to manipulate its contents. All other functionality is implemented by the Information Manager. For example, all programming language specific mappings and sophisticated functionality are provided by the Information Manager.

The reference implementation of the Information Storage uses XML for its internal information representation. Technology-wise, the Information Storage is an XML document, which is implemented as a W3C Document Object Model (DOM) object [DOM, 2003]. This allows efficient manipulation of its content using tree operations, since DOM objects are trees.

When all the information is represented using XML, there is no need for transformations between different representation formats. For example, XML-based markup languages such as JSML [Hunt, 2000b] and Annotation Graphs may be represented as such in the Information Storage. Furthermore, when the Information Access Protocol and the communication protocol use XML as well, all the system information can be manipulated using XML documents. This is the case when the reference implementation of the Information Storage is used together with the XML-RPC communication protocol.

The content or the structure of information inside the Information Storage is not defined by the system architecture, since this is an application and domain specific issue. It is up to application developers to define how they use shared information and how the content of the Information Storage should be structured.
CHAPTER 4: JASPIS ARCHITECTURE

The definition of the shared knowledge is an important phase of the application development process. It is recommended that the contents of the Information Storage is defined by using a suitable formalism, such as Document Type Definition (DTD) or Schema in the case of XML. In this way, application developers can make sure that the information is in the correct form during the design, implementation and evaluation.

4.2.3 Information Manager

The Information Manager is used to provide an application interface to the Information Storage. It uses the Information Access Protocol to access the Information Storage, and provides a programming interface for other components to access the shared system context. Because of this, different programming languages need their own information managers. Data type and data structure conversions are performed by these managers. The functionality of information managers is not defined, except that they should provide the general “process” method. It is up to each manager implementation to decide what other functionality they offer. This section presents some general issues and introduces the overall functionality of the Jaspis reference implementation of the Information Manager.

The most basic implementation of the Information Manager is a wrapper to the Information Storage. This basic component takes an XML document (information storage request), and passes it to the Information Storage. When the Information Storage returns an answer, the Information Manager returns the resulting XML document back to the component. This kind of simple interface may be used in situations when the use of the Information Storage is rather minimal and straightforward. For example, if we use a certain programming language to implement a component which uses the Information Storage only once, there is no point in writing a full-featured manager for that particular language. Instead, we can use XML documents to exchange information.
4.2 - INFORMATION MANAGEMENT

When the Information Storage is used more frequently, it is practical to write a proper interface to access the shared system context. The Jaspis reference implementation of the Information Manager provides over 100 methods to access the Information Storage. These methods offer data conversions and other convenience functions, such as methods to access certain information sources directly without knowing their exact locations. For example, system inputs and outputs may be modified by their own set of methods. Application developers may write new, application specific methods when needed. This is especially useful when the information is used by many components. For example, in many Jaspis applications the Information Manager provides methods to manipulate dialogue data (usually trees), which are used by many components.

Most of the methods offered by the Information Manager are not necessary in a strict sense, because all possible operations can be performed using the “process” method and XML-documents which follow the Information Access Protocol. Still, it is practical to offer a full-featured programming interface for application developers. When a shared system knowledge is used, application developers need to write a lot of code which uses the Information Storage. The development time and errors may be reduced significantly by providing a full-featured implementation of the Information Manager.

In addition to the shared information storage, direct information exchange between certain I/O components is allowed for efficiency reasons. Most notably, the raw audio streams should be passed between components in a cost efficient way to minimize system overhead and delays. Sophisticated audio architectures have been developed [Arons et al., 1989]. Most often, audio-based and other data-intensive information is passed between the system components by using a variant of the broker architecture [Melin, 2001; Seneff et al., 1998]. Currently the Jaspis architecture does not use a broker approach. Instead, the I/O subsystem uses a layered architecture, which is presented in Section 4.5.
4.3 Interaction Management

The interaction management model of Jaspis is focused on adaptivity, modularity and distribution, which are among the key design principles of the architecture. Interaction management in this context means both the overall coordination of systems components, and especially the coordination of those components which implement interaction techniques to be used in human-computer interaction tasks. In practice, this means input, output and dialogue management components in their various forms.

Interaction techniques are implemented by agents, which are software components specialized for certain tasks. Evaluators are used to choose between different agents. Managers are used to coordinate agents and evaluators. Components specialized for related tasks are organized into modules. An overview of the interaction management model is presented in Figure 16.

As illustrated in Figure 16, each system module contains one local manager and several agents and evaluators. It is up to the local managers to decide which agents receive turns in different situations. Instead of centralizing this decision to managers, evaluators are used to evaluate agents and their suitability for different tasks. Thus, there is no central component to make decisions. This makes it possible to construct highly adaptive and modular systems, since all functionality is divided into specialized components which have no predefined execution

Figure 16: General interaction management architecture.
order and relations between them. Next, the principles of managers, agents and evaluators are presented in detail.

4.3.1 Managers: Coordination

Managers form the topmost layer of Jaspis-based applications. Each system contains several modules and there is one local manager inside each module. Each system also contains two kinds of special managers: the Interaction Manager and the Information Manager, as illustrated in Figure 13 (page 120). The Interaction Manager is used for overall coordination of the system, while the Information Manager is used to access the Information Storage, as stated before. Local managers are independent of the other local managers, and their relation to the Interaction Manager is not strong. The Interaction Manager coordinates local managers, but it does not know the internal structure of modules, nor does it coordinate interaction inside modules. The main motivations for this are that layered systems are easier to maintain and they may be more efficient, since local managers need to coordinate only their own components. Evaluations especially are more efficient when evaluators evaluate only those agents which belong to the same module. In general, coordination is noted to be one of the future aspects in agent-based systems [Martin et al., 1999; Blaylock et al., 2002].

The architecture includes a general manager implementation. The General Manager implements the algorithm presented in Algorithm 1. Managers ask evaluators to give a score for each agent in the module. If there is an agent which receives a score above zero, the manager takes the turn (returns true) and activates the agent with the highest score. In the algorithm presented managers multiply evaluation scores, but more elaborate methods may be used as well, as discussed in Section 4.7.
In addition to the General Manager's behavior, local managers are able to extend its functionality, or in general implement their own functionality. For example, some of the reference implementation managers (e.g. the Presentation Manager and the Communication Manager) have extended that basic functionality. This is because they have different needs for the use of agents. The Communication Manager especially uses evaluators and agents in a very different way than the Dialogue Manager, for example. To summarize, the agent – evaluator – manager paradigm is used in all system modules, but in different ways.

The Interaction Manager is a central component in Jaspirs-based systems. It coordinates other components and is responsible for the overall coordination of the interaction. In some ways, the Interaction Manager is similar to certain central components found in other speech architectures. Such components include the HUB in the Galaxy-II architecture [Seneff et al., 1998; Seneff et al., 1999] and the Facilitator in the Open Agent Architecture [Martin et al., 1999]. Similar components can also be found in other distributed architectures. The differences include that there is a division of labour between the Information Manager and the Interaction Manager, and that Jaspirs is a layered (hierarchical) architecture.

When a Jaspirs-based system is initialized, the Interaction Manager starts local managers, and when they have been successfully initialized, the application starts. The Interaction Manager initializes local managers, but not their agents and evaluators. It is up to the local managers to initialize their own components.

```algorithm
for each agent
    agent.score = 1.0;
    for each evaluator while agent.score > 0
        agent.score = agent.score * evaluator.evaluate( agent );
    if max(agent.score) > 0
        agent.process(); return true:
    else return false;
```

Algorithm 1: Generic management algorithm.

130
The Information Manager and the Information Storage are the first components to be initialized. After they have been initialized, other managers will be initialized. After that, each local manager will receive an option to take the turn. The turn is offered on the basis of a priority list.

Jaspis uses a turn-based interaction management approach, in which one manager is active at a time. The Interaction Manager contains a list of local managers, and when the currently running manager ends its processing, other managers are able to assume the turn. The Interaction Manager offers an option for the next manager on the list, and if that manager does not take the turn, the option is offered to the next manager on the list and so on. There are no evaluators at the top level of the architecture. Instead, it is up to the local managers to decide when they are able to take the turn.

When an option to take the turn is offered to a manager, it will check if it is able to handle the situation either by using its own reasoning algorithm and/or by consulting evaluators, as presented in Algorithm 1. For example, if the Information Storage contains output requests, the Presentation Manager checks if it has suitable agents to present these messages. If messages can be presented (i.e. if suitable agents are found), the Presentation Manager takes the turn and suitable agents will be selected to present the messages.

It is possible to include domain specific functionality in the Interaction Manager to decide when each of the managers should take the turn. This approach is used in the Galaxy-II architecture, where scripts can be used to control the interaction, i.e. to select which system components will handle different messages (“frames”). Although this would be possible in Jaspis as well, it is not recommended. In order to coordinate interaction in a flexible way, it is better to use de-centralized control logic. Managers and their components may be more autonomous and the system may adapt to interaction when needed.

When the Interaction Manager selects managers, it uses a list, where each manager has a priority which defines how often and when the Interaction Manager offers an option to take the turn. Managers with a higher priority are
always offered the turn before managers with a lower priority. Managers with the same priority are gone through according to the order of the list. This makes it possible to favor certain managers over others. This feature is mainly targeted at those managers which assist other managers.

Jaspis includes several managers which run at high priority. These include visualization and information logging managers, to be presented in Chapter 6. Normally, managers should run on the same priority. But priority may be used to create a dominant manager, which always takes the turn before other components. For example, the Dialogue Manager may take the turn between all other managers and control other managers in this way. This kind of behavior is used in the MITRE system [Luperfoy et al., 1998]. This is not, however, a recommended way in Jaspis-based applications.

There is no information passing between the Interaction Manager and other system components, since the Information Manager is used to access shared system context. The Interaction Manager only informs other managers when they can take the turn, and it does not contain any decision logic, or offer any facilitation or broker features.

4.3.2 Agents: Interaction Tasks

Agents in the Jaspis architecture are software components which handle various tasks. Interaction agents (e.g. dialogue, input and presentation agents) handle human-computer interaction related tasks by implementing interaction techniques. This general definition leaves a lot of freedom to use agents in different ways in the Jaspis architecture. Even so, there are several assumptions as to how agents in the Jaspis architecture should be designed and used. Next, I present the characteristics of Jaspis agents.

Agents are often understood to handle tasks that are fairly complex. For example, a speech-based system may contain such agents as a dialogue agent, a parser agent and a generation agent. The number of agents in typical setups is
fairly small. For example, in the WITAS multimodal dialogue system there are six agents [Lemon et al., 2001]. Jaspis has been designed with compact agents in mind. Agents are supposed to handle well-defined and single tasks. Because of this, they are fairly small and typical systems contain a lot of them. For example, in the Mailman application there are over twenty dialogue agents and even more output generation agents. There are separate function-oriented agents for reading e-mail messages, navigating between folders and messages and so on (see Section 5.1 for a more comprehensive description).

Typical agents in Jaspis-based systems are specialized for certain tasks, which may be domain specific or general problem solving tasks. Agents may implement interaction techniques for tasks such as presentation of welcome prompts or handling of speech inputs. Some of the agents may be application independent, while some may be closely tied to the application domain. Application independent behavior is usually preferred so that components may be used across applications. Usually, application specific parts can be encapsulated into their own agents.

By using specialized agents it is possible to implement modular, reusable and extensible interaction components that are easy to implement and maintain. For example, we can construct general interaction techniques, such as error correction methods, to take care of error situations in speech applications [Turunen & Hakulinen, 2001a]. Section 6.1 presents how agents and evaluators are used to handle errors in speech applications.

In addition to the co-operational approach, where different agents implement different functionality, agents may be specialized for the same tasks with varying behavior. In other words, agents may provide alternative solutions to the same problems. In this way, we can support different interaction strategies inside an application and adapt the interaction dynamically to the user and the situation. For example, we may have different agents to take care of speech outputs for different languages. If we want to support error handling outputs for different languages, we can construct specialized presentation agents for each language. Language specific features can be incorporated into these agents, and there is no
need to modify the original agents. Similarly, various error handling methods can be included in the system by constructing new agents which handle errors by using alternative approaches.

Jaspis agents keep all the relevant information in the Information Storage. This makes them *stateless*, since they are supposed to retrieve all needed information from the Information Storage when they start their processing, and update the contents of the Information Storage when they end their processing. In this way, every agent may utilize all the information that other agents produce.

Furthermore, there is no need for programming interfaces or messages between agents. This promotes iterative and distributed application development, since developers may freely utilize all the information that the system contains without deciding beforehand what information should be exchanged. New components may be introduced in a plug-and-play manner, and all the information that the system contains will be readily available for them.

If properly constructed, stateless agents can be selected dynamically and they are able to adapt their behavior to new situations. This means that agents should always update their internal state from the shared system context when they make decisions. For example, a dialogue agent may have assumptions of what happens next in the dialogue. When it carries out a set of actions and updates the shared system context, it cannot be sure which agent will take the next turn. Agents should take into account the fact that other agents may behave differently than they expect, and actions may be performed by different agents than they expect.

Jaspis agents are *autonomous* in the sense that they should know when they are able to act. Each agent has a self-evaluation method which checks the current situation (for example, the dialogue context) and returns a positive answer if the agent is able to run in the current situation. To be precise, each agent gives a local estimate which defines how well it can handle the situation.

The self-evaluation method is only one part of the agent selection process. All dependencies between agents, including the overall suitability of competing
agents for different situations, are modeled using evaluators. Evaluators evaluate the overall suitability of agents by using the shared system context. In principle, the final selection of agents is left to managers, but it is recommended that managers do not contain any reasoning beyond general housekeeping and coordination tasks, because if the selection is centralized it is likely that systems become less dynamic and adaptive.

A way to characterize and evaluate agents is the use of attributes. Each agent contains a set of attributes which describes its capabilities and behavioral features. Attributes are name–value pairs which define a single value for each attribute. Attribute values can be either strings or integers. Values are configurable, and they can be between 0 and 10, for example. A simple example follows: We may define that an agent is perfectly suitable for e-mail reading and uses a system-initiative approach by giving a value of 10 for both of these attributes. On the basis of the agent attributes and the current system context, evaluators may determine how well an agent is suitable for handling the given task.

Initial values of attributes are expressed in a static way in the system configuration file, but attribute sets are dynamic objects, which can be manipulated during the application execution. New attributes can be constructed, values modified and even complete attribute sets can be modified. This allows both agents and evaluators to dynamically manipulate attributes and adapt them to the interaction. For example, new values for attributes may be retrieved from the Information Storage, or learned from the interaction.

4.3.3 Evaluators: System Level Adaptation

Evaluators are the key concept to make applications adaptive. They are used to determine which agents should be selected for different interaction tasks. Although the final selection is left to the managers, and agents have also their self-evaluation method, it is up to evaluators to compare different agents and
their capabilities in order to give an evaluation score for every agent in any given situation.

Like agents, evaluators are specialized for certain tasks. In practice, this means that different evaluators evaluate different aspects of agents from different system viewpoints. For example, an evaluator may use the dialogue history to determine which dialogue strategy should be used (i.e. which kind of dialogue agent should be selected), while another evaluator may use the user model to emphasize verbose or brief system outputs.

When Jaspis evaluators are employed at the human-computer interaction level, they can monitor the interaction, and give guidance as to how the interaction is progressing and how it should continue. One example is dialogue strategy: if the interaction is not going on smoothly when the dialogue strategy is user-initiative, dialogue evaluators may give better scores for system-initiative dialogue agents. Similarly, if the user has problems, presentation evaluators may prefer presentation agents which use more detailed and helpful prompts.

When one of the agents is going to be selected, each evaluator gives a score for every agent, as presented in Algorithm 1 (page 130). These scores are then multiplied and the final score, a suitability factor, is given for every agent. Evaluators give scores for agents using a scale between [0,1]. Zero means that the agent is not suitable for the situation. One means that the agent is perfectly suitable for the situation from the perspective of one evaluator. Values between zero and one indicate the level of suitability. Since scores are multiplied, an agent which receives zero from any evaluator is useless for that situation.

It is noteworthy that there is no single evaluator, nor any single component in general, which selects agents for each situation, but instead the selection is always both dynamic and distributed. This makes it possible to keep the program control and interaction flow dynamic and adaptive at the architectural level.

Evaluators may use different strategies and techniques to evaluate agents. The most straightforward evaluators use rules to give binary scores for agents. For example, for telephone-based applications Jaspis contains an evaluator which
4.3 - INTERACTION MANAGEMENT

checks if a telephone connection is established, and if not, the evaluator marks all those agents as unusable which do not deal with the connection setup. This is an example of the evaluation of functional aspects. More advanced evaluators may use methods from machine-learning, for example, to evaluate agents and their usefulness in context. An example of such an evaluator can be found in the Interact system, which is presented in the next chapter.

Evaluators may use the content of the Information Storage when they evaluate agents. This means in practice that they may use information such as current dialogue context, user model and interaction history. Presentation evaluators, for example, may use dialogue history to determine which confirmation agent will be most suitable for the current dialogue state. In conventional dialogue systems, such decisions are taken by the dialogue manager. Jaspis evaluators allow more flexible interaction management to take place, since the reasoning is both distributed and dynamic.

Like agents, evaluators have been designed to be modular, extensible and reusable software components. When different aspects of the evaluation process are divided among different evaluators, application developers may choose the most suitable evaluators and modify or extend their functionality when needed. General evaluators can be used between applications and application specific evaluators can be added when needed. The Jaspis architecture includes several general evaluators. Some of them are used to evaluate the functionality of agents, while some others are used to match agent capabilities and user preferences.

One of the general evaluators is the Attribute Evaluator, which evaluates agents based on their attributes. The Attribute Evaluator has its own set of attributes, which are compared against agent attributes. When the Attribute Evaluator calculates scores for agents, it averages the distance between different attribute values. The non-equivalence between two strings is interpreted as the maximum value (this default interpretation can be changed). If an attribute set contains attributes which are not present in the other attribute set, those attributes are ignored (alternatively, they can be treated as non-equivalent attributes). For example, we may have attributes “gender” and “verbosity”. The attribute
evaluator has values “male” and “5”. Agent A has values “male” and “9”, while agent B has values “female” and “4”. The difference between the attribute “gender” is 0 for the agent A and 10 for the agent B. Similar values for the attribute “verbosity” are 4 and 1. The total difference between attribute sets for agent A is 2 and 5.5 for agent B.

The general attribute evaluator contains a condition rule which defines when attributes are evaluated. This is useful when we do not want to evaluate attributes in all situations. For example, it is not meaningful to evaluate the suitability of different e-mail reading agents if the system has presented confirmation to the user, which asks whether or not the user wants to end the call.

Like evaluators in general, there can be any number of attribute evaluators in the system. Every agent may have only one set of attributes, but attributes can be compared against different attribute sets. For example, the User Preference Evaluator evaluates user preferences only. Other examples of general purpose evaluators are the Language Evaluator and the Modality Evaluator. All of these are instances of the Attribute Evaluator.

### 4.4 Dialogue, Presentation And Input Modules

The Jaspis architecture contains three general interaction modules. These are input management, dialogue management and presentation management modules. These modules contain the components used to implement interaction techniques for human-computer interaction tasks. All of these modules are based on the general agent – evaluator – manager principle, as illustrated in Figure 17. Although all interaction modules use agents and evaluators, they use them in slightly different ways. For example, in the dialogue management module only one agent is selected in each turn, while in the presentation and input management modules several agents are able to act during a single turn. The
communication module, which is presented in the next section, differs more from these modules, and also contains additional components.

The reference implementation of these modules is meant to be a starting point for application development, not a fixed model. In some projects it is necessary to extend these modules, and introduce new ones. For example, in the AthosMail system (Section 5.1) there are several input, output and dialogue management modules. More examples of practical applications are presented in the next chapter. Next, dialogue, presentation and input management modules are presented in more detail.

4.4.1 Dialogue Management

The task of the dialogue management is to update the dialogue state, or in other words, move the dialogue from one state to another. This abstract task representation may be modeled in practice in various ways, and should not be restricted to any single dialogue control model, such as the finite-state model.

In most spoken dialogue systems the dialogue management is done by a single monolithic component. In the Jaspis architecture dialogue agents implement all functionality related to the dialogue tasks. They are basic communication units between the user and the system on the dialogue level. At every dialogue turn, the Dialogue Manager selects one dialogue agent and lets that agent handle the situation. Thus, a dialogue flow is a sequence of dialogue agents. Unlike many
other agents, such as presentation agents and input agents, there is only one
dialogue agent active at any given turn.

Like other agents, dialogue agents are specialized for different dialogue tasks or
provide alternative solutions to the same dialogue tasks. Usually, there are no
general guidelines as to what functionality dialogue agents should implement,
and how tasks should be divided between agents. The dialogue control model is
one way to categorize dialogue agents. For example, in the state-based dialogue
control model each dialogue state can correspond to one dialogue agent, while in
the concept-based model each concept may form its own agent, and in the form-
based dialogue model every slot may be a dialogue agent.
Dialogue evaluators may be specialized for aspects such as general dialogue
level issues, functional aspects of agents, domain specific issues or user
preferences. For example, one dialogue evaluator monitors the dialogue flow and
checks that the interface is consistent for the user, e.g. dialogue management
strategy is not changed for every dialogue turn. Another evaluator may perform
the mapping of dialogue tasks to the functionality of agents. Next I introduce
how dialogue can be modeled in Jaspis-based systems. The dialogue
management principles are presented originally in [Turunen & Hakulinen,
2001b].

4.4.1.1 Multiple dialogue control models
The modular approach to dialogue management makes it possible to use different
dialogue control models, such as state-machines and forms inside the same
applications. The combination of different control models is useful when sub-
dialogues are implemented in different ways. For example, most database
retrieval tasks (e.g. timetable queries) can be modeled efficiently by using forms,
while more open-ended dialogues, such as e-mail reading, may be implemented
more efficiently using state-machines. Complex applications which include
different sub-dialogues may therefore need many different dialogue control
models. In Jaspis dialogue agents may be used to implement different control
models, and evaluators may select appropriate ones for each situation without the need to include these dependencies in dialogue agents. In this way, dialogue agents written using different control models can be combined to form complex applications.

4.4.1.2 Multiple dialogue strategies

Dialogue agents can be used to implement different dialogue management strategies to adapt the dialogue to the user. As stated previously, different dialogue management strategies, such as the system-initiative approach and the mixed-initiative approach, have different benefits and drawbacks, which makes it useful to use both of them in an adaptive way. For example, the interaction could be user-initiative by default, but if the user has problems, the system may use the system-initiative approach to handle the situation and switch back to the user-initiative approach when the interaction is progressing well again. If different dialogue strategies are implemented by different dialogue agents, it is up to the dialogue evaluators to choose which dialogue strategy should be the most suitable in the current situation. The system may use different dialogue strategies in a flexible way, while the overall complexity of the system does not cause complexity to individual dialogue agents. A single agent can be compact and optimized for one dialogue strategy.

4.4.1.3 Alternative dialogue management solutions

Alternative solutions to the same interaction tasks can be implemented by using modular dialogue agents. For example, in an e-mail application we may have a standard set of dialogue agents which handles standard-sized mailboxes. We may have special dialogue agents which are activated when the user has only a few messages, and yet another set of agents for a large number of messages. In these situations, all agents are able to handle all kind of mailboxes, but they are optimized for different mailbox sizes, and they may implement completely different user interfaces. Special situations, such as error handling are an another example where alternative dialogue agents can be used to provide adaptivity to
CHAPTER 4: JASPIS ARCHITECTURE

the interface. Different error handling techniques, such as selections from a list and confirmations may be modeled using specialized dialogue agents.

4.4.2 Presentation Management

The presentation management module is responsible for generating system outputs suitable for the current dialogue state. Usually this is done on the basis of conceptual messages which the dialogue management module produces. Presentation agents produce a representation of speech outputs, which are synthesized by the communication management module. They perform natural language generation, add prosodic information to sentences, decide which (synthesized) voices should be used and add other modality specific features into outputs. Real world metaphors for presentation agents are actors, which perform their roles as efficiently as they can. Different agents have different capabilities, and they are chosen for different roles. Unlike in the real world, the choice of agents can be tailored for each listener, i.e. for each user. A real-world metaphor for presentation evaluators is headhunters, which try to find the most suitable actors to perform the roles in a play. Similarly, presentation evaluators try to find which presentation agents are the most suitable for the presentation of speech outputs. Unlike in the dialogue management, several presentation agents may be active during a single turn. First, there may be multiple output requests in the Information Storage, and all of these will be processed. Secondly, presentation agents may produce several outputs from a single output request. For example, multiple agents may contribute to the output, and a message may be rendered by using multiple modalities. Furthermore, the output generation process can be modular, as presented in Section 6.2.
4.4 - DIALOGUE, PRESENTATION AND INPUT MODULES

4.4.2.1 Role of the presentation management

Natural language generation is a rather old research area and sophisticated systems have been produced for text-based applications. However, relatively little work has been done concerning speech specific natural language generation and multimodal speech outputs. Text-based systems are typically collections of pipelined components, which take any legal system concept and generate a natural language message from it. Modifications may be hard to make to such a system, especially if the original developers are different from those who make modifications. Speech interface specific issues, such as prosody, directed prompts and tapered outputs can be especially hard to add to these components. In speech applications it is important to customize the interface for the given domain and application.

The division of labor between the dialogue management and the presentation management is not strictly defined, and it depends on the techniques used, and on the approach taken by the developers. For example, the presentation management may receive well defined concepts (e.g. the type of the concept is specified in detail, and all parameters are defined), open concepts (overall concept type without parameters), or just a request to present an output to the user. In the last case it is up to the presentation agents to decide what should be presented to the user in the new dialogue state.

An example follows. When the speech recognizer has produced low confidence scores, dialogue agents may produce a request for explicit confirmation. Alternatively, dialogue agents may inform the presentation management that a confirmation should be presented to the user, and let the presentation management decide if it should use implicit or explicit confirmation. In the third case dialogue agents do not give any explicit instructions to the presentation management, but instead the presentation management finds out that there is a possible recognition error and a suitable confirmation should be presented to the user.
4.4.2.2 Multimodal outputs

Besides speech outputs, the presentation management module handles other modalities in the system. Speech-applications are becoming more and more multimodal, and even in telephone applications non-speech sounds (earcons and auditory-icons) are used in conjunction with speech. Multiple modalities may be used in many ways: different modalities may separate outputs which do not have a direct relation between them, the same message may be presented using multiple modalities to offer alternative outputs, and speech outputs and non-speech audio may be used to provide different kinds of information inside the same output.

