Combining Speech User Interfaces of Different Applications

Dissertation

An der Fakultät für
Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

von

Dongyi Song

Tag der Einreichung: 16.11.2006

Erster Berichterstatter:	Prof. Dr. Heinrich Hußmann
	(Ludwig-Maximilians-Universität München)
Zweiter Berichterstatter:	Prof. Dr. Dietrich Klakow
	(Universität des Saarlandes)
Tag des Rigorosums:	13.12.2006

Acknowledgements

This doctoral thesis would not have been able to reach this final stage in less than three years without the continuous support I have had from many people. I would like to express my profound gratitude to all of them.

Especially, I would like to express my deepest gratitude to Prof. Dr. Heinrich Hußmann, who is not only the supervisor and first referee of my thesis but also a good advisor in many aspects of my life. Without his belief in my skills and his numerous ideas this work would not have been realized. He gave me a great deal of invaluable advice leading me to the right way in times of doubt.

This work could not have been carried out without the wholehearted cooperation of Siemens AG.
First, I would like to thank Siemens' former Department of Communications Enterprise Systems and Department of Cooperate Technology for funding my whole research and providing me with an inspiring working atmosphere.
In particular, I want to extend my warmest thanks to my two excellent advisors in Siemens - Mr. Jürgen Trotzke and Dr. Hans-Ulrich Block.
Mr. Trotzke helped me a lot in understanding complicated communication systems, which were necessary to use as case examples for proofing the concept of this thesis. He also helped me a great deal with writing scientific papers and with many administrative aspects of the study.
Dr. Block shared his considerable knowledge about natural language processing with me and gave me much inspiration in solving different problems arising during my research. His group developed the speech dialog system – DIANE, the example dialog system used in this thesis. This dialog system is one of the essential foundations for this work. My thanks go to everyone in this group for their kind support and useful contributions in discussion and, ultimately, a well developed system.

I have to give my exceptional thanks to Prof. Dietrich Klaklow for his support as my second referee and for his good advice about improving the content of this thesis.

I also want to thank my fellow student Anne Schilling who has worked with me on this project for a very fruitful period of time. Special thanks go to Mr. Josef Lörscher for his useful books and patient explanations about lexical analyzers and compilers.

Finally I want to thank all of my friends. Their belief in me was a driving force behind my efforts. At last, I want to thank my parents for their support all through my life and their belief in me all through their lives. Without them, none of my achievements would have been possible. So thank you so much, my dearest papa and mama!

July 2006, Dongyi Song

Abstract

Recent technological advances allow for building real-time, interactive multi-modal dialog systems for a wide variety of applications ranging from information systems to communication systems interacting with backend services. To retrieve or update information from various information systems the user would have to interact – simultaneously – with different man-machine-interfaces. This will inevitably lead to a situation where a user has to interact with multiple speech dialog systems within a single thread of activity. Exposing the users to such an environment with diverse speech interfaces will result in increased cognitive load and thus bad usability. An integrated speech-enabled access layer to all available information from different applications would allow the user to access information more efficiently and easily. This dissertation proposes a novel approach for building such an integrated speech user interface for different applications by combining the existing speech user interfaces of different applications automatically or semi-automatically.

Along with solving the problem of constructing an integrated speech user interfaces for multiple applications by combining their existing speech user interfaces, this dissertation addresses different sub-themes. It first analyzes different possible architectures for combing speech applications and argues that the best way to integrate different speech applications is combining their dialog specifications.

Further, it discusses different approaches for dialog specification according to the flexibility and naturalness of the corresponding speech user interfaces, and analyzes them regarding their suitability for the purpose of combination. The frame-based approach is proposed as the most suitable one for the purpose of combination in the dissertation. However, based on a thorough investigation, this dissertation argues that combining the dialog specifications of current available frame-based systems is not feasible because they are not fully declarative. Therefore, this dissertation proposes a new frame-based dialog model for purely declarative and formal dialog specifications. This dialog model covers simple applications such as information systems, and quite sophisticated applications such as communication systems.

The emphasis of this dissertation is the proposal of a comprehensive combination scheme for combining different speech applications. Different speech applications are combined at the level of dialog specification. A unified dialog specification describing all applications is constructed (semi-)automatically, based on which a dialog manager can provide the user with transparent access to all applications. It means that the user does not need to activate any applications manually and can access any function of these applications simultaneously. Moreover, this dissertation considers different relations particularly functional and semantic overlaps between two applications; and provides a corresponding solution for all possible combination scenarios.

In particular, this dissertation proposes to recognize the functional and semantic overlaps between different applications by comparing their grammars, which serves to describe the possible natural language expressions the user can use to indicate a function or input any value for a system. In this context, this dissertation addresses the problem of comparing context-free grammars used in speech dialog systems. The theoretical backgrounds of grammars and the Chomsky grammar hierarchy are investigated and studied in depth. Further, two different comparison algorithms – finite-state modeling based comparison, and generation-parsing based comparison – are introduced and compared accordingly.

Going beyond the theoretical scheme, a prototype combination tool for constructing an integrated speech user interface has been implemented. Large industrial case studies were carried out to validate the enhanced dialog model and the combination scheme. The approach is proven to be full operational.

In summary, this dissertation provides a novel approach for constructing an integrated speech user interface automatically or semi-automatically by combining the existing speech user interfaces of different applications. By analyzing the dialog specifications of different applications, functional and semantic overlaps between the applications are recognized. The overlaps are successfully solved at the level of dialog specification so that the integrated speech user interface provides transparent access to different applications, and solves the problems of task sharing and enables information sharing among different applications.

Notations

Notation	Description
α , β	Arbitrary strings of terminal or non-terminal symbols
ε	Empty string
Σ	a finite set of terminals, referred to as alphabet
Σ^*	The set of all strings (with any length) over the alphabet Σ
$A \bigcup B$	Union of sets
$A \bigcap B$	Intersection of sets
$A \times B$	Cartesian product of sets
$A \subseteq B$	Inclusion
ϕ	Empty set
$\alpha \rightarrow ß$	Transition from α to β
$\rightarrow^*$	The transitive and reflexive closer of $\rightarrow$
$\in$	Membership
$G = (\Sigma, N, P, S)$ Σ N P S $L(G)$	4-tuple representing a context-free grammar G A finite set of terminals, referred as alphabet A finite set of non-terminals A finite set of derivation rules The start symbol The set of all strings of non-terminals derivable from the start symbol S with the derivation rules of P , referred to as the language defined by the grammar G
$FSA = (K, \Sigma, \Delta, s, F)$ K Σ s F Δ $L(FSA)$	5-tuple representing a finite-state automaton FSA A finite set of states A finite set of input alphabet Initial state $s \in K$ A set of final states Transition function mapping $K \times \Sigma$ to K The set of all string x where $\{x \mid \Delta(s, x) \in F\}$, referred to as the language accepted by the finite-state automaton FSA
$Sim(A, B)$	Stating that grammar A and B are similar
$P(G)$	Parser P checking for any string x whether $x \in L(G)$
$A \approx B$	A and B as transactions, parameters or grammars in the context of dialog modeling are similar
$A \neq B$	A and B as transactions, parameters or grammars in the context of dialog modeling are not similar

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent technological advances allow for building real-time, interactive multimodal dialog systems for a wide variety of application ranges from information systems to communication systems interacting with backend services. To retrieve or update information from various information systems the user has to interact among other man-machine interfaces (simultaneously) with speech dialog systems. This will inevitably lead to a situation where a user has to interact with multiple speech dialog systems within a single thread of activity. Exposing the users to such an environment with diverse speech interfaces will result in increased cognitive load [Pakucs, 2002] and thus bad usability [Nielsen, 1993]. An integrated speech-enabled access layer to all available information from different applications would allow the user to access information more efficiently and easily.

For the sake of better understanding this problem, let us consider a simple example. Suppose we have two separate speech user interfaces, one for air ticket reservation and the other for hotel reservation. If a user wants to book an air ticket and afterwards a hotel, he/she has to call one speech user interface after the other. An integrated speech user interface would in this case provide access to both of these two functions, so that the user could finish both tasks within the same interface. Such an integrated speech user interface has several advantages. First, the user does not have to remember or dial different numbers for different services. Second, common information such as user name, credit card information can be shared by both applications. Furthermore, speech user interfaces of different information systems and services could be integrated into a single unified speech user interface to construct a speech-accessible “smart agent”. This agent is capable of providing different information on, for example, weather, stock, train schedules, etc. and different services such as ticket reservation or hotel reservation. For example, in an integrated speech user interface for different applications, the following dialog is possible (with a little bit of imagination applied):

U: What will the weather be like tomorrow in Munich?

S: Tomorrow will be sunny, and the temperature will be 20 degrees Celsius.

U: That's great. I want to go to Berlin by train.

S: What time do you want to leave Munich?

U: 9 am.

S: Shall I reserve a ticket for you, departing at 9 am on the first of May, from Munich to Berlin?

U: Yes please.

U: I need a hotel too.

S: Do you want to book a hotel in Berlin for tomorrow?

U: Yeah, for two nights.

S: I found the hotel Ibis near the central train station in Berlin for 50 Euros per night, is this ok for you?

U: Yeah. Please book it.

S: The hotel has been booked.

U: Good, now turn off the light and wake me up tomorrow at 7 am.

S: The light is off and the wake up call has been set.

How to construct such an integrated speech user interface is the question to be answered in this dissertation. To facilitate optimal reusability, I claim and argue in this dissertation that the best way to construct such an integrated speech user interface is to combine the existing speech interfaces of different applications.

1.1 Background

A speech user interface is provided by a spoken dialog system, which allows users to interact with computer-based applications such as databases and expert systems by using natural spoken language. To enable the interactions, a spoken dialog system involves the integration of a number of components that typically provide the following functionalities [Wyard et al., 1996]:

- Speech recognition – the conversion of an input speech utterance, consisting of a sequence of acoustic-phonetic parameters, into a string of words;
- Language understanding – the analysis of this string of words with the aim of producing a meaning representation for the recognized utterance that can be used by the dialog management component;
- Dialog management – the control of the interaction between the system and the user, including the coordination of the other components of the system;
- Communication with an external system (backend application) – for example, with a database system, expert system, or other computer application;
- Response generation – the specification of the message to be output by the system;

- Speech output – the use of text-to-speech synthesis or prerecorded speech to output the system's message.

Different applications are endowed with their own speech user interfaces supported by such a spoken dialog system. With the growth of speech accessible applications, the user is exposed to an environment consisting of diverse speech user interfaces. This environment can cause increased cognitive load and thus bad usability. For example, a user wants to book a cheap hotel and there are several systems providing reservation services in different hotels. So he/she calls each system to ask the price there. In this process, the user has to remember all the numbers of the different systems and dial them one after the other. Second, the user has to write down the price information provided by each system because he/she has already forgotten the offering of the first system after his/her last call. Third, the user has to reenter the details, such as arrival and departure date, for each hotel. This process is obviously cumbersome.

An integrated speech user interface for different applications would solve this problem. With an integrated speech user interface, the systems providing reservation services can be joined together, so that the user only needs to call one number to get different offerings during the same call.

Such an integrated speech user interface across different applications requires a multi-application dialog system. A multi-application dialog system allows the user to access different applications in a dialog. There exist several architectural approaches for a multi-application dialog system such as GALAXY architecture [Seneff et al., 1999] or the multi-domain Mandarin dialog system [Lin et al., 1998]. However, the corresponding speech user interfaces are not application transparent, which means the user has to activate each application explicitly and can not access the functions of different applications simultaneously. Therefore, these speech user interfaces provide only limited flexibility.

A clear reason for limited flexibility in existing integrated speech user interfaces is that the applications are integrated as independent and/or closed units. The applications are not analyzed and integrated corresponding to their dependencies in functions and semantics. The consequence is that the user has to switch applications explicitly and cannot benefit from the advantages of an integrated speech user interface. A notable exception is the dialog-modeling approach for multi-application dialog systems provided in Bui et al. [2005], which allows the user to access different applications transparently without having to switch the domain explicitly. However, this approach does not analyze the functions and information provided by each application. As a consequence, the problems of task sharing and information sharing

are left for future work. A more detailed description about different multi-application dialog systems can be found below in Chapter 7.1.
Compared with existing integrated speech user interfaces, the goal of this dissertation is to construct a flexible speech user interface which enables transparent access to different applications simultaneously, solves different conflicts between them and allows for information sharing among them.
Nowadays applications are all developed in a distributed style, and many applications are endowed with their own speech user interfaces. To integrate these applications is one of the greatest challenges for software engineers. Distributed developed applications must not follow the same architectural and conceptual schemes and vary also in their implementations. Integrating different applications in the user interface layer is a novel idea being presented in this dissertation. Though the applications are developed independently from each other and can therefore have different conflicts with each other, by means of the integration in a unified speech access layer they will appear to be a seamlessly integrated unit to the user. How to integrate the applications, and how to recognize as well as handle the conflicts between different applications, are the main issues addressed in this thesis.

1.2 Overview

In this thesis, I design and implement a novel approach to build an integrated speech user interface by combining existing speech user interfaces automatically or semi-automatically. Based on the analyses of functions and semantics of applications, different relations between applications are declared and solved accordingly. As a result, the generated speech user interface allows the user to access different applications transparently, solves task-sharing problem in different applications and supports interoperability between different applications.
Construction of an integrated speech user interface involves the integration of different applications. There are different possible architectures for combing different applications: different speech applications can be integrated at the level of the actual backend applications or at the level of the dialog manager, which controls the interaction between the user and each application, or at the level of dialog specification, which specifies the domain specific information of each application for the dialog manager. Chapter 2 analyzes these three different possibilities and argues that the best architecture integrates different applications at the level of dialog specifications.
This dissertation seeks to construct a *natural, flexible and composable* speech user interface. There exist different dialog-modeling approaches to specify the domain

specific information of each application. Some provide a rather robust and static speech user interface; some provide a flexible and quite natural speech user interface. So in Chapter 3, I discuss these approaches according to the flexibility and naturalness of the corresponding speech user interfaces and analyze them according to their suitability for the combination issue. At the end of the chapter, I argue that the frame-based approach is the best one for this purpose.

However, it is not feasible to combine the dialog specifications of current available frame-based systems. In order to be combined with each other, a dialog specification must be specified declaratively and formally. The current frame-based dialog specifications involve more or less native coding and are not fully declarative. This is partly due to the fact that the basic frame-based dialog model is still not powerful enough. So in Chapter 4, I extend the frame-based dialog model and propose an enhanced frame-based dialog model, such that only declarative and formal dialog specifications are needed to specify an application, which can range from simple applications such as information systems to quite sophisticated applications such as communication systems. In Chapter 5, I explain how to construct a speech user interface for an application based on the enhanced frame-based dialog model.

The combination approach proposed in this dissertation applies to any frame-based dialog systems, which fulfill the requirements of being formal and declarative. Under the assumption that the speech user interfaces to be combined are all supported by a single frame-based dialog system following the requirements, in Chapter 7 I analyze the dialog specifications of each application. Different applications are compared with each other at the level of functions and semantics. In this context a problem arises - how can the dependencies between the functions and semantics of different applications be recognized? The dialog specifications, which describe the application in the aspect of speech user interface, are exploited for this purpose. I argue that the most suitable element in the dialog specification to indicate any dependencies between different applications is the grammar, which serves to describe possible natural language expressions the user can use to indicate a function or input any value for a system. Chapter 6 then discusses how two grammars can be compared with each other in order to find out their similarities, which can indicate the dependency of two functions/semantics and two applications respectively.

Based on grammar comparison, the relations between two applications can be determined once they are specified with a frame-based dialog model. In Chapter 7, several possible relations between applications are identified, and, accordingly, a solution to each relation is proposed. Going beyond a theoretical scheme, I implemented a prototype combination tool for constructing an integrated speech user interface based on the DIANE speech dialog system [Block et al., 2004]. This tool is also introduced in Chapter 7.

Chapter 8 closes the thesis with a discussion of related work, a summary of achievements and an outlook to further open research issues.

1.3 An Example

As an example let us consider two applications, hotel reservation and flight reservation.

A hotel reservation application provides the user with the possibility of reserving a hotel via speech. Since there is payment information stored for each user, the user is required to log in to the system at the beginning. Furthermore the user can check his/her credit card information.

A flight reservation application provides the user with the possibility of booking a flight ticket via speech. The user must log in to the system at the beginning and if required check his credit card information.

When combining these two applications to build an integrated speech user interface for both applications, the following overlaps arise:

1. In both applications, the user has to log in.
2. In both applications, there is credit card information, which the user can change or check via speech.
3. The destination city of the flight reservation is most likely the same as the city of the hotel reservation.
4. The arrival date of the flight is most likely the same as the check in date at the hotel.

The integrated speech user interface should not only provide both flight reservation and hotel reservation services but also solve the overlaps appropriately.

One possible solution for the task- and information-sharing issues in these two applications is that the duplicated functions will be provided only once in the user interface but will be executed by both applications; the common information will be shared in both applications.

So the integrated speech user interface acts as follows:

1. The user must log in first in the interface; thereby the user will be logged in to both systems.
2. The user can book a hotel or reserve a flight ticket in the interface.
3. If the user has reserved a ticket, the information will be reused if he/she continues on to book a hotel.

4. If the user asks for his credit card information, both systems will provide the stored information.

Such an integrated speech user interface will provide a dialog such as the following:

S: Welcome to systems for hotel and flight reservation, please enter your ID first.

U: Two four five seven.

S: Welcome, Mr. Miller! What can I do for you?

U: I want to book a flight from Berlin to Munich tomorrow.

S: What time do you want to leave tomorrow from Berlin to Munich?

U: 9 am.

S: Shall I reserve a ticket for you, tomorrow 9 am, from Munich to Berlin?

U: Yes please.

U: I need a hotel too

S: Do you want to book a hotel in Berlin for tomorrow?

(The information of the flight is reused.)

U: Yeah, for two nights.

S: Hotel Hilton for 150 Euros per night located in the centre of Berlin?

U: Yeah, that sounds good.

S: The hotel has been booked.

U: Which credit card has been taken?

S: Flight reservation system: the credit card is the Visa card with number one two three four. Hotel reservation system: the credit card is the MasterCard with number five six seven eight.

U: Ok, thanks! Good bye!

To generate such an integrated speech user interface, there are several problems to solve:

1. **At which layer are the applications integrated?**
   They can be first integrated at the core application layer, and then a speech user interface can be constructed for the integrated application; or they can be integrated at the speech interface layer. In this dissertation, I propose and argue that the best way to integrate two applications with speech user interfaces is to integrate the applications at the level of dialog specifications, which describe the application for speech user interfaces.
2. **How are the applications specified for the speech user interface?**
   There are different approaches to specify dialog specifications. I analyze different approaches and claim the most suitable one for my approach is the frame-based approach. An application can be viewed as a set of functions. Each function

corresponds to a frame in the frame-based dialog system. The required information for a frame is represented as parameters of a frame.

3. **Is it feasible to integrate two frame-based dialog specifications?**
   The dialog specification must be declarative and formal, so it is feasible to combine them. I explain in this dissertation the limitation of existing frame-based dialog-modeling approaches and propose an enhanced frame-based dialog model, in which the declarativity and formality are ensured.
4. **How can two frame-based dialog specifications be combined?**
   The dialog specifications of two applications can be integrated trivially, by merging two sets of frames. This dissertation is inspired by this idea.
5. **How can duplicated functions (like "logging in" and "checking credit card information" in the example) and shared information (like "city" and "date" in the example) be determined?**
   The comparison and dependency of combined applications are among the core themes addressed in this dissertation. I elaborate in this dissertation how to determine the dependency based on the formal specification.
6. **How can the different overlaps be handled automatically?**
   A concrete combination scheme and an interactive combination tool are introduced; the latter has been prototyped within the scope of this dissertation. Different patterns are given for different overlap scenarios.

1.4 Thesis Contributions

Along with solving the problem of combing different speech user interfaces in order to build an integrated speech user interface, this dissertation makes several contributions:

- It gives a thorough analysis of different architectures for building an integrated speech user interface.
- It provides an enhanced frame-based dialog model, which is capable of modeling different applications formally and declaratively, and serves as a preserved model for the combination issues addressed in the dissertation.
- It analyzes different scenarios for the task of combining two applications.
- It proposes a novel approach for constructing an integrated speech user interface and solves the task-sharing and information-sharing problems, which are addressed as future work in the current scientific literature.

Chapter 2

Architecture

A speech user interface is enabled by a dialog system. To build an integrated speech user interface for different applications involves first of all the integration of an appropriate dialog system and further the integration of different applications. This chapter introduces the basic architecture for a speech user interface, discusses different possible architectures for integrating applications, and, finally, proposes an appropriate architecture for building an integrated speech user interface for different applications.

2.1 Overview

An integrated speech access layer involves first of all a corresponding spoken dialog system, which enables conversation between human and machine. The spoken dialog system provides a voice portal allowing users to access information services or engage in transactions provided by backend applications via speech. From a simplified point of view such a spoken dialog system can be regarded as an integration of three main parts: the speech recognition component, the dialog management component and the speech generation component. The dialog manager integrates the other components, initiates transactions, controls the interactions between users and system and interacts with backend applications. In doing so, the dialog manager needs a description of the application, for which the speech user interface is provided. This application is usually called "backend application". Currently these descriptions (usually called dialog specification) are mostly proprietary, depending on each spoken dialog system. The emerging VoiceXML [McGlashan, et al., 2003] is a standard for development and specification of speech applications. A speech application described in VoiceXML can be interpreted by any VoiceXML-supported dialog systems. These systems are also called VoiceXML browser [Teppo & Vuorimaa, 2001]. Some dialog systems [Block, et al., 2004] can generate run-time resources for speech recognition and generation in

adequate formats automatically based on the dialog specifications of individual applications.
The following figure illustrates the simplified architecture of a spoken dialog system:

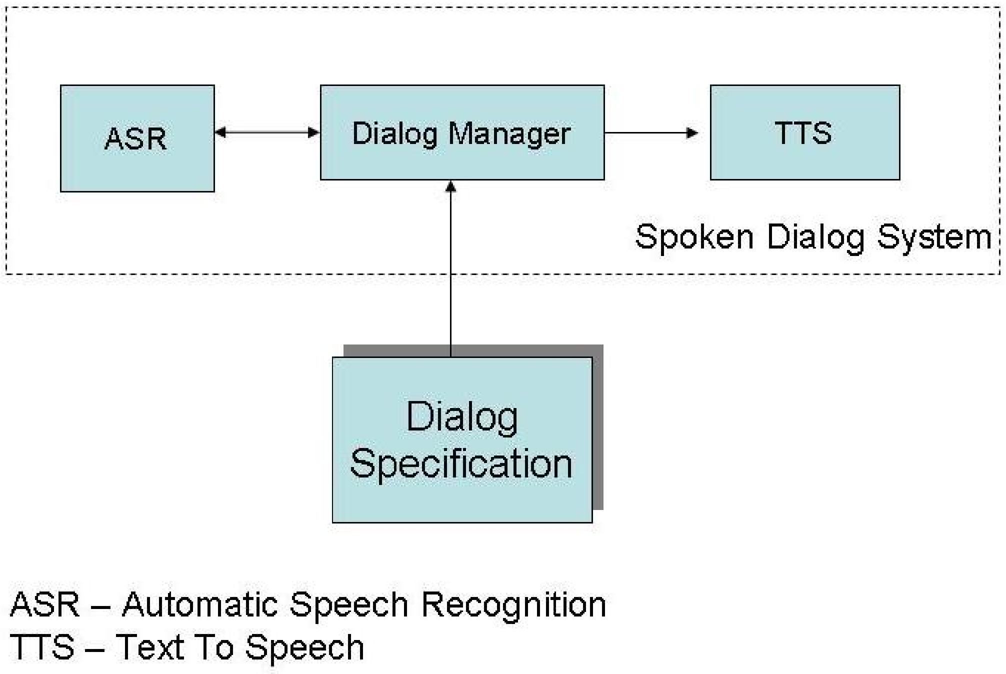


Figure 2-1 Simplified architecture of a spoken dialog system

Based on the architecture of a spoken dialog system we can see that building an integrated speech user interface for different applications means using one proper dialog manager to engage in conversations between users and different applications. Since the key purpose is the integration of different applications, there are two kinds of integration paradigms to consider. One is to use a meta-dialog manager to control the different dialog managers of underlying applications. That means each application to be integrated in the speech user interface must provide its own dialog manager, which enables the interaction between the user and this application. I call this approach “dialog manager level integration”. An example of this kind of integration is the turn-management mechanism introduced in Seneff et al. [1999]. The other possible paradigm is to use a single dialog manager based on a unified specification for all applications to enable access to different applications via speech. The core component of this paradigm is the unified specification. This specification integrates different applications, so that the dialog manager can provide one integrated speech user interface for them. There are two different ways for generating such a unified specification. One way is to integrate/combine the different dialog specifications of the underlying applications, which is called “dialog specification level integration”. The other possible approach, called “application level integration”, integrates different applications directly and generates the unified dialog specification based on the integrated application afterwards.

I will elaborate on these three different architecture approaches in the following sections and discuss the pros and cons of each approach, in particular according to the following criteria:

- Feasibility of the approach
- Extensibility of the architecture to a new application
- Interoperability of different applications – to enable different applications to interact with each other, meaning different applications can share certain information with each other
- Suitability for different kinds of applications – It is not realistic to consider all different applications together, so this thesis assumes that the integrated applications follow certain preconditions. So I will discuss each architecture according to which kind of applications the approach is particularly suited, and for which kind of applications the approach causes extra complexity
- Flexibility for dialog design of each application – whether each application can configure and personalize the dialogs according to their particular requirements
- Integration with the speech recognition component – based on dialog specifications, in particular the grammars, some speech user interface development platforms can generate the run-time recognition resources for speech recognition components automatically. Different speech recognition components support different formats. If the speech recognition resources are provided prior and proprietary before integration, it turns out to be a non-trivial problem of how these resources can be integrated. So I will compare each architecture according to the fact of whether or not it is feasible to integrate or generate the run-time recognition resources of different applications for an integrated speech access layer

According to these criteria, I will propose an adequate architecture at the end of this chapter.

2.2 Dialog Manager Level Integration

The dialog manager level integration assumes that each application is speech-enabled by an appropriate dialog manager. The integration of different applications takes place at the level of these dialog managers. A meta-dialog manager integrates and manages different dialog managers, which are responsible for the interaction between the user and the applications. The meta-dialog manager does not engage in concrete application-specific conversations, and only delegates user utterances to a corresponding sub-dialog manager. Figure 2-2 shows the architecture of this approach.

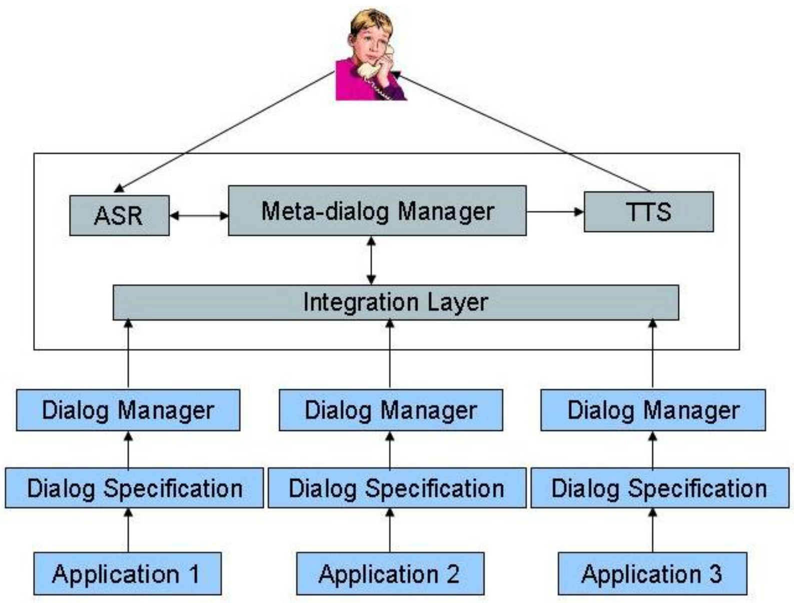


Figure 2-2 Architecture of dialog manager level integration

In such a system, the speech recognition component first transfers the user utterances into a string of words, and then the meta-dialog manager tries to interpret the words. If the meta-dialog manager is able to understand the user request, e.g. “tell me what I can do”, “which information may I ask here”, then it generates the answer and gives the feedback. Otherwise the meta-dialog manager broadcasts these words to all sub-dialog managers. If exactly one sub-dialog manager is capable of interpreting the user request, the meta-dialog manager passes the control to this dialog manager and follows up. If more sub-dialog managers can interpret the user request, the meta-dialog manager has to identify the actual task by engaging in conversations with the user. For example, a user utterance “What about Munich?” can mean “What will the weather be like in Munich?” or “How far is Munich from here?” After the task identification, the sub-dialog manager controls the dialog and the meta-dialog manager only keeps listening till the user switches the task. Then the meta-dialog manager is active again.

Pros of dialog manager level integration

A major advantage of this architecture is its plug-in feature. As long as a dialog manager implements the required interface for interacting with the meta-dialog manager, it can be plugged into the integration layer simply. In addition, the applications encapsulated by the dialog managers do not need to be analyzed for integration. So it is simple to extend the integrated speech user interface to a new application. This architecture is of particular interest for legacy applications with an integrated proprietary speech user interface.

Another advantage of this approach is that it is easy to personalize each application with a different synthesized voice, because different TTS components can be plugged to the meta-dialog manager and the underlying dialog managers.

Cons of dialog manager level integration

The difficulty of this architecture is the implementation of the meta-dialog manager. This meta-dialog manager has to control and communicate with all the underlying sub-dialog managers, and it should also be able to engage in general purpose conversations with the user.

For those applications that are developed by the same team and have no integrated speech user interface initially, this approach increases the complexity. In this case the integration could be achieved already, at a lower level such as at the application level, relatively easily; therefore only one dialog manager will suffice.

In order to be integrated into this meta- and sub-dialog manager architecture, existing dialog managers must implement certain interfaces. This means for applications with an integrated dialog manager, it is not trivial to integrate them into the integrated speech access layer. An extra interface must be implemented for integration. There exist many voice-enabled web services supported by different VoiceXML browsers. To integrate these services, the VoiceXML browsers must implement the interface as well. However, there is no standard meta-dialog manager, and a meta-dialog manager is realized rather in the integration stage. It means that different existing dialog managers or VoiceXML browsers must be re-implemented according to certain requirements and the interface of the meta-dialog manager.

To achieve interoperability between different applications would be a tedious and difficult task in this approach. First, each dialog manager could have a proprietary data format, so to achieve data format transparency for interoperability, some common repository with standard data format and some data format transmission modules for different data formats would have to be designed and integrated in the integration layer. Secondly, after delegating a task to one sub-dialog manager, the meta-dialog manager has still to be always informed by this sub-dialog manager about every successful user input, so that some information (e.g. username, account password, address) can be saved centrally and shared among different applications/sub-dialog managers.

To generate run-time resources for speech recognition automatically is not trivial in this architecture, because each proprietary dialog manager may have a different format for the grammar specification. A solution for this problem is to adopt the standard grammar specification language SRGS (Speech Recognition Grammar Standard) for grammar specification and a SRGS-compatible speech recognition component [Hunt & McGlashan, 2004].

2.3 Dialog Specification Level Integration

A dialog manager uses the dialog specification describing the task and domain of the backend application to provide a speech user interface for that application. The integration based on dialog specifications involves only one dialog manager, which enables the interactions between the user and the backend applications based on a unified dialog specification, which describes all underlying applications. This unified dialog specification will be generated in the integration layer based on the dialog specifications of different applications. This unified dialog specification describes the tasks and domains of all the underlying applications. A dialog specification can be formulated in different formats. This approach assumes that the underlying dialog specifications use the same format among a group of dialogs to be integrated. Figure 2-3 illustrates the architecture of this approach.

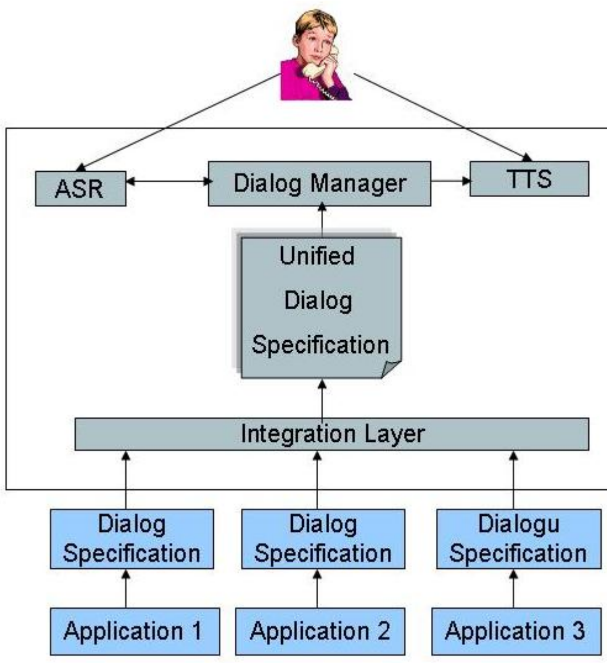


Figure 2-3 Architecture of dialog specification level integration

Pros of dialog specification level integration

It is feasible to implement this approach with moderate complexity as long as the corresponding dialog specifications can be combined without having to change the existing structures defined in each dialog specification. The integrated unified dialog specification distinguishes from a normal dialog specification only in the fact that it describes more applications. And, if more applications can be regarded as a “big application” consisting of all functions from different applications, it is natural that a portable dialog system supporting the original applications will support the “big application” as well.

Extending the integrated speech user interface to a new application requires that the application provides its own dialog specification and a new generation of the unified dialog specification. I claim and argue later in this thesis that the unified dialog specification can be generated automatically or semi-automatically, so this approach remains simple to extend to new applications.

The interoperability of different applications can be supported by the unified dialog specification, which could specify the dependencies between the applications accordingly.

This approach is particularly qualified for applications without any speech user interface where it is intended to have an integrated speech user interface. These applications need only to provide their dialog specifications for the integration in the unified speech user interface. Based on these specifications a unified dialog

specification will be generated more or less automatically. Another advantage in this architecture is that each application is itself able to configure the dialog behaviors according to its business logic and user groups. This makes it easily possible for the dialog structure of each application to remain independent of each other.
Based on the unified dialog specification, it is possible to generate the adequate run-time resources for the speech recognition component automatically. Because the speech recognition component will be integrated afterwards, it is relatively free to choose an adequate grammar specification format in the stage of dialog specification. Correspondingly, a speech recognition component supporting the format can be applied.

Cons of dialog specification level integration

Legacy applications with integrated speech interfaces may have an ambiguous separation between business logic and user interface. Thus it is not adequate to split the user interfaces and the business logic parts from each other. Therefore the legacy applications would be forced to wrap a dialog specification around the original speech interface.
Voice-enabled web services have already explicit descriptions about the dialog flow in VoiceXML. If VoiceXML is adopted for dialog specification, the integration becomes seamless. However, it is a critical point as to whether VoiceXML is suitable for the combination purpose at all. If, rather, it is not feasible to combine different VoiceXML dialog specifications, it will be time- and effort-consuming to wrap one dialog specification, which is best tailored for the integration issue, around the VoiceXML specification.