The presentation management module performs fission, i.e. it decides which modalities should be used for outputs. In some systems the dialogue management selects modalities, but it is beneficial to leave this decision to the presentation management. It is easier to utilize different modalities in the system and take user preferences into account, since dialogue management components do not need to be modified when available modalities are changed, and user preferences are learned or given by the user.

4.4.2.3 Prosody

Prosody is an essential element in speech-based communication between humans, and when properly used it makes synthesized speech outputs more pleasant and intelligible. Different prosodic elements, such as speaking rate and pitch variations have different effects. For example, the use of pauses seems to affect intelligibility more, while variations in pitch can bring pleasantness to the speech [Hakulinen et al., 1999]. Examples of the use of prosody include presentations of lists, such as e-mail messages or bus timetables. These are further discussed in the next chapter.

4.4.2.4 Consistent use of language

Presentation agents should know the language model of the speech system (e.g. the one that the speech recognizer uses). As noted in Section 2.3, speech outputs
have a strong influence on the user's choice of words, and so also on the interaction as a whole. Similarly, presentation agents should know what kind of language the user prefers (e.g. what word the user has used before) and use similar language in speech outputs. For example, if the system says “you have three new messages”, it is more likely that the user will use the word “messages” rather than the word “mails”. Likewise, if the user says “read the fifth mail” it would be inconsistent for the system to answer “message number five was sent by…”. Overall, presentation agents should produce consistent speech interface and use knowledge from the dialogue management, user inputs and the user model in this process.

4.4.2.5 Multilinguality

Multiple languages can be hard to handle in speech applications. A simple approach where a single component produces outputs for all languages is usually not suitable, since multilingual output generators are not common, especially for minor languages, and the developers of a new language version are often different from the developers of the original version. The modular structure of the presentation management module helps to separate language specific issues into their own agents, and the developers of a new language version are able to translate the system in an iterative way and keep the process manageable. We used this approach in the case of the Mailman application, which is presented in the next chapter.

4.4.2.6 General presentation agents

Although the presentation management is a highly domain and application dependent process, the Jaspis architecture contains some general presentation agents which may be customized for specific applications. The Canned Phrase Agent takes its parameters from the configuration file. It supports the use of variables, pointers to the Information Storage locations, and basic arithmetic and string operations. It can be used to produce fairly complex outputs, and it is widely used in many Jaspis-based applications.
CHAPTER 4: JASPIS ARCHITECTURE

Another example of general presentation agents is the List Presentation Agent. It takes a list and a set of variables (the latter one is not mandatory). Then it turns these into a set of output messages containing the list elements presented in suitable chunks and equipped with meta-information and prosodic elements. All domain specific information is presented in the configuration file along with language specific information. There may be several agents for different lists and languages, each using the List Presentation Agent.

4.4.3 Input Management

Input agents and evaluators work similarly to presentation agents and evaluators. They take conceptual input requests from the Information Storage, select appropriate modalities, and add the necessary control information, such as device configuration parameters. The resulting control messages are then processed by the communication module. In this way, conceptual input requests, modality selection and device control are separated from each other. Control information is modality specific, and in speech applications the construction and selection of recognition grammars (language models) are the main task of input agents and evaluators.

4.4.3.1 Selection of modalities

Modality selection is often a fairly straightforward task in conventional speech applications. For example, in telephone applications there are only two input modalities: speech inputs and DTMF keys. Most often both of these are selected, so that the user may enter multimodal inputs in exclusive fashion, i.e. input modalities are sequential and independent (see Table 2, page 87).

When speech is used in other areas, for example in ubiquitous applications or advanced mobile phones, more input modalities may be used. Even in telephone applications it would be possible to use DTMF keys and speech inputs in the alternate manner, i.e. to use them in combined and sequential fashion. For example, in applications where users may perform numerous operations, and a
lot of numbers need to be entered, it makes sense to use speech for operations and the keypad for entering numbers. It should be noted that input management components do not perform multimodal fusion, since this is the task of the communication module, but instead input components select which modalities are available for the user, and therefore they should be aware of the fusion mechanisms available.

The selection of input modalities is a highly application specific process. Although there are theories of how modalities should be selected (see [Bernsen, 1997] for a speech oriented approach), the selection of appropriate modalities is an iterative task in which different modalities are experimented with and the most suitable selected when their usefulness is determined (by usability studies, for example). It is also common practice to work with text or graphical modalities in the development phases of speech applications. By selecting input modalities in specialized agents, dialogue level components do not need modifications. It should be noted, however, that modality change is not just a configuration parameter, but instead may involve complex relations between input, output and dialogue modules.

4.4.3.2 Selection of modality specific parameters

Input management components take care of the selection and creation of the input vocabulary. In speech applications this means mostly recognition grammars, or language models in general. In the most straightforward cases there is only one fixed grammar, which is used for all speech inputs. This is suitable for situations where the dialogue management strategy is user-initiated and the language model is not complex (i.e. its perplexity is low). Context-sensitive grammars may be used in system-initiative dialogues, meta-dialogues and in other situations where users are expected to speak in various ways. Predefined grammars may be selected among existing ones or new grammars may be created dynamically to suit the situation at hand. The possibilities are naturally limited by the capabilities of speech recognizers.
CHAPTER 4: JASPIS ARCHITECTURE

Input agents represent different recognition vocabularies, and input evaluators choose which of these should be used in given situations. As with other evaluators, input evaluators model relations between agents and the system state, and input agents are able to focus on their own tasks. This helps to write and maintain reusable input agents, although it is not straightforward to construct efficient and domain independent recognition grammars, and in fact, the most efficient recognition grammars should be customized to the application domain.

Recognition grammars should be consistent with the way that the system speaks to the user. If a single fixed recognition grammar is used, this is mainly a task for interface designers, but in cases where grammars are dynamically constructed and/or selected, they should be consistent with the output messages that the presentation management produces. Even in simple cases, such as when synonyms and yes/no dialogues are used, a wrong lexical selection may lead to problems if output and input languages are not consistent [Hockey et al., 1997]. Since the order in which Jaspis managers are executed is not fixed, application developers may decide how they model topics, speech styles and lexical selection in different modules.

4.5 Communication (I/O) Management

Communication, i.e. low-level input and output management in Jaspis differs in many ways from other parts of the architecture, although it shares the agent – evaluator – manager paradigm. In addition to these, more components are involved. In addition to I/O agents and evaluators, the Communication Manager includes a layered architecture for handling communication between devices and other external resources. This architecture consists of devices, clients, connections, servers, handlers and engines. Devices and engines are I/O specific components, while clients, connections, servers and handlers can be used in other parts of the system for distribution of components. Figure 18 illustrates I/O components.
4.5 - COMMUNICATION (I/O) MANAGEMENT

Figure 19 illustrates how I/O components are related to the other parts of the system. The left side of Figure 19 illustrates how dialogue, input and presentation agents and evaluators generate I/O messages. Dialogue agents produce output and input requests, which are translated by presentation management and input management components into control messages.

Figure 18: I/O components.

Figure 19: I/O components and their relation to the architecture.
CHAPTER 4: JASPIS ARCHITECTURE

The right side of Figure 19 illustrates the overall principle of communication management. When the evaluators receive control messages, they process them, and include possible device specific control information. Control messages are then processed by the devices. They return result messages, which are interpreted by the agents. Next, the evaluators determine what to do with the results. They may provide feedback (i.e. control messages) to devices, and suggest the communication manager to start new devices or stop running ones. The evaluators decide when it is time to stop input gathering. Interpretation of speech inputs and multimodal fusion is performed by agents and evaluators. Next, these components are presented in detail.

4.5.1 Communication Manager

The Communication Manager coordinates input and output devices, as well as I/O agents and evaluators. In many spoken dialogue systems inputs are processed in a separate natural language processing module and there is no time-dependent interaction between I/O devices and other components. This kind of approach is based on the batch-sequence model and assumes that the interaction between the human and the computer is based solely on turns. In order to create more naturalistic interfaces we need more flexible models for handling interaction [Allen et al., 2001b]. The Communication Manager uses I/O agents to interpret and conceptualize inputs, and input evaluators to coordinate devices. This process is carried through in a timely manner, and feedback may be provided to the devices while they are still gathering inputs or presenting outputs.

Like other managers in the Jaspis architecture, the Communication Manager does not include any reasoning beyond infrastructure housekeeping tasks. It starts and stops devices, receives their results, stores the results in the Information Storage and starts I/O agents and evaluators, but all other functionality and reasoning is done by other components.
4.5.2 I/O Agents And Evaluators

I/O agents receive results from the input and output devices. The output devices usually return status reports, e.g. that they have completed their tasks. Input devices, on the other hand, may produce fairly complex results which need to be interpreted and conceptualized. I/O agents perform tasks such as natural language understanding and multimodal fusion. For example, typical I/O agents take recognition results, perform syntactic and semantic analyses and return conceptualized results. Other examples include checking recognition confidence scores and combining different recognition results (e.g. multimodal fusion and data fusion).

Several I/O agents may be active at the same turn, and they may work in a cooperative way. Agents are invoked in sequential order and each of them may work on the basis of the original results, or continue on the basis of the results of previous agents. For example, a natural language understanding process may consist of multiple agents, which correspond to the different stages of NLU. Input results are presented in a two-dimensional array, which contains all inputs and all their interpretations.

I/O agents may be specialized for modality specific processing. For example, some agents may handle speech inputs, while other agents process results from the telephone keypad and so on. I/O agents also perform multimodal fusion. Some of the I/O agents may be specialized for fusion, which is done on the basis of the I/O results. As a general architecture, Jaspis does not define which kind of multimodal fusion techniques should be used, but instead provides an explicit model for the coordinated interpretation of multimodal inputs.

I/O evaluators receive results from input agents, and decide what to do with the results. It is up to the I/O evaluators to decide when current input results are sufficient, and when devices should be stopped or restarted. As a part of the process, I/O evaluators prioritize the received inputs. Rather than making decisions based on single results, I/O evaluators operate on the basis of multiple results. In this way, they also interpret inputs, but rather than interpreting the
meaning of single inputs, they interpret the results together and according to the overall system context (e.g. dialogue history). In this way, I/O evaluators take part in multimodal fusion on a global level.

When needed, I/O evaluators provide feedback to devices. For example, multiple recognizers can be started and stopped based on the results of those recognizers, which return before others. For example, in the Interact bus timetable application there are two recognizers with different vocabularies. The first one is customized for legal sentences which the system can perform, while the other model consists of domain related out-of-vocabulary words. If the first recognizer returns with a good result (i.e. confidence scores are high), the second recognizer is terminated. This is presented in more detail in Section 5.2.2.

4.5.3 Devices And Engines

Devices and engines form the logical and physical ends of the Jaspis communication architecture. Virtual devices represent concrete interaction components (i.e. speech recognizers, microphones, telephone cards etc.) on an abstract level. Like many other Jaspis components, they are highly specialized, and fairly simple and more compact than fully featured and complex. The Jaspis communication model assumes that inputs and outputs are represented as different logical devices, even if the actual interaction devices, such as a telephone card, usually handle both inputs and outputs together, and may return inputs when only outputs are requested. For example, there are multiple virtual devices to represent the different functions of a physical telephone device: one device records speech inputs, another receives DTMF keys and so on. By keeping virtual devices compact it is possible to model inputs and outputs in a flexible way in other parts of the system.

Compact logical devices can be combined to form more complex devices. This feature supports reusability. Jaspis contains devices which represent audio input and output sources, and devices which perform operations on audio sources. For
example, microphone and telephone audio inputs may be connected to the speech recognition device to provide a basic speech recognizer. Similarly, these devices, along with the loudspeaker output device and the telephony cards output device, may be connected to the voice activity detector (VAD) device to form a VAD activated speech recognition device, which may also be used for outputs (such as feedback sounds).

Engines are components which perform low-level operations, such as starting and stopping of physical devices. While virtual devices are specialized and compact, engines tend to be fairly large and complex. There is usually one engine for each concrete input or output component, such as a mouse, keyboard or speech recognizer. Since other components are distributed (logical distribution of virtual devices and physical distribution of clients and servers), engines should handle all the functionality related to their distributed and concurrent use. They should, for example, make sure that devices are not used in a way which makes their state inconsistent for multiple clients. A basic sharing mechanism is supported by the general Jaspis engine implementation, while specific engines should provide additional features when needed.

All engines work in a similar way. They receive a control document, which represents operations, their parameters and optional configuration data. The document may include multiple operations (called actions). When actions are performed, the engine returns a result document, which may contain multiple results. The use of control documents makes the middle-layer (i.e. clients and servers) thin, and new connection methods may be easily added without modification to engines or devices. Since all documents are represented using XML, formal document definitions (DTD) may be used to define the used markup.

Engine interface definitions (i.e. document definitions for engine actions) of similar engines are represented in general classes. For example, there is a generic speech recognizer definition which represents elementary speech recognition operations. Engine specific features are presented in individual engine implementations. Engine specific converters are used to translate between
different representation formats. Jaspis includes several converters for both input and output engines. For example, there is a converter which translates between JSML and SAPI speech synthesis markup languages. In a similar way, the architecture includes a converter which transforms various speech recognition results into Annotation Graph format. Like in other parts of the system, all common formats are XML-based.

4.5.4 Clients, Connections, Servers And Handlers

Clients and servers make it possible to distribute Jaspis components. Although they are designed to work primarily between virtual devices and engines, clients and servers may be used to distribute other Jaspis components as well. In this way, different Jaspis components, such as managers and agents, may be distributed to multiple computers without modifications. Only XML-based configuration files specifying relations between components need to be changed to define the communication methods used.

Clients use connections to route requests made by virtual devices or other Jaspis components to the appropriate server, and so to the engine or in general to another Jaspis component. The actual delivery of the request (XML document and/or binary data) is up to the connection which implements the functionality needed for communication. Connections are very general, i.e. they implement the communication method (e.g. XML-RPC), but no other functionality. Clients provide methods to transfer information between devices and connections. When a result from the server is received, the client transforms the connection specific information back into the device specific format.

Servers are counterparts to clients, i.e. they wait for requests from clients, route the requests to the engines, and return the engine results back to the clients. Servers are multithreaded and they may serve multiple clients. This is an important feature for engines, which may serve multiple virtual devices, as well as multiple applications. It is also an important feature for other than I/O
components, such as for Information Storage, which may get requests from multiple components.

Servers do not process requests themselves, instead they use handlers for that purpose. There is a corresponding handler for each connection type. Handlers perform similar tasks to clients, but in the reverse order, i.e. they transform information back and forth to the connection specific format. When handlers receive messages from servers, they transform the information into the general Jaspis format, and call related Jaspis components to process messages. When components return results, handlers transform results back into the connection specific format, and return them to the servers.

All components (devices, clients, connections, servers, handlers and engines) are dynamically interconnected together, and alternative implementations may be provided for communication layers (clients, connections, servers and handlers) without modifications to devices and engines. The separation between I/O specific communication layers (clients and handlers) and general communication layers (connections and servers) keeps all layers compact and usable for multiple purposes. For example, connections and servers are also used to provide distribution services for other than I/O components. Currently Jaspis includes implementations for plain socket and XML-RPC based communication. Other formats, such as SOAP are fairly straightforward to implement if needed.

### 4.6 System Framework

In the previous sections of this chapter I have presented the general principles of the Jaspis architecture. In this section I present the main issues regarding the reference implementation. The Jaspis architecture is a fully functional and freely available application framework, and it has been used in many practical applications, as described in Chapter 6.

The Jaspis architecture is implemented using the Java programming language and XML technologies. The Java language was chosen because the virtual
CHAPTER 4: JASPIS ARCHITECTURE

machine approach makes portability across operating systems easier than with many other languages. Java is also suitable for agent-based applications, and it has strong support for XML, which is used in all parts of the Jaspis architecture. Together, Java and XML provide a strong foundation to support other emerging standards. From the viewpoint of software engineering, the Java language has other advantages as well, including a wide range of suitable software libraries for various purposes. All parts of the core Jaspis classes are implemented using Java. Some parts of the Jaspis extensions are written using other languages, but in total over 99% of the source code is written by using Java.

Jaspis runs on J2SE environments with versions 1.4. and upward. The architecture was tested on Windows 2000/XP and Linux systems. Naturally, such technical details are subject to change, and the latest release contains the most accurate information on technical requirements.

4.6.1 General System Structure

Technology-wise, the Jaspis framework is based on dynamic objects and XML documents. The architecture consists of a set of core infrastructure classes and various extensions. The core architecture contains general manager, agent and evaluator components, the I/O subsystem, generic XML components and the Interaction Manager, Information Manager and Information Storage components. There are also reference implementations of presentation, input and dialogue components, all of which extend the general classes. Currently there are more than twenty extensions which add functionality to the core system. Figure 20 illustrates the core infrastructure and some of the extensions.

The modular architecture allows the distributed development and use of Jaspis-based applications. In most cases, dependencies between extensions are small. Mostly they implement interfaces, defined by the core package or other extensions. For example, the XML-RPC extension implements the communication interface, and in this way provides an alternative to the core
4.6 - SYSTEM FRAMEWORK

communication classes. When the XML-RPC extension is used, only configuration changes are necessary. Similarly, various speech recognition extensions implement the general speech recognition interface.

The core system classes and their relations are presented in Figure 21. There are two parent classes, `JaspisComponent` and `XML`. The JaspisComponent class is parent to all Jaspis components, excluding most of the XML classes. It provides general functionality, such as reference to the Information Manager, and several convenience functions for the creation of new class instances, logging of information and so on. It also defines the common “process” method.

The Jaspis XML class offers general purpose functionality for the handling of XML documents. This core class provides essential methods, while its children provide more specific functionality. A similar use of object-oriented techniques for representing information is reported in [O'Neill et al., 2003]. The use of XML is presented in more detail in the next section.

When the system is initialized, the construction of the system is done recursively. The Interaction Manager reads the XML-based configuration file, creates the Information Storage and puts the contents of the configuration file into storage. After that, it reads the configuration of other managers from the storage. Managers are created by the generic manager class, which receives a
description (configuration) of a manager and creates a runtime instance of the manager. Similarly, the manager reads descriptions for its agents and evaluators, and generic agent and evaluator classes create the runtime instances of the components described.

4.6.2 XML In Jaspis

XML was chosen for information representation because of its many advantages: it is structured, widely accepted and fairly mature technology with numerous tools, and it integrates perfectly with the Java language. Furthermore, most of the markup languages, linguistic resources and other data sources are released in the XML format. The advantages of XML in natural language applications has been noted widely, see for example [Zadrozny et al., 2000; Herzog et al., 2003].

The use of XML is based on JAXP (Java API XML Processing) [Mordani & Boag, 2002]. JAXP is a generic API, which contains both DOM (Document Object Model) [DOM, 2003] and SAX (Simple API for XML) [SAX, 2003]
document models. Jaspis uses a JAXP compatible XML parser, such as Apache Xerces, for all elementary XML operations. By using JAXP the Jaspis architecture is not tied to any specific XML parser, which makes it possible to choose the parser freely, as long as it is JAXP compatible.

In addition to the elementary JAXP components Jaspis contains specialized XML classes which offer higher level functionality for certain purposes. The basic XML class provides functionality for operations such as the construction and manipulation of XML documents. Various more specific XML classes extend this functionality by providing higher level operations. For example, Jaspis contains classes for the representation of inputs and outputs. In these classes special methods are provided for handling the task specific functionality. The reference implementation of the Information Storage is also a child of the XML class.

XML is used in all system configuration and log files as well. In fact, a Jaspis configuration file is an XML document, which is the initial state of the Information Storage. The configuration file is read from a URI (e.g. file or URL address), and the Information State is initialized from the content of this file. In this way, the system state can be stored in the file, and used later for analysis or to start up the system in a specific state. This is a frequently used method in interpreted languages, and is used extensively in many Jaspis tools, as described in Chapter 6.

4.6.3 Information Access Protocol

The Information Access Protocol defines all Information Storage operations, and in this way defines the interface between the Information Storage and the Information Manager. It contains the following operations: query, add, replace, count and delete. They respectively retrieve, add, replace, count and remove information elements inside the Information Storage. Next, these operations are defined in detail along with examples.
Each operation contains at least one parameter, which is the location in the Information Storage. When the reference implementation of the Information Storage is used, this is a path (route) to a node in the XML tree. Similarly, this may be an SQL statement when a relational database is used, or a reference to an object when an object-based information representation is used. In addition, “add” and “replace” operations take a second parameter. This parameter defines what information should be added or replaced, i.e. the new content of the location.

All operations and their parameters are expressed using XML, i.e. all Information Storage operations are XML documents. The root node of the object defines the operation to be performed. The first parameter and the optional second parameter are children of the root node. Nodes correspond to tags in the XML document. The first parameter contains a set of “route” nodes, which represent paths to the information source. Each node may contain an optional index, which is used to resolve situations when multiple nodes exist with the same name, i.e. when a node contains multiple children with the same name. The contents of the second parameter may be any legal XML content.

The following examples demonstrate the use of information management operations. The examples contain two XML documents. The first document represents different operations which the Information Storage receives, and the second document represents the results returned by the Information Storage. The content of the Information Storage before operations is presented in Example 1.
Example 1: Original content of the Information Storage.

First the name of the user is retrieved. The “query” operation is used for this purpose. The route to the name of the user is defined by listing the nodes which lead to the desired information. It is noteworthy that the root node is omitted. The resulting query document is presented in Example 2.

Example 2: “Query” example.

When the Information Storage receives the document presented in Example 2, it returns the reply document, which is presented in Example 3.
CHAPTER 4: JASPIS ARCHITECTURE

Example 3: “Query” example result document.

In the second case the number of e-mail addresses that the user has is retrieved. This is done by using the “count” operation. This is presented in Example 4.

```xml
<?xml version="1.0"?>
<result>Markku</result>
```

Example 4: “Count” example.

When the document of Example 4 is received, the Information Storage returns the answer presented in Example 5.

```xml
<?xml version="1.0"?>
<count>
  <route>
    <tag>internal</tag>
    <tag>user</tag>
    <tag>address</tag>
    <tag>e-mail</tag>
  </route>
</count>
```

Example 5: “Count” example return document.

In the third case the second e-mail address of the user is retrieved. Example 6 shows the document which retrieves the desired information. The index attribute is used here to indicate the exact location of the desired address.
Example 6: “Query” example with “index” attribute.
As expected, the Information Storage returns the result document, which is shown in Example 7.

Example 7: Return document for Example 6 query.

In the fourth case the contents of the Information Storage are modified using the “replace” operation. It changes the name of the user by replacing the old content. This is presented in Example 8, while the result document is shown in Example 9.

Example 8: “Replace” example.
Example 9: “Replace” example return document.

In the fifth case new information is added to the Information Storage by using the “add” operation. Example 10 represents this operation. The result document is the same as in Example 9. Unwanted nodes can be removed using the “delete” operation. This is presented in Example 11. Again, the result document is the same as in Example 9.

Example 10: “Add” example.

```xml
<?xml version ="1.0"?>
<add>
    <route>
        <tag>external</tag>
        <tag>system</tag>
        <tag>technical</tag>
    </route>
    <content>
        <host>Jaspis.cs.uta.fi</host>
        <port>8200</port>
    </content>
</add>
```
Example 11: “Delete” example.

The content of the Information Storage after all of these operations is presented in Example 12.

Example 12: Final content of the Information Storage.

As seen in Example 10, the content of the second parameter can be any legal XML node, e.g. it may contain child nodes. When the add operation is used, the path to the destination may be non-existent. When necessary, new nodes will be created by the Information Storage.

The difference between the “add” and the “replace” operations is that the “add” operation always creates a new node. Otherwise, they perform similarly. This
CHAPTER 4: JASPIS ARCHITECTURE

means that the “replace” operation can be used to create new nodes, but it cannot
be used to add new nodes when nodes with the same name exist. For example, if
we want to add a third e-mail address to the Information Storage (as presented in
the example 1), we should use the “add” operation. However, we could have
used the “replace” operation instead of the “add” operation in Example 10, since
there are no nodes with the same name.

The reference implementation of the Information Manager provides a more
convenient way to express routes needed in information management operations.
It provides a different format to express paths, which are transformed into proper
XML documents as defined by the “route” node of the Information Access
Protocol. The following string illustrates this syntax by representing a path
which defines the second address in Example 1:

“internal->user->address->e-mail[2]”.

4.6.4 Standards And Markup Languages

The use of standards and general solutions are essential in modern software
development. The development of spoken dialogue applications is often a data
intensive process and corpus collection is needed as a part of the application
development process. Suitable standards may facilitate the corpus collection
process enormously, and make it easier for developers to use the corpus for
application development. Standard markup languages also aid everyday
development tasks, and support reusability.

The markup language support in Jaspis is based on XML-based markup
languages and converters, which perform transformations between these
languages and other markup languages. In this way, the benefits of sophisticated
languages may be utilized, and a generic interface may be provided for
application developers. At the same time, practical markup languages may be
utilized to control speech technology components.
One example of this approach is synthesizer markup languages. The Speech Group in Sun Microsystems Laboratories has created JSML (Java Speech Markup Language), an XML-based markup language for speech synthesis [Hunt, 2000b]. JSML is well featured and suitable for general purpose synthesis control. Unfortunately, there are not many synthesizers which support JSML, and significantly there are no Finnish synthesizers which support it, but instead all currently available Finnish synthesizers support the SAPI standard [Microsoft, 2003]. JSML can be used, however, since the Jaspis framework contains a synthesizer converter, which transforms JSML markup into SAPI compatible markup as closely as possible.

Another example is Annotation Graphs [Bird & Liberman, 2001]. Annotation Graphs is a well defined, formal framework for the representation of linguistic annotations. Linguistic data is represented in Annotation Graphs by using acyclic directed graphs, which can be expressed using XML. Although they are not designed to support speech recognition results and other dynamic interaction data, we have adapted them successfully for this purpose. The support for Annotation Graphs is based on a set of converters which perform transformations back and forth to AG format. This is presented in more detail in Section 6.5.