2.4 Application Level Integration

An integrated speech access layer based on application level integration involves only one dialog manager, which is able to control conversations with users based on a unified task and domain specification across all applications. The integration is based on the applications directly. Interfaces for speech access will be defined in the integration layer. Each application to be involved in the speech user interface must implement these interfaces. For example, each application must provide a list of speech-accessible functions and provide the necessary parameters for each function according to the speech interface. Supported by the speech interface, a unified dialog specification for all functions provided by different applications will be generated in the integration layer. The following figure illustrates this architecture:

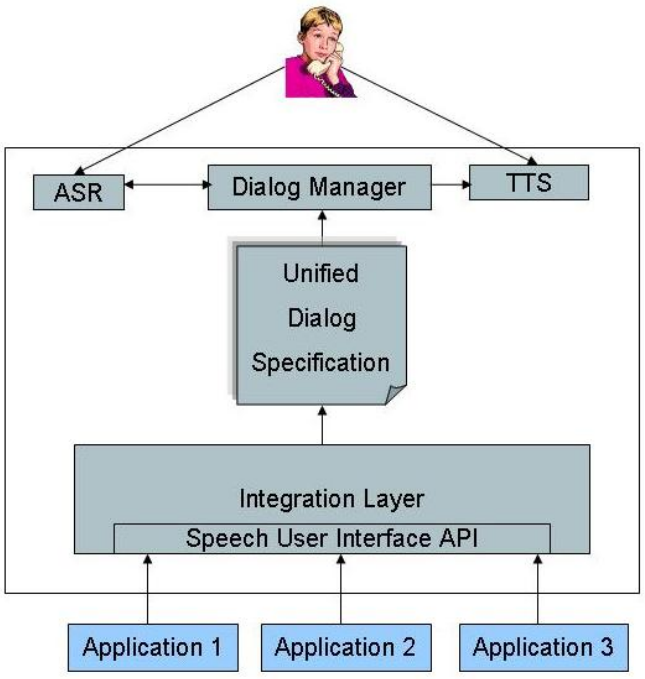


Figure 2-4 Application level integration

Pros of application level integration

This approach is independent of the dialog manager used. Based on the speech interface, a unified dialog specification could be created according to different dialog models. Thereby, it is possible to choose any appropriate spoken dialog system without affecting the integration and the applications. The development of the applications is isolated from the development of the integrated speech user interface.

The applications need not describe concrete dialog behaviors. This is the trade off of the implementation of some extra interfaces for speech access.

This architecture is simple to realize if the requirement for the dialogs between the user and the system remains low, so that it is feasible to generate a dialog specification for the dialog manager automatically, based on the application programming interface for speech user interface.

Extending the integrated speech access layer to a new application requires the interface implementation of the new application only and the new unified dialog specification can be generated automatically under the mentioned assumptions.

Data sharing should be achievable with the help of the speech interfaces without huge efforts. For example, each application can provide a list of shareable information so that the management component in the integration layer can enable the interoperability between different applications.

Cons of application level integration

The main problem of this architecture is the fact that it is critical whether a unified dialog specification can be generated automatically, based on the applications. The description about dialog issues in the application can be only very limited, and it is an open question as to whether, based on the limited description, a much more sophisticated and comprehensive dialog specification can be generated automatically. A further disadvantage of this architecture is that the applications cannot construct their individual dialog behaviors in case of need. The dialog attitudes of all different applications will be generated in the same style if at all possible. Due to this automatism, the dialog flexibility and naturalness of the integrated speech user interface will be rather limited.

This architecture is not suitable for applications which do not provide a clear application programming interface. For example, web sites cannot be integrated easily. Also, legacy applications may have no clear programming interface which can be extended with speech programming interfaces. Voice-enabled web services are hard to integrate in this solution because the speech access is already integrated in the web services, e.g. using VoiceXML standards. It is effort- and time-consuming to abstract the actual functions of these services from VoiceXML files and re-describe them by means of the speech interfaces defined in the integration layer. However if the dialog manager used is compatible with VoiceXML standards, and therefore the speech interfaces are also VoiceXML-compatible, it will be feasible to integrate voice-enabled web services in this approach with only moderate complexity.

2.5 Architectural Proposal

In summary, application level integration promises interoperability between the applications. However, it requires support from applications and thus affects the implementation of the applications. The dialog manager level integration facilitates the plug in of different speech user interfaces into the integrated speech access layer but it does not support the interoperability between different applications well. The dialog specification level integration allows for automatic generation of a unified dialog specification based on the individual dialog specifications of different applications; thus, it does not affect the core application development, and promises certain interoperability as well.

Regarding the objective of this dissertation – combining different speech user interfaces to build an integrated speech access layer which promises both usability and interoperability, the dialog specification level integration is the most suitable approach. Each application to be combined in the integrated speech user interface provides its dialog specification. All applications must specify their dialog specifications following the same dialog-modeling concept. This dialog-modeling concept is supported by the dialog manager used.

However this architecture is, of course, not applicable for legacy applications with existing speech user interfaces.

I therefore propose to build a mixed architecture of the dialog manager level integration and the dialog specification level integration. Thereby, the legacy applications will also be supported by this architecture. The following figure shows the proposed architecture.

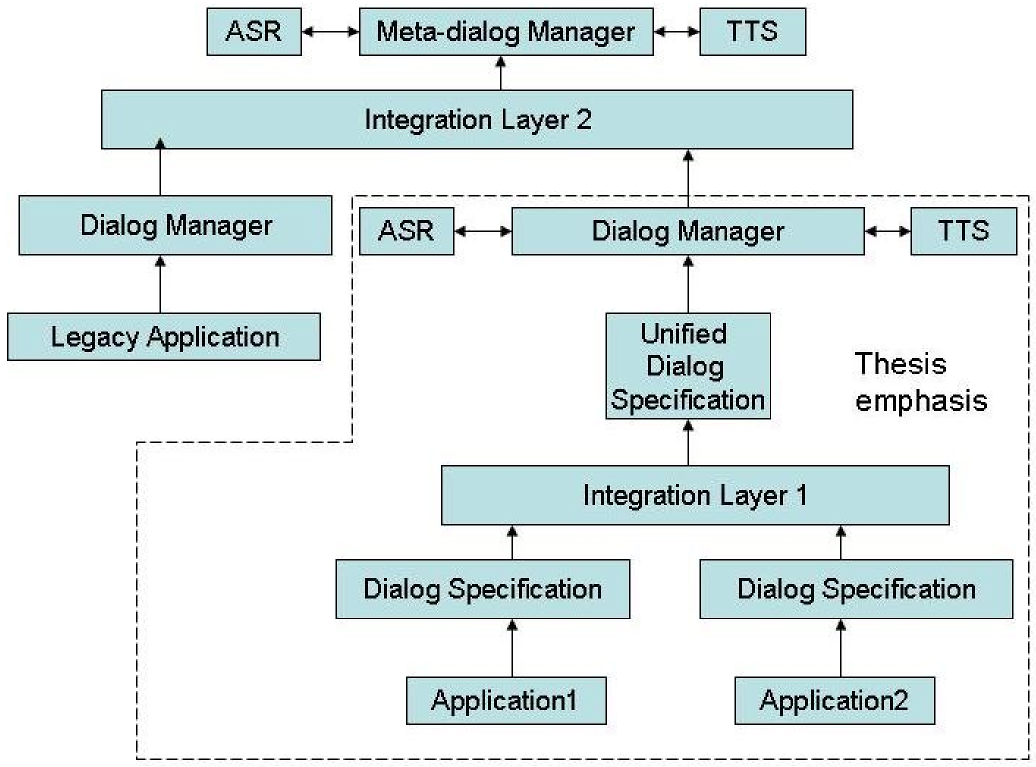


Figure 2-5 Combination architecture

This architecture contains two integration layers. The first integration layer is based on the dialog specification. Each application provides its own dialog specification, which describes the domain and tasks of the application. Based on these specifications, one unified dialog specification for all underlying applications will be generated. With this unified dialog specification, a normal dialog manager is able to provide an integrated speech user interface to different applications. The advantage of this approach is its simplicity and extensibility as well as the interoperability. With this approach, sharing of the common data can be achieved relatively easily, e.g. through a repository for all common data in the “integration layer 1” in Figure 2-5. Some applications could also have overlapped functions. In this approach, the dialog structure of each application can be analyzed, so the conflicts or redundancies in their dialogs can be solved accordingly. In Chapter 7, I will give a detailed elaboration of dialog structure analyses and solution of conflicts in dialog specifications.

For other legacy applications with an integrated speech interface provided by a proprietary dialog manager I suggest, as an extension, to build a second integration layer upon different dialog managers and to use a meta-dialog manager to control all of them. The interfaces between the meta-dialog manager and the underlying normal dialog managers will be defined in this integration layer.

This mixed architecture ensures the tight intelligent integration of related applications and also solves problems with existing legacy speech applications. It supports interoperability, consistency and scalability. The overall architecture is quite interesting but too challenging for one dissertation. Therefore in the remainder of this thesis I will focus on the first layer – dialog specification level integration – and evaluate the architectural and conceptual approach of this integration layer based on prototypical implementation.

Chapter 3

Dialog-modeling Concept

In the first integration layer of the proposed architecture, the dialog specification level integration is based on dialog specifications of different underlying applications. A unified dialog specification is generated by merging individual dialog specifications. Using this unified dialog specification a spoken dialog system provides an integrated user interface for all underlying applications. In this integration layer, it is assumed that all dialog specifications are specified with the same model. Therefore, an adequate dialog model is a precondition for dealing with the combination issue.
Traditionally there are three different types of classic dialog models [McTear, 2002]:
1. Finite-state-based dialog model
2. Frame-based dialog model
3. Agent-based dialog model

In recent years some new advanced dialog-modeling approaches are emerging, e.g. object-oriented dialog modeling [O'Neil & McTear, 2000] and information state approach to dialog modeling [Larsson et al., 2001]. In this chapter, different dialog-modeling approaches will be introduced and in particular it will be elaborated whether these dialog models are appropriate for the purpose of combination.

3.1 Classic Dialog Modeling

Each dialog model involves some elementary specifications. For example, the main element of a state-based dialog-modeling approach is state and the main element of a frame-based dialog-modeling approach is frame. The speech user interface of an application is specified as a composition of these elementary specifications. Combining different dialog specifications means combining different elementary specifications. The problem is that different applications can have overlapping elementary specifications, and the combined speech user interface should take this interoperability into account. But is the elementary specification the adequate unit for combining different speech user interfaces? Or there are some bigger clusters composed of several elementary specifications, which can be better reused in the combined speech interface? In this section I will analyze the structures of the three basic dialog models: state-based, frame-based and agent-based dialog models, and address some issues of combining different speech user interfaces based on the specification elements.

3.1.1 State-based Dialog Modeling

Basic concepts and specification elements

The state-based dialog model [Doest et al., 1996] is based on a network of dialog states and transitions. In a state-based system the user is taken through a dialog consisting of a sequence of predetermined states. Thus, a state-based system is mostly based on system initiative. The basic dialog element is the state. Each state defines possible user utterances, system prompts and next states. The system controls the dialog based on the dialog state graph. Figure 3-1 is an example of a dialog state graph. It is a bank application which contains two functions: balance query and money transfer.

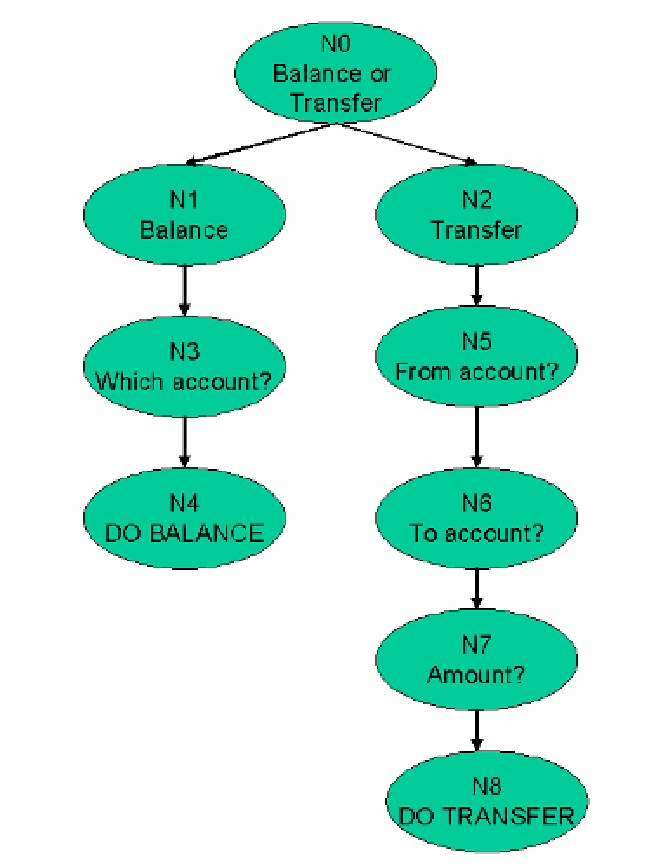


Figure 3-1 Dialog state graph of a bank application

Structure of state-based models

The basic element of a state-based model is a dialog state. The smallest units are system prompts and grammars, which constitute each dialog state. Sub dialogs can be constructed within the transition network to encapsulate several dialog states into a closed module. The UML class diagram 3-2 depicts the structure of a state-based dialog model.

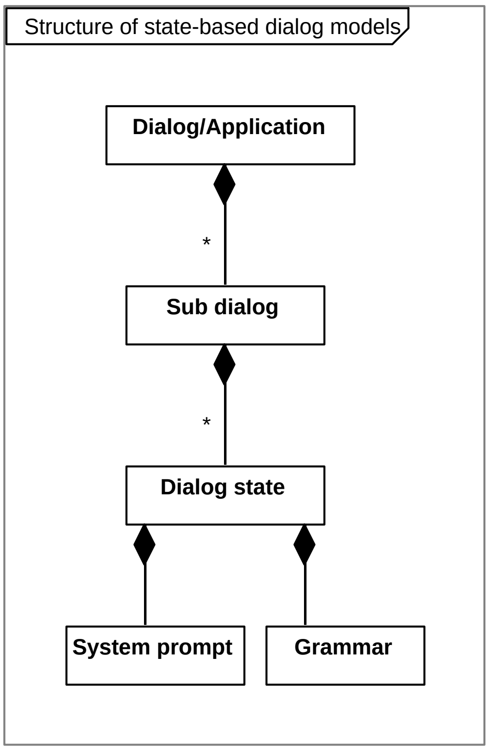


Figure 3-2 Structure of state-based dialog model

Issues of combining different state-based dialogs

To combine the speech user interfaces of different applications, different specification elements - dialog states in each dialog specification - must be reorganized into a global state transition network. The main disadvantage of the state-based dialog model is its inflexibility and bad scalability. To combine dialog states of different applications if possible into one dialog graph will cause the dialog graph to grow exponentially. Further, it is extremely time- and effort- consuming to reconstruct a new dialog graph based on the elementary dialog states of different applications. The cost for combining them will not be much smaller than that of designing a new dialog state graph from all applications from scratch.

One optimizing consideration is to build sub dialogs consisting of several dialog states. The speech user interface of an application will be specified as a network of sub dialogs. Each sub dialog carries out a sub task, e.g. checking a balance, making a money transfer. Then the combination of different speech applications will be based on sub dialogs instead of dialog states. This will simplify the combination process and improve the reusability. Currently there are some providers for reusable speech components, which are actually reusable sub dialogs. For example, the “speech objects” [SpeechObjects, 2000] and “dialog modules” [Larsson, 2001] are sub dialogs for enquiring specific data such as digit, zip code, and so on.

3.1.2 Frame-based Dialog Modeling

Basic concepts and specification elements

In a frame-based system, the user is asked questions that enable the system to fill slots in a template in order to perform a task such as providing train timetable information. In this type of systems the dialog flow is not predetermined but depends on the content of the user’s input and the information that the system has to elicit. The frame (or template) keeps account of the items for which the system requires information from users. Commonly these items are referred to as parameters. In a task-oriented dialog, the user must provide information for all parameters, before the system can execute the task. Naturally the user cannot provide all the information in one utterance, so system prompts are involved. But these prompts do not have to be made in a particular sequence. It depends on the current context which sequence is used. Sometimes the questions that the system might ask are bound to some preconditions [Austin et al., 1995], or they are associated to each parameter [Caspari, 2003], so that the system always asks for the first unknown parameter. A frame-based system supports both system-initiative and mixed-initiative modes. Because the dialog flow is not predetermined, the user is free to provide any information no matter what the system has asked for. The system does not always lead the dialog and the user is not always passive; we call this mixed-initiative.

In Allen et al. [2001], a dialog-modeling technique distinguishable from normal frame-based systems – sets of contexts – is introduced. According to Allen et al. [2001], a frame-based system consists of only one frame (or template) while with the sets of contexts more frames can be used to model a series of contexts (or a series of different tasks). Each context corresponds to one actual frame. The set of contexts can be regarded as an extension of the simple frame-based system. In this thesis, I propose to extend the definition of a frame-based system in Allen et al. [2001] to cover systems with one or more frames.

Structure of frame-based dialog models

A frame-based system consists of one or more frames. Each frame represents an individual task such as “book a hotel” or “rent a car”. A frame contains parameters representing the necessary information required for executing the task. In a simple case, a parameter encapsulates the prompt needed by the system to ask for this parameter, and the grammar defining the legal user inputs to this parameter. Sometimes there is more information associated with a parameter, e.g. the confirmation message or the help prompt for that parameter [Caspari, 2003]. Alternatively, the parameters do not encapsulate information about prompts and grammars and the dialog control is driven by rules, which define actions such as making a system prompt or executing a database query together with preconditions [Austin et al., 1995]. For example, a precondition could be that the destination city for a train trip is unknown, and the corresponding action will be prompting the user for the destination city. The following UML diagrams show the structure of both these variants.

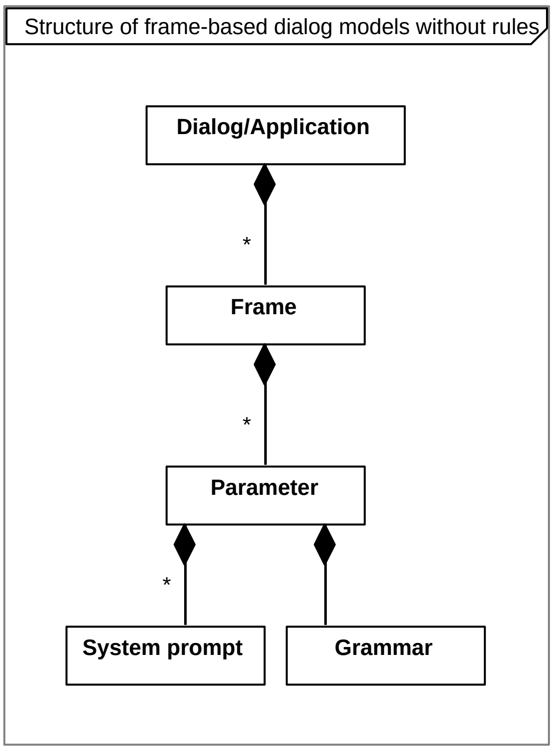


Figure 3-3 Basic structure of frame-based systems without rules

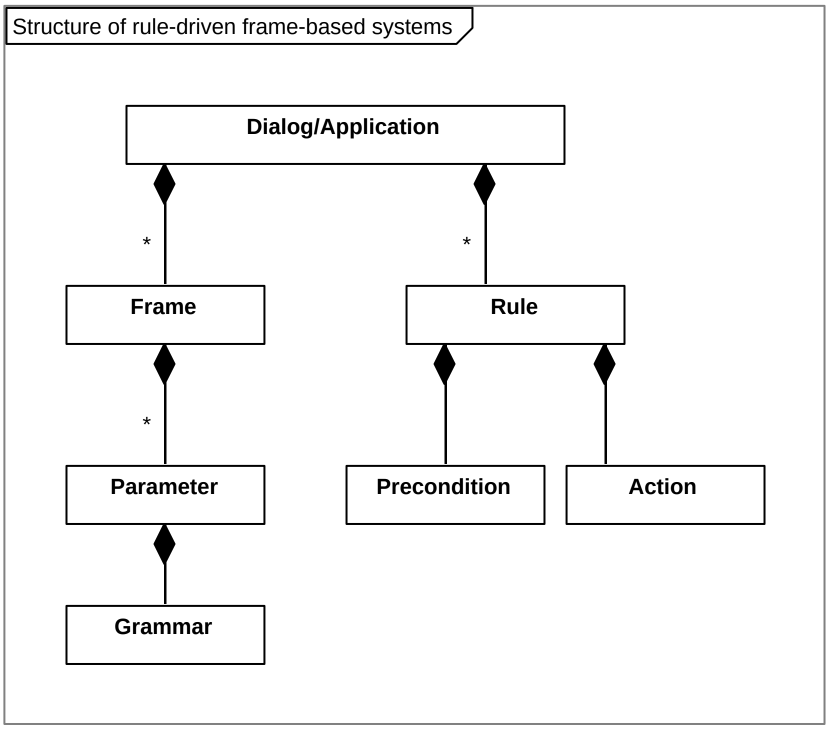


Figure 3-4 Structure of rule-driven frame-based systems

Issues of combining different frame-based dialogs

Let us first consider frame-based systems without rules. In a trivial case, the combination of different speech user interfaces means a unification of all frames. This is only applicable when the applications do not have any overlaps. When the applications overlap with each other, the problem becomes sophisticated. The frames of different applications can be identical or partially overlapping. If the frames are identical, which means they represent the same task, they are to be merged into one. If the frames are only partially overlapping, their parameters have to be analyzed and adapted with regard to interoperability issues.

The combination problem is more difficult with rule-driven frame-based systems. Not only the frames but also the rules are to be combined. To merge different rules, it is necessary to check their preconditions and actions. To check if two rules have the same preconditions, the comparison of parameters involved in the rules is required.

Frame-based dialog systems with or without rules achieve similar flexibility and effects in the dialogs between the user and the applications. In a dialog system without rules, many preconditions and the associated actions are integrated in the dialog management component, so that the applications do not have to specify them externally. Therefore, I consider the first variant (without rules) for frame-based dialog system as a better and more advanced one and I will refer only to this kind of frame-based dialog systems in the following.

3.1.3 Agent-based Dialog Modeling – Plan-based Dialog Modeling

Agent-based systems are designed to permit complex communications between the system, the user and the underlying application in order to solve some problem or task. There are many variants of agent-based systems, depending on what particular aspects of intelligent behavior are included in the system. In agent-based systems communication is viewed as interaction between two agents, each of which is capable of reasoning about its own actions and beliefs, and sometimes also about the actions and beliefs of the other agent. According to McTear [2002], agent-based systems include systems using theorem proving, planning, distributed architectures and conversational agents. I do not consider all these classes in this thesis. First, it goes beyond the scope of the current work; secondly, the agent-based dialog systems are mostly still at the research laboratory stage and not mature enough for combination. Heuristically, I discuss, in the following section, the most developed and widespread agent-based system – the plan-based dialog-modeling approach.

Basic concepts and specification elements of plan-based dialog models

The key idea of plan-based approaches is the modeling of utterances as speech acts [Traum et al., 2003]. Speech acts are fundamental communicative units. A plan to achieve a goal using language would typically involve chaining together a series of such speech acts, including acts representing the intentions and communicative actions of another agent. A plan-based system recognizes the speech acts behind the user utterance, thus the actual goal of the user. Then the system statically find or dynamically constructs its own plan consisting of different speech acts to achieve the goal according to the beliefs, obligations and intentions of both interlocutors. Based on the plans the system takes actions such as responses to the user or queries to the database.

Structure of plan-based dialog models

The plan-based dialog model is based on plans. Each plan is specified by speech acts. The plan can be constructed dynamically in the discourse. To realize the plan construction, there has to be some specifications of desired behaviors of the system including the actions it will typically be asked to perform, what obligations it has, and a specification of how to perform actions. These specifications are regarded as the domain-specific information within a plan-based system [Allen et al., 2000]. They are normally specified by some logical rules representing the basic states and behaviors of the backend application.

Issues of combining different plan-based systems

It is an open question at the moment whether it is achievable to combine different speech user interfaces modeled using plan-based approaches. First, there are few application independent plan-based spoken dialog systems. Most of the plan-based spoken dialog systems are still ad hoc designed and implemented for specific applications. A generic dialog shell proposed by Allen et al. [2000] is based on domain specifications. The specifications are represented by some logical rules. Combining different logical rules is a challenging and complicated task.

3.2 Advanced Dialog Modeling

Based on the basic concept of dialog modeling, there are some novel approaches for bringing advanced software engineering concepts such as object orientation into the development of a dialog system. They aim to clarify the relationship between different functionalities and the corporation of different components in a dialog system. The proposed dialog systems strive to be more domain-transparent and intelligent. In this section I discuss some advanced concepts for dialog modeling, particularly in regard to the issues concerning combination of different speech user interfaces modeled with these concepts.

3.2.1 Object-oriented Dialog Modeling – Inheritance

Inheritance of dialog states

Randall Sparks [Sparks et al., 1994] proposed an object-oriented dialog-modeling concept based on a hierarchy of dialog states. There is an inheritance relationship between general and domain-specific dialog states. The dialog is represented by a dialog plan consisting of different dialog states. This approach has been adopted in the implementation of a prototype for a dialog-based driving information service, called “voice navigation”, which gives users driving directions. The following pictures illustrate the dialog states' inheritance; a dialog state graph called “dialog plan” describes the dialog flow consisting of different dialog states.

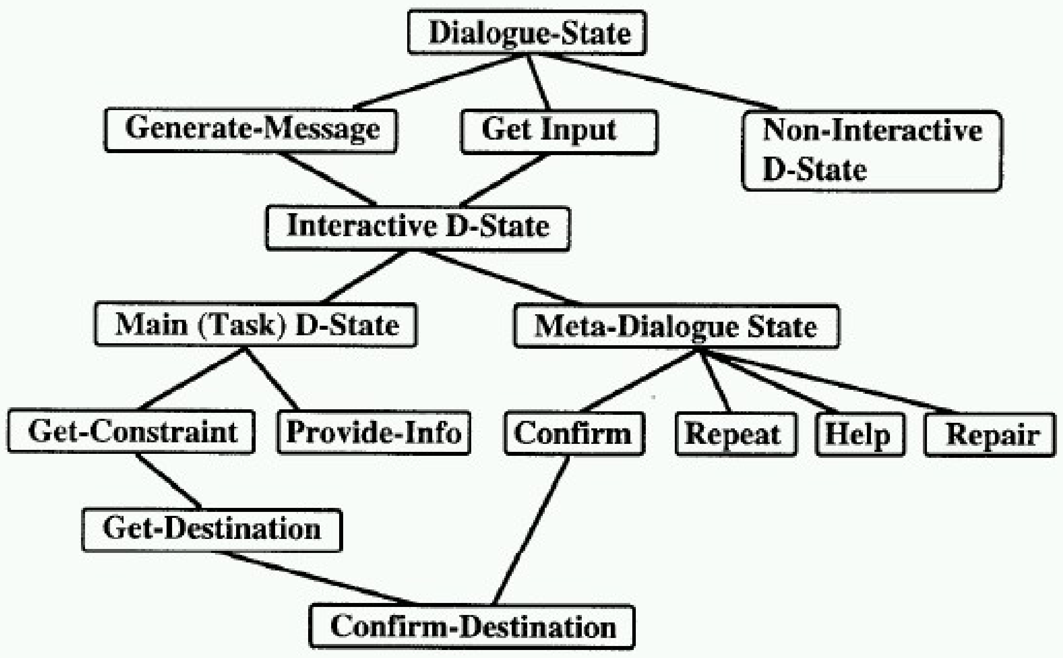


Figure 2. Part of the Dialogue State Inheritance Hierarchy.

Figure 3-5 Dialog state hierarchy of a spoken dialog system

Source: Sparks et al. [1994]

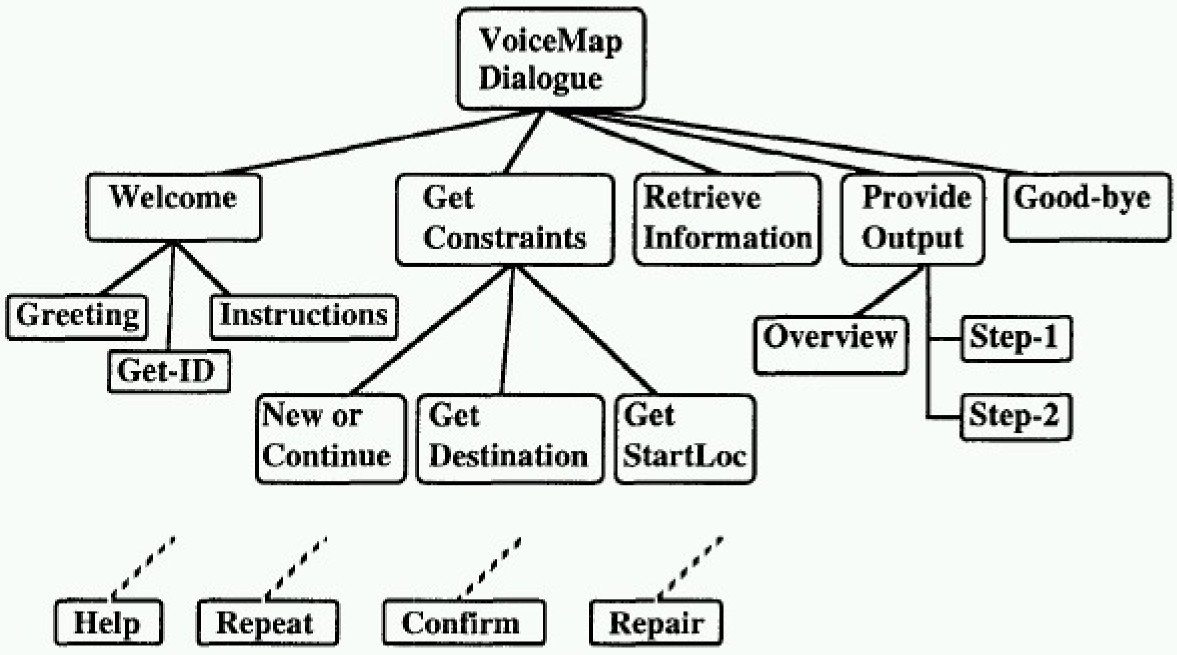


Figure 4. The default dialogue plan.

Figure 3-6 Dialog plan for a voice navigation system

Source: Sparks et al. [1994]

This approach can be regarded as a combination of state-based and plan-based approaches. The basic specification element is dialog state. Instead of explicitly specifying the dialog flow via a transition network, the dialog control is implicitly described by a dialog plan. Besides the properties defined by each dialog state, each dialog state inherits common communicative abilities and properties from its parent states. Based on the availability of general dialog states, the complexity for building a new speech user interface for an application is essentially reduced.

Inheritance of dialog components

Ian M. O'Neil and Michael F. McTear [O'Neil & McTear, 2000] introduced an approach to object-oriented modeling of spoken language dialog systems. They use object-oriented modeling techniques in the creation of spoken dialog management systems. The key idea is the separation and relationships of generic and domain-specific components in the dialog management systems. Classic UML diagrams are used to elaborate the relationships and corporations of different components.

As illustrated in Figure 3-7 use case diagrams are used to document the behavior users expect from the systems and help to understand the relationships between the system's main areas of functionality. The diagram represents the interactions between different groups of users and a dialog system providing travel and event information, which are regarded as the specialized use cases of certain domains. In the system, high-level functionality is abstracted to form generalized use cases such as "Message I/O" for transferring the messages between the user and the system, "Manage Discourse" for general dialog management process such as confirmation and repeating, "Log discourse" for storing and recalling the semantic content of the system's and the user's utterances and "Identify domain" for identifying the right expertise which matches the user's intention. Using the "generalization-specialization" relationship in use case diagrams, the domain specific "travel enquiry" und "event enquiry" use cases inherit all necessary functionality from the general use case "Manage Discourse".

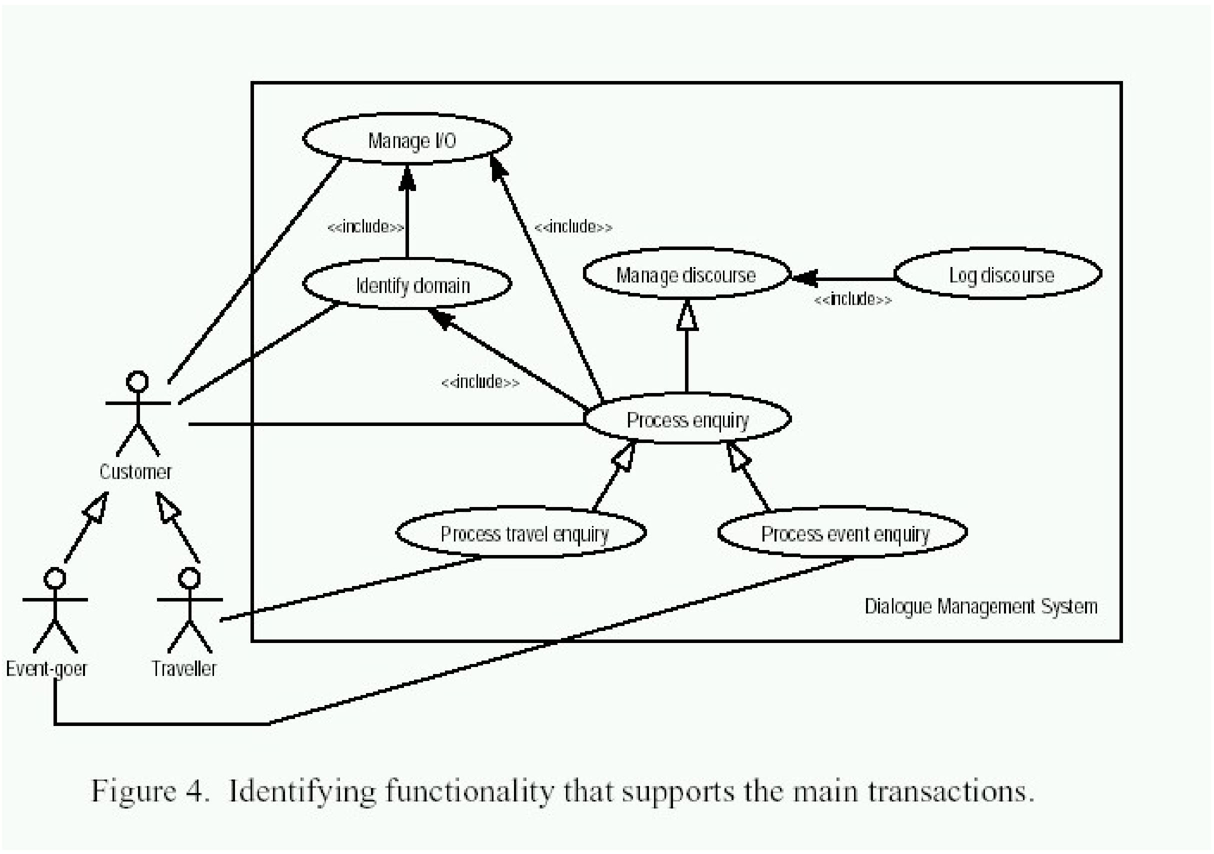


Figure 3-7 Use case diagrams for a spoken dialog system

Source: O'Neil and McTear [2000]

Furthermore, the class diagram helps to understand the implementation of a spoken dialog system with object-oriented techniques. Based on the use case diagram, the classes for implementation can be constructed easily. Figure 3-8 gives an example for a class diagram according to the use case diagram in Figure 3-7.

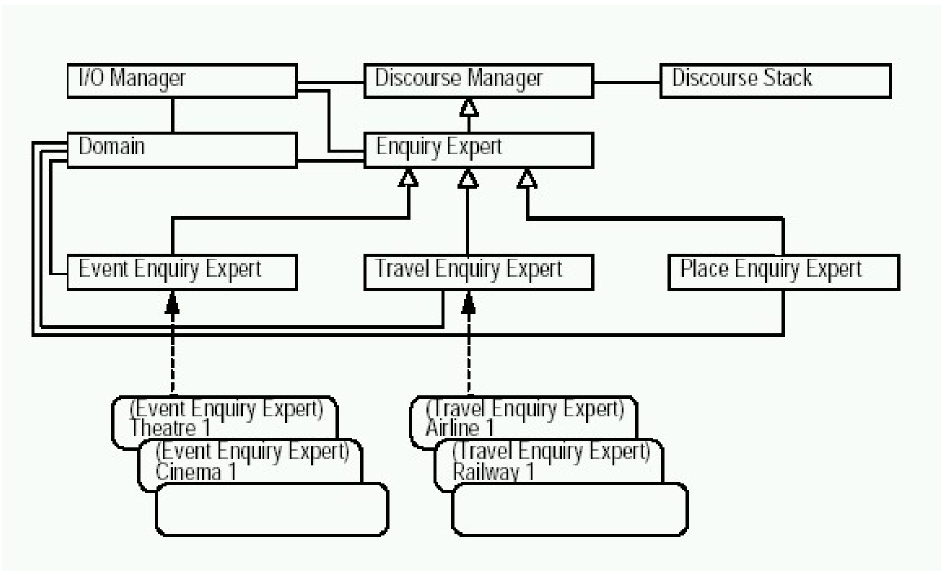

I/O Manager
Discourse Manager
Discourse Stack
Domain
Enquiry Expert
Event Enquiry Expert
Travel Enquiry Expert
Place Enquiry Expert
(Event Enquiry Expert) Theatre 1
(Event Enquiry Expert) Cinema 1
(Travel Enquiry Expert) Airline 1
(Travel Enquiry Expert) Railway 1

Figure 5. Main system components.