Currently Jaspis does not support any specific speech recognition markup language. Although there are several proposals, such as VoiceXML (for general dialogue control) and JSGF (for recognition grammars), they are not suitable for general purpose speech recognizer control as such because of the limited support from speech technology providers. The Jaspis architecture includes a custom XML DTD to control speech recognizers, but in the future this may be replaced by a suitable standard markup language.
4.7 Summary And Comparison With Other Architectures

The features of the Jaspis architecture presented in this chapter provide solutions to challenges arising from technology, human-computer interaction and application development viewpoints, as stated in the architecture design principles in Section 4.1. In this section I summarize how the Jaspis architecture addresses these challenges. I also compare it with other architectures and introduce further challenges for speech systems. Many of these are addressed in the Jaspis' architecture, which is presented in Chapter 7.

The Jaspis architecture is designed to be a generic, extensible and practical infrastructure for application development. Both a conceptual architecture model and a concrete infrastructure are provided for designing and implementing practical speech applications. In this sense it is similar to architectures such as Galaxy-II [Seneff et al., 1998; Seneff et al., 1999] and Open Agent Architecture [Martin et al., 1999]. The core infrastructure includes only the system components which are essential for communication between components, information sharing and adaptation. In this way, it does not force application developers to adopt any predefined solutions, such as certain dialogue control models or multimodal fusion techniques, as in some other systems (e.g. CSLU Toolkit [Sutton et al., 1998]). Instead, it provides an extension mechanism which allows technology, domain and application specific components to be added to the core system. The core system components provide interfaces and reference implementations for various tasks, and they may be replaced when necessary.

Unlike most general architectures Jaspis also provides support for speech technology components in its various extensions. In this sense it offers similar functionality as the ATLAS architecture [Melin, 2001]. Jaspis uses a layered I/O subsystem, offering separation of logical and physical devices, multiple changeable communication protocols and a time-dependent model for input interpretation, including multimodal fusion. The I/O subsystem allows the
development of highly distributed heterogeneous systems which may be used in the speech-based pervasive computing applications, for example.

The main design principle of the Jaspis architecture is explicit system level adaptivity, and in general support for adaptive components and interaction techniques. Jaspis is designed to support alternative components and collaborative development, unlike many other systems, which usually assume that applications are developed from several black-box components that have well-defined task descriptions and interfaces. In reality, interaction level issues are scattered around the system, and multiple components and modules provide solutions to the same tasks. The agent – evaluator – manager model is available for all system modules, although it is designed especially for interaction modules. Nevertheless, all system modules and components may benefit from it, even if adaptive features are not needed, since the model offers a strong basis for distribution, modularization, reusability and extensibility.

When compared to other adaptive speech systems, such as adaptive dialogue management by MIMIC [Chu-Carroll, 2000], the Jaspis approach is more general, and it may be applied to all system components. In most other systems, the architecture itself does not support adaptivity in any particular way. Jaspis also provides extensible and distributed adaption, i.e. the adaption mechanism itself is distributed between multiple components, and may therefore be extended without modifications to existing components. This is a major step away from the black-box type adaptivity which most systems offer.

The basic evaluation method, as presented in Section 4.3, is suitable for most purposes, but when necessary, it can be extended in several ways. First, scaling factors may be used to give preference to certain evaluators over others. For example, it is usually meaningful to give preference to functional properties over user preferences (but not in all cases). Secondly, a custom function may be applied instead of multiplying the evaluator scores. This is a more general version of the scaling factor. Finally, evaluators may form trees, in which child evaluators are executed before their parents, and in this way their results are available for parents, and more control may be applied for the evaluation
process. This may be useful when multimodal inputs are evaluated, for example, and in a way it is similar to the melting pots approach [Nigay & Coutaz, 1995], but provides a more open and flexible way to combine results. Yet another option is to use evaluators at the top level of the architecture.

Evaluators are a key aspect in adaptation. Most evaluators evaluate agents based on the current system context and agent properties. It is also possible that they evaluate the results of agents directly. For example, if the user prefers brief messages rather than long and informative ones, an evaluator may ask all suitable agents to produce output messages, from which it selects the shortest. This kind of black-box comparison could be useful in situations where several similar agents exist. In this way agents can be compared without explicit specifications of their functionality, and evaluation parameters may be learned by using machine-learning techniques.

Flexible interaction management is related to adaptivity. Most of the current speech systems are still pipeline systems, or more precisely, batch-sequence systems, even if these architectural models are not particularly suitable for interactive systems. More flexible interaction models have been presented, and most of these are based on client-server and agent approaches, such as Galaxy-II and OAA. In these systems agents and/or servers solve problems in a collaborative manner, and the interaction management is based on messages between agents. In practice, the interaction is usually controlled by a central component, such as a dialogue manager or hub. Direct messages create explicit relations between components, which limit the systems' ability to behave adaptively. Examples of this approach are MITRE [Luperfoy et al., 1998] and WITAS [Lemon et al., 2001] system. WITAS has expanded recently to support asynchronous dialogue management [Lemon & Cavedon, 2003]. The TRIPS system [Allen et al., 2001a] offers a more flexible interaction model. Still, these systems do not support an explicit model for coordination and organization of components which is needed, for example, in system level adaptation. This issue is addressed in the TRIPS system [Blaylock et al., 2002].
Interaction management in the Jaspis architecture addresses the question of coordination in the form of the Interaction Manager, which provides a flexible and coordinated interaction model, while the Information Storage makes it possible to utilize shared system context and use indirect messaging between components. These components allow the interaction to be flexible, but still coordinated.

Jaspis-based systems may contain any number of managers. Their type is not limited in any way, e.g. there may be several managers with the same functionality, and in this way they may work collaboratively. This may be useful when multiple simultaneous dialogues are taking place. However, the system does not provide any particular support for multiple similar managers. A major limitation of the first version of the Jaspis architecture is that only the I/O subsystem allows concurrent interaction to take place. This limitation is removed in the Jaspis² architecture, and in general, concurrent dialogues are better supported.

Modularity in speech systems is usually limited to the separation of several monolithic modules, as presented in many examples in the previous chapter. The dialogue management module especially is usually a single component, which covers almost all the functionality of the system. The TRIPS architecture provides a more modular approach, and in fact the system does not contain such a component as a dialogue manager, but instead several agents which perform dialogue related tasks. Nevertheless, the agents in the TRIPS architecture are fairly complex and monolithic components. The Jaspis architecture goes further by using compact agents which are targeted at handling single tasks. The system architecture encourages application developers to keep the interaction components as compact as possible. Jaspis is also a layered architecture, which means in practice that it contains several layers, with their own organizational structure. These features provide a highly modular, and still organized and coordinated system architecture.

Agents may help monolithic, multi-function components to be more transparent, modular and adaptive. For example, when new functionality or alternative
CHAPTER 4: JASPIS ARCHITECTURE

solutions for an existing dialogue manager are added, this can be done by adding new agents and evaluators without the need to modify the existing code. This is especially useful when components from different sources are used, and original components cannot be modified. In the early phases of the development process basic functionality can be included in applications by using standard components. System functionality may be later extended by including more advanced and specialized components. Application developers may also experiment with different approaches to the same problems without any need to know details of the other agents in the system or change their behavior. It should be noted that speech application development is an iterative process and the interface should be tested with users as soon as possible. A modular agent-based approach helps developers to experiment with different strategies even if the system is not yet fully functional.

From the human-computer interaction viewpoint, Jaspis agents offer a way to structure interaction level tasks. They provide a layer between the system functionality and interaction tasks. Even if the functionality of several agents may be performed by a single piece of program code (e.g. a conventional dialogue manager or a response generator), it is recommended that all interaction level tasks should be separated into their own agents. For example, if we have an existing dialogue manager, which is implemented as a single monolithic component, we may have several dialogue agents which all use this single component, but which are seen as different agents by the rest of the system.

General domain-independent agents support reusability. Since most of the speech systems use monolithic components, which handle multiple tasks and do not separate domain specific and general tasks, it is extremely hard to reuse them in other domains and applications. General Jaspis agents can be reused between applications when their selection is left to evaluators. In addition to reusability, this supports internationalization and localization. These are important issues in speech applications, in which differences between languages are greater than in graphical applications.
Completely domain independent agents may be hard to design and implement, and in practice domain specific modifications are usually needed. For example, error handling components may benefit from the domain specific dialogue knowledge. However, it will be easier to modify and extend modular agents than large and complex components.

In a way, Jaspis agents are similar to reusable dialogue components, such as XML-based SpeechObjects [Burnett, 2000]. Jaspis agents, however, are true software components, not static descriptions. More similar approaches are taken in the Agenda/RavenClaw [Rudnicky & Xu, 1999; Bohus & Rudnicky, 2003], the Queen's Communicator [O'Neill et al., 2003] and the SesaME [Pakucs, 2003] dialogue management models. These models also contain methods for dynamic and non-trivial selection of agents. Compared with these approaches, the Jaspis approach is not specific for any dialogue management model, but instead provides an explicit infrastructure for implementing such models: an interesting future task would be the implementation of the dialogue management models mentioned using the Jaspis architecture. By providing the general agent – evaluator – manager mechanism Jaspis also differs from generic infrastructures, such as Galaxy-II, OAA and the MULTIPLATFORM Testbed [Herzog et al., 2003], which do not offer such infrastructure for internal coordination of components inside modules.

In addition to agents, evaluators and managers, shared information is utilized in many ways in the Jaspis architecture. In order to adapt to different situations, components utilize information such as dialogue history and user profiles. Modularity is easier to achieve when a common system context is provided for all components. The shared Information Storage also aids collaborative and iterative development by encouraging application developers to specify what information each component stores in the shared storage. This allows rapid prototyping of components, because interfaces between components can be minimized, and components may be changed dynamically. In order to facilitate the application development process, Jaspis offers interactive tools for the
visualization and manipulation of the content of the Information Storage. These tools are presented in Section 6.3.

The benefits of shared information are apparent in other systems as well. For example, MALIN [Dahlbäck & Jönsson, 1999], WITAS and TRIPS all use shared resources, but either the knowledge sources are connected to the dialogue manager, or they do not use information resources in a coordinated way, such as in Jaspis. Unlike some agent-based architectures, such as the OAA, the Jaspis architecture does not use any specific agent communication language in order to be generic. A similar approach is used in the MULTIPLATFORM Testbed [Herzog et al., 2003].

Jaspis-based systems contain usually one Information Storage. This approach has several limitations: for example, the use of a single blackboard may introduce communication bottlenecks in large systems. The Jaspis architecture does not explicitly support multiple Information Storages, but when needed, additional storages may be used. The first version of the architecture does not provide any particular support for this. The support is provided in the Jaspis² architecture.

The role of the Information Storage in the Jaspis architecture is rather passive, i.e. it only processes synchronous operations requested by other system components. In the Jaspis² architecture the Information Storage is a more active component.
5 JASPIS-BASED APPLICATIONS

In this chapter I present several applications implemented using the Jaspis architecture. These applications cover many areas, including information services, multilingual systems and pervasive computing applications.

These applications demonstrate how the Jaspis architecture is used for the construction of practical speech systems. Both functional and technical aspects are covered. Likewise, I describe how the challenges encountered during the development have guided both the development of the original architecture as well as the design of the Jaspis2 architecture introduced in Chapter 7.

Jaspis-based applications cover many human-computer interaction aspects presented in Chapter 2. These include multilinguality, adaptive interaction techniques, dialogue initiative and the use of non-speech audio, among others. Findings from the evaluation of applications are presented, along with their relation to the application design and the system architecture.

Many of the application issues addressed, both technical and human-computer interaction related, are introduced in general form in the next chapter, which describes various models and tools to support application development and evaluation.

5.1 E-mail Applications: Mailman And AthosMail

Our research group has been involved in the development of several speech-based e-mail applications. Historically, the motivation for the Jaspis architecture comes from this domain. When we started the development of our first e-mail
application, Mailman (Postimies in Finnish), there were no tools for the construction of such multilingual applications. We also realized the need for adaptivity at the architecture level. After the initial prototype of the Mailman application, we started the architecture development, focusing on adaptivity in general and multilinguality in particular.

The e-mail domain is an especially suitable area from both research and practical perspectives. First, it provides many challenges for technical and human-computer interaction oriented research. This was already realized in the 1980’s [Schmandt, 1982]. The e-mail domain is also a practical research area. E-mail itself is one the most successful applications in the history of computing, and entails many issues which are relevant for other information services as well. Speech-enabled services such as Mailman may provide universal access for various users in different settings. Thus, the results are relevant to a wide range of applications and users.

The first version of the Mailman application is presented in our research article [Turunen & Hakulinen, 2000b]. More recently we made a completely new version. The second version is presented as a part of a master’s thesis by Esa-Pekka Salonen [2002]. The development of the third version is in progress in the EU-funded DUMAS project. In this project, the existing Mailman application serves as a base for a more advanced AthosMail application [Turunen et al., 2002]. In all cases, the Jaspis architecture is used to provide the system framework.

Next, I describe the functionality of the Mailman application, along with additional functionality provided by the AthosMail application and other development issues. Then human-computer interaction aspects are considered, and issues related to multilinguality in this domain are discussed. Finally, findings from user studies are presented.
5.1 - E-MAIL APPLICATIONS: MAILMAN AND ATHOSMAIL

5.1.1 Mailman Functionality

The main purpose of the Mailman application is to allow the user to access his/her mailbox using a standard mobile or desktop phone. Mailman provides the most common e-mail client functions except the possibility to create new messages. Users can browse among their messages, and read and delete messages. In order to provide an efficient interface for telephone use, messages are automatically organized into manageable groups, and divided into sections in which the user can navigate when the message is being spoken. The system provides both speech input (speech recognition) and DTMF interfaces which may be used in multimodal fashion. Next the functionality is presented in detail, and demonstrated in Dialogue 1.

S: “Postimies toivottaa sinut tervetulleeksi, anna käyttäjätunnuksesi.”

“Welcome to the Mailman system. Please give your four digit user ID.”

U: “XXXX” [DTMF]

S: “Anna salainen tunnuksesi.”

“Please give your secret password.”

U: “XXXX” [DTMF]

S: “Terve Demokäyttäjä, otan yhteyden postipalvelimeesi. Ei uusia viestejä, 12 vanhaa viestää.”

“Hello demouser, I'll contact your mailserver. No new messages, 12 old messages.”

U: “Listaa kansiot”

“List folders”
S: “Neljä kansiota. Kansio yksi viestit otsikolla 'reply Tutkimustoimintanne',
kansio kaksi viestit otsikolla 'reply Tervehdys', kansio kolme otsikolla 'reply
Projektikuvaus', kansio neljä otsikolla 'reply Opetustoimintanne'.”

“Four folders. Folder one message subject is 'reply Your research area',
folder two message subject is “reply Hello”, folder three message subject is
“reply Project description”, folder four subject is “reply Your teaching
area.”

U: “Kolmas kansio”

“Third folder”

S: “Kansion kolme viestit otsikolla 'reply Projektikuvaus'.”

“Folder three, message subject is 'reply Project description'. ”

U: “Kerro lisää”

“Tell me more”

S: “Kansion kolme, kolme viestiä otsikolla 'reply Projektikuvaus'.”

“Folder three contains three messages, subject is 'reply Project
description'. ”

U: “Toinen kansio”

“Second folder”

S: “Kansion kaksi viestit otsikolla 'reply Tervehdys'.”

“Folder two, message subject is 'reply Hello'. ”

U: “Listaa viestit”

“List messages”
5.1 - E-MAIL APPLICATIONS: MAILMAN AND ATHOSMAIL


U: “Kolmas viesti”

“Third message”

S: “Viesti kolme lähetäjältä Heli Rikala, otsikolla ‘reply Tervehdys’.”

“Message three from sender Heli Rikala, subject is ‘reply Hello’.”

U: “#1” [DTMF] (the user selects the first message)

S: “Viesti yksi lähetäjältä Päivi Majartanta, otsikolla ‘reply Tervehdys’.”

“Message one from sender Päivi Majaranta, subject is ‘reply Hello’.”

U: “Lue toinen”

“Read second”

S: “Viestin otsikolla ‘reply Tervehdys’ on lähetänyt Toni Pakkanen.”

“Message subject is ‘reply Hello’, sender is Toni Pakkanen.”

S: “Lainaus edellisestä viestistä.“

“Quote from a previous message.”

S: “Arvoisa Kollega. Huomasin Tietojenkäsittelyopinlaitoksen sivuilta yhteystietosi ja olisin halusen kuulemaan lisää, sillä intressimme näyttävät olevan hyvin samanlaiset, eli kerrotko lyhyesti itsestäsi ja työstäsi.”

S: “Lainaus päättyy.”

“Quote ends.”
CHAPTER 5: JASPIS-BASED APPLICATIONS

S: “Olen 28-vuotias tietojenkäsittelyn opiskelija ja toimin tutkijana täällä. Tällä hetkellä tutkin....”

U: “6” [DTMF] (the user skips the paragraph)

S: “Lainaus edellisestä viestistä.”

“Quote from a previous message.”

S: “Lisäksi haluaisin jakaa ulkomaalaisen ystäväni kanssa tietosi, joten voisitko vielä muutamalla virkkeellä esitellä itsesi englanniksi.”

S “Lainaus päättyy.“

“Quote ends.”

S: “Hi, I’m 28 year old student of computer science University of Tampere and currently I’m working as researcher in Human-Computer Interaction group in University of Tampere, you can contact me...”

U: “0” [DTMF] (the user ends the call)

S: “Heippa soitellaan.”

“Bye, please call again.”

Dialogue 1: Mailman example.

5.1.1.1 Login and retrieval of messages

When the user makes a call the login process begins. The user is asked for a four digit user ID and password. This is done using telephone keys to simplify the recognition process and to make the login process robust and secure. We have planned to use additional identification methods, such as speaker identification, combined speech recognition and speaker verification, and identification of calling-number. All of these approaches have their own disadvantages, however, and DTMF input will always be used at least as a backup method.
5.1 - E-MAIL APPLICATIONS: MAILMAN AND ATHOSMAIL

When the user is identified, the system connects to the user’s mailbox and retrieves messages. Messages can be retrieved from IMAP and POP3 servers. This makes the system client independent, i.e., it is not tied to any particular e-mail client such as Elm, Pine, Eudora or Outlook.

When messages are being retrieved, some of them are filtered out depending on user preferences. Filtering is done on the basis of the logical rules defined by the user. Rules may contain references to all header and content information which the messages contain. We found filtering to be a crucial feature, since typical mailboxes contain a lot of messages which users do not want to hear when a speech interface is used. One example is mailing lists, which are not particularly important in mobile use. Visually impaired users, however, may prefer to access all their messages, since for them the speech interface is their primary access method. We have developed a www-interface for the definition of filtering rules. Like all our www-pages, it has been tested to be compatible with the screen reader applications used by visually-impaired people.

5.1.1.2 Handling of e-mail messages

When the messages are retrieved from the server they are transformed into a structural XML format. We have defined an email-markup language for this purpose [Turunen et al., 2002]. A set of content processing agents is used in a pipelined fashion to process messages. Mailman contains agents for identifying segments (e.g. paragraphs, quotations, structural elements such as addresses etc.) and the language used in each segment. It is important that at this point in the processing the original content is preserved, and the identified parts marked with conceptual markup codes. Later on, when the content is spoken to the user, the presentation agents will decide how these elements are to be synthesized. This makes the agents both application and language independent.

After the e-mail messages have been processed, they are grouped into folders of at most $n$ messages per group. The number of messages is a configuration variable. In both versions of the Mailman system we used at most nine messages,
which makes it possible to use the DTMF interface efficiently for message selection.

The Mailman system contains various agents which implement different grouping algorithms to make coherent and balanced views of the mailbox. Grouping agents mainly use header information, e.g. message senders, recipients (to detect mailing lists, for example), message subjects and chronological information (e.g. date and time of the message). In the AthosMail system machine learning techniques will be used to form groups on the basis of both the content of the messages and user profiles [Sahlgren, 2003].

Two possible approaches will be used to select between groupings. In the first approach evaluators select the grouping which is going to be used after all groupings are formed by different agents. This is the approach which Mailman uses currently. In the AthosMail approach all groupings are preserved, and evaluators choose suitable groupings to be presented to the users. In this way, different groupings can be presented to the user. The benefit is that grouping principles can be changed dynamically, for example from subject based to sender based. On the other hand, this may confuse users, and the feature should be used with caution.

5.1.1.3 Navigation in folders, message lists and messages

When the messages have been retrieved, filtered, structured and grouped into folders, the system presents a summary of folders to the user. From this point forward the dialogue is user initiative. The user may navigate between the folders and inside the folders (e.g. between the messages), ask for more information about them, read and delete messages, ask context-sensitive information or leave the system. The interface is not hierarchical or modal, i.e. the user may initialize almost all functionality in any given situation. Some situations, most notably listening to messages, contains functionality which is not available in other situations.
When the user navigates between folders, or asks for information about them, a list of folders and their messages is presented. In this way, most of the system outputs are in the form of lists and the interaction consists from navigation in these lists. When the user selects a message, it will be spoken in chunks corresponding to identified paragraphs of the original message. The user may navigate between these segments, or initiate other functionality when the system reads the message. All this functionality is modeled using separate dialogue and presentation agents.

### 5.1.1.4 Advanced features

The functionality of the Mailman system, as presented here, describes the two first versions of the application. The first version contains all this functionality, and the second versions includes added functionality, such as context-sensitive help, universal commands, and other refinements. In the AthosMail application extended functionality will be added to cover more flexible ways to choose messages e.g. based on their content and other characteristics, as well as providing dynamic ways to group messages and present summaries of their content. All this functionality is presented in separate agents, and no modifications to the other parts of the system are needed. For example, using Jaspis agents and evaluators it was possible to construct a complete interactive online tutoring feature without modifying Mailman at all [Hakulinen et al., 2003].

### 5.1.2 Application Architecture

In this section I present system overviews of Mailman and AthosMail applications. The overall organization of components, along with the most relevant components is presented. Some components, such as those communicating with e-mail servers, are only of technical interest and are not discussed here. Other components are more interesting from the perspectives of human-computer interaction and speech applications, and will be presented in
more detail next. The focus here is on the Mailman application, since the AthosMail application is still under development.

### 5.1.2.1 Mailman architecture

Figure 22 illustrates the overall structure of the Mailman application. The system contains many standard Jaspis components, which are tailored to the e-mail domain by specifying the necessary parameters in the system configuration file. Language specific components are customized for each language (Finnish, English and Swedish) in the same way. This means that the runtime architecture contains more components than the static class architecture. The system configuration file, which is an XML document, plays an important role and is fairly large consisting of multiple files. For example, it contains all the relations between components (such as connections between I/O devices), as well as content information for many prompts and other system outputs. The general presentation agent, which is part of the Jaspis distribution, can handle fairly complicated dynamic speech outputs for multiple languages, and is used extensively in the Mailman application.

The Mail Manager coordinates components related to the handling of e-mail messages. The Mail Manager receives requests from the dialogue components and performs operations on the mailbox. For this purpose it utilizes a collection of components for the handling of e-mail messages. These components offer services for communicating with the IMAP and POP3 servers, transforming text messages into XML documents, filtering out unwanted messages, modifying the contents of messages and providing views (folders) for messages, as previously presented. Most of these generic components are not Mailman specific, and some of them are Jaspis extensions, which may be reused between applications. We also have plans to release all general e-mail components as Jaspis extensions, which means that they will be freely available.
5.1 - E-MAIL APPLICATIONS: MAILMAN AND ATHOSMAIL

All Mailman agents are function-oriented, i.e., they correspond to different functionality in the application domain. This differs from the concept-oriented agents of the timetable applications and from the state-oriented agents of the Doorman application. Agents are also stateless, i.e., they do not store any information inside themselves. If they need to keep records, they use the Information Storage. This has facilitated the reusability of components between applications, and many Mailman components have also been used in other applications.

5.1.2.2 AthosMail architecture

The AthosMail application is based on the Mailman applications, but it has a more complex system structure, as illustrated in Figure 23. Most interestingly, it includes components from the original Mailman application, and in addition various modules for both the same and different purposes. For example, there are various modules and agents for input parsing, dialogue management and presentation of system outputs from other project partners [Cheadle & Gambäck,
2003; Ramsay et al., 2002] which provide alternative solutions to the same problems as do the Mailman components. All of these components operate on the basis of the shared Information Storage and utilize the same dialogue information.

The AthosMail application contains an offline MailServer, which continuously retrieves e-mail messages from mail servers, and processes the retrieved messages. This is necessary because some of the techniques used for message processing are resource intensive, and it is not possible to perform them in real-time [Gambäck et al., 2003; Sahlgren, 2003]. The MailServer also contains components for user modeling. The online AthosMail application and the offline

**Figure 23: AthosMail and MailServer modules.**

The AthosMail application contains an offline MailServer, which continuously retrieves e-mail messages from mail servers, and processes the retrieved messages. This is necessary because some of the techniques used for message processing are resource intensive, and it is not possible to perform them in real-time [Gambäck et al., 2003; Sahlgren, 2003]. The MailServer also contains components for user modeling. The online AthosMail application and the offline
MailServer communicate using the Jaspis communication interface, e.g. by using XML-RPC. The details of the offline system are presented in a DUMAS research report [Turunen et al., 2002].

5.1.2.3 Dialogue management

The dialogue management components of the Mailman application do not use any specific dialogue control model as such, but are instead fairly independent agents, which react to different input and dialogue state combinations. Mailman dialogue components do not contain any language specific code, since they work on the basis of the conceptualized information, and produce only conceptual information. Thus, the dialogue management in the Mailman system is completely language independent, and it is not necessary to modify dialogue components in any way when new languages are introduced, unless a new language specific functionality is added. We have made English and Swedish versions of the Mailman system to be used as a starting point for the AthosMail application. All versions use the same dialogue components, and it would even be possible to change languages dynamically during the interaction, but this does not make sense in this application domain.

Since the dialogue management strategy is based on the user initiative approach, function-oriented agents have been proved to be suitable units for modeling dialogue states. All dialogue agents handle single tasks, and a new agent is selected for each turn. In practice this means, for example, that there are separate agents for different states of e-mail reading. The dialogue history is stored in the Information Storage and utilized when necessary, for example to continue the already started operations.

In the AthosMail application a similar approach is used, and in addition to the Mailman agents the system contains a legacy dialogue manager [Ramsay et al., 2002], which is seen by the system as a set of Jaspis agents. It takes care of some of those situations which the Mailman agents do not handle. All of these components use the same dialogue history, which is represented as a discourse tree in the Information Storage.
CHAPTER 5: JASPIS-BASED APPLICATIONS

5.1.2.4 Adaptivity and extensibility

One particularly interesting feature of the function-oriented agents of the Mailman application is that the dialogue strategy may be dynamically changed, and new features may be added when appropriate. A typical example is that some users prefer that e-mails are read to them automatically. For this purpose we can use a special agent which navigates between folders and messages and initiates reading of those e-mails which have not been read already. This agent is chosen by the evaluators depending on the user model. If the user interacts with the system, the evaluators give a score of zero, so it is activated only when no other interaction is present.

In the presentation components different prompts are used for different users. This is implemented in such a way that each type of prompt (for example, a brief and a verbose version of a welcome prompt) has its own agents, and the appropriate ones are selected during runtime. Presentation agents for multiple languages are implemented in this same way. This approach makes the selection of agents more dynamic, and it helps to build modular components.

In the first two versions of the Mailman system, adaptivity (i.e. the selection of agents) is based mostly on profiles. Typical attribute values are related to speech outputs, for example they describe what kind of prompts a presentation agent is able to produce (e.g. a welcome message), how (e.g. brief) and in which language (e.g. in Finnish). In the AthosMail application machine learning techniques will be used to provide more adaptivity. For example, there will be a machine learning-based dialogue act classifier to select dialogue agents based on the collected corpus [Black et al., 2003].

The most current addition to the system are tutoring agents, which give guidance to the user by introducing the Mailman system and monitoring how well the user interacts with the system. Tutoring components take the initiative in certain situations. They utilize the dialogue history, the user model and their own tutoring plan in this process. Otherwise, the Mailman application is working normally, and no modification has been done. The tutoring feature is included in
Mailman by adding the definitions of the tutoring components in the system configuration file. The feature can be turned dynamically on and off. The implementation of tutoring agents is presented in detail in [Hakulinen et al., 2003].

5.1.3 Speech Interface Issues

In this section I present the most interesting and challenging interface issues from the e-mail domain as we have encountered them in the case of the Mailman application. These include dialogue strategy, universal commands and speech outputs. Further human-computer interaction issues are presented in sections which present user experiences and the handling of multilinguality.

Most of the work done with Mailman concerns speech outputs. In the e-mail domain the main challenge is how to read messages and present other information so that outputs are both intelligible and pleasant for the user. Although these are general issues for speech applications, they are especially crucial in this domain. The user may achieve his/her goals by using a simplified spoken language or telephone keys, but if the speech outputs are not understandable or otherwise poor, interaction will not be successful. Thus, this domain is mainly about speech outputs, including the processing of e-mail messages to a readable form, although there are challenges for speech input processing as well, such as the handling of names [Marx & Schmandt, 1994].

5.1.3.1 Dialogue strategy

The interaction model of the Mailman application is based on the user initiative dialogue strategy with some mixed-initiative features. In such open-ended dialogues tasks are not well structured, the user may interact with the system fairly freely, and his/her goals may differ between sessions according the mailbox content. Therefore the user initiative dialogue strategy is suitable, since the user is the active participant in the dialogue. Furthermore, there is a little room for mixed-initiative strategy, i.e. places where the system may take the
CHAPTER 5: JASPIS-BASED APPLICATIONS

initiative. As in most mixed-initiative systems, the system takes the initiative in
special situations, such as in the login procedure and in error situations.
Although the dialogue strategy is user-initiative, the system should provide
information actively to the user, so that the user can obtain relevant information
without needing to ask everything from the system. For example, Mailman offers
an overview of the folders automatically. Similarly, as much information as
possible is provided to the user when messages are listed, so that the number of
dialogue turns is minimized.

5.1.3.2 Universal commands

As already stated, the application is task-oriented. The first version of the
application contained only the most essential functionality. The second version
of the application introduced additional features, such as context-sensitive help
and universal commands. These features provide meta-functionality which may
be activated in any situation, for example to obtain more information or guidance
on how to proceed from the current situation. It is noteworthy that these
commands, such as “what next” and “tell more” are not only theoretically
motivated, but we have observed that they are used spontaneously. From the
application development viewpoint, this functionality is implemented using a set
of additional agents and evaluators. Tutoring agents are another example of the
added functionality.

5.1.3.3 Static system outputs

There are three kinds of speech outputs in the Mailman system, and in e-mail
systems in general. The first category consists of static system messages. Typical
examples are login messages and generic error messages. These messages can be
hand-tuned and tested to be as fluent as possible. Since these are mostly
something that users learn quickly, they may be annoying if poorly constructed.
On the other hand, they are valuable sources of information for novice users. For
these reasons, it is important to know the users, and to test the system with them.
We have worked together with visually impaired users, and learned that their
preferences are quite different from those of other users, especially from normally sighted novice users. We use brief system messages with the “tell more” universal command, which gives more information in an incremental way. In this way the novice user can obtain more information in all situations, while the actual prompts are short. Static outputs are implemented using the standard Jaspis Canned Phrase Agent.

5.1.3.4 Structured system outputs

Different lists, such as lists of e-mail messages and folders are the second category of system outputs. The key question is how these lists can be spoken efficiently. The first version of the Mailman system used six messages in each folder, and a maximum of six folders was used (if messages could not fit into six folders, then the last folder contains more than six messages). These were supposed to match the upper limit of short-term memory [Miller, 1956]. However, various design guidelines based on this law may lead the (speech) interface design in the wrong direction [LeCompte, 2000; Suhm et al., 2001a]. Briefly, users do not need to memorize all the list items, and in many cases they prefer longer lists, which can be provided by a single speech output instead of multi-layer menus.

Nevertheless, the interface should not cause too much cognitive load for the users. In the Mailman system we wanted all functionality to be accessed by the DTMF interface as well. Thus, in normal situations only elementary numbers (e.g. 1-9) are used to select messages and folders. In this way, both the speech and DTMF interfaces are robust and simple to use. If the user has a large number of messages, more complex recognition grammars are selected by input evaluators to access the messages. These grammars contain all possible numbers up to thousands. The DTMF interface, on the other hand, becomes clumsy, since users need to press multiple keys to access messages beyond 9.

Mailman uses tapering to reduce the length of the lists, and include only the meaningful information into them. As a result of this, the speech outputs sound harsh and “unnatural”, but we have learned from user feedback that this is the
CHAPTER 5: JASPIS-BASED APPLICATIONS

way that experienced users want to hear the system speaking. Separate presentation agents provide more verbal outputs for novice users.

5.1.3.5 E-mail messages as system outputs

The third type of outputs in e-mail systems is e-mail messages. They are the most challenging for many reasons. First, their content is not limited in any way. Even with single users content may vary enormously between messages. Secondly, e-mail messages differ from both written and spoken language. This makes it difficult to use existing language technology tools to handle e-mail messages. Third, e-mail messages are often multilingual in some countries, and multilingual content may appear in various forms, as described later on in this chapter. Fourth, e-mail messages contain non-linguistic elements, such as tables, ascii-graphics and so on. Fifth, messages also have relations to structures, such as quotations, which are not explicitly, properly or consistently marked. All of these elements are increasingly used, because other non-textual formats, such as HTML are becoming more common. Most of these elements are highly visual and difficult to present using speech.

For e-mail messages to be successfully read, those elements which are not suitable for speech should be omitted either by the system or the user. For example, reading a paragraph which contains 50 asterisk marks is not meaningful and may be either omitted or shortened to the form of “50 asterisk characters”. On the other hand, complex elements such as signatures may contain vital information, which is not easy to extract automatically.

In the Mailman application these elements are identified by the content processing agents when messages are retrieved from the server, and presented to the user by various e-mail presentation agents. They read signatures and other similar elements so that the user is able to skip irrelevant parts and focus on those parts which are relevant to him/her.
5.1 - E-MAIL APPLICATIONS: MAILMAN AND ATHOSMAIL

5.1.4 Multilinguality In The E-mail Domain

The e-mail domain is an especially challenging example of a multilingual application. The unlimited content of e-mail messages may involve multiple languages in numerous combinations. Especially in multilingual countries e-mail messages often contain multiple languages. Multiple languages may appear both among messages (i.e. different messages are written in different languages) and within single messages (i.e. single messages contain multiple languages). Structured and other non-linguistic elements also mean additional challenges for the handling of multilingual content. As noted previously, we constructed several multilingual versions of the Mailman system. In this section I consider issues specific to this domain.

5.1.4.1 Multilinguality in Mailman

Originally Mailman was bilingual, using Finnish for inputs and Finnish and English for outputs. Although Swedish is the second official language in Finland, English is more important for most users. The current version of the Mailman system can handle either Finnish, English or Swedish speech inputs and can produce any combination of outputs in these languages. System messages are read to the user in the same language as the inputs he/she gives, and e-mail messages are read in the language in which they are written. When the language identification is not accurate, priority is given to the language of the user.

Mailman also includes the problem of initial logging. When the user calls the system, we do not know his/her language until the login procedure has been completed. Thus, all initial prompts must be given in multiple languages. With three languages this is a fully working solution, but problems are likely when we have more languages, or when we do not know the identity of the user.

5.1.4.2 Speech inputs and dialogue management

The handling of multilinguality in the Mailman application concerns mainly speech outputs. Since the identity of the user is always known, there is no need
to identify his/her language during the interaction. Therefore, the question is mainly how multilinguality is handled in dialogue and presentation modules. For dialogue management this is not a problem, since all language dependent information is processed by the I/O agents, and the dialogue management works on the basis of language independent concepts. Language independent conceptual form is also used in presentation components. The only exception is some output components which use synonyms to tailor the outputs to the language of the user. For example, if the user tends to use the word “mail” rather than the word “message”, the system also uses the word “mail”. Thus, the original content of the user input is always available.

Things may become slightly complicated when the system functionality is extended. For example, if the user is allowed to use references to message subjects and senders, foreign names and subject words bring many additional problems. The problems with the use of names in speech interfaces have been already identified [Marx & Schmandt, 1994]. When multilingual messages are used, there may be even more problematic cases, since parts of the user input may contain different languages. For example, the user may say something like “lue aiheen ‘project meeting’ toinen viesti”. One possible solution would be to use multiple word-spotting recognizers to detect fragments of different languages and combine these. Unfortunately, not all current recognizers, such as Finnish ones, support word-spotting, and in general speech recognition is not robust for such utterances.

5.1.4.3 Basic functionality for all languages

When different technologies with varying capabilities are used, not all functionality can be used for all languages. To share components and maintain portability between languages, a common set of core functionality is needed. More advanced interaction methods could be built on top of that if technology, most importantly speech recognition, allows it. In the Mailman system, all three languages have similar basic functionality even though the English recognizer is superior compared to the other recognizers. As noted before, most of the
language differences are handled in configuration files and only a small number of language specialized components is needed.

5.1.4.4 Types of multilinguality in e-mail messages

The handling of multilingual speech outputs is the most crucial and challenging aspect in the e-mail domain. There may be many forms of multilingual messages. A typical mailbox in most non-English speaking countries consists of messages written in multiple languages. For example, most Finnish users receive messages written either in Finnish or English. Furthermore, it is not uncommon that a single message contains multiple languages. In fact, this is quite a common situation, since people prefer to comment in their mother language, even if the original message is written in English. In these messages, language switches between paragraphs or between chunks of text, such as between quotations.

There are also cases where languages may change between small units, such as sentences or words. Sentences written in different languages often appear when people refer to something written by a foreign author. Foreign names, terms and technology jargon are examples of common foreign words. Fortunately, these are not especially problematic. First of all, it is usually not dramatic if the foreign part of the text is spoken in the user's native language. For example, English text spoken by a Finnish synthesizer is quite intelligible, even to the extent that some users may prefer to hear it this way. On the other hand, if the text written in the first language of the user is identified incorrectly, it is almost impossible to understand the spoken output given in the foreign language. Because of this, our language identification components are biased towards the first language of the user.

5.1.4.5 Use of multiple synthesizers and voices

To solve problems of multilingual content, the optimal solution would be to use truly multilingual synthesizers, such as the quad-lingual synthesis by Traber et al. [1999]. Such synthesizers are not widely available, however, and there is always a risk of incorrect identifications, since the language still needs to be
recognized. We use several monolingual synthesizers, one for each language. In simple cases, when multilingual content is only a small portion of the content, one solution would be to alter the pronunciation of foreign elements so that they would sound reasonably intelligible when read with the primary synthesizer. For most situations, this is not necessary, but would help the outputs to be more pleasant and intelligible in the long run.

The use of multiple synthesizers causes the problem of multiple changing voices. When a speaker – synthesized or human – changes, the user needs time to adapt to the new voice. This is especially the case with speech synthesizers, since in general they demand more attention and cognitive capabilities from users [Lai et al., 2000]. It is also not possible to change synthesizers between small units without totally losing the prosody of the output. This is also a technical challenge, since the change of synthesizers takes some time, and thus unpredictable and unwanted pauses are likely to occur.

For the reasons mentioned, unnecessary and frequent synthesizer changes should be avoided. In the Mailman application we change synthesizers only between paragraphs in order to reduce both incorrectly identified sentences and make the interface more consistent. It is also possible to utilize the properties of different voices. We use male voices for Finnish and female voices for English. In this way the user immediately knows how the system has identified the language, even if the output does not sound intelligible.

The use of both human voices and synthesized voices is an interesting way to utilize the advantages of both approaches. In the study by Gong and Lai [2001] the mixing of synthesized and human speech was favored by the users, but users performed better with only TTS interface. Their mixing of human and synthesized speech was fairly elementary, and we found in the initial experiments on the tutoring feature of the Mailman application that use of recorded speech together with synthesized outputs is a very promising approach [Hakulinen et al., 2003].
5.1 - E-MAIL APPLICATIONS: MAILMAN AND ATHOSMAIL

5.1.4.6 Structural multilingual elements

When multilingual content occurs inside structural elements the situation can be particularly problematic. Structural elements, such as e-mail and www-addresses, tables, and signatures are non-linguistic elements, and may contain other linguistic and non-linguistic elements. For example, there may be individual words or complete sentences inside non-linguistic elements, and vice versa. As presented previously, we have many problems with structured elements even without the problems of multiple languages.

First, most of these elements are hard to identify in the first place. One example is tables, which are far from trivial to identify from pure textual form, although work has been done in this field [Hurst, 1999]. Another question is how these elements should be spoken. For example, in which order rows, columns and individual cells should be spoken, what meta-information should be included to the outputs and so on. All non-linguistic elements, such as special characters and graphics should be transformed into verbal form, since they may contain vital information.

When structural elements are spoken to the user, the presentation agents need to decide which parts of the structure are spoken using which languages. For example, if punctuation marks, table headers and other similar elements are read in the native language of the user, while the linguistic elements are read using a foreign language, we may help the interpretation of non-linguistic elements, but we will face the problems of frequent voice changes. In the Mailman system we try to use the same language for all elements as far as possible, although some elements, such as meta-information (information about the start of a quotation, for example) are read in the first language of the user.

5.1.5 User Experiences

We have arranged several user experiments related to the e-mail domain and and Mailman and AthosMail applications. Most of these experiments have been informal, and their main purpose has been to find the most important technical
and human-computer interaction problems as a part of the iterative development cycle of applications. In addition, user feedback has been received from the public use of the Mailman application, which has been available at a public telephone number since 1999.

5.1.5.1 Presentation of e-mail messages and prosody

Before the development of Mailman was started, we arranged a design experiment to identify the most challenging questions of the domain. In this experiment human-computer interaction experts from our research unit examined e-mail messages. The most challenging e-mails contained sections written in different languages, and different kinds of structured elements, such as tables, e-mail addresses, and web addresses. From this perspective, the research challenge was to construct a multilingual e-mail application which could handle real life e-mail messages.

Since we identified the presentation of e-mail messages to be the main issue in the domain, we arranged an informal experiment to find out the main problems when real-life e-mail messages are read to the user, and how different Finnish synthesizers would perform in this task. In this experiment we ran a listening test in which recordings of three Finnish synthesizers and a human reader were demonstrated to test subjects. The synthesizers used were Mikropuhe, Sanosse and Infovox 230. The quality of these synthesizers is not comparable to the most advanced synthesizers for English.

We did not compare recordings in a statistical sense, but instead elicited subjective ratings of recordings and subjective preferences. We found that the most problematic issues were mostly the same as in the design session, i.e. structured elements such as tables, lists, addresses and foreign names. Test subjects emphasized the need for better use of pauses, and prosody in general. Surprisingly, misspellings were not noticed or complained about at all, which suggests that there is no need to put a lot of effort into trying to correct misspellings and missing accented characters. This finding is not generalizable
for all languages, for example in the case of the Hungarian e-mail application a lot of effort was needed to restore accented characters [Németh et al., 1999].

In order to find the most relevant prosodic elements to make the reading of e-mail messages more intelligible and pleasant we arranged a controlled experiment on the basis of the listening test [Hakulinen et al., 1999]. We had 16 subjects. This time we focused on the use of prosody in structured elements. This study was motivated by the fact that listeners in our previous experiments had mostly problems in structured elements, and they suggested that prosodic elements may help them to understand speech outputs better. Also, the utilization of prosody is not the strongest feature of current synthesizers.

In the prosody experiment our overall research task was to find out how e-mail messages in general and structural elements in particular should be read in order to be both intelligible and pleasant. We wanted to find out how prosodic elements (i.e. pitch, volume, speaking rate and pauses) affect the segmental intelligibility, comprehension and pleasantness of speech outputs. We wanted especially to find the ways which human readers utilize. By finding these we could make our presentation agents more efficient.

The results from the prosody experiment indicate that the use of pauses increases the intelligibility of spoken outputs, while variations in pitch and speaking rate made outputs more pleasant to listen to, but did not have much effect on comprehension. In addition, we found that the synthesis was more intelligible than one of the human speakers and it also performed surprisingly well in other areas. On the basis of a detailed analysis of the speakers it is possible to create highly specialized and unique software components to encapsulate the rich use of prosodic elements of human speakers. Jaspis presentation agents provide an efficient way to implement this. Presentation agents were first realized in the Mailman application.

5.1.5.2 Wizard of Oz experiment

We used the Wizard of Oz method with the Mailman application as a part of the AthosMail data collection task [Black et al., 2002]. This was done in order to
find out how users want to interact with the system with unrestricted and perfect speech recognition. We used the Jaspis WOZ tool to replace the speech recognizer with a wizard controlled GUI. The WOZ tool had also been used in other Wizard of Oz experiments. It is presented in more detail in Section 6.4.

Since the nature of the WOZ method would reveal the content of the e-mails to wizards and other experimenters, we were unable to use real e-mails in the experiment. We learned from the previous experiments that if non-relevant e-mails are presented to users, the situation is highly artificial. This is something that subjects indicated in the listening tests, in which they needed to understand and memorize the message content [Hakulinen et al., 1999].

In order to introduce some relevance into the messages, a scenario approach lasting one week was adopted. Six test subjects were assigned to different roles, and they dictated their messages (responses to the received messages or new messages) during the experiment. No other users were involved in the communication, so we were able to record all messages and their content, as well as all interaction with the system.

It turned out that the situation was still not ecologically valid, and the interaction with the system was rather limited and unnatural, when compared to experiences from real use of the Mailman application. First, the content of mailboxes was completely different from real content, and especially the number of messages and their length was significantly smaller. As the e-mails were mostly dictated (with perfect recognition accuracy, since the recognition was simulated), their content also differed from that of the written messages. As a result, there was no navigation between message groups and most of the problematic elements that real e-mail messages contain were not present in the experiment. Furthermore, even though we offered unlimited speech recognition capabilities, the use of speech recognition was very command oriented. We obtained similar results from experiments with other systems, for example in the experiment with bus timetable systems.
The Wizard of Oz experiment showed, however, that it is possible to use the Jaspis architecture and its configurable WOZ tool to arrange Wizard of Oz experiments. This is described in more detail in the next chapter.

5.1.5.3 Data collection from a working application

We have gathered user feedback from the working versions of the Mailman system. The two first versions of the system have been in a daily use for interested users since 1999. Since the users have used their own mailboxes, we have not been able to collect any message related data for reasons of privacy. This has proved to be one of the main obstacles in user tests in this domain, since the interaction is heavily dependent on the contents of the mailboxes. We encountered this same problem in the AthosMail WOZ experiments.

In order to identify usability problems we have collected feedback from representative users by asking specific questions and free-form comments on usability. These are good ways of identifying specific problems, e.g. if certain e-mail contents have been read correctly and so on. We worked in collaboration with the Finnish Federation of the Visually Impaired and received detailed feedback from them. Visually impaired users, especially those working in the information technology industry, may be considered to be expert users of speech applications, and they are in many ways also a very demanding user group. Other expert opinions were gathered from students on a usability course with no prior experience of speech applications, but with ability to analyze user interfaces.

We collected a lot of useful information from end users, and the results have been used to improve the speech interface. This includes new filters to eliminate unwanted segments of messages, ways to make tapering more efficient, and ways to read certain message elements. Most of these findings apply to other domains than e-mail, for example to the reading of web-pages. Comparison of the results from different sources has shown that different user groups have very different needs and preferences and thus adaptive techniques are needed. We have found it efficient to work with the modular Jaspis agents, which have
CHAPTER 5: JASPIS-BASED APPLICATIONS

allowed us to make changes to specific components without modifications to other system components and to implement alternative components for same tasks to support adaptivity.

One interesting finding is that among average users not familiar with DTMF interfaces (a typical situation in Finland, where IVR applications are not widespread), the keypad layout had to be learned. The DTMF interface is quite important, since both the visually impaired users and students preferred it. We tried different keypad layouts, some of them designed in collaboration with end users, and the feedback states clearly that novice user do not understand layouts intuitively, no matter what layout and metaphor is used. We used several variations of layouts and found that most users found the layouts to be randomly assigned to the application functionality. Although the situation might change when speech applications become more popular, this shows that novice users need support and guidance.

To summarize, tests and experiments with the Mailman system have provided essential information for the iterative application development. Jaspis agents have made it possible to efficiently realize the results in software components. The Jaspis WOZ was found to be useful for conducting Wizard of Oz experiments. However, the WOZ method is not particularly suitable in the e-mail domain. The Wizard of Of method, among other tests in which the test conditions are not ecologically valid, should be considered especially carefully.

5.1.6 Summary

In the e-mail domain multilinguality as well as processing and presentation of e-mail messages are key issues. During the development of multiple versions of the Mailman and AthosMail applications we faced various challenges related to this domain. Using Jaspis agents it was possible to provide a common functionality for all languages, and extend applications by adding language specific agents with varying behavior. Language specific agents are used in all
system modules, including input, dialogue and presentation management components.

Function-oriented agents, and Jaspis agents in general allowed the implementation of several extensions, such as universal commands and tutoring agents. Like in the case of language specific agents, it was possible to add these agents without modifications to existing agents. In this domain, where dialogues are open-ended, function-oriented agents are a natural and efficient solution for dialogue management, and they are also an efficient way to take care of complex outputs of e-mail messages. Many of the Mailman agents are used in other applications as well.

Several user studies were conducted. Although the WOZ tools that the architecture contains were useful for arranging tests, the WOZ method itself was found to be problematic in this domain. From studies with a working system and from listening experiments we found out that usage patterns are very diverse, and a flexible and adaptive user interface is needed. The Mailman application is highly configurable, and both speech input and DTMF interfaces were provided, for example. In the AthosMail application various ways to bring more adaptivity are examined.

In the AthosMail application the distributed nature of the Jaspis architecture with built-in adaptation mechanism allowed us to build a complex system with components from various sources. The AthosMail system contains a legacy dialogue manager and a lot of specialized dialogue agents, which work together. Similarly, it contains several input interpretation and presentation components.

5.2 Bus Timetable Systems

In addition to e-mail systems, our research group has been working in various projects with several bus timetable systems. While e-mail systems are user-initiative and focus on the efficient presentation of multilingual outputs, the focus in timetable systems is more on mixed-initiative dialogues and
CHAPTER 5: JASPIS-BASED APPLICATIONS

interpretation of user inputs. The two systems presented differ in their approach. For example, dialogue control and task management are modeled differently in these applications, and they also differ in their use of the Jaspis architecture.

As with the e-mail domain, the timetable domain is practical, provides plenty of research challenges, and the results can be used in other similar application areas. Timetable services, and information services in general, are very suitable for speech and mobile use. Similar human-operated systems are widely available, and there is a need to replace existing ones with automatic systems. Furthermore, with automated systems new services may be produced, which are either not possible or economically viable with human operators.

In this section I present two different systems, the Busman system, which is our internal project, and the Interact system, which is being implemented in collaboration with various other universities. The Busman system gives information on Pirkanmaa Region bus timetables, while the Interact application is based on Helsinki bus and tram timetables. After the system descriptions, user experiments are reported.

5.2.1 The Busman System

The Busman system provides information the Tampere area bus route and timetable information. The functionality of the system is similar to other timetable services, such as MALIN [Dahlbäck & Jönsson, 1999]. The user may request information such as bus routes (e.g., “which bus goes to the university hospital?”) and timetables (e.g., “when does the next one leave?”). Like the Mailman application, the Busman system has a speech interface and can be used with mobile and desktop telephones, but unlike Mailman, the Busman system is unimodal, i.e. it does not use DTMF keys. Dialogue 2 demonstrates the system.

S: “Tervetuloa toivottaa Bussimies, Tampereen bussiaikataulujärjestelmä!”

“Welcome to the Busman, Tampere bus information service!”
Dialogue 2: Busman example.