Figure 3-8 Spoken dialog system components hierarchy

Source: O'Neil and McTear [2000]

Last but not least, sequence diagrams help to model the interactions of different components involved in a transaction. Figure 3-9 illustrates the interaction and invocation process of different components for the situation where the user's utterance indicates only that he/she wants to make a booking and gives no further details. First the domain spotter for domain identification is programmed to interrogate the "Enquiry Expert" sub classes to find out which one can handle bookings. The "Enquiry Expert" forwards the analyze request to its sub classes again, so the ones which can handle booking response to the domain spotter, so the dialog would continue with a clarification between "travel booking" and "event booking".

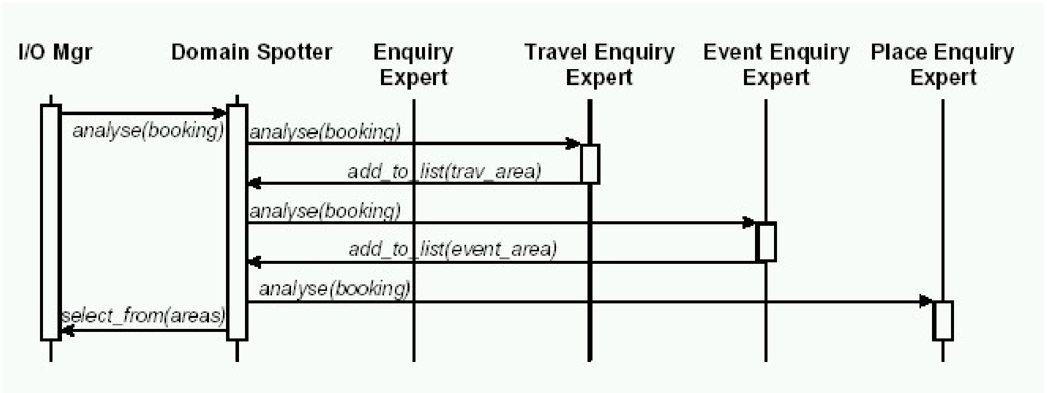


Figure 7. Finding the relevant subclass.

Figure 3-9 A example sequence diagram

Source: O'Neil and McTear [2000]

The object-oriented techniques help to analyze the functionality and interaction of different components. The modeling of dialogs is still frame-based. The behaviors of the systems are described by different rules. In a class hierarchy a component is able to inherit the general rules from its parents and in the mean time define its own domain-specific rules.

Combining such systems together is similar to combining different frame-based dialog systems, which has been described in Section 3.1.3.

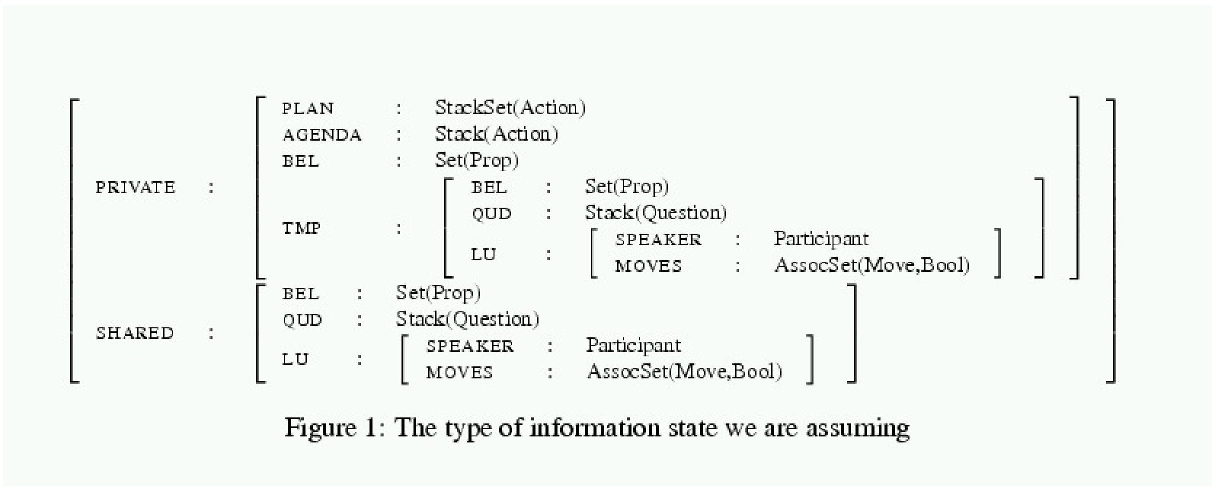
3.2.2 Information-state Approach to Dialog Modeling

Information states [Traum et al., 1999b] represent the information available to a dialog participant at a given stage of the dialog. The “information state” of a dialog represents the information necessary to distinguish it from other dialogs, representing cumulative additions from previous actions in the dialog, and motivation for future action. Every utterance in the dialog leads to one or more information state updates. This approach models the interactions in terms of information state updates. An information state theory of dialog modeling consists of the following components:

- A description of the informational components of the information state
- Formal representations of the informational components
- A set of dialog moves that will trigger the update of the information state.
- A set of update rules, which govern the updating of the information state, given various conditions of the current information state and performed dialog moves. Some of these rules will also select particular dialog moves for the system to perform.
- A control strategy for deciding which rules to select at a given point from the set of applicable ones.

It is important to distinguish the information state approach from the well-known structural dialog state approach. While the state-based approach describes the information implicitly in the state itself and the relationships it plays to other states in the transition network, the information state approach describes the information of the participants declaratively and explicitly. Furthermore a state-based approach can only model a finite set of states and transitions in a transition network. There is no finiteness restriction on information states and the motivations for updating and making dialog move may rely on only a part of the information available in a information state, which is different from the state-based approach where the whole information of a state is needed for making a transition decision.

The information state approach is somehow similar to a plan-based approach, since the mental notions such beliefs, intentions and plans may but must not necessarily be included in the information state. It distinguishes itself from a classical plan-based approach, because it also includes aspects of dialog states and specifies the updates of the information state explicitly with update and selection rules.
The following is an example of the information state purposed in the TRINDI project [Traum et al., 1999b]:

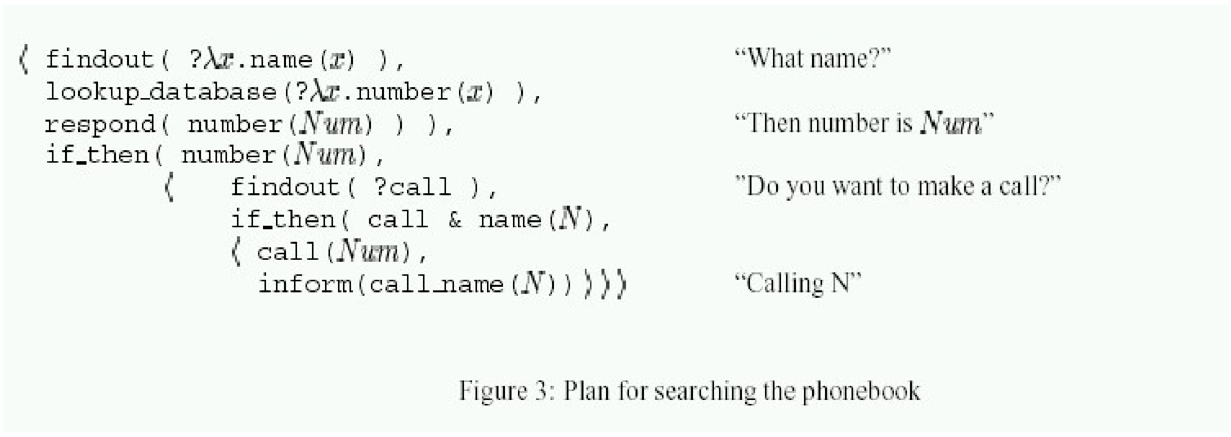


$$
\left[\begin{array}{ll}
\text{PRIVATE} : & \left[\begin{array}{lll}
\text{PLAN} & : & \text{StackSet(Action)} \\
\text{AGENDA} & : & \text{Stack(Action)} \\
\text{BEL} & : & \text{Set(Prop)} \\
\text{TMP} & : & \left[\begin{array}{lll}
\text{BEL} & : & \text{Set(Prop)} \\
\text{QUD} & : & \text{Stack(Question)} \\
\text{LU} & : & \left[\begin{array}{lll}
\text{SPEAKER} & : & \text{Participant} \\
\text{MOVES} & : & \text{AssocSet(Move,Bool)}
\end{array}\right]
\end{array}\right]
\end{array}\right] \\
\text{SHARED} : & \left[\begin{array}{lll}
\text{BEL} & : & \text{Set(Prop)} \\
\text{QUD} & : & \text{Stack(Question)} \\
\text{LU} & : & \left[\begin{array}{lll}
\text{SPEAKER} & : & \text{Participant} \\
\text{MOVES} & : & \text{AssocSet(Move,Bool)}
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Figure 1: The type of information state we are assuming

Figure 3-10 Information state used in TRINDKIT
Source: Larsson et al. [2001]

GoDiS [Larsson et al., 2001] is an experimental dialog system implemented using TRINDKIT [Larsson & Traum, 2000], which is a toolkit for building and experimenting with dialog move engines and information states. The following figure shows the dialog plan used for representing the domain in the GoDiS system.



```
⟨ findout( ?λx.name(x) ),                        "What name?"
  lookup_database(?λx.number(x) ),
  respond( number(Num) ) ),                      "Then number is Num"
  if_then( number(Num),
          ⟨    findout( ?call ),                  "Do you want to make a call?"
               if_then( call & name(N),
               ⟨ call(Num),
                 inform(call_name(N)) ⟩⟩⟩        "Calling N"
```

Figure 3: Plan for searching the phonebook

Figure 3-11 Dialog plan for function “searching the phonebook”
Source: Larsson et al. [2001]

The domain of an application is described by several such dialog plans. Each plan consists of rules specifying the correct behavior of the system. We can see from Figure 3-11 that the dialog plan describes the flow of dialogs with logical rules.
Combining two speech-enabled user interfaces modeled with information state approach means combining the dialog plans of an application with those of the other. This will be problematic because the plan is not specified in a declarative and or task-oriented way. It rather specifies a dialog flow explicitly. Combining such plans is similar to combining transition networks of state-based dialog systems, which has been discussed in Section 3.1.2. Therefore, this approach is not particularly interesting to use for the combination issue.

3.3 Summary

In this chapter different dialog-modeling concepts have been introduced and analyzed according to their suitability for combining different dialog specifications. The state-based approach is inflexible and not scalable. As a result, the state-based approach is not recommended for the purpose of combination. The agent-based approach is based on logical rules, and combining logical rules is a very difficult task. The frame-based approach enables flexible and free dialogs and facilitates the combination of two different dialog specifications. The advanced dialog-modeling approaches address the implementation issues and the reusability and flexibility of the dialog manager, and do not actually provide new fundamental dialog models.
As a consequence, concerning the combination objective I propose to use the frame-based approach for the dialog modeling and aim further in this dissertation to combine different dialog specifications based on frame-based dialog models.

Chapter 4

Extended Frame-based Dialog Model

As it was shown in the previous chapter, the frame-based approach to speech dialog systems is the one which is best suited for the purpose of combination. In a frame-based approach, the algorithm for the dialog management is defined and implemented in the dialog manager. The domain information needed for accessing the individual application is specified in a set of frames. Each frame corresponds to a task provided by the application. The additional information required to achieve the task is modeled as parameters of the frame. The deployment of a frame-based spoken dialog system for a specific application requires only the declarative description of that application. Thus the combination of different speech applications means the merging of their frames, taking into due consideration their concurrency and interoperability.

However, not all applications can be modeled by existing frame-based approaches. The applications which are covered by the frame-based approaches are mostly only simple information-providing applications. The integrated speech user interface proposed in this thesis aims to cover as many different applications as possible. Therefore, there should be no evident restrictions for the applications, which can be combined into the integrated speech user interface. So, as a first step, I have to examine the limitations of current frame-based approaches, and propose in this chapter an enhanced frame-based approach, which covers most applications ranging from simple to sophisticated and from stateless to stateful applications.

In what follows, I will first introduce frame-based approaches in brief and give an overview of current frame-based dialog systems. Afterwards I will analyze different restrictions of existing frame-based approaches for modeling certain sophisticated applications. Then I will propose an enhanced frame-based dialog model to solve the restrictions. I will provide different application scenarios to show how the enhanced model can be applied. At the end of the chapter, I will compare the features of the enhanced frame-based dialog model with different existing frame-based dialog models and summarize the chapter.

4.1 Frame-based Dialog Models – State of the Art

Frame-based approaches are based on the slot-filling concept. Slots are containers for the information that must be elicited from the user. Slots are stored in structures called frames. A frame normally corresponds to a task which can be executed by the actual backend application. Started from single-frame dialog system, the up-to-date dialog systems support normally multiple frames [Caspari, 2003; McGlashan et al., 2003].

In a frame-based dialog system, the dialog flow is controlled by the dialog manager, which is independent of the concrete application logic. Therefore, it is relatively simple to deploy a frame-based dialog system to a new application. Normally only the dialog specification for the application must be provided.

However, different frame-based systems provide different dialog models, which are differently powerful in modeling different applications. Also, the required dialog specifications vary in their declarativity. In the following, I offer a survey of several representative frame-based dialog systems:

One of the first frame-based dialog systems was the Philips train timetable system [Aust et al., 1995]. This system uses a frame consisting of slots to specify the information needed to query a train timetable. The questions that the system might ask are listed together with the conditions under which that question should be asked. The conditions normally check whether some slots are filled by the user utterance or not. In this way, the dialog system manages the interactions with the user. Such a system is restricted to only one frame and does not consider the dependencies between different slots of the frame or the influences from any backend application with its dialog model.

In the Mercury flight reservation system [Seneff & Polifroni, 2000], the dialog manager manipulates linguistic and world knowledge represented in the form of semantic frames. In each dialog, it begins with an E-form representing the constraints of the current query as a set of (key: value) pairs. This E-form provides the initial values in the dialog state and evolves over the course of the turn. The dialogs are controlled by a set of rules, which defines actions to different conditions. For example, a rule for prompting source of a flight is defined as follows:

!source → prompt_source

Though the dependencies between different slots can not be specified explicitly, they can be handled by rules operatively. It is an advantage of this dialog system that the

designer can construct a more complex system by carefully designing different rules. Moreover, this system considers the dialog history so it is able to fulfil a part of the E-form based on its memory of the dialogs with the user. However, whether or not it is able to use the dialog history explicitly in rules is not described.
A similar concept of E-form is used in the GALAXY II project for dialog control in the turn manager [Seneff et al., 1999].

Thompson and Bliss [2000] proposed a concept to nest frame descriptions to enable reusability and flexibility. They extend the basic frame concept by allowing frames to be hierarchically structured in each other. A frame is not only a set of slots to be filled, it contains both domain information and dialog information (such as grammars, prompts and goals). This kind of extension to basic frame-based model makes the dialog specification more transparent and more modular, so it is easier to build a new dialog system.

Bohus and Rudnicky [2003] developed a sophisticated task hierarchy for dialog management based on the basic frame. They propose to describe a domain as a tree consisting of sub trees, where the leaves of the tree are comparable to a frame. Each node in the tree has a set of preconditions to be activated; triggers to fire it and a completion criterion for finish it. Figure 4-1 shows such a tree:

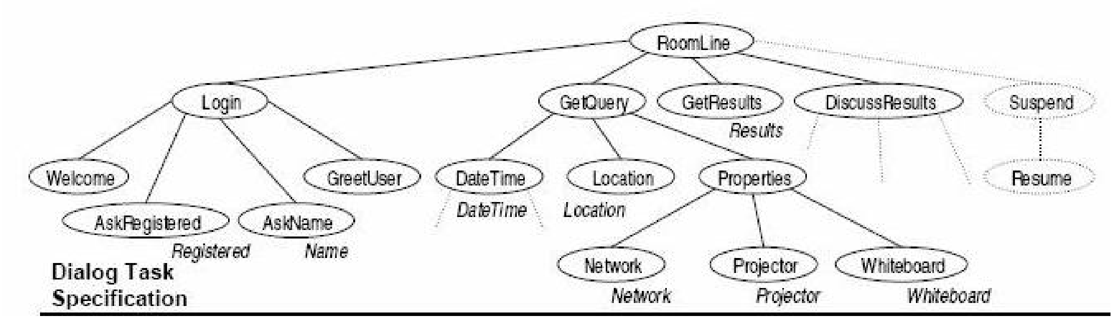


Figure 4-1 Task tree specification
Source: Bohus and Rudnicky [2003]

By extending the frame-based model with a hierarchical representation, the relations between different sub trees (frames) can be specified directly and intuitively.
As an open standard dialog specification language for speech applications, VoiceXML [McGlashan et al., 2003] has recently been widely used in speech applications. A VoiceXML speech application is a set of related documents. Each document defines a part of the dialog flow. A document is a finite-state machine consisting of dialogs and transitions to other documents (URIs). The building blocks

of the dialog in the VoiceXML documents are forms and menus. A form is a dialog containing fields (slots to be filled by the user) while a menu is a dialog containing choices. VoiceXML supports sub dialogs and both external and embedded speech recognition grammars.

VoiceXML actually uses both state machines and the frame-based model in its underlying dialog control model. A speech application consisting of multiple documents is modeled as a global state machine. Each document constitutes a state in this global state machine. Such a document can be regarded as a state machine and may consist of different forms which are comparable to the frames introduced earlier in this chapter.