5.2.1.1 Application architecture

Figure 24 illustrates the Busman architecture. The system contains standard Jaspis components, and an additional manager, the Database Manager, which is used to communicate with the timetable database. The Information Storage contains the domain and concept model, which is used for all interaction tasks. Most of the code implementing the Busman system is general and released as an extension of the Jaspis architecture. This extension may be used to construct similar information retrieval applications by defining concept and domain models. The actual Busman components are mostly related to the handling of database information and domain specific interaction tasks.
5.2.1.2 Domain and concept models

The Busman application is based on a generic dialogue model, in which applications are constructed by defining concept and domain models. The model is realized by implementing generic agents which use concepts to handle various interaction tasks, such as interpretation of user inputs. Application specific concepts are defined in the system configuration file. All parts of the system, including dialogue management, output generation and input parsing, operate by using these same concepts. The advantage of this approach is that there is no incompatibility between the system modules. For example, the system recognition grammars cover only those sentences that the input interpretation components are able to handle. In other words, it is guaranteed that the application is able to handle all the utterances that the recognizer produces.

5.2.1.3 Dialogue control model

As in many other database oriented applications, the Busman system uses forms for dialogue control. The form is stored in the Information Storage and contains
a set of concepts representing user inputs and other information needed to perform database queries. The heart of the dialogue management is the Concept-based Dialogue Agent, which handles generic situations (such as filling slots of the form on the basis of the user input) by using the concepts that the form contains to request database queries or user inputs.

In addition to the Concept-based Dialogue Agent, the system contains 14 other dialogue agents, which are specialized for certain specific cases in the dialogue. The system has dialogue agents for handling greetings, help requests and instruction messages, out of domain place names, confirmations, requests for repeated information and so on. For some situations there are alternative agents, such as for handling help requests.

Dialogue agents produce output requests (e.g. responses to user queries), input requests (the system needs more data or the initiative is given to the user) and database requests. Output and database requests are presented by using the concepts of the concept and domain models, and processed further by the Database Manager and the Presentation Manager. The Input Manager generates suitable input concepts for the current dialogue situation.

5.2.1.4 Dialogue initiative

Each dialogue agent has an initiative attribute, which indicates its suitability for system-initiative and user-initiative dialogues. In the current implementation one dialogue evaluator is used to give preference to system-initiative agents when the interaction proceeds normally, and user-initiative agents when errors are present. In this way, the system tries to adapt to the interaction by changing the dialogue strategy on the basis of the success of the interaction. As reported in Section 5.2.3, we have found this to be the key aspect to make user-initiative dialogues work in this domain.

It is possible to model dialogue initiative in more detailed ways. For example, dialogue history, such as number and percentage of errors, and time of last error may all contribute to the selection of dialogue initiative, as well as aspects such as user preferences. These all could be implemented as their own dialogue
evaluators to make the initiative selection more distributed and adaptive. The real challenge is to find the correct scaling factors for these evaluators, i.e., to find out which parameters affect user satisfaction. A lot of data from the real usage of the system is needed for this, as is a suitable method of analysis, such as the PARADISE method [Walker et al., 1997].

5.2.1.5 Task management

The Database Manager transforms concept queries into native database queries. It is a passive component in the sense that it waits for requests from dialogue components without any application specific logic. In practice, this is done in such a way that the dialogue agents write requests in the Information Storage, which are retrieved by the Database Manager. After that, database specific queries are executed, and conceptualized results are stored in storage. The Database Manager is similar to the Mail Manager in the Mailman application.

5.2.1.6 Speech outputs

The Presentation Manager receives a set of concepts which are transformed into textual form with added prosodic information. As in dialogue management, there is one generic presentation agent, the Concept-based NLG Agent, which handles most basic situations. This agent takes a list of concepts (value pairs). Then it uses shared concept and domain definitions from the Information Storage to realize expressions used in speech outputs. In its current form this agent does not utilize any additional information, such as which concepts are new, but this functionality may be easily added by extending the presentation agents.

Beside the general NLG agent, the Busman system contains several specialized presentation agents. They are used to generate messages for error situations, confirmations and so on. Most of these use common Jaspis agents with a domain specific configuration, but some of them are specific to the domain.
5.2.1.7 Input parsing

As in the presentation management and dialogue management modules, user inputs are interpreted on the basis of the shared concepts. The generic component for processing user inputs is the Concept NLU Agent, which seeks known words from the input utterances and uses shared concept definitions to extract concepts from the utterances. Specialized I/O agents are used to handle synonyms, resolve confirmations and expand temporal references, for example.

One specialized I/O agent solves references to user requests for return trip inquiries. The system is able to handle questions such as “how about the return journey?”. There may be time constraints, such as “I like to be back before four fifteen”. This functionality is implemented in a specialized input agent, which changes the destination and departure locations in the form, and updates the time constraints. This is a good example of how the system may be extended by adding new agents to bring additional functionality.

5.2.1.8 Recognition grammars

In order to make the task of the recognizer manageable, the Busman system uses several speech recognition grammars for different situations. This is a commonly used method to improve recognition accuracy. In addition to the default recognition grammar, special grammars are used for greetings and confirmations, for example. Special recognition grammars are selected by the input agents and evaluators so that input agents represent grammars, while input evaluators choose between agents. The selection of the recognition grammar is based on the current dialogue state. In the Busman system there is an explicit mapping between dialogue states and recognition grammars.

Recognition grammars are generated offline. There are two reasons for this. First, the recognizer used does not support dynamic grammars, and thus grammars must be compiled beforehand, because there are no practical ways to compile grammars in runtime. Some other recognizers, however, support dynamic grammars so this is not a fundamental problem (for example, in the AthosMail application we have implemented dynamic grammars to handle
Finnish inflected names). A more problematic issue is that fully automatic grammar generation on the basis of concepts used is far from trivial. In practice, hand-tuning and extensive testing with real users are needed to find out how well the recognizer performs in real situations and what kind of language is used.

The selection of recognition grammars in the Busman application is done a priori, while it would be also possible to do that a posteriori. This can be done by using several grammars, which first recognize the overall topic of the utterance, and, based on this result, a suitable task oriented recognition grammar is then selected by evaluators to recognize the content of the utterance. This kind of solution was used in the Interact project to detect out-of-topic utterances, and is presented in the next section.

5.2.2 Interact System

The Interact project was a Finnish national applied research project carried out by several Finnish universities and funded by the National Technology Agency (TEKES) and several information technology companies. The main purpose of the project was to investigate natural interaction methods which may be used in speech applications. As a national project, we focused on spoken Finnish. Adaptivity was the main theme of the project, and this was explored in various ways, including the combination of rule-based and statistical approaches to interaction management. The Jaspis architecture was used to combine different approaches and techniques into a working spoken dialogue system. The project and its results are presented in our research article [Jokinen et al., 2002].

The Interact system provides Helsinki area bus timetable information. The functionality of the Interact system is very similar to other bus timetable applications, such as the Busman system, i.e. users may elicit information such as bus routes and their timetables. This is demonstrated in Dialogue 3. The system uses speech for both inputs and outputs and is telephone-based. The system is Finnish, and there are additional English outputs for demonstration
purposes. The system also has multimodal extensions, which include a graphical touchscreen interface for information kiosk style usage, and a short-message interface for mobile non-speech usage.


"Welcome to the Helsinki automatic bus timetable service. You can request bus numbers, departure and destination times and places. Please speak after the tone. How may I help you?"

U: "Milloin bussi numero 6 lähtee Hakaniemestä?"

"When does bus number 6 leave Hakaniemi?"

S: "Mikä tulopaikka olikaan kyseessä?"

"What is the destination?"

U: ""Hietalah – Hietalahti"

S: “Bussi 06 Hakaniemestä Hietalahteen lähtee kello 10:59”

“Bus 06 from Hakaniemi to Hietalahti leaves at 10:59”

U: “Koska bussi numero 6 saapuu Hakaniemestä Hietalahteen?”

“When does bus number 6 from Hakaniemi to Hietalahti arrive?”

S: “Bussi 06 Hakaniemestä Hietalahteen on perillä kello 11:14”

“Bus 06 from Hakaniemi to Hietalahti arrives at 11:14”

Dialogue 3: Interact example.

5.2.2.1 Application architecture

The overall Interaction system layout, which is illustrated in Figure 25, is similar to other Jaspis-based applications. It uses, however, an older version of the architecture, which means that some of the components have different names.
There is one additional manager, the Task Manager, which communicates with the application back-end. The application back-end is a relational database serving Helsinki area bus timetable information. Other components produced in the project are implemented as agents and evaluators, and they operate under different managers of the system.

5.2.2.2 Dialogue state and form

The core of the interaction management in the Interact system is the dialogue state, which is modeled using a set of concepts. The concepts used are speaker, dialogue act, topic, new information, goals and expectations. These concepts are present in each turn that the system contains. A list of proceeding turns forms the dialogue history. Domain specific knowledge is modeled using a form. These data structures are stored in the Information Storage.

Figure 25: Interact system overview.

The Interact application is a fairly complex system and utilizes multiple computers to run. We have organized several project demonstrations in which the system has been running on several computers which are connected via sockets. For practical reasons (some of the components need special hardware) these components have been physically distributed both in Helsinki and Tampere. Nevertheless, the final system was able to run within reasonable response times.

5.2.2.2 Dialogue state and form

The core of the interaction management in the Interact system is the dialogue state, which is modeled using a set of concepts. The concepts used are speaker, dialogue act, topic, new information, goals and expectations. These concepts are present in each turn that the system contains. A list of proceeding turns forms the dialogue history. Domain specific knowledge is modeled using a form. These data structures are stored in the Information Storage.
5.2 - BUS TIMETABLE SYSTEMS

5.2.2.3 Dialogue management

The dialogue management strategy is user-initiative with mixed-initiative features, as demonstrated in Dialogue 3 (the system takes the initiative in enquiring the destination). Instead of functional agents, as in the Mailman application, or concept-oriented agents, as in the Busman application, dialogue agents in the Interact system model dialogue acts. In practice this means that the system contains dialogue agents for asking information, confirming possibly misunderstood concepts and informing the user of results found (e.g. bus lines). There are also agents for meta-communication, such as for asking rephrasing from the user.

Dialogue agents generate new dialogue turns representing certain types of dialogue actions. The actual functionality of dialogue agents is therefore very straightforward. Agents are quite autonomous, and the selection of agents is done on the basis of their self-evaluation methods, i.e. agents check the current dialogue state, and use heuristic rules to give an estimate of how well they think that they can handle the situation.

In order to make the dialogue management more adaptive, the use of reinforcement learning has been planned. In this approach, the evaluators are trained on the basis of the collected corpus. These evaluators can focus on different issues, such as the evaluation of functional suitability and user preferences. The details of the dialogue management and the adaptation method have been reported by Kerminen and Jokinen [2003].

5.2.2.4 Task management

The Task Manager takes care of performing database queries on the basis of the content of the form. It contains several task agents which take care of different database queries. Task agents check that all the required concepts are present in the form so that database queries can be performed. If all required concepts are present, the query is executed and results are stored in the form. Beside the general task agent, there are agents for handling the navigation in results sets (e.g. bus lines) and ensuring that the names used in queries are legal.
The Task Manager operates in close collaboration with the Dialogue Manager. This collaboration is not straightforward pipelined processing. Instead, agents of these two modules communicate by switching turns on the basis of the form which is stored in the Information Storage. Agents check the content of the form and decide when they should take the turn, and when they should release the turn and let other agents continue processing. This can be thought of as a set of negotiating agents.

In the first version of the system all database operations were included in the dialogue management components. In the current version database operations form their own module, which is similar to the Mail Manager in the Mailman application and to the Database Manager in the Busman application. This distribution makes the dialogue management components application and domain independent in most cases, and allows the system level adaptation to take place, as presented previously.

5.2.2.5 Speech outputs

The Presentation Manager operates on the basis of concepts produced by the Dialogue Manager. Each concept is marked as new information or topic. In addition to these concepts, results from the database query are stored in the form in the Information Storage. Presentation agents generate speech outputs from these concepts.

The presentation management module contains two kinds of presentation agents: first there is a natural language generator, which is the default output generation agent. This monolithic component produces outputs in normal situations, i.e. it takes care of generic messages, such as greetings and other similar prompts. In addition, we included several specialized presentation agents in the system during its iterative development cycle. These include agents for presentation of lists, and other agents which produce basic system prompts as an alternative to the main generator agent.
During the user experiments we added a set of new agents which implemented more informative outputs than the brief main generator agent. We also used prosodic elements to make outputs more pleasant for the user. By using compact agents it was possible to add new outputs and experiment with them without the need to modify the main generator agent, which was built by other project partners than the system evaluators. We also found it very useful to utilize the dialogue and task knowledge from the Information Storage to make speech outputs more efficient, as presented in the following sections.

5.2.2.6 Input parsing

The communication components, i.e. the I/O agents and evaluators conceptualize speech recognition results into system concepts. Different I/O agents take care of basic morphological analysis, topic and focus detection, ellipsis resolution, anaphora resolution, dialogue act classification and combination of input modalities. These agents operate independently of each other, but they naturally utilize information produced by other agents, so, for example, the discourse-level agents utilize the information produced by sentence-level parsing agents.

Since current speech recognizers are not good at handling of out-of-vocabulary words, and the topics that a single application can cover are limited, there is a need to detect the most common requests for out of domain information. In order to do this, the Interact system has a special agent for detecting the topic of the utterance. This is done by using two recognizers, one having a limited task oriented grammar and the other having a more open grammar. If the overall topic, which is identified from the open grammar, is such that the system can handle it, the content of the utterance is recognized using the task oriented grammar.

The open grammar covers a large set of words relevant to the domain. The topic agent uses document maps, a methodology based on the Self-Organizing Maps algorithm [Kohonen, 1995] for mapping utterances to topics. Document maps are trained from the real usage data by collecting conversations between users and the human operator [Lagus & Kuusisto, 2002].
CHAPTER 5: JASPIS-BASED APPLICATIONS

Topic identification is an example of robust input handling in which the overall content of the utterance is analyzed first, and more detailed analysis is performed after that, if appropriate. Since the topic identification component forms its own agent, it is transparent to the rest of the system, and can be used in a plug-and-play fashion. This kind of general agent may be ported to other domains. Of course, the training of document maps must be made separately for each domain.

It is also possible to use multiple task oriented grammars, as in the case of the Busman system. When the overall topic has been identified, a suitable task grammar may be used to recognize the detailed content of the utterance. A similar approach is used in the MIT Oxygen project [Oxygen, 2003].

5.2.3 User Experiences

We conducted an informal user study with the Interact system. The purpose was to find the most important technical and interface problems rather than study any specific interaction issues. The overall goal was identify the bottlenecks of the interaction, both from the technical and user interface perspectives. We conducted eight test sessions, each session consisting of one user making four calls. A scenario-based approach was used, in which the user played the role of a visitor traveling around in Helsinki using buses and trams. In order to successfully complete the given tasks, the user needed to use all the main features of the system. Several problems were found after each test, and the interface was found to be inefficient - the average number of user utterances for each task was five, although only one or two would have been needed. On the basis of the findings Jaspis agents were used to improve usability and add new functionality to the system.

One of the causes of problems was recognition accuracy. The recognizer initially had problems with the vocabulary of 1200 words (the application database contained over 1100 place names) and relatively open grammar. Therefore we reduced the number of names to around 50. Other identified reasons were the
system's inability to take the initiative and to offer sufficient feedback to the user in error situations. This is demonstrated in Dialogue 4. These two issues are somewhat related, and it could be that in a system-initiative mode brief, non-informative prompts and system feedback could be more acceptable. Similarly, if the system is more informative, the users might be able to solve recognition errors in the user-initiative mode.

S: (a welcome prompt omitted from the transcription)

U: "Koska lähtee seuraava bussi Hietalahdesta Munkkiniem öö korjaus Hietalahdesta rautatieasemalle?"

"When does the next bus leave from Hietalahti to Munkkinie ehhh correction from Hietalahti to Railway Station?"

S: "En valitettavasti löytänyt yhtään bussia Hietalahdesta Munkkiniemeen."

"Unfortunately I could not find any buses from Hietalahti to Munkkiniemi."

U: "When does the next bus leave from Hietalahti to Railway Station?"

S: "Bussi 06 Hietalahdesta keskustaan lähtee kello 10:58."

"Bus 06 from Hietalahti to downtown leaves at 10:58."

U: "Koska lähtee ensimmäinen sellainen bussi keskustasta Munkkiniemeen joka lähtee myöhemmin kuin 10:58?"

"When is the next bus from downtown to Munkkiniemi which leaves after 10:58?"

S: "Valitettavasti bussi 44 ei mene keskustasta Munkkiniemeen."

"Unfortunately bus 44 does not go from downtown to Munkkiniemi."
CHAPTER 5: JASPIS-BASED APPLICATIONS

Dialogue 4: Interact user study example.

Since there were errors in most of the dialogues, the interaction would have been system initiative most of the time, if the system took the initiative in error situations. Furthermore, since “perfect dialogues” were rare, it is very likely that the system would have performed much better in the system-initiative mode. When similar tests were made for the Busman system and the same problems occurred, we concluded that it is more efficient to use a system-initiative dialogue strategy as a default. This approach is used in our latest timetable information service called Pysäkkimies. For the Interact system it was too late to implement a more system initiative dialogue strategy, so we focused on improving the language model and system outputs.

In addition to recognition accuracy, the experiment revealed other problems with the language model. A corpus recorded from real calls to the Helsinki local traffic information service center was used as a basis of the recognition grammar
and input analysis components [Jauhiainen, 2001]. However, the language used by the users in the experiment was different from the language used in the recorded conversations between human operators and users. The lesson learned here is that the language model should not be acquired directly from human-human conversations, because the language used in human-computer communication is really different. The users tried to adapt their language to the system. Some users tried to use a slot filling strategy, and they used utterances such as “departure: railway station [pause] destination: Arabia”. The latter sentences are similar to universal commands as presented by Rosenfeld et al. [2001]. We modified the recognition grammar to include such utterances and added new I/O agents to help the main interpretation component to interpret the utterances.

In order to help the users in knowing what to say to the system we implemented an incremental help feature using presentation agents. However, we found that the users did not listen to the help messages at all. When they realized that there was a help message presented to them, they quickly interrupted the system. One possible solution would be to use context-sensitive help which gives only the information needed. It is not easy, however, for the system to understand the situation of the user, and in many cases it would be better for the system to take the initiative. The use of interactive tutoring agents, which we have tried out in the e-mail domain [Hakulinen et al., 2003], is a promising context-sensitive technique to teach users to use the system.

As stated previously, the main generator agent used brief outputs. The disadvantages of brief system outputs were obvious when the system misunderstood the user. When everything went well, the system responses made perfect sense, and no unnecessary information was exchanged between dialogue participants. However, since the system did not confirm what it understood, it also happened that the system response was perfectly appropriate to the situation, but in reality the system misunderstood the user, and because of brief outputs, the user did not realize this. In order to provide more feedback we added several presentation agents to the system. They presented more verbose speech outputs,
and especially confirmed recognized concepts implicitly by adding “redundant” information to speech outputs, e.g. repeating already recognized concepts. This is similar to the confirmation approach presented in [O'Neill & McTear, 2002]. Overall, we found Jaspis agents useful for extending the system to overcome problems found during the user test.

5.2.4 Summary

The Busman and Interact applications present two approaches to timetable systems. They both provide similar functionality, and in many ways similar interfaces as well, but their implementation and utilization of Jaspis agents is different. In the Busman system a generic domain and concept model is used in all system modules. General agents provide functionality for common operations, while domain and application specific information is presented in the shared information storage. The use of shared concepts helps to keep the system coherent, although the system is otherwise highly modular and distributed. The domain and concept model is a Jaspis extension, and it has been used in other applications as well. The Busman system contains also additional agents which provide domain specific functionality and operate together with general agents.

In the Interact system agents are used to model dialogue acts. Like in the Busman system, dialogue agents operate on concepts. The output module contains one legacy output generator and several compact presentation agents. The legacy generator produces basic system outputs, while specialized agents handle various individual situations. We found it very useful to work with compact agents, especially during the user experiments. We modified and extended specialized presentation agents without the need to modify the legacy generator. We also utilized the shared information storage, and especially dialogue information to modify speech outputs on the basis of user feedback. In this way, the architecture facilitated work in this multi-site project.
The evaluation of the Interact system revealed several serious usability problems. Based on our experiences, user-initiative dialogues are hard to implement with current Finnish speech technology. In the Busman application a truly mixed-initiative dialogue management was experimented with. Both applications separate dialogue management from task and database management in their own modules. The Interact system also utilizes the distributed nature of Jaspis applications, and runs on several networked computers. In the Interact system reinforcement learning techniques were studied to implement evaluators and provide adaptation on the basis of the human-human corpus collected. Both systems utilized context-sensitive recognition grammars. In the Busman system the selection of recognition grammars is done a priori, while in the Interact system several recognition grammars are done a posteriori. In both systems evaluators are used for the selection. The architecture provided strong support for all of these tasks.

5.3 Pervasive Speech Applications

Doorman is an ubiquitous computing system which uses speech and audio as its main modalities. The system is actually more a research playground than an application, and we use it to experiment with how speech interfaces could be implemented in pervasive computing settings. Several extensions, such as an auditory awareness information service, have been added to the system. The system, as presented here, consists of many applications which are designed to work together. The basic guidance system is fully functional, while other functionality is more experimental, and in some cases is not integrated completely to the main system at the moment.

In this field many new challenges have been encountered, and in this section I discuss the challenges of such systems, and especially what kind of system architecture is needed. Experiments from this domain have greatly influenced the development of the Jaspis$^2$ architecture.
CHAPTER 5: JASPIS-BASED APPLICATIONS

5.3.1 Doorman System

The Doorman system serves staff members and visitors in an office environment. The system controls access to our premises by identifying staff members and helping visitors to find the place or person that they are looking for. The system gives guidance to visitors about how to reach their destination in our premises.

The Doorman system uses speech recognition, speaker verification and speaker identification to recognize the user and his/her speech. Synthesized speech and pointing gestures are used for multimodal outputs. The presence and location of users are tracked with different sensors, which use infrared and EMFi (Electro-Mechanical Film) technologies. Other information, such as keyboard activity, is received from desktop computers and mobile devices. The system has features similar to the Office Monitor [Yankelovich & McLain, 1996] and PER [Pakucs & Melin, 2001] systems. Next, the functionality of the Doorman system is presented.

Figure 26: Doorman system.
5.3 - PERVASIVE SPEECH APPLICATIONS

Figur 26 illustrates the basic functionality of the Doorman system. The user interacts with the system for the first time when he/she arrives at the front door of our premises (1). The system activates the speech recognizer when it detects the presence of the user. The user may also push the doorbell, which causes the system to wake up if the sensors have failed (for example, if there has been a lot of traffic recently). A welcome prompt is presented to the user, and the purpose of the visit or the name of the staff member is asked. Dialogue 5 demonstrates the basic interaction with the Doorman system.

In the case of visitors:

U: (Pushes the doorbell)

S: “Olen Ovimikko. Sano etsimäsi henkilön tai paikan nimi äänimerkin jälkeen.”

“I'm the Doorman. Please say the name of a person or a place after the tone.”

U: “Markku Turunen”

S: “Tervetuloa. Lukko on nyt auki.”

“Welcome. The door is now unlocked.”

U: (Enters inside to front of the guidance puppet)


“The person that you are looking for, Markku Turunen, can be found from room 432. In order to get there turn left. Go to the front of the meeting room. Turn right, and go to the sofa. Markku's room is two meters from there, in front of you, at left.”
In the case of staff members:

S:  (Detects movement and starts recognition)

U: “Turusen Markku tässä huomenta.”
   “This is Markku Turunen, good morning.”

S: “Terve Markku. Lukko on auki.”
   “Welcome Markku. The door is unlocked.”

Dialogue 5: Doorman example.

5.3.1.1 Staff members

If the person arriving does not press the doorbell, but initiates the dialogue, the system assumes that the person is a staff member, and expects to receive his/her name either in isolation or as part of the response to the system greeting. When the user utterance is recognized the result is compared to the speaker verification result. If these results match, access is granted to the premises and the door is unlocked. When the staff member enters, the guidance puppet gives additional information, such as the number of messages which have been left for the user.

5.3.1.2 Visitors

If the person arriving press the doorbell the system assumes that the person is a visitor, and asks the name of a staff member or the name of a place. In our office environment most visitors are either students or research partners, and they are assumed to know the name of the place (such as a speech laboratory or a meeting room) where they are going, or the name of the person that they are seeking. Of course, visitors may give both of them (“I should meet Mr. Turunen in the lobby”).

When the object of the visit is identified, the door is unlocked. If the purpose of the visit cannot be identified, i.e., no known object can be recognized, the system asks the user to repeat the information, this time in the basic form (“Please give
the name of the place or the person you are looking for in basic form, such as Professor ...”). If the communication still fails, the system says that it cannot open the door, and notifies staff members. The visitor may also ring the doorbell whenever he/she wants, and the system will stop.

5.3.1.3 Guidance

When the visitor enters inside, the puppet is used to give guidance to the user. The premises are modeled in such a way that the system is able to generate various alternative route descriptions with varying levels of detail. In the current setup the guidance is given in one place. There may be more than one guidance puppet. The puppets may be located in strategic places in the office, such as intersections. In this way, the guidance is given in small chunks, which are easier to understand and memorize. The user could also ask for more help from the system.

In addition to the presented basic functionality, we have implemented several extensions to the Doorman application, which are described next.

5.3.2 Awareness Information

One group of extensions to the Doorman system is applications for supporting awareness. In these applications auditory information is presented in a way which helps to keep users aware of events occurring in their surroundings. This is done so that auditory icons and other auditory information are used to indicate important changes in the environment, such as arrival of visitors, group members and incoming messages.

Recently we have implemented an application which monitors the presence of group members and informs other group members about the activity in the group. The system uses activity information from desktop computers, so we know when users are typing or using their computers in other ways. The awareness information is based on other sources as well. We have placed EMFi sensors around our premises, so we can track where people move. This tracking
CHAPTER 5: JASPIS-BASED APPLICATIONS

is transparent to the users. The same information sources that were used by the Doorman application can be used as well. For example, if a group member is spotted at the door, this information can be used by the awareness application.