However, VoiceXML is not best suited for the combination issue of this dissertation due to the following reasons:

- VoiceXML uses a mixture of state-based and frame-based models. This causes its document to contain both declarative specifications such as “form”, “field” and structural dialog flow specifications such as “goto”, “submit”.
- A VoiceXML dialog specification is not task-oriented, but rather dialog-flow-oriented. This is more like a state-based dialog system, which is not suitable for the combination purpose. (See Chapter 3.1.1)
- A VoiceXML application is by default system-directed. To enable a mixed initiative dialog, the developer must specify extra form elements.

In summary, VoiceXML does not provide a purely declarative frame-based dialog specification, which is required for the purpose of combination.

DIANE (DIAlog maschiNE) [Caspari, 2003] is a frame-based dialog system developed by Siemens CT.

In DIANE, an application is modeled as a set of transactions. A transaction is similar to a frame. Each transaction corresponds to a function provided by the backend application. The necessary information required for the execution of each transaction is modeled as parameters of the transaction. For instance, the transaction “train ticket reservation” needs information about departure, destination, date and time of the itinerary. This information will all be modeled as parameters of the transaction “train ticket reservation”. Each parameter defines its own grammars and prompts for query or confirmation. Further, each transaction has trigger grammars, which define words that indicate the transaction directly, thus distinguishing one transaction from another. For instance, a transaction for a train ticket reservation may be triggered by the words “book ticket”, “ticket reservation”, etc. These words will be defined in a trigger grammar.

Besides these specification elements – transaction, parameter, grammar, prompt, etc., in order to describe dependencies between parameters and influences from the backend application, program codes such as Java classes can be used. DIANE provides two standard interfaces for such Java code. With these interfaces the dialog designer can implement the required callback functions to specify inference rules, consistency conditions, the repair mechanism and invoke the corresponding functions in the backend applications,. These factors influence the dialog flow; however, they must be programmed in Java code and – in the DIANE model before the new extensions described below - they cannot be described declaratively like the other elements of the DIANE dialog model.

Figure 4-2 illustrates an overview of DIANE specification elements, depicted as white boxes, and Java classes for describing inference rules and execution commands, depicted as grey boxes.

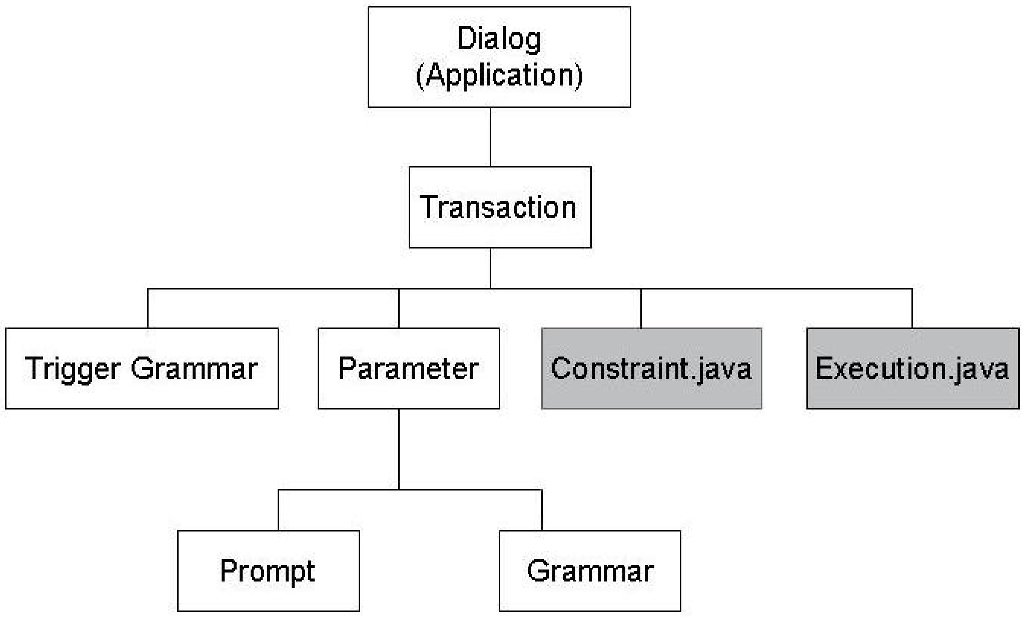


Figure 4-2 Dialog model of DIANE

The DIANE dialog management allows for mixed-initiative dialogs. The mixed-initiative strategy is a user-initiative strategy with system-initiated error handling. The user dominates in the dialog; he/she determines the task to be executed and provides the required information for the execution. The system only takes the initiative in asking about the unknown parameters when the user has not provided all information or when there is an ambiguity in the user utterances, so the system must initiate a clarification sub dialog.

Most systems introduced above are mostly similar due to their frame-based nature. Some systems use rules for dialog controlling, and some pack the dialog information within the frame. The RavenClow system [Bohus & Rudnicky, 2003] is a little different because it mixes a graphical tree mechanism with a frame-based scheme. VoiceXML differs from the others because it mixes a state machine with frames. I find DIANE to be an outstanding frame-based dialog system. Its dialog model including the mixed-initiative strategy can represent most frame-based dialog systems, whereas it uniquely provides Java callback interfaces. With these interfaces, DIANE allows the application to define their sophisticated dependencies between parameters, and between user interface and backend applications respectively. Therefore, I use DIANE as the example dialog system for the discussion of frame-based dialog systems in this dissertation.

The features provided by VoiceXML, the GALAXY II dialog management component and DIANE are compared in Chapter 4.6.

4.2 Limitations of Frame-based Dialog Models

Most frame-based approaches are best tailored for stateless applications such as information seeking. The dialog flow is determined by the central dialog control algorithm based on the presence of required information of a task. The system action is either asking for an unknown parameter or calling an operation of the backend application. Many logical dependencies and other influences in more sophisticated applications are outside this scope and cannot be described declaratively within the existing frame-based models. In DIANE, this problem is partly solved by the Java classes – constraint and execution.

The constraint class serves to check the consistency of different parameter values and to determine the value of an unknown parameter (inference of a parameter) based on the constellation of available parameter values. For example, for a flight reservation, the departure city and the destination city must be different. Such a condition is a kind of consistency check. And, given the departure time and the duration of the flight, the arrival time can be computed automatically. Such a process is called inference. The following code is an example for a section in the corresponding constraint class:

```
Public class constraint{

  /* check the parameter values according to different conditions and
```

```
   return the inconsistent parameter.
*/
public String getFirstInconsistentParam(Kontext kontext,
                                        Interpreter interpreter){

  // get the parameter values of destination city and departure
  // city
  String dest_city = kontext.getParamSem("DEST_CITY");
  String dep_city = kontext.getParamSem("DEP_CITY");

  //check if the destination city and the departure city are the
  //same. In this case, report the parameter “destination city” as
  //an inconsistent parameter.
  if(dest_city.equals(dep_city)){
           return “DEST_CITY”;
  }
  …
 }

/* return the error message prompted by the system in case of
   inconsistence.
*/
public String getInconsistencyMessage(Kontext kontext,
                                         Interpreter interpreter){
   String param = getFirstInconsistentParam(kontext, interpreter);
   if (param.equals(„DEST_CITY“)){
       return “sorry, the destination city can not be the same as
               the departure city.”;
   }
   ……
 }

/* infer unknown parameter values based on existing parameter
   constellation.
 */
 public Kontext infer(Kontext kontext, Interpreter interpreter) {

    String dep_time = kontext.getParamSem("DEP_TIME");
    String dest_city = kontext.getParamSem("DEST_CITY");
    String dep_city = kontext.getParamSem("DEP_CITY");
```

```
        String duration = computeDuration(dest_city, dep_city);
        String arr_time = computeArrivalTime(dep_time, duration);
        // set the parameter “ARR_TIME” to the given value “arr_time”.
        kontext.setParamInferred("ARR_TIME",arr_time);
        ……
    }
}
```

The execution class serves to invoke the corresponding function in the backend application and to return an appropriate message to inform the user about the result of the execution. For example, the following Java code illustrates a section from the execution class of a transaction providing weather information:

```
public class Execution{
    public String getMessage (Kontext kontext,
                              Interpreter interpreter){
       String date = kontext.getParamSem("DATE");
       String city = kontext.getParamSem("CITY");

       // get the information from the backend application
       String weather = backend.getWeatherInformation(city, date);
       return weather;
    }
}
```

These two classes handle some influences from the backend applications and some dependencies between the parameters. There are yet other dependencies and backend influences not covered by these classes. In what follows I elaborate on various dependencies and influence factors existing in different applications, which are not covered by the current frame-based dialog model, except that a part of them is supported by Java callback function in DIANE already. However, this section seeks to give an overview of all dependencies and provide a foundation for an extended dialog model which is capable of representing these factors declaratively.

4.2.1 Transaction Dependencies

In some multi-task applications, there are dependencies between different transactions. An analysis of different applications shows that there exist two kinds of dependency - conditional dependency and contextual relation.

Conditional dependencies

A conditional dependency between different transactions means that the successful completion of a transaction is a precondition for the execution of another transaction. For instance, in many applications user authorization is required. The user has to log in first, before he/she can access any other functions provided by these applications. Between the function login and the other functions, there is a conditional dependency. Only if the function login has been executed successfully, the other functions are activated for user access. Such a situation is suitable rather for a state-based dialog model, where all the other functions can be defined as successor states of the login function. Such a stateful philosophy is not included in the frame-based models, where all frames are equally activated in different stages of a dialog.

Contextual relation

In a multi-task application, it is normal that the user expresses his “command” according to the current context. Therefore, a group of transactions belonging to the same context should be accessible in a series without requiring the context information for each transaction every time. The comparison between the current dialog context and the context information of a transaction gives an implicit hint for the transaction identification. In a dialog of a stateful application, this is an important factor, which can be considered before the system prompts to clarify the right transaction in case of ambiguity. Consider the following scenario: The user first says that he/she wants to listen to the next email. After that, he/she only says, “delete” to indicate that he/she wants to delete that mail. In the current frame-based approaches, there is no context modeling. So if the system understands that the user says “delete”, and there are two tasks “delete email” and “delete appointment” associated with the keyword “delete”, then the system will not know, although it is obvious in this case, that the user means “delete email”. The action the system usually takes is to initiate a clarification dialog to find out which task the user means exactly. The available context information is not exploited yet.

4.2.2 Parameter Dependencies

Besides the dependencies between different transactions, there are also dependencies between parameters of a transaction. In frame-based approaches, the parameters may be given by the user or may be inferred by the system. The inference is supported by the Java callback functions in DIANE, but cannot be described declaratively. Other frame-based approaches provide only limited support for the inference between parameters. For example, in VoiceXML a default value can

be provided for a parameter. In the Mercury flight system [Seneff & Polifroni, 2000], a rule’s condition can refer to more parameters so that a rule more or less present a relation of these parameters implicitly.
An analysis of different systems shows that there are two kinds of parameter dependencies which influence the dialog flow.

Conditional dependencies

A conditional dependency between different parameters means that the value of a parameter determines whether the other parameter must have a value or not. For instance, in an application for train ticket reservation, the user is first asked if he/she wants to make a seat reservation. If yes then he/she is asked if he/she wants to sit at the window or aisle. If the user does not wish to reserve a specific seat, the parameter for seat position will have to be left empty. In this case, the user input as to whether he/she wants to reserve a seat determines whether the seat position parameter should be asked for or not.
Such a dependency is resolved by the inference mechanism in DIANE. The inference is defined in Java code. There is still no declarative description method for such a dependency.

Logical dependencies

Logical dependency between parameters is more complicated than conditional dependency. In a logical dependency, a parameter can directly infer the value of the other parameter. For example, in an application for meeting room reservation there are usually parameters for start time, end time and duration. Of course it suffices if the user provides information for any two of these three parameters. The duration is equal to the end time minus the start time. So given the availability of any two parameters, the third parameter can be inferred automatically.
Such a dependency is also resolved by the inference Java class in DIANE.

4.2.3 Influences of the Backend System

The backend system is the application that actually executes the tasks. In the information-seeking system, the backend system is a database. In other cases, the backend system can be a Java stand-alone application or an electronic appliance, etc. In the current frame-based approaches, the interactions between the dialog manager and the backend system are restricted to task execution at the end of the transaction. However, states of the backend system at runtime of the dialog may be crucial for the interaction and thus influence the dialog flow.

Influence of the backend system on the active transactions

The states of the backend application can influence the set of accessible transactions. For instance, an event ticket reservation system may only provide a reservation service at certain times such as from 10 am to 6 pm. At other times, the system only provides information for different events, and the user cannot book any ticket. Here the time, which is managed in the backend system and can be regarded as a state representation of the backend system, influences the available transactions to the user.

Validation of parameter values

All possible values of a parameter are defined by its grammar. However, not all values are valid in different situations. The validity of a parameter value may depend on the state of the backend system. Let us consider the following scenario: in a communication system the user can send messages to his/her contact. Each contact may have an SMS, email and fax address. The contact does not have to have all of these addresses. So if the user tells the dialog system that he/she wants to send a message to a contact, the dialog system has to interact with the backend system to be informed by the backend system about which addresses are available for this contact. Then the dialog system can ask the user for the ways (SMS, email or fax) to send the message accordingly or validate the user input to check if the corresponding address is saved in the backend system. Whether the parameter indicating the way to send a message is valid is dependent on which addresses are saved to that contact in the backend system. So only with help of the backend system is it possible to verify the parameter.

Information on system state

In current frame-based approaches the system assistance for task completion is very limited and in fact is restricted to error handling. In a more sophisticated application it is sometimes necessary to assist the user during the completing of a transaction. One possibility is that the system should inform the user about its current state to facilitate the user completing the task. For example, in an application for event ticket reservation, there is a function for cancelling existing reservations. In this function, if the user just says that he/she wants to cancel a reservation, the system can assist the user to complete the cancellation by telling him/her his/her current reservations, which ones can be cancelled, and maybe also which cannot. Such an action for information on system status is a kind of system-initiated action, which is related to a certain dialog state. Such a system action is not included in the existing frame-based dialog models.

4.3 Extensions of Frame-based Dialog-modeling Approach

A very important feature of the frame-based approach is the declarative description of the applications and a common dialog control algorithm used by the dialog manager. This feature facilitates the specification of a new application and increases the portability of a generic spoken dialog system. Currently the common dialog control algorithm mainly seeks to complete the frame by prompting the user for unknown parameters. For more sophisticated applications, this dialog control algorithm does not suffice. Besides prompting the user for unknown information, managing dependencies between different transactions and parameters and providing system assistance during the dialog should be supported as well. Different factors influencing the dialog flow have been elaborated in the last section.

There are two ways to solve this problem:

One way is to separate the data structure for storing the dialog states and the specification of dialog policy [Bohus & Rudnicky, 2003]. This solution proposes to manage the dialog state in a form-like structure, so that over informative user utterances are supported in the same way as normal frame-based approaches. Instead of using a general dialog policy, each application can specify its own dialog policy in different ways. Some use rules with preconditions involving the dialog states and actions that determine the next dialog step [Seneff, 1997; Traum & Larsson, 2003]. Others use transition networks to specify the dialog policy [Lemon et al., 2001; Catizone et al., 2003; Wu et al., 2002]. The latter solution mixes the advantages of a frame-based approach and a finite-state-based approach. However, for each application the dialog policy has to be specified from scratch. This inhibits the portability of spoken dialog systems and increases the complexity of creating a new speech user interface. The use of transition networks may lead to combinatorial explosion in the case of sophisticated applications.

An alternative solution is to extend the common dialog control algorithm and the data structure, such that various dialog control mechanisms are supported. As far as the investigation within the scope of this dissertation has shown, this second solution has not yet been adopted and explored in academic research.

My goal is to use the frame-based approach to specify different applications, so that their dialog specifications can be combined together to build an integrated speech user interface with minimal effort. So a comprehensive spoken dialog manager is required, which abstracts the common dialog policy. In addition, declarative dialog specifications are necessary, so that it is feasible to combine different applications with only moderate complexity. For this purpose, I follow the second route and extend the dialog model and the general dialog control algorithms respectively.

In this section I elaborate on an enhancement proposal for frame-based dialog models. The DIANE dialog model is taken as a reference example of frame-based dialog models, and the enhancements are explained based on the DIANE model.

Figure 4-2 illustrates the dialog model used in DIANE, in the following Figure 4-3 illustrates the extended dialog model. All black boxes are extended specification elements.

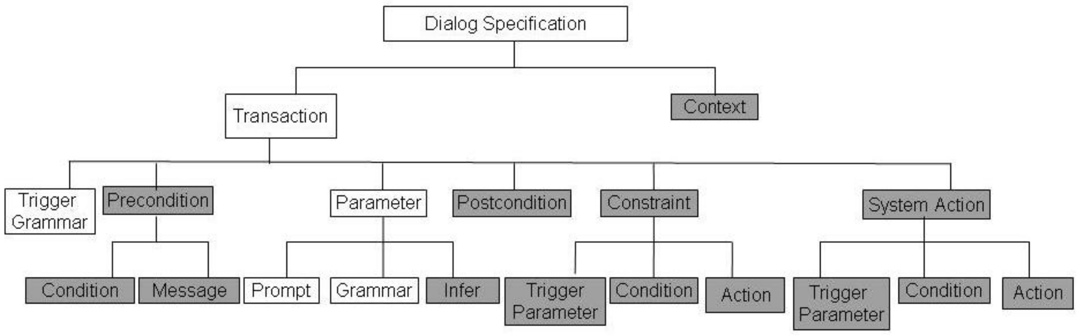


Figure 4-3 Enhanced frame-based dialog model

In the following all extended specification elements are explained in detail.