There are two basic ways to present awareness information. First, a graphical layout is used. Figure 27 illustrates a situation in which several group members and their presence during the day are displayed. This preliminary visualization uses the same information sources as the Doorman application, and especially its geographical model of our premises.

![Figure 27: Awareness visualization.](image)

The second way to present information is the use of environmental audio. We have placed EMFi-loudspeakers at several locations on our premises. Loudspeakers are placed both in public places and in the rooms of group members. Loudspeakers are shared with the Doorman application.

The loudspeakers are used to provide information similar to that from the graphical interface, but in a more transparent and non-obtrusive manner. Both continuous and temporal information are presented, such as auditory icons representing the activity of group members, and sounds which are played when
somebody enters the premises. Natural sounds, such as walking sounds, are used in auditory presentations [Mäkelä et al., 2003].

5.3.3 Additional Functionality

The second extension is a communication application using the same infrastructure as the Doorman and awareness applications. This is implemented so that each group member has a microphone connected to his/her computer, and the communication is started by saying the name of another group member. When the name of the receiver is recognized, the system establishes a communication channel (via TCP/IP) between the group members. Then conversation is continued so that the application routes audio information between the users. The application also allows group communication, so messages can be broadcasted to selected or all group members. Communication can be also archived by the system.

We have plans to extend the system functionality in various ways. Since all communication points have both microphones and loudspeakers, they can be used to perform additional functions and serve other communication purposes. For example, a staff member may ask the system “Are there any new messages for me?”, and the system identifies the staff member and informs the user about his new e-mails and other messages. The users may also track other staff members, for example by querying “Is John still in his office?”. We also have plans to integrate other applications, such as the Mailman and Busman applications into the Doorman application. In this way, visitors may query timetables when they are going to leave our premises, for example.

Multilinguality is another issue in the Doorman application. Many offices are multilingual these days, and in our case there is an increasing number of non-Finnish speaking visitors and students visiting our premises. The system needs to be multilingual in order to be useful. There are currently versions for Finnish and English, and plans for a truly multilingual version are in the making. It is noteworthy that since there can be communication between multiple participants
and the system, the language should be always user specific, and a single session may include multiple languages.

All of the applications mentioned are implemented on top of the Jaspis architecture. They have fairly straightforward dialogue agents, and more complex presentation and inputs agents which convey the information to users. There are few conventional dialogues between the user and the system, and the awareness application especially is more a proactive system than a dialogue system. A system overview is presented next in more detail.

5.3.4 Application Structure

Figure 28 illustrates the Doorman application structure. The system contains the standard Jaspis managers and several additional managers. The Communication Manager runs on a specific computer which handles all sensor information and

![Figure 28: Doorman application structure.](image-url)
other hardware resources. The External Context provides information about the environment and is a separate Jaspis application.

5.3.4.1 Dialogue management
The dialogue control model of the Doorman application was originally based on the state-machine approach. The basic functionality of the application was quite limited and tasks were clearly defined, so the state-machine approach was a natural choice for dialogue control. The problems of this approach were soon discovered, however. First of all, the number of states increased rapidly when error management and other meta-communication were added. Modifications to the system were also problematic. For these reasons, the dialogue control model was extended.

The current dialogue model has been extended to be agent and event oriented in addition to the finite-state model. Current dialogue agents are more related to the interaction situations than system states. In this way it is more flexible to handle special situations and add new functionality to the system. Examples of dialogue agents include the Doorbell Agent, which handles all doorbell related situations regardless of the context. Another dialogue agent handles speaker recognition results. In the future the rest of the finite-state model will be replaced by more independent agents.

5.3.4.2 Knowledge sources
The Context Manager keeps track of the user activity on the premises. Part of this information is collected by the Doorman application, while other parts of the information are retrieved from other sources. The main information source is the External Context, which is a Jaspis installation running on a separate computer (since it is shared between multiple applications). Doorman uses information from this database, and stores new information in the database as well. In addition to Doorman, it serves other applications, such as the awareness application. The Person Manager manages the information related to the staff members.
CHAPTER 5: JASPIS-BASED APPLICATIONS

The Route Manager serves route descriptions. The core of the system is a two-dimensional geographical model, which is represented as an XML document. When the user needs guidance, a request from one place to another is given and the Route Manager produces a conceptual representation for the requested route. There are several levels at which routes can be represented. Finally, a surface generation is done for several languages (currently Finnish and English are supported) in their own agents. Similarly, graphical representations are generated by the awareness application from the same conceptual information.

5.3.4.3 Limitations of the interaction model

There are two major problems in the current dialogue model. First, it is turn-based rather than event-based. In the Doorman application, as in other environmental applications, turns are not a very efficient concept, but instead all kinds of events occur continuously. Although the application is targeted mainly at spoken interaction, there are no explicit, well defined tasks, and the user may initiate and stop the dialogue in unexpected ways. The changes in the environment also affect the dialogue. For example, if the target person of the visit leaves his/her office while the guidance is going on, the situation needs to be handled immediately.

The second problem is that the application is not capable of handling concurrent dialogues. There are many places in which concurrent dialogues are needed. For example, there are often many visitors coming to separate meetings, and the system must serve all of those visitors. While one visitor is being served inside our premises, another visitor may appear at the door. Similarly, multiple staff members may communicate with the system at the same time.

5.3.4.4 Multiple distributed systems

One solution for handling concurrent dialogues is to split the dialogue into sub-dialogues which form their own applications and communicate with each another. The Jaspis architecture provides strong support for this through its shared system storage and efficient communication model. For example, a door
application could take care of visitors at the door, and update the information to the shared External Context. A first guidance application then receives notification from the External Context, gives the first part of the guidance and updates the External Context. A second guidance application then continues from this phase onward and so on. This is technically straightforward, and in fact the only difference would be that there are several instances of the Doorman system running, each of them containing only a selected set of agents and evaluators. There is also an additional shared Information Storage (External Context) in addition to the local ones.

Since each Doorman installation is its own entity, it can serve multiple users at the same time. For example, when the door application stops and the guidance application starts, the door application is able to serve new visitors. Nevertheless, they are not able to react to external events dynamically. For example, if the door agent sends notification of another incoming visitor while the guidance is in progress, the guidance application is able to react only when the guidance has been given. There should also be several systems running at the same time, which may be impractical when the number of dialogues increases.

5.3.4.5 Concurrent distributed agents

A working solution would be to allow several agents to run concurrently, each of them having their own dialogue in progress. Furthermore, agents should be invoked in dynamically, so that they can serve multiple users without the need for queues, delays and unnecessary instances of components. Of course, physical resources such as microphones cannot be used at the same time. The presented limitations of the Doorman application have led to the second generation of the Jaspis architecture, which allows concurrent agents.

5.3.5 Wizard Of Oz Experiment

The usability of the Doorman system was evaluated as a part of the development process by performing a Wizard of Oz experiment, in which speech recognition
and speaker recognition were simulated. Otherwise the guidance system was functional, and the wizards were not able to affect the interaction. The test subjects consist of staff members (around 30 people), visitors and students. The study lasted one week. In total, 74 dialogues were analyzed. The details of the test setup and the Jaspis WOZ tool are given in the next chapter, while our research article presents the experiment in detail [Mäkelä et al., 2001].

The main findings from the user study concerned the way that users spoke to the system. First of all, it surprised us that none of the visitors (who did not know the real nature of the application) used place names. Instead, the users were always looking for people. In most cases this was not problematic, but there were also cases in which the person that the visitor was looking for was not a staff member. For example, our usability laboratory is used extensively by students to accomplish assignments, and in several cases students were seeking usability experiment conductors, not the usability laboratory. In these cases the system often failed to serve visitors, since it does not recognize the names of these people. The users did not use the names of places even though they were explicitly instructed by the system to give the name of a person or the name of a place.

Another surprising finding was that although it was assumed that only visitors use the doorbell, the staff members used it as well in many cases. This was problematic, since when they pushed the doorbell the system assumed that they were visitors. This led to situations from which there was no way out. In some cases staff members did not seem to notice this, however, and they were satisfied that the system let them in. In the following version this incorrect assumption was corrected. There is still a possibility that the system misrecognizes the user utterance and confuses staff members and visitors. Obviously, these situations may be solved by using confirmations, but this brings additional turns to the interaction, and in this application the interaction should be as fast as possible. Otherwise, staff members use their keys to access the premises and do not use the system. Visitors are easily frustrated as well.
5.3 - PERVASIVE SPEECH APPLICATIONS

We also spotted problems with the spoken guidance. The original guidance algorithm used directions to guide visitors. It was soon found that this does not work. If directions, such as “walk five meters forward and turn right, then walk seven meters forward and turn left, ...” are used, most guidance prompts will be too long for the users to understand and memorize them. Instead, landmarks should be used, and long prompts should be spoken in shorter chunks. There may also be a dialogue going on while the user is searching for the destination, for example “where should I go now?” or “was it the second or the third door from the left?”. For these reasons, the guidance algorithm was modified to use landmarks and relative measures, and the dialogue was divided to consist of smaller dialogue chunks. At this point we concluded that although it would be possible to add functionality to the system using agents to overcome the problems mentioned, the finite-state based control was not flexible enough. Furthermore, in order to implement guidance efficiently, we needed support for concurrent dialogues. This motivated the new version of the Jaspis architecture.

5.3.6 Summary

The Doorman application, and its various extensions, differ in many ways from other applications presented in this chapter. Most importantly, concurrent dialogues are needed. Although it is possible to model multiple dialogues in their own applications, it is more efficient in many cases to allow concurrency inside single applications. Experiences from this domain resulted in the second generation of the architecture.

The Doorman system used originally a state-based dialogue control strategy, which was found to be a non-optimal solution when the application was extended. Because of this a function-oriented dialogue control strategy was adopted. Still, it shows that state-based dialogues can be efficiently constructed using Jaspis agents. In pervasive applications distributed knowledge sources are needed. In the Doorman application the shared information storage of the Jaspis architecture is used successfully to serve multiple applications.
CHAPTER 5: JASPIS-BASED APPLICATIONS

When the Doorman application was tested, the spoken guidance given by the system was found to be hard for users to understand. The algorithm was modified to make the guidance consist of several chunks. The experiment also showed that the Jaspis WOZ tools are suitable for arranging user studies with such systems.
6 INTERACTION TECHNIQUES AND TOOLS

A system framework is not the only requirement for productive application development: interaction techniques and development tools are needed as well. To implement successful applications developers need reusable components, methods and interaction techniques, which may be used across multiple applications and domains. In this chapter I present general models for error management and modular output generation. In addition to general principles I also present how these may be implemented efficiently by using the Jaspis agent – evaluator – manager paradigm.

For iterative application development and evaluation, application developers need supporting tools. Speech system components usually come from multiple sources, and the development, testing and understanding the internal working of spoken dialogue applications is often a laborious task. Here I introduce a set of Jaspis tools for visualization, debugging, evaluation, demonstrations and corpora collection. These tools are extensions to the generic Jaspis infrastructure, and are called together JasViz tools. In the long run, the ultimate goal is to provide a complete development environment for speech applications.

In addition to the presentation of the techniques and tools mentioned, this chapter also reports experiences from various projects and applications in which the presented solutions were used. The techniques and tools are implemented using the Jaspis architecture, but their principles are general.
CHAPTER 6: INTERACTION TECHNIQUES AND TOOLS

6.1 Support For Error Handling In Speech Applications

Errors are unavoidable in speech-based communication. There are various styles of errors both in human-human and human-computer communication. Speech-based human-computer interaction is especially error-prone, since the technology is not mature, and the interaction technology is still in its infancy. Communication skills for speech-based human-computer interaction are also developing, and they differ significantly from human-human communication [Fraser & Gilbert, 1991].

It is often said that the major problem in speech applications is their inability to detect and correctly handle different types of errors. For these reasons, error management is essential for successful interaction in speech applications. In this section I introduce a general error handling model for speech applications. The model was originally presented in our research article [Turunen & Hakulinen, 2001a]. Here I present first the general requirements for error handling, and then introduce the model in detail.

6.1.1 Requirements For Error Handling

There are three requirements which an error handling model should support. The first one relates to dialogue management. Error management is often understood as error recovery only, and it is handled independently of the main dialogue. Some error-handling solutions even hide the errors from the other parts of the system by solving them locally [Mankoff et al., 2000]. This may cause several problems. Most importantly, the dialogue strategy may need to be changed when errors occur, for example from the user-initiative to the system-initiative. The dialogue management components may also benefit from awareness of errors, so that they may learn and adapt their behavior. To conclude, error management is not only an input management level issue. The first requirement for an error
handling model is that error handling should be seen as a part of the dialogue management and used as a part of it.

The second requirement concerns modularity. Error handling may increase the complexity of interaction, especially dialogue management, and lead to complex and problematic dialogue structures. For example, including error management features in state-based systems may multiply the number of states. Other dialogue control structures do have similar effects in their internal structures when error management is added to the system. The second requirement for an error handling model is that it should support modular components that take care of error handling independently of the main dialogue management. Nevertheless, error management should not be handled locally and hidden from the main dialogue. Instead, common reusable components for error management reduce the need to implement error management every time from scratch, and helps to keep the main dialogue more simplified and focused on the task.

The third requirement relates to the scope of the error management. As stated, it is not a dialogue or input level issue only. Instead, different parts of the error management process can be found on different levels of interaction processing. Duff et al. [1996] analyzed error management and found five different phases which take place in the input, dialogue and output parts of speech systems. We further studied the error management process and found seven phases [Turunen & Hakulinen, 2001a]. In order to handle errors efficiently, all error handling phases should be supported. The is the third requirement for an error handling model. Next I introduce an error handling model which supports all the error management requirements mentioned.

6.1.2 Jaspis Error Handling Model

The Jaspis error handling model is divided into the seven phases presented in Section 2.3. This model provides an explicit and modular view of the error management process. Error management is realized using Jaspis agents and
6.1.2.1 Error detection

Error detection components are implemented in Jaspis as I/O agents. Recognition of an error can be accomplished by the user or by the system. There are several ways that I/O agents can be used to recognize errors. In the simplest cases it is possible to examine recognition confidence scores and use threshold levels for binary decisions, i.e. to decide if a recognition result is accurate enough or not. The problem is that in practice recognition confidence scores are not especially strong candidates for error detection, and it is not advisable to rely too much on these scores when a single recognizer is used.

More robust methods include the use of multiple recognizers. The results of multiple recognizers can be used for various comparisons. For example, an error detection agent may use results from different types of speech recognizers, such as a combination of a normal, restricted vocabulary recognizer and an unrestricted vocabulary phoneme recognizer [Gustafson et al., 1999a] to detect possible errors. If the recognition results are inconsistent, errors may be present.

Figure 29: Jaspis error handling model.
Voting can also be used, if there are more than two recognizers [Schwenk & Gauvain, 2000].

Error detection agents have a close connection to the speech recognition engine. Basic agents are able to operate without any domain knowledge by just using the recognition results from recognizers as described above. More advanced error detection agents may utilize context knowledge (e.g. dialogue history and user model) to detect possible errors. For example, an error detection agent may check if the result is meaningful in the current context. In the Mailman application it could happen that a recognizer result is “Read fifth message”, even if the user has only four messages. If the recognition accuracy is high, this is a possible semantic error and should be handled.

I/O agents are also used when the user acts to correct errors. Special grammars may be used for the detection of user initiated errors. This approach is used in the Busman application, in which user initiated error corrections are detected by using a special grammar, which includes error handling related words. For example, if the user says something like “but I’m leaving from the central park, not from the central station”, this agent detects the error correction attempt, and changes the system information state (e.g. dialogue state) so that the rest of the system deals with the corrected information (departure=central park). Similarly, if the user prefers alternative modalities (such as pushing the doorbell several times in the Doorman application), the error detection agent may alert dialogue components that the recognizer is not performing well, or the user is disoriented. Other methods include the use of voice parameters such as content and duration [Hirasawa et al., 2000] or prosodic features [Litman et al., 2000] of the user responses to detect user initiated error corrections.

One particular problem in speech applications is out-of-vocabulary words and utterances, i.e. utterances that are not covered by the system language model. In the Interact system we used two recognizers, one for recognizing the task oriented utterances and one for recognizing the topic of the conversation. If the recognition results were incoherent, an out-of-topic sentence was spotted. For example, if the task oriented recognizer returned “when does the number 6 bus
leave”, and the topic recognizer returns “reductions”, it is easy to reason that the user is requesting information about discounts and the task oriented recognition result is erroneous. In this situation a correct response would be that the system does not know about discounts.

6.1.2.2 Error reasoning

Error reasoning is done by the I/O evaluators. In some cases the reason relates to the accuracy of the recognizer, but many other reasons may also exist. For example, in the previous bus system example the reason is out-of-topic utterance, not a speech recognition error. Various information sources, such as user model and dialogue history are needed to resolve error reasons. For example, when an error detection agent has spotted a possible error but the recognition confidence scores are high, we can conclude that the error is semantic, i.e., the user is disoriented or has simply used the wrong words. This may easily happen in information systems, where the user navigates between menus and lists, such as folders in e-mail applications. Such situations should be handled on the dialogue level. Because of the need for application specific information, error reasoning is often a highly application dependent task.

6.1.2.3 Error repair design

Error repair design takes place after the cause of the error has been deduced. Since error correction is about changing the route of the dialogue, this phase is performed by the dialogue evaluators. Special error-planning evaluators check the type of error, dialogue state and information on past errors and repairs. Based on these information sources they choose the most suitable error correction agent.

Domain independent error-planning evaluators can be used in many cases for choosing between error correction agents. For example, “yes/no” confirmations can be used when we have to either accept or reject the utterance, while various techniques for the selection of items from a list can be used when we have
6.1 - SUPPORT FOR ERROR HANDLING IN SPEECH APPLICATIONS

multiple alternatives. Such generic techniques can handle most error situations in a domain independent way.

One important part of error correction design is to avoid error spirals, in which an error correction procedure leads to new errors, making the interaction an endless loop of errors. To prevent error loops, an error-spiral evaluator checks what kind of error repairs have been done previously, and with what results. In this way, the error-spiral evaluator can try to terminate error cycles by favoring different kinds of error-correction agents. This evaluator may in many cases be application independent and reusable.

In some cases the error-planning evaluators may even completely ignore the error correction. This is the case when error correction costs are higher than the consequences of the error. Especially if explicit confirmations are used, the interaction may become too clumsy. Another example is real-time applications, where speech is used for temporal control [Turunen, 1998]. In these cases it is the task of the presentation agents to inform the user about the situation, for example by using implicit confirmations.

Different dialogue management strategies, i.e. user-initiative and system-initiative strategies, may be also used for error correction. For example, instead of selecting an error correction agent the dialogue strategy may be changed from user-initiative to system-initiative to solve the problems in the interaction. The need for this was very marked in the case of Interact user experiments, as reported in the previous chapter.

6.1.2.4 Error correction

Error correction is a dialogue itself. It may be a confirmation, a yes/no question, rephrasing, respeaking, spelling or a selection from a list. The error correction procedure is carried out like a normal dialogue. If more errors occur during the correction dialogue, they can be processed in a recursive manner.

It is possible to use application independent error correction agents. For example, most selections from lists can be done in an application independent
CHAPTER 6: INTERACTION TECHNIQUES AND TOOLS

way, as can confirmations. These agents are similar to general error handling procedures in systems such as CSLU Toolkit [Sutton et al., 1998].

Although many cases can be handled by generic techniques, specific techniques with domain knowledge are also needed. For example, when we want to clarify where the user wants to go to in a bus time table system, a special agent which includes district knowledge can be used. This can be an alternative to a general agent, which presents a list of bus-numbers from an n-best recognition result. The district knowledge is domain specific information which the general error repair agent does not have, and which can make the error correction more efficient. A domain specific error-design evaluator chooses these domain specific error correction agents when available.

6.1.2.5 Informing the user

Special presentation agents and evaluators are used to inform the user in error situations. First, specialized presentation agents are used to produce outputs related to certain error correction techniques. For example, explicit confirmations and selections from lists have their own presentation agents. Special presentation evaluators choose these agents when appropriate.

The second case is when the error correction is about to end. For those situations separate presentation agents are used. They inform the user that the error has been handled, and that the main dialogue will continue. In some situations this may not be needed, for example when a selection from a list continues the dialogue in a natural way. However, in cases where the course of the dialogue has changed, or errors have accumulated, the user may have lost his understanding about the current dialogue state. In these situations it is useful to present a message to the user about the current situation and implicitly or explicitly inform that the error handling process is over.

6.1.2.6 Return to the original dialogue

Return to the original dialogue involves reasoning how to continue. The situation may have changed dramatically, and the original dialogue flow may need to be
adjusted. For example, there may be a need to change the dialogue strategy or goals according to the information acquired during the error handling process. The selection of the following dialogue step is up to the dialogue evaluators. They select the next dialogue agent, which continues from the current situation. As an example, in the Busman system a special dialogue evaluator favors system-initiative dialogue agents if other errors have occurred recently in the dialogue. It is noteworthy that since there is no single evaluator which chooses agents, the selection of dialogue strategy is also a distributed process, which takes into account other factors as well.

6.1.2.7 Error prevention

Error prevention may take place on all interaction levels. Input agents and evaluators may try to reduce errors by changing recognition grammars when errors are present. For example, if they detect many recognition errors, reduced or alternative recognition vocabularies can be used. These components also select special error handling grammars for tasks such as confirmations and selections from lists. In addition, input components map conceptual input requests to devices and modalities. In multimodal systems alternative modalities can be used to reduce errors [Oviatt, 1999a].

At the dialogue level, alternative dialogue agents can be selected to be more fail-safe in error situations. The dialogue style can be also changed. These “fallback” dialogues are not usually very efficient, but in problematic situations they may be the only option for successful interaction. Special output agents can be designed to provide alternative information in situations where errors are likely to occur. It is the duty of presentation evaluators to check if the user needs these agents.

6.1.3 Implementation Issues

The error management model described covers all seven phases of the error management process, but the borders between the phases are not strict. For
example, error prevention can be included in other phases as well, especially when the user is informed about the continuation, and when the next dialogue state will be selected. When possible, it is still advisable to keep these in separate components, so that the system stays as modular as possible.

The error management model is heavily based on Jaspis agents which implement and represent different interaction techniques. The main advantage of using agents beside modularity is adaptiveness. Here the evaluators play a central role. Evaluators should be modular as well, so that dynamic selection of error handling components can be achieved. For example, when the dialogue strategy is going to be changed, there are many issues which should be modeled. These include the proximity of errors, type of errors and user preferences. These factors have a varying effect on different applications and domains, so static modeling would not be optimal. When they are modeled as separate evaluators, the balance between them can be customized to the case in hand by modifying the evaluation values which each evaluator has in its configuration parameters.

### 6.2 A Model For Modular Output Generation

The generic presentation management scheme of the Jaspis architecture assumes that the process is performed in one step, i.e., that conceptual messages from the dialogue management are processed into control messages in a single phase by a single presentation agent. In more sophisticated systems the response generation process may have more steps, as is sometimes the case with text-based natural language applications. There may be different stages in which messages are processed from certain points of view, until a natural language message is finally produced with all control information needed, such as prosodic elements.

This kind of modular generation can be done in several ways in the Jaspis architecture. In the first approach, all generation functionality can be encapsulated into agents, which means that when a message is going to be produced, the selected presentation agent calls various natural language
generation components one by one. In the second approach the output generation process is repeated several times, and each time a single stage will be performed, i.e. presentation agents for different purposes will be selected each time. This is a more flexible approach and allows the exploitation of the adaptive nature of the Jaspis architecture. Nevertheless, there is unnecessary complexity because heterogeneous components are used inside a single module. In the preferred approach there are several presentation management modules, each of them focusing on different aspects of output generation. The basic principle is the same for all of these modules. A model which realizes this approach will be presented next with examples.

Typical phases in natural language generation systems are strategic planning, sentence planning and surface realization. These can be further divided into more detailed phases. In multilingual and multimodal speech applications we need additional high-level phases for modality selection, language selection and prosodic information processing (or in general other modality specific processing). Figure 30 illustrates an example set of output generation phases, each of them corresponding to a complete presentation management module with its own set of presentation agents and presentation evaluators.

![Figure 30: Modular output generation phases.](image)

Examples of each of these phases follow. The examples given illustrate how the output evolves in these phases. The results, i.e. the various elements of the output message, are presented as trees, which can be expressed in Annotation Graphs format, as presented in Section 6.5.
In the **topic generation** phase overall topics of the output, i.e. what the system is going to say to the user, will be generated. In Figure 31 (1) two topics have been generated, a confirmation and a message listing. Two presentation agents have
been selected, one for each topic. The order of the topics is not important at this phase, and can be changed later if necessary.

After the topics have been generated, the next presentation manager will take its turn and agents for strategic planning will be selected. In Figure 31 (2) a confirmation agent has produced an implicit confirmation, which will be given before the actual content, i.e. before the list of messages. A message listing agent has produced a sub-tree in which messages are preceded by an opening phrase and followed by a closing phrase. The order of nodes can be manipulated later, if necessary.

In the modality selection phase suitable modalities will be selected. In Figure 31 (3) speech has been selected for the confirmation and the message subjects, while non-speech audio will be used for the opening and closing elements of the list.

After the modalities have been selected, the actual content of the individual sentences will be produced. When speech is used this means that sentence content will be determined. For non-speech audio outputs the style of the audio (e.g. earcons and auditory icons) is selected. This is illustrated in Figure 31 (4).

When the contents of the sentences are produced, suitable languages will be selected for each element. For speech elements this usually means that the language of the user will be selected, unless there is a reason to use some other language.

In Figure 31 (5) Finnish has been selected for all except one node. This node will be read in English, since its content is in English (an e-mail written in English). For audio objects a suitable sound scheme, such as a standardized telephone audio set will be selected.

In the next phase modality specific information will be generated. For speech this means prosodic elements, such as speaking rate and different voices. In Figure 31 (6) the message number has been emphasized along with the message subject. For auditory messages, the closing sound will be twice as long as the opening sound.
Finally, the surface generation will take place. Actual sounds are mapped to the audio messages and natural language sentences will be generated, as illustrated in Figure 31 (7).

6.2.1 Implementation Issues

The generation model presented has several advantages over black-box systems, which include all the processing inside monolithic components. Most importantly, all the adaptive and distributed features of the Jaspis architecture can be used in the process. Since generation phases are not strongly coupled, some of the phases can be skipped, and in general there is more freedom in the division of the workload between generation phases. Using this model flexible controlled systems can be constructed, and features such as feedback between phases can be implemented.

From the implementation viewpoint, the output tree can be represented using Annotation Graphs, as presented in Section 6.5. In this way, the generated output tree will contain all the information from each stage in one place, and several outputs can be combined and represented as a single graph. When the output graph is presented to the user, the temporal information can be added, and the resulting graph can be stored for analysis purposes.

The error management and generation models presented are examples of how Jaspis architecture can be used for modular and coordinated handling of interaction tasks. Using similar approach we have modeled other tasks, such as database management and handling of multilingual inputs and outputs.

6.3 Application Visualization And Control Tools

The development of spoken dialogue applications may be a laborious process, especially if no suitable tools are available. Previous work on speech applications development tools includes special tools, such as SUEDE to aid the
design process [Klemmer et al., 2000]. Some other systems, such as the CSLU Toolkit [Sutton et al., 1998] support interactive dialogue design and testing capabilities for state-based dialogues. These tools were discussed in more detail in Section 3.6.

Existing tools leave room for improvement. First, most of the existing tools are external to the application development frameworks, and thus their results may be hard to utilize when actual applications are developed. This makes them mostly usable for design tasks, and for the development of early prototypes. Ideally, development tools should be integrated to the application framework. Some development tools are also missing almost completely. For example, dynamic visualizations of speech applications are rare.

Here I present several tools for aiding Jaspis-based speech application development. These tools are part of the JasViz toolkit, which is an extension to the Jaspis architecture. In this section I present tools for dynamic visualizations and for the control and manipulation of speech applications. In the next two sections I present evaluation and data collections tools.

JasViz tools utilize the Jaspis architecture in many ways. Using the shared Information Storage, data from the interaction can be collected to be used later for demonstrations, user studies and evaluation. Modular architecture structure is used to control execution of applications, for example, to run applications step by step. These are used together to make it possible to execute applications backwards and forwards by recording all the necessary information, and using component changes as breakpoints.

Technology-wise, JasViz tools are Jaspis components, i.e. managers, agents and evaluators. There are two types of JasViz components: extended versions of the general Jaspis components and new managers, evaluators and agents. Application developers are able to include JasViz components in applications by including their definitions in the system configuration file. Developers can define how and when different JasViz tools are used. By controlling the priorities of the components developers are able to achieve precise control over
the interaction. For example, application designers may specify that between the execution of certain managers the system pauses for a certain amount of time, or that before a manager of their interest (e.g. a dialogue manager) the system waits for user actions.

6.3.1 Visualization Tools

The internal behavior of an application is not easy to understand from its external behavior, especially when applications are based on speech inputs and outputs. In modular and adaptive systems it can be hard to understand which components are operating in different situations, especially if the application designers and the evaluators are different people. Software visualization tools can help developers to understand and debug applications.

Program visualization has been studied widely in recent decades. Various visualizations have been developed for different needs, especially for application development and for educational purposes. While program visualization tools try to present how executable programs behave, algorithm visualization tools try to aid the understanding of the logic behind programs. JasViz stays in the middle of these approaches. It visualizes concrete applications, but it leaves low-level details to generic tools, such as programming language development environments. It helps to understand how information is exchanged and how interaction proceeds in speech applications. Its methods for visualization are motivated by previous work on algorithm animation [Brown, 1988].

JasViz visualization components include a controllable tree view of the content of the Information Storage. Using this component, information can be modified, highlighted and annotated while applications are running. JasViz also contains a tool for the visualization of the system components, which helps to understand which managers, agents and evaluators are active during interaction.

The most general visualization tool is the Information Storage Visualizer. It shows both textual and graphical representations of the content of the
6.3 - APPLICATION VISUALIZATION AND CONTROL TOOLS

Information Storage, and also how system components use the storage. Since the default Jaspis Information Storage is implemented as an XML document, it is possible to use both text and tree representations. The Information Storage Visualizer is an extended version of the Information Storage. It monitors components which access the storage and provides visualizations. This tool can be seen in Figure 32.

![Figure 32: Information Storage Visualizer.](image)

The Information Storage Visualizer uses several parameters to emphasize information events occurring in the system. Events are visualized by using different colors for different operations. For example, information retrieval and update operations have different colors. To enhance visualizations and provide temporal information the intensity of colors is altered as time passes. This means in practice that the results of the most recent operations are visualized by using pure colors and when the time passes, the colors fade. This method is a simple but powerful way to convey information [Leponiemi, 2000].

Other features of the Information Storage Visualizer include the marking and monitoring of interesting information. Using this feature, we can see how certain
pieces of information are used and modified when the application execution proceeds. It is also possible to manipulate the content of the Information Storage interactively. Even the complete Information Storage content can be changed in this way. This may be used to control applications, as described in the following sections.

There is also an option for system components (agents, evaluators and managers) to provide additional information for the developer by annotating the Information Storage. These annotations act like comments in a program code. For example, when preparing speech outputs a presentation agent may include information on why it chooses a particular sentence. Developers are also able to make annotations, for example to provide comments to be used later by developers or usability researchers. Annotations are visualized by using cartoon balloons, as illustrated in Figure 32, and included in the Information Storage as XML comments. In this way they are not separate from the other information, and are stored with it.

In addition to the Information Storage Visualizer, JasViz contains a tool for the visualization of system components. The Component Visualizer shows how system components take turns during the execution of applications. Figure 33 illustrates how managers, agents and evaluators are visualized. The colors of components reflect temporal information in the same way as in the Information Storage Visualizer, i.e. components that have been active more recently are visualized using pure colors, and the colors fade as the execution proceeds.

The activity of components is recognized from the operations that they perform on the Information Storage. Therefore, all components that access the Information Storage are automatically visualized. One of the principles behind the Jaspis architecture is that components are ideally stateless and store all their information in the Information Storage. In this way they access the Information Storage every time when they do something. Obviously, this principle does not apply to the internal data of components, but instead to the shared knowledge. In this way all the important data resides in the Information Storage.
6.3 - APPLICATION VISUALIZATION AND CONTROL TOOLS

6.3.2 Application Control Tools

In addition to the visualization tools, other key features of the JasViz toolkit are the ability to control the execution of applications in various ways, and an option to modify the content of the Information Storage dynamically. Together with everyday application development tasks, these are useful for demonstrations.

Controllers are used for application execution. The general application controllers monitor developers' actions or other specified conditions, such as rules, timers or external components. JasViz includes several default components, such as those waiting for keyboard actions and GUI events. Simple rules can be used to control applications or give the developer an option to react to interesting events. Time controlled execution of applications is possible using timers. Controllers can be combined, for example, to wait for user actions or events for a certain period of time.

One powerful feature to understand and test applications is the ability to browse the applications' execution history, and restart the application execution from selected points. This makes it possible to backtrack when executing applications and also to go forward when re-running applications. This can be implemented

![Component Visualizer](image)

Figure 33: Component Visualizer.
CHAPTER 6: INTERACTION TECHNIQUES AND TOOLS

by keeping a record of changes in the Information Storage. In this way we can recreate the Information Storage as it was at a certain point in time. Of course, system components must be stateless and store all their information in the storage.

There are several interface techniques to control application execution in the ways mentioned. Application execution history can be browsed by using various methods. The basic method is the browsing of one step (i.e. manager change) forwards or backwards. Another method is browsing between bookmarks, i.e. running the system until a bookmarked manager is selected. In this way applications can be examined in terms of interesting managers. For example, the developers of the dialogue manager can skip all other managers and focus on the dialogue manager. The execution history can be browsed by using sliders for free selection of states, or by using temporal bookmarks.

Since all dynamic information, including user inputs, database results and exchange of information between system components is stored in the shared Information Storage, applications can also be controlled by modifying the content of the Information Storage. This feature can be used as a design tool to simulate certain parts of the system. JasViz provides basic editing capabilities as a part of the Information Storage Visualizer, as stated previously.

The Information Storage Visualizer can be used for many purposes. We have used it extensively for the simulation of unfinished or unplugged system components, such as speech recognizers and dialogue agents. We have also used it for the design of applications. Various scenarios can be designed by modifying the system outputs and inputs while the application is running. This is a powerful method when applications are designed interactively with end users. This is discussed further in the next section.

Currently the JasViz tools include basic methods for application control. More advanced methods, such as a graphical interface with bookmarks and sliders are planned. In the Jaspis² architecture the implementation of this functionality will be slightly different, because the system allows concurrent components.
6.4 Evaluation And Demonstration Tools

We have used JasViz tools as a part of our daily work when we have been building and testing applications mentioned in the previous chapter. JasViz tools have been used in all speech user interface development tasks: in the iterative design of application logic, in the design of system prompts, in the testing of interface prototypes in demonstrations, in corpus collection and in Wizard of Oz experiments. In this section I describe how Jasvis tools have been used to support these activities. First I introduce tools for Wizard of Oz experiments and describe experiences from several experiments. Then I discuss experiences of everyday development tasks.

6.4.1 Wizard Of Oz Experiment Tools

Wizard of Oz experiments are useful for speech application development, as a usability testing and data collection method, as discussed in Section 2.4. There are several tools to aid such experiments, such as SUEDE [Klemmer et al., 2000]. One particular problem with such external tools is that by using them the whole system is simulated, i.e. the application used in experiments is different from the actual system. In most cases, however, at least a partly implemented system is needed or wanted. Thus, it would be useful to run the actual system in a WOZ mode. This is the purpose of the JasViz WOZ tools.

We used the JasViz tools to set up a WOZ experiment when we tested the Doorman and AthosMail applications. The Doorman experiment setup is illustrated in Figure 34. There were two places in our premises in which the interaction took place. The first was the front door, where the system controlled access to the premises, and the second was the lobby, where multimodal guidance was given. The wizard was located in a different room, so that the true nature of the experiment was not revealed to users. The experiment is described in detail in our research article [Mäkelä et al., 2001].
CHAPTER 6: INTERACTION TECHNIQUES AND TOOLS

The Doorman system was fully functional, except that we had not finished the construction of the speech recognizer grammars and the speaker recognizer was still in development. Besides evaluating the system functionality and multimodal guiding instructions we wanted to collect data about how people speak to the system. The WOZ setup was created by replacing the missing technology components with simulated ones. This was done using a WOZ applet. There were no modifications to the Doorman components.

The experiment was successful and we were able to find many ways to improve the system. The simulated GUI interface (similar to the Mailman WOZ GUI illustrated in Figure 35) was easy to use and implement. The use of distributed WOZ GUI to control the interaction was an especially powerful feature, since the wizard was in a different room than the main application. This will be more important in the future, when we will need several wizards to take care of multiple users in different physical locations, as we have planned to do with the Doorman application.

We also used the WOZ method to test the AthosMail application. A similar setup was used, but this time the WOZ GUI was modified to enable text-entry answers and manipulation of the Information Storage. In this way complex simulated
functionality, such as information retrieval of e-mail content was possible. The resulting WOZ GUI is illustrated in Figure 35. It shows the customized GUI which allowed the wizards to react quickly for the most common operations, such as reading of messages. Otherwise the system was fully functional, and test users called the system by telephone.

As in the Doorman experiment, the simulated nature of the experiment was not revealed to the user. It surprised us that people believed that the system was real, and only few users doubted this. In this sense the WOZ setup is an ecologically valid method, but in the case of the AthosMail experiment we had other problems with the use of artificial e-mail messages, as reported in more detail in Section 5.1.

From a technical viewpoint, the wizard simulates one or more system components. For example, to carry out an experiment without a speech recognizer, as in the previously presented examples, the task of the wizard is to replace the speech recognizer. Similarly, if the dialogue module or output module is not available, the wizard simulates these missing components.

From the viewpoint of Jaspis-based systems, simulation of components consists of the manipulation of the Information Storage, since all interaction is based on
the content of the storage. Sometimes a direct control of the application is also needed. For example, it may be useful to select some specific components and skip some other components. For these reasons, the Jaspis WOZ tools are mostly customized user interfaces for the Information Storage and the application control tools presented in the previous section.

In order to be useful the WOZ interface should be easy and efficient to use, so that wizards can react fast and accurately. This means in practice that interfaces are highly application dependent. Always building user interfaces from scratch, on the other hand, is laborious. In order to help to build WOZ interfaces, JasViz includes a generic WOZ GUI, which can be customized for different purposes using an XML-based description language. This way setting up a WOZ GUI requires a definition of the interface, and no programming is necessary.

To summarize, the WOZ method is a powerful way to evaluate applications while they have not been fully implemented. In most cases it is more useful to implement the WOZ interface as a part of the actual application than to use separate WOZ tools. When the system architecture is distributed and modular it is possible to use WOZ components in a plug-and-play fashion, as shown in the two examples presented. It should be noted, however, that in applying the WOZ method tasks and other test conditions must be considered with caution, as the results from the e-mail domain show.

6.4.2 Design Sessions And Demonstrations

Iterative design with end users is another important part of speech application development, but it is often hard and time consuming to set up design sessions. Currently we lack both tools and best practices to set up such sessions.

We used JasViz tools in collaboration with target users in a design session of the Mailman application. We arranged a session with visually impaired users. The purpose of the session was to introduce the components implemented and to design speech inputs and outputs by experimenting with different alternatives.
The first use of the JasViz tools was to set up mailboxes for demonstration purposes without a network connection to the server where the actual messages were stored. This problem was solved for messages by running the system before the meeting and storing the content of the Information Storage. A snapshot of the storage was the starting point for simulated sessions. Similarly, different scenarios can be experimented with by taking a specific snapshot of the storage. Since all interaction components are stateless, it is possible to continue almost at any execution point. We have used this feature extensively for various purposes.

The design of the system outputs was done by overriding certain branches of the system Information Storage using the JasViz Information Storage Visualizer. The telephone server and the speech recognizer server were replaced by graphical components, which made it easy to experiment with different input sentences without modifying recognition grammars and touch-tone key mappings. These tools are similar to the WOZ interface presented in the previous section.

We have found it useful to conduct early user tests with partially implemented applications. This has proved to be an easy and fast way to experiment with different inputs and outputs. Application developers are not usually keen on arranging such sessions, since they tend to need a lot of extra work. We have managed to set up design sessions with JasViz tools without any modifications to the application code. This significantly speeds up the process and allows a wide range of experiments to be conducted interactively.

We have also used JasViz tools for demonstration purposes. In some setups we have simulated external connections, such as e-mail servers, in a way that is similar to the interactive design session. We have also exploited the option to modify the system Information Storage when necessary. In this way we can keep the demonstrations on track even in case of speech recognition errors.

Since Jaspis is a distributed architecture and applications may run on several machines over network connections, it is possible to use JasViz in a collaborative way between several research groups working on different aspects
of the same applications. In this way applications can be designed, tested and evaluated collaboratively from different physical locations. In addition, this could be useful for teaching purposes.

### 6.5 Support For Speech Corpora Collection

Corpus collection is crucial for speech technology and speech applications. Not only do we lack resources for base technology development, but also for application development. Spoken interaction data is needed to construct recognition grammars and language models, to study how systems are used in a real context, and to study which interaction techniques are useful and which are not. At the same time, the use of machine learning techniques increases the need for data recorded from real usage.

For a long time the focus has been on human-human corpora collection, but this is not adequate for all purposes, and in many cases it may lead the development in the wrong direction, since the communication between the computer and the human differs from human-human communication. Artificial setups, such as Wizard of Oz studies also have their own problems, as demonstrated in the previous sections.

Overall, there is a great need to collect data using implemented spoken dialogue systems in realistic settings. In order to do this efficiently, tools for collection of data, and markup languages for the annotation of the data collected are needed. So far methods, tools and annotation schemes have not been standardized, and even in large-scale research projects ad hoc methods and annotations are used. This applies especially for multimodal speech systems, and a recent example of this is the SmartKom project [Wahlster et al., 2001] and its data collection process [Türk, 2001].

In this section I present data collection methods which have been included in the Jaspis architecture and used in the development of Jaspis-based applications. I
describe how XML-based technologies can be used for the representation of all linguistic information in speech applications. I present especially how the Annotation Graphs framework [Bird & Liberman, 2001] can be adapted to the needs of speech applications.

6.5.1 Data Collection

The starting point for corpora collection is the collection of data while the system is running, either in a (partly) simulated or fully functional mode. Even if all system components are simulated by human wizards, the system should offer functionality for data collection tasks. Here we can distinguish between two kinds of data: those coming from outside the system and those produced within the system. Examples of external data include inputs given by the user and database results, such as bus timetables or e-mail messages. It is meaningful to capture both, even if it would be possible to reproduce the system data afterwards. These may be non-trivial and laborious tasks, however, if the system architecture does not support the process.

From the viewpoint of the Jaspis architecture we can distinguish between the data stored in the Information Storage and the data processed by the I/O components. The content of the Information Storage can be stored efficiently by the core system architecture, but the I/O data must be stored at component level.

There are two ways to record the whole information state of the system. The first method is to record the content of the Information Storage every time a component change occurs. It is up to the Information Storage to implement the storing method, since the architecture does not define how the storage should be implemented. Since the default implementation of the Information Storage is an XML document, the system history consists of series of XML documents. The implementation specific storing method for other than XML-based storages can likewise be used.

The second method is to record the initial state of the Information Storage and all information requests which follow. This way all information exchange between
system components can be captured, and any system state can be reconstructed by performing the captured operations in the same order.

In both of the ways mentioned we are able to capture modifications to the storage, which are made by developers when they execute applications, since the tools used to manipulate the storage (e.g. the Information Storage Visualizer) use standard methods to access the storage.

While the Information Storage is used to store all conceptual data, there is a separate file for storing internal system data, such as information about component activity and other application specific information. The logging mechanism is implemented by the Information Manager. It uses XML, but in this time in a temporal manner, which means that system events are logged as time-stamped nodes. The logfile stores information which is needed when applications are tested and debugged. This includes requests received by the Information Storage. Using these two sources it is possible to reconstruct the application execution in a temporal manner.

The use of XML has several advantages. First, it is possible to use the rich set of existing XML tools to access, visualize, modify and analyze interaction history. Second, we do not need separate formats for different purposes, and different information sources can be combined and manipulated easily. Nevertheless, there is a need to collect data in other formats as well: we cannot escape the fact that data sources such as speech signals need to be stored in their own specific format.

The Jaspis architecture encapsulates all input and output management in I/O components. Virtual devices, controlled by the Communication Manager, exchange information between the user and the system. Other managers, such as database managers, exchange information with the outside world. To collect external information Jaspis uses a mechanism whereby each of the managers using external information should maintain own archives. Different information sources are synchronized using information obtained from the Information Manager. In this way Jaspis components running on separate computers can
6.5 - SUPPORT FOR SPEECH CORPORA COLLECTION

synchronize their information. This is especially useful in pervasive applications, where different interaction devices often run in highly distributed environments. Beside corpus collection, the information collected can be used for evaluation purposes and demonstrations. Using the manager-based application control strategy it is straightforward to use the recorded information to simulate input devices, databases and other components using external information. For example, spoken utterances or devices such as speech recognizers and relational databases can be simulated by retrieving the recorded information when the Interaction Manager gives the turn to the manager which uses simulated components.

By recording and/or simulating some devices and re-running other devices in the normal way it is possible to experiment with alternative recognizers, natural language understanding modules and databases. For example, we can simulate the speech-recording device and re-run the recognizer to obtain new results from the recognizer by using already collected data, or simulate the recognizer and test new natural language or dialogue components with the collected data.

Several problems emerge when some of the system components are simulated and recorded information is used. If the simulation changes the flow of the interaction we need to record data for all possible dialogue paths. In some situations, for example when speech utterances are simulated, this is not possible. When accessing external information, such as database connections, we should record either the entire contents of the database or all information exchanged. In some cases we cannot be sure that the information will be the same when the application is executed next time. Thus, we should record the entire database content, if we want to obtain exactly the same results. If we want to use current information, we need to use recorded database queries, not stored results.

We have encountered these problems in both e-mail and timetable domains. For example, to organize and read e-mails we are dealing with information such as dates, which affect the interaction. The current date is one of the key principles
CHAPTER 6: INTERACTION TECHNIQUES AND TOOLS

when messages and their summaries are displayed to the user. In the bus
timetable domain the date and time of the interaction is the dominant factor for
most queries, since these applications mostly serve temporal information.
Similar experiences have been reported by other researchers [Glass et al., 2000].

6.5.2 Representation Of Interaction Data

One of the essential issues in corpora collection is the format of the data, and
especially the markup language used. Many different annotation schemes have
been used, and almost every major speech corpus has its own annotation
language. Well known examples include TIMIT and Switchboard. Overall,
annotation schemes are not compatible with each other. Most of the formats are
based on ad hoc notations, and each of them requires its own parser when the
corpus is used.

The use of XML technology solves the problem of different base formats by
offering a common base format. Although XML solves this basic issue, i.e., what
the overall format is for marking and exchanging data, there is still a need to
define the structure and elements of the markup, i.e. to define the markup
language for annotation of the data.

6.5.2.1 Requirements for the representation of data

There are several issues which must be taken into account when the annotation
language is defined. The first is the temporal nature of speech. Speech data is
temporal, i.e., it takes place in time. The first requirement for the markup
language is that it must include a timeline. All other elements should be
anchored in this timeline. The second requirement is that several annotation
levels should be supported. For example, phoneme level annotations and
sentence level annotations are present in speech-based communication. Although
in many cases these levels can be represented hierarchically, they may also
overlap, for example when words are mapped into concepts, the concept level
annotations do not necessarily map hierarchically into word level annotations.
The third requirement is that elements must be allowed to overlap. This is necessary when interruptions and back-channeling take place, for example. In multimodal applications different modalities may overlap in various ways.

When these requirements are considered from the viewpoint of data structures, it is clear that hierarchical structures, such as trees, are not particularly suitable because they would limit the expressive power of annotations. This is the problem in some XML-based annotation schemes because of the tree-based data structure of XML.

When other data structures are considered, the most suitable structures are graphs. They are expressive enough so that all elements of the interaction may be placed in the timeline and their relations can be efficiently expressed. Because of the temporal nature of the data there is no need for cyclic graphs, and since the relations between elements (such as sentences and words) are symmetric, graphs can be non-directed. From the computation viewpoint, acyclic non-directed graphs are a computationally strong model.

The next question is how graphs should be represented. A natural choice would be an XML-based variant of some existing graph markup language, such as XGMML (eXtensible Graph Markup and Modeling Language) [Punin & Krishnamoorthy, 2001]. The problem of this approach is that such markup languages are generic, and therefore their use can be complicated, especially when humans annotate the data. Nor do they take into account the special properties of speech-based interaction.

### 6.5.2.2 Annotation Graphs

Annotation Graphs (AG) [Bird & Liberman, 2001] are a linguistically motivated framework for the annotation of speech data. They use acyclic graphs as the structure of the annotations, and although not originally based on XML, graphs can be expressed using XML. Annotation graphs address all the requirements presented above, and are therefore suitable for the representation of speech-based interaction.
An annotation graph consists of a set of nodes and a set of edges. Nodes are called anchors, and may contain an optional label, which is a reference to a timeline. Edges are called annotations, and they contain a set of features. Features are user defined, and they are used to mark linguistic entities, such as phonemes, words or sentences. Each timeline may contain multiple graphs. Annotation Graphs have XML binding, which means that they can be exported and imported in XML format, defined by an AG DTD.

In addition to the theoretic framework, the AG framework provides a set of software components, the AG toolkit, to aid the use of annotation graphs. The heart of the toolkit is the core AG library, which provides elementary functionality, such as construction and manipulation of annotation graphs. There are also several concrete tools for manual annotations, such as the TableTrans [Bird et al., 2002], which provides a spreadsheet-like annotation interface.

6.5.2.3 Annotation Graphs in Jaspis

Annotation Graphs are targeted originally at the annotation of existing speech data, i.e., for offline annotations. We have adapted the idea for spoken dialogue systems, i.e. for the dynamic annotation of spoken interaction. Originally, we developed a similar format to be used as a general representation format in the Interact application, but when we found out the existence of annotation graphs, which had already gained international recognition, we adapted it for our purposes. Although AG is not widespread or generally used, it is still a serious attempt to create a common format for the linguistic markup of speech data.

The Jaspis architecture contains the necessary tools to use annotation graphs in applications. The support is based on the Java version of the AG library. XML representation is used to store the graphs in the Information Storage, and either the XML or graph representation can be used to manipulate graphs. Other similar representation languages, such as NITE Object Model Library [Carletta et al., 2003] can be used in addition to Annotation Graphs using converters. In
6.5 - SUPPORT FOR SPEECH CORPORATIONS COLLECTION

the future we will consider adding new extensions to support other markup languages as well.

6.5.2.4 Representation of speech recognition results

The most obvious targets for annotation graphs are speech recognition results. We modified the general speech recognition interface of the Jaspis architecture to store the speech recognition results of different recognizers in a common AG format.

In the common recognition result format n-best speech recognition results are represented as an AG set, i.e., a set of annotation graphs. Each graph contains an interpretation of the utterance at word and sentence levels, along with the confidence scores and temporal information, when available. In its most generic form a recognition result graph consists of a set of nodes, which are used as anchors, of a set of edges, which are word annotations, and of an edge, which is used as a sentence annotation. Figure 36 presents an example in XML and graphical forms. It should be noted that some elements have been omitted to make the example more readable.

In the example given there are two graphs, which are anchored to the same timeline. Each graphs has its own set of nodes and edges. It would be possible to represent multiple recognition results inside a single graph, but this would lead to a complex structure, since items of the n-best result may differ from each other. For example, the number of elements, such as words, may differ, as can their temporal information. The example illustrates such a situation, as the first result contains two words and the second result contains only one word. It would also be possible to represent recognition results in the lattice format, if the recognizer supports this functionality.
Figure 36: Annotation Graph example.
6.5 - SUPPORT FOR SPEECH CORPORA COLLECTION

6.5.2.5 Representation of multimodal inputs and outputs

The representation of speech recognition results is just one use of annotation graphs. They can also be applied to the representation of the other interaction elements in spoken dialogue systems. Other inputs, such as pointing gestures, can be similarly represented. The same applies for outputs, such as synthesized speech outputs.