Precondition

Pre- and post-condition-style reasoning are frequently used approaches for web services composition [Sheshagiri et al., 2003; Traum & Larsson, 2003]. The dependencies between different web services and their sequential or parallel relationships are described by pre- and post-conditions. I adapt this idea to the frame-based approaches and introduce a specification element **Precondition** to each transaction. Before a transaction can be accessed by the user, its preconditions must be checked and have to be fulfilled. Each precondition consists of two sub

elements – **condition** and **message**. The **condition** expresses the requirement for accessing the transaction such as that the time is between 8 am and 6 pm. The **message** specifies the prompt to be uttered by the system, if the evaluation of the **condition** results in false. For example, a message could be "sorry, you can't book a ticket right now, the service will be accessible in short time. "
The following form gives a precise definition for precondition:

$Precondition = < Condition, Message >$

$Condition$ is a term according to a defined syntax (refered as CON - SYNTAX)

$Message \in \Sigma *$

$\Sigma = \{A, \ldots Z, a, \ldots z, 0, 1, \ldots. 9\}$

Postcondition

Each transaction provided by the speech user interface corresponds to a function in the backend applications. At the end of a transaction, when all necessary parameters have their values the corresponding function should be invoked. In many existing frame-based dialog systems, the invocation of the functions in the backend application is directly coded in the dialog management system [Austin et al. 1995]. Thus the spoken dialog systems are proprietary and cannot be adapted to any application without modification in the core of the spoken dialog system. In DIANE, the invocation of the functions in the backend application is coded in a Java class. For a purely declarative specification of applications, I introduce here a specification element – postcondition. This element serves to specify the execution of the transaction in the backend application and the subsequent information prompt to communicate the result to users. With this element the general dialog policy is freed to specify how each transaction should be executed, so that the spoken dialog system remains general enough to be able to be applied to any application.
The following form gives a precise definition for postcondition:

$Postcondition$ is a term according to a defined syntax (refered as ACTION - SYNTAX)

Context

Users are used to specify a task with reference to the currently discussed context. In a multiple task application, different tasks can be grouped together and assigned to the same context. For instance, "read email" and "delete email" can be grouped together and assigned to the same context "email manipulation". In frame-based approaches this context information is underspecified. I introduce a specification element, **Context,** to represent the contextual relation between different transactions.

The context is an element of the application. Each application can have one or more contexts, and each context consists of a set of transactions, which are referred to via their unique names.

The following form gives a precise definition for context:

$CONTEXT = \{TransactionID_1, ..., TransactionID_n\}$ $TransactionID_i \in \Sigma^*$

Parameter Infer

For each parameter, I introduce further the possibility of adding an **Infer** element. This element defines different ways to obtain the value of a parameter. In most existing frame-based approaches the only way to obtain the value of a parameter is to prompt the user. In more sophisticated applications, the value of a parameter can be inferred based on the current dialog state or by the backend application. With the **Infer** element, different ways of obtaining the value of a parameter can be specified declaratively in a logical form. More precisely,

$Infer$ is a term according to a CON - SYNTAX.

Constraint

To check the validity of the parameter values, I introduce a **Constraint** element as a sub element of the transaction. Each transaction may define different constraints. Each constraint consists of three parts – **trigger parameter**, **condition** and **action**. The trigger parameters are the ones which are checked in the condition. The element **action** defines the operations to be executed if the condition is not fulfilled. A constraint can be formally defined in the following form:

$Constraint =< TriggerParameter, Condition, Action >$

$TriggerParameter = "Parameter_1;...;Parameter_n"$

$Condition$ is a term according to CON - SYNTAX

$Action$ is a term according to ACTION - SYNTAX

System Action

Sometimes the system should inform the user about the current dialog state or the state of the backend application. The specification element **System Action** is proposed to specify such system activities. System action is defined as a sub element of a transaction. Each transaction may define more system actions. Each system action consists of **trigger parameter**, **condition** and **action**. When the trigger parameters change their values, the condition will be checked. The defined actions are executed if the conditions are evaluated to true.

$SystemAction = < TriggerParameter, Condition, Action >$

$TriggerParameter = "Parameter_1; ...; Parameter_n"$

$Condition$ is a term according to CON - SYNTAX

$Action$ is a term according to ACTION - SYNTAX

Backend Application Reference

Besides the extended specification elements, a way to specify interactions with the backend applications is needed for describing the various influences of the backend applications on the dialog flow. For this purpose, I assume that there is a clear interface between the speech user interface and the backend execution application. All methods defined in this interface can therefore be referred to in the dialog specification directly. As a reference, the name of the backend application, for instance, can be used in the dialog specification. When the name of the backend application occurs in the specification, the dialog engine knows that it is an invocation of the corresponding operations defined in the interface. With such a prerequisite, the interactions with the backend applications can be specified formally in the dialog specification.

In the extension, three specification elements are quite similar – **constraint**, **system action** and **postcondition**. The **constraint** and the **system action** differ from each other only in the condition for executing the defined actions. In constraint, the actions are executed if the condition is not fulfilled, so these actions are violation actions. In system action, the actions are executed if the condition is fulfilled. Though the difference is small, they serve to two different purposes. So I strongly suggest the introduction of two elements instead of merging two elements with different purposes into one more general element. The element **postcondition** can be regarded as a special case of **system action** with the trigger parameters as all mandatory parameters and the condition to be defined as that all mandatory parameters have their appropriate values. However, postconditions are required for each transaction, while system actions are just optional assistance actions.

In summary, a dialog specification based on the enhanced frame-based dialog model can be defined with the following forms:

$$A = < \{T_{1,} T_{2,} ... T_n\}, \{CONTEXT_1, ..., CONTEXT_m\} >$$

$$T_i = < ID, tg, PRE, PROMPT, P, CSTR, SYS, Post >$$

$$PRE = \{Precondition_1, ... Precondition_{n2}\}$$

$$PROMPT = \{prompt_1, ..., prompt_{n1}\}$$

$$CSTR = \{Constra\text{int}_1, ..., Constra\text{int}_{n3}\}$$

$$SYS = \{SystemAction_1, ..., SystemAction_{n4}\}$$

$$P = \{p_1, ..., p_{n5}\}$$

$$p_i = < PID_i, BOOL_i, PPROMPT_i, ig_i, dg_i, Infer_i >$$

$PID_i \in \Sigma^*$,
ig_i is the grammar used to understand user input in a user initiative utterance
dg_i is the grammar used to understand user input in system initiated dialog

$$BOOL_i = \{b_{i1}, ..., b_{in}\}$$
$$b_{ij} \in \{true, \ false\}$$

$$PPROMPT_i = \{pprompt_{i1}, ..., pprompt_{in1}\}$$

$$CONTEXT_i = \{TID_1 ... TID_i, ..., TID_l\}$$
$$TID_i = ID, where \ T_i = < ID, tg, PRE, PROMPT, P, CSTR, SYS, Post > \in A$$

There is a little offset between this form and Figure 4-3. Figure 4-3 focuses on the meaningful structures while this form considers additional elements for implementation based on DIANE. These elements comprise the identifiers for transaction and parameter, together with a set of Boolean values indicating the function of the parameter in a transaction; for example, whether a parameter is optional in a transaction. The details about implementation of this formal specification will be elaborated in Chapter 5.

4.4 Dialog Management

For interpreting the dialog specifications based on the enhanced frame-based dialog model, I propose a dialog management, which combines frame-based and event-based approaches. The task of the dialog management is to make the right response according to the user utterances and the system states. The basic mechanism in a frame-based approach is to query the unknown parameter of a transaction. This simple system response does not suffice for sophisticated applications, where different constraints or system actions exist. Combining the event-handling mechanism into the frame-based dialog management solves this problem. In this section, I first introduce the emerging dialog events in the enhanced frame-based dialog model, and then different dialog control mechanisms supported by the dialog manager.

4.4.1 Dialog Events

In the enhanced dialog model introduced in this dissertation, the parameters are the key for different constraints or system actions. Thus the change of parameter values is the relevant dialog event. A parameter may have one of three different statuses: undefined, valid defined, or invalid defined. If the status of the parameter changes to valid defined, the bound constraint or system action will be triggered for verification or execution. So the dialog event to be considered is the change of parameter status from undefined or invalid defined to valid defined. This dialog event should be captured by the dialog manager and triggers the evaluation of constraints or system actions.

4.4.2 Transaction Identification and Activation

Besides the normal mechanism for identifying the right transaction to be executed such as the one used in DIANE [Caspari, 2003], the context specifications help to avoid unnecessary ambiguity arising in dialogs thus to identify the right transaction. An application may contain different context $A = \{C_1,...C_n\}$ and a context is defined as a set of related transactions $C_i = \{T_1,...T_n\}$. After the execution of a transaction T, all contexts Ci with $T \in C_i$ will be saved as active contexts C_{active} in the dialog management system. The next user utterance is then interpreted concerning the actual contexts. This means, if the user utterance is ambiguous in sense of triggering more transactions ($T_1,...T_k$), the transactions Ti with maximal context overlapping are always preferred

$$|\{C \mid T_i \in C \wedge C \in C_{active}\}| \geq |\{C \mid T_j \in C \wedge C \in C_{active}\}| \, (i, j \in \{1,...k\})$$
$$\rightarrow \text{Prefer}(T_i)$$

If there is only one transaction Ti with maximal context overlapping, this will be taken by the system as the right one without asking the user for disambiguation. This feature leads to a more intelligent dialog system, which allows the user to express his/her request more efficiently related to the current context.

After the transaction has been identified, its preconditions will be verified. If the preconditions are not fulfilled, then the defined message for precondition violation will be prompted. Otherwise the transaction will be processed normally when all preconditions are fulfilled.

4.4.3 Parameter Query and Verification

If the status of a parameter is undefined, the system will prompt for this parameter. But before prompting for a parameter value, the dialog manager first checks the value element of the parameter. If there is any defined logical dependency, this logical form will be estimated. According to the evaluation result of the logical form, the value of the parameter will be inferred or prompted.

After the value of the parameter has been initiated, the constraints bound to this parameter will be checked. If the constraints are violated, the constraints violation action will be executed by the system.

The following Pseudo code illustrates the query and verification process of a parameter p = $< Grammar, QueryPrompt, ..., Infer >$

```
Procedure $query(p)$:
BEGIN
IF p is undefined
IF $Infer$ of p is not null
THEN V = evaluate ( $Infer$ )
IF V is not null
THEN p = V
verify(p);
ELSE system prompts QueryPrompt
P = USERINPUT
verify(p)
ENDIF
ELSE system prompts QueryPrompt
P = USERINPUT
$verify(p)$
ENDIF
ENDIF
END

Procedure $verify(p)$
BEGIN
IF $\exists Constraint =< TriggerParameter, Condition, Action > [p \in TriggerParameter]$
THEN evaluate (constraint)
IF Condition = false
THEN execute (Action)
```

```
ELSE p is defined.
ENDIF
ELSE p is valid
ENDIF
END
```

Figure 4-4 Query and verification process of a dialog parameter

4.4.4 Interactions with Backend Applications

The backend application can be referred to in the dialog specification. The developers use the backend application name as reference for this. When the dialog engine engages in a dialog based on the provided dialog specifications and encounters the backend application name, the defined expression will be sent to the backend application. The dialog engine will wait for the results from the backend application, and resume as normal after that.

With the element **system action**, the assistance of the backend application for task completion can be specified. In the **action** part of the element **system action**, the operations of the backend application can be invoked and thus the help messages dynamically generated by the backend application can be prompted to the user.

4.5 Scenario Examples

In this section I introduce a concrete example to explain how to specify the speech user interface of an application using this dialog model. It shows what the concrete dialogs look like, and in addition, a particular example to illustrate the function of context specification is introduced.

4.5.1 A Complete Application Example

A widespread telephone-based application is the unified messaging system. A case study assessing the extended frame-based dialog model with the Siemens unified messaging system – “HiPath Xpressions” has been carried out in the scope of this dissertation. The example functions and scenarios given in this section about unified messaging systems are based on this case study. A unified messaging system provides a speech user interface for users to access their messages via telephone. Currently the speech user interface of this system is an IVR system based on DTMF input. The interface is menu-driven and system-initiated. Building a very comprehensive menu system for such a messaging system is unavoidable. Each possible interaction between the user and the system has to be covered by a menu. Constructing a flexible speech user interface based on the extended frame-based dialog model is thus highly recommended in this case. A unified messaging system may provide many different functions; here my aim is to explain the enhanced dialog specification, so only the basic functions are concerned in what follows:

- Identify (log in)
- Listen to all new messages
- Listen to the messages from a certain person
- Send a message

An application is modeled as a set of transactions at the dialog level. Each transaction corresponds to a function provided by the backend application – here a unified messaging system. So that means that the functions are mapped to a transaction one by one. The dialog specification of this application involves four transactions: log in, listen to all new messages, listen to the messages from a certain person and send a message. In the following section, each transaction will be elaborated.

Identify

The **identify** function will be modeled as a transaction. This transaction can only be executed if the user is not logged in yet. This is the **precondition** for this transaction. The user has to input his/her telephone number and a password. These are the **parameters** of this function. As soon as all required information is present, the login function of the backend application (the unified messaging system) will be invoked, and afterwards the execution result will be prompted. These actions should be defined as postconditions. This modeling can be summarized in the following table:

Transaction	Identify
Precondition	The user has not been logged in.
Parameters	Username (telephone number), Password
Postcondition	Invocation of login function in the unified messaging system Prompt information about the login status (failed or successful)

Listen to all new messages

This function provides the user with all his/her new messages and is to be mapped to a transaction. To access his/her messages, the user has first to be successfully logged in to the system. This is the **precondition** for accessing this transaction. The application context of this transaction is the application "unified messaging"; more precisely, this transaction deals with all new messages. Thus, the context – “unified messaging: all new messages” for this transaction can be modeled with the element **context.** To execute this function, the backend system needs no information from the user, thus this transaction has no parameter. To perform this transaction, the corresponding function in the backend application has to be invoked and a suitable message has to be generated and prompted accordingly. The following table gives an overview of the modeling of this function:

Transaction	Listen to all new messages
Precondition	The user has already logged in successfully.
Postcondition	Invocation of function (get all new messages) in the unified messaging system Generate a corresponding information message Prompt the message(s)

Listen to all new messages from a certain person

This function provides the user with all new messages from a certain person. This function is also to be modeled as a single transaction. To access his/her messages, the user has first to be successfully logged into the system. This is the **precondition** for accessing this transaction. To execute this function, the system needs to know from the user what his/her name is ('person name'). So this transaction has one **parameter** – requested person name. To perform this transaction, the corresponding function of the backend application has to be invoked and a suitable message has to be generated and prompted accordingly. The following table gives an overview of the modeling of this function:

Transaction	Listen to all new messages from a certain person
Precondition	The user has already logged in successfully.
Parameter	Name of the person whose messages are requested
Postcondition	Invocation of function (get all messages from a certain person) in the unified messaging system Generate a corresponding information message Prompt the message(s)

Send a message

With a unified messaging system, the user may send a voice message to a contact. Also this function has to be mapped to a transaction in the enhanced frame-based approach. Just as with any other function, to access this transaction the user has to be logged into the system successfully. This has to be modeled as a **precondition** of this transaction. To send a message, first the message has to be input by the user. The message to be recorded would be the first parameter of this transaction. After the user has input the message, the system should provide the user with the possibility of checking the recorded message. In the case of a human-like system, the user should have the initiative to decide if he/she wants to check the recorded message or not. Thus, this will be modeled with a parameter (e.g. “listenYesNo”), which specifies the information as to whether or not the user wants to listen to the recorded message. If the user wants to check the message, then the system should prompt the recorded message. This action has to be modeled as a system action, which is triggered by the parameter “listenYesNo”. After the user has checked the message, the system should support possible re-recording of the message in case the user thinks that the message is erroneous. Thus, the information concerning whether or not the recorded message is correct has to be modeled by a further parameter (e.g. “confirmMessage”). If the user does not want to check the message after recording, then this message is indirectly confirmed. That means there is a dependency between the parameter “listenYesNo” and “confirmMessage” – if “listenYesNo” has a value of “no”, then the “confirmMessage” has an inferred value of “yes”. This dependency has to be defined as an inference rule in the **Infer** part of the parameter “confirmMessage”. If the user thinks the recorded message is not right, then a **system action** has to be defined to enable the user to re-record the message. More precisely, a **system action** has to be defined to remove the value of the parameter “message”. Further, to send a message, the receiver must be input by the user. This is a further parameter in this transaction. As a kind of additional information, the user can also mark a message as confidential or urgent, etc. Such special settings can be modeled by an optional parameter. Finally, before the system sends a message, it confirms the sending process with the user and waits for a positive reply from the user to execute the transaction. The user reply is specified by a parameter as well. The **postcondition** for this transaction sends the message and prompts corresponding information.

The following table gives an overview of the modeling of this function:

Transaction	Send a message
Precondition	The user has already logged in successfully.
Parameter	message – the message to be sent listenYesNo – if the user wants to check the recorded message confirmMessage – if the recorded message is correct receiver – the receiver of the message specialSetting – special setting of the message, e.g. confidential, urgent confirmSending – last confirmation before sending the message
Parameter Inference (Infer)	confirmMessage: Message is confirmed indirectly if the user did not choose to listen to the recorded message a second time. (listenYesNo=”no” -> confirmMessage=”Yes”)
System action	Prompt the recorded message if the user wants to check it. Clear the value of the message parameter if the user says the message is not right.
Postcondition	Invocation of function (send message) in the unified messaging system Prompt the corresponding message

Based on this dialog specification, an enhanced dialog manager is able to provide a dialog as in the following example:

S1 (System): Welcome to the unified messaging system, please identify yourself first.
U1 (User): 12345
S2: Telephone number 12345, please enter your password.
U2: 12345.
S3: You are logged in successfully. What can I do for you?
U3: Please give me all messages from Harry Hanks.
S4: Messages from Harry Hanks…a moment please…
S5: There are two messages from Harry Hanks: First message …; second message…
U4: Please send Harry Hanks a message.
S6: Please input your message and press # when you are finished.
U5: This is a test …. (# pressed)
S7: Do you want to check the message?

U6: No.

S8: Send the message normally?

U7: Yes.

S9: The message has been sent.

The following figure illustrates the changes of parameter values in the dialog session from U4 to U7. The active changed parameters after each user utterance are highlighted.

U4:

Action	Send a message
Parameter	Message: unknown listenYesNo: unknown confirmMessage: unknown ***Receiver: Harry Hanks*** specialSetting(optional): unknown confirmSending; unknown

U5:

Action	Send a message
Parameter	***Message: "This is a test…"*** listenYesNo: unknown confirmMessage: unknown Receiver: Harry Hanks specialSetting(optional): unknown confirmSending; unknown

U6:

Action	Send a message
Parameter	Message: "This is a test…" ***listenYesNo: no*** ***confirmMessage: yes (inferred)*** Receiver: Harry Hanks

	specialSetting(optional): unknown confirmSending; unknown

U7:

Action	Send a message
Parameter	Message: “This is a test…” listenYesNo: no confirmMessage: yes Receiver: Harry Hanks specialSetting(optional): unknown ***<u>confirmSending: yes</u>***

4.5.2 Scenario for Context Application

Let us consider an organizer system with the following transactions:

- Check new messages
- List new messages
- Check appointment
- List appointments

The “check” transactions only tell the user if there is any new message or appointment, while the “list” transactions list all messages or appointments.