Another advantage of graphs is that multimodal information can be expressed in the same way as unimodal information. This can be applied for both inputs and outputs, thereby resulting in a common representation language for multimodal interaction. When multimodal inputs are represented, different input elements can be represented inside the same annotation graph, or alternatively they can form different graphs, which can later be aligned to form a common representation, for example when the fusion of input elements is performed. Graph-based fusion is more expressive than trees, such as melting-pots in the PAC-Amodeus model [Nigay & Coutaz, 1995]. Trees are not especially suitable for the representation of overlapping elements, which are common in multimodal inputs and outputs.

When annotation graphs are used for multimodal speech outputs, such as combined speech outputs and pointing gestures in the Doorman application, different elements can be expressed so that output elements are represented as annotations inside a single graph, and are aligned to the timeline. In this way, both relations and temporal aspects of multimodal output elements can be expressed. For example, the nodes can have index and temporal information as their offsets. If an offset contains temporal information, the connected annotation (i.e., the output element) will be presented to the user at exactly that time. If the offset is not temporal, the node is used to represent relations between elements, such as that the pointing gesture will be presented after the speech output ends. This is an expressive way to represent complex multimodal outputs, such as the behavior of speaking computer characters (e.g. “talking heads”).

269
6.5.2.6 Representation of dialogue level data

In addition to the representation of single interaction elements, such as inputs or outputs, annotation graphs can be used to include different levels of information, such as syntactic and semantic information of recognized sentences inside a single graph. For example, when the utterance is recognized and represented in AG format, the parser includes its syntactic analysis as a new annotation, while the semantic processing component adds a corresponding semantic annotation, and the dialogue manager adds an annotation about the dialogue act and so on. All information is stored inside the same graph, and components can focus on those annotations which interest them.

In Figure 36 there is no temporary information available, but if the recognizer supports it, temporal information can be included in the nodes. At this point, the information is anchored on the utterance timeline instead of the global dialogue level timeline. When other temporal information is gathered from the interaction, a common timeline for the whole interaction can be made available. In this way, a dialogue between the user and the system will be represented as a single annotation graph set which consists of inputs, outputs and dialogue data. This provides a good starting point for further analysis and use of the collected interaction data.
7 JASPIS$^2$ ARCHITECTURE

The features of the Jaspis architecture have proven to be successful when applications, interaction techniques and tools have been built on top of the architecture, as described in the previous chapters. The architecture is limited especially by one feature, however: concurrent interaction is allowed only in the communication management subsystem, not in all parts of the system. This is default behavior in most spoken dialogue systems, and often seen as a desired feature. However, in new application areas, such as ubiquitous computing applications and multimodal applications, along with sophisticated spoken language applications, concurrent interaction models are needed, as presented in the case of the Doorman application.

In order to better support sophisticated speech-based systems, the Jaspis architecture has been extended in several ways. The main new feature of the Jaspis$^2$ architecture is the support for concurrent interaction in a coordinated and synchronized way. At the same time, the architecture is more blackboard oriented, and the distribution of components is extended to support sharing of components between modules. All of the changes have been made in way which makes it possible to use the adaptive features of the original architecture. In this chapter I present the new architecture. I will first discuss the challenges of pervasive application areas, and then describe the changes to the architecture. The Jaspis$^2$ architecture was originally presented in our research article [Turunen & Hakulinen, 2003].
7.1 Challenges Of New Application Areas

In mobile and ubiquitous applications dialogues may be distributed, concurrent, open-ended, dynamically constructed, and involve multiple participants in dynamic environments. The Doorman application is one example of this. Further examples of these applications and their challenges are presented in recent research articles [Lemon et al., 2001; Matsusaka et al., 2001; Stent et al., 1999]. Near-future applications will also introduce new challenges [Luperfoy et al., 1998]. At the same time, there is also a need to make speech interaction more natural in the conventional application areas. Issues such as back-channeling constitute similar challenges, as described by Allen et al. [2001b].

In Table 3 I compare conventional and novel speech applications. The key features of the novel applications are the distributed and concurrent nature of dialogues, rich multimodality, multilinguality and the active role of the computer. Furthermore, when the user interface is embedded in the environment, issues such as context awareness, privacy, non-intrusiveness and fusion of sensed and learned information are important aspects in the interaction.

In order to support spoken interaction in new application areas we need more flexible interaction methods than those based on current practices. The key issues from the architecture perspective are the distributed nature of dialogues, coordinated concurrent dialogues and system activity. At the same time, the main principles of the original architecture, adaptivity and information sharing, should be preserved. Next, these needs are discussed.
### 7.1 - CHALLENGES OF NEW APPLICATION AREAS

<table>
<thead>
<tr>
<th></th>
<th>Conventional applications</th>
<th>Novel applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialogue participants</td>
<td>single user and single computer</td>
<td>multiple users and multiple computers</td>
</tr>
<tr>
<td>environment</td>
<td>desktop and telephone</td>
<td>office and home environments, mobile settings</td>
</tr>
<tr>
<td>dialogue characteristics</td>
<td>single deterministic dialogue</td>
<td>open-ended, dynamically constructed concurrent dialogues</td>
</tr>
<tr>
<td>active participant</td>
<td>active user</td>
<td>user + (pro)active computer</td>
</tr>
<tr>
<td>dialogue control</td>
<td>centralized dialogue management</td>
<td>distributed interaction management</td>
</tr>
<tr>
<td># of modalities</td>
<td>mostly unimodal</td>
<td>mostly multimodal</td>
</tr>
<tr>
<td>fusion mechanism</td>
<td>alternative / exclusive (sequential) modalities</td>
<td>concurrent / synergistic (parallel) modalities</td>
</tr>
<tr>
<td>modalities</td>
<td>speech, text, graphics</td>
<td>speech, sensors, haptics, gestures</td>
</tr>
<tr>
<td>languages</td>
<td>monolingual</td>
<td>multilingual</td>
</tr>
<tr>
<td>interaction paradigm</td>
<td>&quot;natural” methods based on human-human interaction</td>
<td>”innovative” methods based on human-computer interaction</td>
</tr>
<tr>
<td>usability metrics</td>
<td>efficiency, user satisfaction</td>
<td>privacy, non obtrusiveness, context-awareness</td>
</tr>
<tr>
<td>information sources</td>
<td>explicit information, such as databases</td>
<td>fusion of sensed and learned information</td>
</tr>
</tbody>
</table>

Table 3: Conventional versus novel speech applications.
CHAPTER 7: JASPIS2 ARCHITECTURE

7.1.1 Distributed Dialogues

Dialogues may be distributed in many ways. First, the dialogues may take place in multiple physical places, such as in different rooms. For example, the user may interact with the system first in the kitchen, and then continue the dialogue in the living room. Second, dialogue management may be distributed over different components instead of a single dialogue management module. For example, when the user interacts with the system in his office and continues the interaction at a meeting room, it is unlikely that the dialogue will be managed by the same dialogue component. Nevertheless, the users should hear the conversation as one unified dialogue.

7.1.2 Concurrent Dialogues

In many new application areas multiple dialogues may appear at the same time. The first requirement for the system architecture is the ability to support concurrent dialogues. Since the dialogues may be related, coordination and synchronization of dialogues must take place at the architecture level. For example, in the Doorman application there can be multiple concurrent dialogues. They may be related in a logical sense, since the users are talking about the same issue (e.g. finding a person), or in a technical sense, since the resources are shared (e.g. a microphone). Without coordination and synchronization resources are likely to be wasted, and interaction level problems are likely to occur.

7.1.3 Active Systems

Most speech applications are based on the assumption that there is synchronous dialogue going on between the system and the user. The dialogue initiative may be either user initiative or system initiative, but in both cases the interaction consists of synchronous turns. In these settings the computer is passive, i.e. it is the user who starts the session.
In future scenarios the system is the active participant and serves the user. In the following two dialogue examples the interactions in passive service-oriented systems and proactive systems are contrasted. First, an example of a conventional application is presented:

**User calls a bus information service**

S: “Welcome to the bus timetable system. How may I help you?”

U: “I want to go to the hospital”

S: “Which hospital do you mean? There are three hospitals.”

U: “The northern one”

S: “What is your departure location?”

U: “The railway station”

S: “Bus number 17 leaves at 10:12”

U: “Bye”

**Time passes, no bus comes, the user calls to a taxi information service**

S: “Welcome to the taxi service. How may I help you? …”

The next example illustrates how a proactive speech system handles the same situation:

**System contacts the user**

S: “You have an appointment with your doctor at the Northern Hospital. You need to hurry to catch the bus. It leaves the central station at 10:12”

U: “Thanks”

**The system is notified that the bus is going to be late, so it contacts the user**

S: “I’m sorry, the bus is going to be late. The next tram leaves seven minutes from now.”

U: “No, I'll take a taxi”

S: “OK, I'll find a taxi for you, I'll be back soon!”

S: “The taxi is coming in 2 minutes, please wait in front of the theater building”

U: “Thanks”
CHAPTER 7: JASPIS2 ARCHITECTURE

In active systems the key aspect of the interaction is event-based communication, which means in practice that the various system components are notified depending on changes in the system knowledge base. The changes in the system knowledge base may be caused by external sources, as in the example dialogue presented.

7.1.4 Adaptation And Information Sharing

As described in the previous chapters, applications should be able to adapt their behavior to the different needs and communication patterns of users and to different environments. In new application areas this aspect is even more present. For example, multimodality and multilinguality are desired and needed features in most public services. Multimodality will also find richer forms, and various modalities can be combined with speech in innovative ways. The challenge is how highly distributed, concurrent, and proactive interaction can be adapted at the architecture level.

In pervasive computing settings shared information sources are necessary in many application areas, but their use should be considered carefully. The user's privacy, for example, should be ensured. In highly distributed systems the information sharing must also be efficient between system components.

7.1.5 Jaspis And New Challenges

In order to address the challenges mentioned, the Jaspis architecture has been extended in various ways. The distributed nature of the original architecture has been extended so that agents and evaluators can be shared between system modules (collections). Concurrent interaction is supported in a coordinated and synchronized way by using triggers and transactions. This event-based approach makes the system more blackboard-oriented and better supports active applications.
The information sharing mechanism has been extended to support access control to prevent unauthorized access to information sources. The adaptive nature of the architecture is preserved by using indirect communication between components, and by applying the adaptation mechanism similarly as in the original architecture. Next, the new features of Jaspis$^2$ are presented in more detail.

### 7.2 Jaspis$^2$ Overview

The core infrastructure of Jaspis$^2$ is based on the same principles as the original architecture, but in addition it introduces more flexibility to the interaction by supporting concurrent, event-based interaction. This is done by bringing the nature of the architecture closer to the blackboard architecture paradigm. The main changes concern information and interaction management components.

Blackboard architectures are data-driven, i.e. components (knowledge-sources) are activated by events sent by the blackboard when certain conditions are met. The original Jaspis architecture is not data-driven in this sense. The drawback of the blackboard approach is considered to be that it does not provide coordination when it is needed [Martin et al., 1999]. We address this issue by using an explicit mode for coordination and synchronization.

The new system infrastructure is illustrated in Figure 37. The Information Storage and the Information Manager components are similar to those in the original architecture, but they support additional functionality. In contrast to the original architecture, where interaction was based on turns, system activity is based on *triggers*, which react to changes in the Information Storage. The triggers may activate multiple components at the same time, and therefore interaction can be concurrent. *Transactions* are used to synchronize concurrent actions.
7.3 Triggers And Transactions

The key concept of the new architecture are triggers. They are used to provide indirect messaging between system components and enable concurrency. This is illustrated in Figure 38. All managers and agents are able to register triggers (1,4). When a trigger becomes active, the manager that has registered it is acknowledged (2,5), so that it can evaluate the situation and give its agents an option to react (3,6). It should be noted that even if an agent has registered the trigger, the local manager would be notified. This makes the system level coordination and adaptation possible as in the original architecture, i.e. managers use evaluators to choose agents when triggers are activated.

Figure 37: Overview of the Jaspis² architecture.
An example of a typical scenario follows. When new input results from I/O agents become available to the Information Storage, the Interpretation Manager will be notified. The Dialogue Manager, on the other hand, will be notified as well, so that back-channeling may take place. The Interpretation Manager chooses a suitable interpretation agent which starts the analysis of the input results. The Dialogue Manager selects an agent which produces an output request for back-channel information. The Dialogue Manager will be notified again when the interpretation of the inputs becomes available.

Triggers are based on the information in the Information Storage. When a manager registers a trigger, it gives the Information Manager a set of rules that need to be satisfied for the trigger to become active. The rules contain pointers to locations of the Information Storage and operators for these values (which can be abstract concepts). For example, the Dialogue Manager may set a rule that there must be new input results available, and that they must be interpreted. Such rules can be expressed by using Information Storage references and Boolean logic operations. The rules do not need to define all details, but instead the detailed analysis is done by the dialogue evaluators and agents.
Synchronization is done using transactions, a concept adapted from database management systems. A set of related triggers belongs to the same transaction, and is activated together. For example, in the previous example the Interpretation Manager and the Dialogue Manager triggers belong to the same transaction, and thus both of these managers are notified at the same time. The transaction mechanism makes sure that the triggers work at the correct times (e.g. that notifications are not done for uncommitted operations), and that the triggers that are closely related (i.e. active during the same transaction) are handled in a synchronized way.

Transactions consist of Information Storage addresses which define their coverage. For example, a transaction may define its coverage to be the input section of the Information Storage (i.e. the section where inputs from the user are stored). Now all triggers which are based on this information source will be activated when the transaction closes. When the transaction starts, it locks the corresponding Information Storage area, which means that other components are not able to modify it. In this way we can make sure that the Information Storage will be coherent all the time. When agents operate, they usually start a transaction, do all their processing within that transaction, and close the transaction when they end their processing.

Transactions, like triggers, are able to pinpoint multiple locations in the Information Storage. This makes it possible to refer to various kinds of information sources without blocking the whole Information Storage. This is important in concurrent applications, so that components can access the storage concurrently. Components can also register multiple transactions if they want to provide more concurrency. In this way those operations which do not affect each other can be encapsulated into their own transactions.

Another benefit of transactions is that they reduce the number of times the trigger rules need to be evaluated. If transactions were not used, the Information Manager would need to evaluate all the trigger rules every time the Information
7.3 - TRIGGERS AND TRANSACTIONS

Storage is modified. By using the transactions the trigger rules needs to be evaluated only when the transaction ends.

Triggers and transactions are stored by the Information Manager, because they can refer information found inside multiple information storage components. Local managers are notified by the Interaction Manager, however. It would be possible for the Information Manager to notify local managers directly, but because several triggers may become active during the same transaction, the Interaction Manager is used for coordination. The Interaction Manager notifies local managers according to their priorities. The use of the priority system is similar to that in the original architecture, i.e. managers with high priority will be given an option to react first. Nevertheless, managers are able to operate concurrently; only their notification methods are scheduled. With these methods they can reserve necessary shared resources before the Interaction Manager notifies other managers.

7.4 Information Access

The second major improvement to the architecture is the support for access control. This is done because methods to support privacy are deployed in many application areas, for example when the system monitors the user and his interaction with the environment and other users in ubiquitous computing applications. Access control is implemented using information access levels for different information sources. In practice this means that the Information Manager has a list which defines how components and users may access different information sources. For example, it may be defined that the user model part of the Information Storage is available only for certain components, and not available to others, such as information logging components.

Information access control can be accomplished on the basis of system components or users. When the Information Manager receives a request, it checks both the calling component and the user who is served by this
component. If either of these checks fails, i.e., the access to the information source is blocked from the calling component or its user, the request will not be served. The access rules can contain inclusive and exclusive rules, e.g., access can be granted to certain entities or blocked from certain entities. For example, the user model may be available to all components except the Log Manager, or to the user themselves and the system administrator but not for anybody else.

Another way to control information access is to use separate information storages. Multiple information storages can be supported using a directory containing locations of different information sources. For example, the directory may define that all user related information is stored separately from other information. In this way, the Information Manager knows in which storage it can find the information, and sensitive and non-sensitive information can be stored at separate physical locations. This feature also supports distribution of information, but preserves its centralized access.

### 7.5 Collections

The third main addition to the architecture is the introduction of collections. Agents and evaluators are grouped into collections which contain components related to similar tasks. For example, agents used in dialogue tasks belong to the dialogue component collection. The difference between collections and the modules used in the original architecture is that it is possible to share components between the collections, so, for example, dialogue agents may belong to more than one collection. This makes it possible to handle multiple dialogues inside the same system in a distributed way (e.g. to use separate collections for dialogues), and at the same time share a common system context and components between the dialogues. This is important in an ubiquitous system, in which we may need hundreds of concurrent dialogues and we need to minimize an explosion of object instances.
The sharing of components is done in such a way that each component has a primary collection, to which it belongs. Other managers may include this component in their own collection as well. The component is prevented from simultaneous access by multiple managers by synchronizing it just as transactions bar access to the Information Storage when they process certain parts of the storage. In this way only one manager may access the component at any given time. The purpose of the sharing is not to support simultaneous use of object instances, but to provide a way to minimize the need for multiple instances of the same components.

It would be possible to use dynamic creation of components. For example, when a component is shared, and it is accessed by multiple managers, there is an option for the creation of a new instance of the component. Now the component acts like a local one and will be disposed of when it ends its processing. In this way the peak times of the interaction, e.g. when multiple people access the system simultaneously, may be served, but the requirements for off-peak times are kept low.

The Jaspis\textsuperscript{2} architecture presented is still experimental, and many of its features are subject to change. The challenges and problems of ubiquitous speech applications, and pervasive computing applications in general, are still based more on theory than practice. Although many challenges, such as concurrency are addressed in areas such as parallel computing and databases, only practical systems will show what kind of methods are needed to build working and usable speech-based pervasive computing applications. We are developing the Jaspis\textsuperscript{2} architecture in close relation to the Doorman system in order to find how new application areas can be better supported by the system architecture.
In this dissertation the research challenges related to the development of speech applications have been addressed. The main research question is: **what kind of an architecture do advanced speech applications require?** In particular, I have addressed the challenges of **how the system architecture could support advanced interaction techniques** and **how application development could be supported by suitable models, methodology and tools**. The main contribution of this dissertation, the Jaspis architecture, addresses these questions. It is both a conceptual model and a concrete development framework. It has been used to construct practical speech applications, and it contains several tools and extension modules for various development tasks.

### 8.1 Review

Speech applications are a diverse field of research and development. Both the human-computer interaction and software development perspectives must be taken into account when speech applications are designed, developed and evaluated. For example, technology limitations, the nature of speech-based communication and interface design principles affect the design. From the development perspective, the role of different system components, their relations to each other and to user interface issues, the architectural models for organizing these components, and different dialogue management approaches need to be covered.
CHAPTER 8: CONCLUSIONS

Interface and development aspects of speech applications were discussed in Chapters 2 and 3, and in particular I presented how they are related to the system architecture. Using selected examples from existing speech systems, various approaches to these issues were reviewed, and needs for the system architecture were derived. Based on these observations, the design principles of the Jaspis architecture were formed. According to these principles, a speech system architecture must be generic, extensible, adaptive and flexible, modular and distributed, and it should support collaborative application development, reusability and standards.

Based on the design principles mentioned, I presented the Jaspis architecture in Chapter 4. I introduced both the conceptual basis and the concrete implementation of the architecture including its various subsystems and components. The human-computer interaction and development perspectives were taken into account and the architecture was compared to previous work.

The applications presented in Chapter 5 provided concrete examples of how practical speech applications can be implemented using the architecture. Applications and their user experiences have also raised many questions in human-computer interaction. Various ways to implement interaction techniques using the architecture, and in particular dialogue management techniques, were demonstrated in this chapter.

In Chapter 6 I presented general models for agent-based error handling and modular output generation. I demonstrated how the Jaspis architecture can be used for their efficient implementation. I also presented several Jaspis tools for the visualization and debugging of applications and for arranging Wizard of Oz experiments. The support for corpora collection and information representation was also presented.

Finally, in Chapter 7, I presented the challenges of emerging application areas, such as pervasive computing applications. I explained how Jaspis was extended to meet challenges and introduced the principles of the Jaspis² architecture.


8.2 Contribution

The main contribution of this dissertation is the Jaspis architecture. It addresses the challenges of advanced speech applications using novel solutions. The usefulness of the architecture is demonstrated in several Jaspis-based applications and tools. The challenges of emerging applications areas are addressed in the Jaspis² architecture. Next I present the contribution of this dissertation in more detail.

Adaptation and flexible interaction management are central themes in this dissertation. The Jaspis architecture provides a modular and distributed model through its agent – evaluator – manager paradigm. Agents are used to model interaction tasks, and tasks in general. In contrast to many other systems, Jaspis agents are compact and targeted at handling single tasks. Still, they are true software components unlike the static descriptions found, for example, in many VoiceXML systems.

The interaction model is flexible, but coordinated. The shared information management allows indirect communication between components, and overcomes the problem of lack of coordination found in many blackboard architectures. The use of standards is an essential part of modern applications development, although surprisingly many systems use ad hoc solutions. The Jaspis architecture is based on general solutions, and XML-based solutions, such as markup languages, are utilized throughout the architecture, and in particular in information management components.

Adaptation is achieved using evaluators. This offers an explicit and extensible system level adaptation. Unlike many black-box systems, it offers adaptation at the architecture level, and is available for all system components. The adaptation mechanism is general, and application specific adaptation techniques can be applied for different needs. Examples of adaptation techniques include profiles and machine learning techniques, as discussed in Chapters 4 and 5.

The communication subsystem of the Jaspis architecture provides a layered model for efficient distribution of components, and allows multiple
communication protocols to be used simultaneously. The benefits of this are demonstrated in several Jaspis-based applications, and in particular in the AthosMail, Interact and Doorman systems. The architecture also includes separation of logical and physical devices, and contains an explicit model for the interpretation of inputs, including multimodal fusion. Technology support is missing from most general architectures. The Jaspis architecture supports several speech technology interfaces, which facilitate the development of practical applications.

A development environment is needed for efficient application development. In speech application development corpus collection may be a laborious and time-consuming task. The Jaspis architecture provides support for data collection in general, and in particular it supports an existing linguistic annotation framework, which has been adapted to the needs of speech applications. In order to support testing, debugging and evaluation of applications, Jaspis contains several tools for everyday application development. Visualization and application control tools make it possible to control application execution in various ways, and modify the content of the shared Information Storage interactively. Using these general tools design and demonstration sessions with end-users can be efficiently arranged, as reported in Chapter 6.

User experiments are an important part of the application development process. A general Wizard of Oz tool included in the Jaspis architecture was used successfully in the development of the AthosMail and Doorman applications. In the Doorman application the WOZ method was successful, and yielded valuable information on how users interact with such ubiquitous speech systems, and especially how spoken multimodal guidance should be implemented. In the case of the Mailman application the Wizard of Oz approach caused several problems, and the conclusion is that the method is not particularly suitable for such domains, in which the WOZ approach makes the situation to be ecologically invalid and artificial. Still, the tool itself was found useful and the architecture suitable for conducting WOZ experiments.
8.2 - CONTRIBUTION

Multilinguality is addressed in this dissertation in various ways. One of the key features of the architecture, system level adaptation, was originally motivated by the lack of tools for constructing multilingual applications. In the Mailman and AthosMail applications speech outputs may take various forms, and multilingual content of messages poses many challenging interaction level questions. Concrete examples of their solutions were discussed in Chapter 5. In particular, Jaspis agents are used to provide common functionality for multiple languages, while language dependent agents add language specific functionality in these applications. The general modular output generation model presented in Chapter 6 takes multilinguality (among other issues such as multimodality) into account.

Most general system architectures provide elemental components only, such as components for interaction management and communication. In addition, specialized components for dialogue, input and output management are needed. As discussed in Chapter 4, each of these system modules has its own needs, but they still utilize common architectural principles. Jaspis provides both conceptual models and concrete implementations for these tasks. In addition, the agent-based error management model presented in Chapter 6 demonstrates how the architecture can be used to implement general interaction techniques in an efficient way. Still, the architecture does not force developers to follow predefined solutions, such as particular dialogue control models, as in many high-level application development frameworks.

General Jaspis agents and evaluators support reusability, alternative interaction techniques and collaborative application development. Agents and evaluators include the basis for the handling of interaction level issues, as presented in several Jaspis applications. The Mailman and AthosMail applications show how function-oriented agents can be used for dialogue management, and for adding new functionality, such as universal commands and interactive tutoring without the need to modify existing agents. In the Interact system a legacy output generator and compact presentation agents work together. It also demonstrates how Jaspis agents can be used to modify and extend system functionality as a part of the iterative development process without affecting other components.
CHAPTER 8: CONCLUSIONS

The Busman application shows how Jaspis agents implement a general concept and domain model, which can be used for input processing, dialogue management, output generation, and selection of recognition grammars. The use of shared concepts and compact agents keeps the system coherent, highly modular and distributed.

The usefulness of the Jaspis architecture has been demonstrated in many research prototypes. In addition, a Jaspis-based bus timetable service is in daily use. Further challenges are encountered, however. In the case of the Doorman application the interaction model was found to be limited when multiple simultaneous dialogues are needed in this kind of pervasive computing applications. In order to provide better support for pervasive computing applications, and more natural and flexible applications in general, a new version of the architecture was introduced to address this challenge. It introduces triggers, transactions and various other improvements in the architecture. The resulting Jaspis$^2$ architecture allows concurrency, but preserves the coordinated architecture level adaptation of the original architecture.

The Jaspis architecture is freely available, and it will be released simultaneously with this dissertation as an open source software to facilitate speech application development. An important part of future work will be to make it more accessible to other researchers and speech application developers. This work, which includes more technology interfaces, comprehensive documentation and examples, is already in progress. The work with the experimental Jaspis$^2$ architecture continues in close collaboration with the development of pervasive applications such as Doorman.
REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