In such a system, the following situation is possible:

The user first asks if there are any new messages for him/her, then in the case that there are some new messages, asks for a list of all new messages. Since the user has first checked if there are any new messages, he/she maybe just says “list them” to tell the system that he/she wants a list of all messages. However, the sentence “list them” is ambiguous, because it triggers two transactions – "list messages" and "list appointments". A normal frame-based spoken dialog system will initiate a clarification dialog to ask the user if he/she wants to list messages or appointments. However, according to the current context – the user has just executed the transaction “check new messages” – it is actually clear that the user wants a list of new messages.

With the context specification introduced in the extended dialog model, this problem can be easily solved:

Two contexts can be defined in the corresponding dialog specifications:

$C1 = \{CheckMessage, ListMessage\}$

$C2 = \{CheckAppointment, ListAppointment\}$

After the execution of "CheckMessage", when the user utters "list", the following steps take place:

1. The context C1 with $CheckMessage \in C1$ will be stored in the dialog management component, and {C1} is the set of active contexts.
2. Two transactions will be triggered – "list new messages" and "list appointments".
3. The contexts of both transactions will be checked by the dialog system, and the following results will be found:

   $ListMessage \in C1$ – context match ….1 (C1 is in the set of active contexts)

   $ListAppointment \in C2$ - context match … 0
4. The transaction with the most context matches – "ListMessage" – will be chosen by the system without clarification.

A dialog example for this system is as follows:

S1 (System): Welcome to your personal organizer, please identify yourself first.

U1 (User): 12345

S2: Telephone number 12345, please enter your password.

U2: 12345.

S3: You are logged in successfully. What can I do for you?

U3: Is there any new message for me?

S4: Yes, there are two new messages.

U4: *List them* please:

S5: *First message …; second message…*

U5: *List* all my *appointments* for today please.

S6: *First appointment…; second appointment…;*

After U5, though the active context is still C1, the user utterance is unambiguous, so the transaction "ListAppointment" is executed without clarification.

4.6 Summary

An enhanced frame-based dialog model and the corresponding dialog management strategies have been introduced in this chapter. With this model and strategies it is possible to describe various dependencies in different sophisticated applications declaratively. A speech user interface for a new application can be developed by means of specifying all tasks, their parameters and the relationships between the tasks and parameters declaratively. This new dialog model improves the power of a frame-based dialog system to be able to model sophisticated applications completely declaratively without affecting the portability of the general dialog system.

4.6.1 Comparison to Related Approaches

There are different frame-based approaches aiming to improve the dialog model in different ways. I compare the enhanced frame-based model (EFM), with the existing VoiceXML model [McGlashan et al., 2003], the basic DIANE model [Block et al., 2004], and the Galaxy II dialog control mechanism developed in a DARPA project [Seneff et al., 1999]. The features comparison is summarized in Table 4-1.

Feature / Dialog Management	**EMF**	**DIANE**	**VoiceXML**	**GALAXYII**
Basic model	Frame-based	Frame-based	Frame-based + finite-state machine	Frame-based with rules
Precondition	Declarative specification	Not supported	Not supported	Not supported
Postcondition	Declarative specification	Supported by Java callback Function	Not specified	Not specified
Context specification and application	Declarative Specification	Not supported	Not supported	Not supported
Parameter Inference	Declarative specification	Supported by Java callback function specification	Only static default value supported	Not supported
System initiative assistances	Declarative specification	Not supported	Supported by control element	System action can be initiated by rules

Dependencies between different parameters	Declarative specification	Supported by Java callback function	Not supported	Can be specified with rules
Interaction with backend application	Integrated into the declarative specification	Supported by the java interface	Integrated in the declarative specification	Not introduced

Table 4-1 Feature comparison of different frame-based models

Based on the comparison, the outstanding features of the enhanced frame-based dialog model proposed in this chapter have been highlighted. With the proposed model it is possible to describe different dependencies in various applications declaratively and thus to provide a general and portable dialog system, which can be deployed to an enormous range of applications from weather information to management of communication profiles. A speech user interface for a new application can be developed by specifying all tasks, their parameters and the relationships between the tasks and parameters declaratively.

The next chapter introduces a concrete implementation approach for this dialog model based on the existing DIANE dialog system. This implementation proposal has been adopted in the meantime by Siemens, based on the concepts developed in this work, in developing a new version of DIANE supporting the enhanced frame-based model. In the next chapter, I introduce the way that speech user interfaces of different applications can be specified declaratively based on this dialog specification, and how the new DIANE system engages in a conversation with the user. Furthermore, the way in which the power of this dialog model has been assessed will be commented on.

Chapter 5
Speech User Interface Development for Applications

A dialog model suited for the combination purpose of this dissertation was introduced in the last chapter. In this chapter, I give a detailed approach for implementing the extended frame-based model. A specification language (called enhanced DIANEXML) is constructed by extending the existing DIANEXML. Several typical examples are introduced to explain how the language is applied and how the speech user interface for applications can be specified within the enhanced frame-based model.

5.1 Specification Language – Enhanced DIANEXML

DIANEXML developed by Siemens is an XML-based dialog design language for simplifying the development process of speech user interfaces. To provide a speech user interface for an application, the tasks and domains are specified using DIANEXML. The runtime resources for DIANE dialog system [Caspari, 2003] can be generated by the system automatically, based on the DIANEXML specification. To implement the enhanced frame-based dialog model, I extend the specification language DIANEXML to “enhanced DIANEXML”. In this section the existing DIANEXML will be introduced first, and then the enhanced DIANEXML will be elaborated.

5.1.1 DIANEXML

The DTD definition of DIANEXML can be found in Appendix 1. A speech user interface is specified with three kinds of files – a transaction file, a parameter file and a set of grammar files.
Figure 5-1 shows the file structure for a dialog specification.

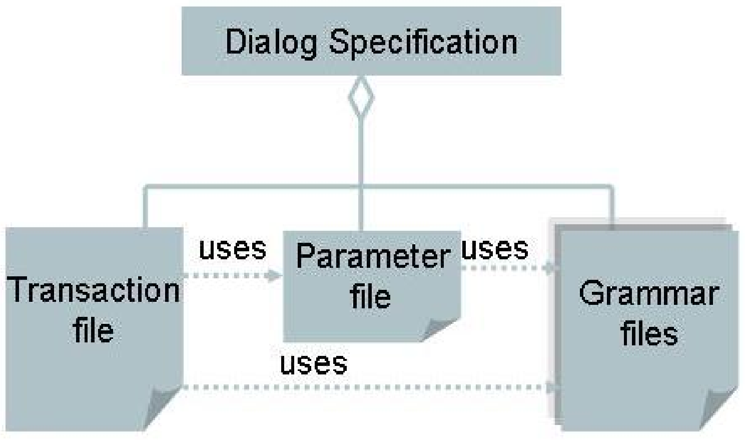


Figure 5-1 File structure of DIANEXML specification

The transaction file defines the speech user interface as a set of executable transactions. The necessary information for executing the transaction is defined as parameters. The parameters are first specified globally in the parameter file. In the transaction file the parameters are referred to via names. For speech recognition and language understanding, the grammar files are defined. In the transaction and parameter files grammars are referred to via names.

Transaction definition

In the DIANE model there is no concept for an application: the set of transactions represents the application. So the top-level element in a transaction file is “transactionL” representing a set of transactions (See Appendix 1). The element “transactionL” has no attributes and contains one or more transactions.
A transaction (see Appendix 1) represents an executable task provided by the backend application of the speech user interface. Table 5-1 gives an overview of the elements of a transaction:

Specification Element	**Function of the element**
Name	The unique identity of the transaction
ExeFunc	The function to be invoked when all necessary information of the transaction is available
ExePromt	The message to be prompted before the execution of the function in the backend application
TrPrompt	The prompt used to identify this transaction in a clarification dialog, e.g. “book a flight ticket” or “query

	the timetable of a train”
TrConfirmBool	The Boolean value to specify whether an extra confirmation from the user is required before the system addresses the transaction as the desired one
TrConfirmPrompt	The prompt for confirming the transaction
TrGrammar	The trigger grammar of the transaction
TrParameter	The parameter of the transaction. This element can be involved more times to define different parameters. The parameters are defined globally in the parameter XML file. Here the parameters are referred to by names. Several transaction-specific features are defined as sub elements. These elements are introduced in Table 5-2.
TrAddCode	Java code to be added as procedure attachment in the dialog
CstrFunc	A function defining constraints and consistency rules of the transaction in java
SemLessGrammar	Grammars defining expressions without semantics, which should be understood by the system such as “I mean”, “huh”, etc.
SemLessStartGrammar	Grammars defining expressions without semantics, which should be understood by the system when it interprets the user initiative utterance such as “I want”, “I would like to”, etc.

Table 5-1 Elements of transaction in DIANEXML

Table 5-2 shows an overview of the sub elements of the element TrParameter. They define the transaction specific features of a parameter:

Specification Element	**Function of the element**
Name	The identity of the parameter; this is also the name defined in the parameter XML file
RecBool	A Boolean value indicating whether the parameter is used recursively
OptBool	A Boolean value indicating whether the parameter is

	optional
DefaultVal	The default value of the parameter
InfBool	A Boolean value indicating whether the parameter can be inferred by other parameters
AlwaysConfBool	A Boolean value indicating whether the parameter value should always be confirmed with the user
UserConfBool	A Boolean value indicating whether the parameter value should be confirmed when it is given by the user
ConfIfInfBool	A Boolean value indicating whether the parameter should be confirmed when its value is inferred by the system
ConfIfDefBool	A Boolean value indicating whether the parameter should be confirmed when it owns a default value
InterruptBool	A Boolean value indicating whether the transaction should be interrupted when the parameter violates any consistency rule
TestAllBool	A Boolean value indicating whether the parameter value will always be proven against the backend application. This has only effect when the backend application is implemented in prolog.
OutOfTaskBool	A Boolean value indicating whether the parameter is only modeled in the speech user interface but not in the backend application. That means the backend application cannot handle the parameter and the user will be informed that the input is out of task.
CopyPermittedBool	A Boolean value indicating whether the parameter value can be copied from the dialog memory

Table 5-2 Elements of transaction parameter in DIANEXML

Parameter definition

All parameters used in dialog are defined globally in an extra XML file. The exact definition can be found in Appendix 1. All parameters are defined in one file, and the top-level element in this file is “parameterL” containing one or more parameter elements. All global information of a parameter is defined with sub elements that are listed in Table 5-3:

Specification Element	**Function of the element**
Name	The unique name of the parameter
PmType	The type of the parameter; it can be “generating” or “not generating”. When set to generating, the help prompt will be generated from the parameter grammar automatically.
IGrammar	Grammar used to understand the parameter in the initiative user input
DGrammar	Grammar used to understand the parameter in the dialog when the system asks for it
PmPrompt	Prompt to query the parameter
PmRekPrompt	Prompt to query a recursive parameter
PmRekDiscardPrompt	Prompt to be uttered by the system after the user discards a part of the recursive parameter
PmConfirmPrompt	Message for confirming the parameter value
PmHelpFunction	Function defining the help prompts in java code
PmHelpPrompt	The help message for the parameter when the user does not know what he/she should provide as values
PmInferredPrompt	The message to be prompted together with the confirmation prompt if the parameter value is inferred by the system.

Table 5-3 Elements of parameters in DIANEXML

Grammar definition

The grammars are context-free grammars allowing only right-recursive rules as recursive rules. In Chapter 6, the grammars will be discussed in detail. The DIANE grammar formalism can be found in Appendix 1. The grammars will be referred to in the transaction and parameter definition files via names.

Application example

I introduce a simple application which allows the selection of any radio channel (defined by music genres) via speech as an example here. This example illustrates how to define a speech application with DIANEXML.
The following is the transaction definition file:

```
<TransactionL>
```

```
 <Transaction>

  <Name>select_radio_channel</Name>

  <ExeFunc>select_radio_channelExe</ExeFunc>

  <TrPrompt>select a radio channel</TrPrompt>

  <TrGrammar>radio.grm</TrGrammar>

  <TrGrammar>select.grm</TrGrammar>

  <TrParameter>

    <Name>radio_channel</Name>

    <CopyPermittedBool>true</CopyPermittedBool>

  </TrParameter>

  <CstrFunc>select_radio_channelCstr</CstrFunc>

</Transaction>

</TransactionL>
```

The following is the parameter definition file:

```
<Parameter>

  <Name>radio_channel</Name>

  <PmType>generating</PmType>

  <IGrammar>radio_channel.grm</IGrammar>

  <DGrammar>radio_channel.grm</DGrammar>

  <PmPrompt>which channel?</PmPrompt>

  <PmConfirmPrompt>Do you want to listen to $radio_channel?

  </PmConfirmPrompt>

</Parameter>
```

For this application three grammars are needed. They are “radio_channel.grm” for different radio channels, “radio.grm” and “select.grm” for triggering the transaction “select a radio chanel”. I offer the grammar radio_channel.grm as an example here:

```
$radio_channel =
        hip pop {: "hip pop" :}
      | classic music {: "classic music" :}
            | sport {: “sport”};
```

5.1.2 Enhanced DIANEXML

To describe a speech user interface as declaratively as possible, the frame-based dialog model has been extended in this thesis to an enhanced frame-based dialog model which includes pre- and post-condition specification, specification of the contextual relation of different transactions, formal specification of parameter inference and influences of the backend system. To implement an application specified with the extended frame-based model, I introduce a specification language based on DIANEXML. The DTD file of enhanced DIANEXML can be found in Appendix 2. Here the added specification elements are described briefly to provide an overview.

Precondition

Precondition defines conditions for executing a transaction. The system only activates a transaction if its preconditions are all fulfilled. A **precondition** contains two elements – **condition** and **message**. The **condition** defines the actual logical rule. The logical rule follows a syntax defined as enhanced DIANE script, which can be found in Appendix 2. The **message** defines the prompt in case of precondition violation. For one transaction one or more preconditions can be defined.
The following is an example of a precondition, which defines that the user can only access the transaction if he/she has logged in to the backend application successfully:

```
<Precondition>
  <Condition> @xpressions.isLoggedIn() == true </Condition>
  <Message>You have to log in first. </Message>
</Precondition>
```

Postcondition

Postcondition defines the execution of the transaction including the invocation in the backend application. Each transaction is represented as a form with one or more parameters in the enhanced DIANE model. From the dialog view, the transaction is finished if all required parameters have got a unique value. But from the point of view of task accomplishment, a transaction contains also the invocation of the corresponding methods in the backend application performing the task, and the report about the task execution results to the user. These two parts are defined in the postcondition of a transaction.

A **postcondition** can be a method invocation in the backend application or a prompt to inform the user about the task execution status. A **postcondition** can also be encapsulated by an **if-else-statement**.

The following is an example for the postcondition of a login transaction, which invokes the login function in the backend application (which is “xpressions” here) and gives the user an appropriate message about the login result:

```
<Postcondition>
     <BackendExe expr="@xpressions.login($telnumber,$password)" />
     <If cond="@xpressions.isLoggedIn()==true">
      <Prompt> You are successfully logged in. </Prompt>
     <Else/>
     <Prompt>The   id   or   password   is   wrong,   login   not
      successful.</Prompt>
     </If>
</Postcondition>
```

Context

The **context** element serves to define the contextual relation between different transactions. By using this contextual information in the dialog, the usability and intelligence of the speech user interface can be enhanced. The following is an example of the context specification for the “address book search”, “address book edit” and “delete a contact from the address book” transactions:

```
<Context>
  <Transacton>AddressbookSearch</Transaction>

  <Transaction>AddressbookEdit</Transaction>

  <Transaction>AddressbookDelete</Transacton>
```

```
</Context>
```

The context is a set of transactions which are contextually related.

Infer

The **Infer** element is a sub element of the specification element **TrParameter**. With this element it is possible to describe how the parameter can be inferred by other parameters or system. The **infer** element can be either a simple term or an if-else-expression. The term or expression should be written with the extended DIANE script language. (See Appendix 3)
The following is an example of parameter “phoneStatus” representing the telephone status, whose value can be inferred by another parameter “phoneStatusYesNo”.

```
  <TrParameter>
    <Name>phoneStatus</Name>
    <InfBool>true</InfBool>
    <Infer>
     <if cond="$phoneStatusYesNO=='no'">
       NOINPUT
     <else/>
       USERINPUT
     </if>
    </Infer>
    <UserConfBool>true</UserConfBool>
    <CopyPermittedBool>true</CopyPermittedBool>
  </TrParameter>
```

The parameter “phoneStatusYesNo” indicates information as to whether the user wants to define the phone status in a transaction. In the Infer element this parameter will be checked. If the user does not want to define the phone status then the parameter “phoneStatus” will not be asked but instead inferred to a dummy value. This mechanism increases the dialog intelligence, and the *infer* element keeps the specification declarative.

Constraint

The **constraint** element serves to define the consistency rules of the parameters. In practice, some parameters must have certain values in certain situations.

A **constraint** element contains one or more **trigger parameters** as sub elements. When one ***trigger*** **parameter** changes its value, the constraint will be checked. Moreover a **constraint** element can have one or more **conditions** as sub elements. The **conditions** define the actual logical consistent rules. This **condition** should be defined with the script language (See Appendix 3). Finally a **constraint** element may have a further sub element – **action**. This element defines the actions to perform if the consistent rules are violated.

The following is an example for a constraint, which declares the consistency rule between the parameter “destination city” and “departure city” of the transaction “flight reservation”. Particularly, these two cities must be different for a flight:

```
 <Constraint >

    <TriggerParameter>destinationCity</TriggerParameter>

    <TriggerParameter>departureCity</TriggerParameter>

    <Condition>$desctinationCity!=$depatureCity </Condition>

    <Action>

      <Clear name =" destinationCity" />

      <Prompt> The destination city can not be the same as the
               departure city of a flight. Please provide a new
               destination city.

      </Prompt>

   </Action>

</Constraint>
```

This constraint specifies that the destination city must be different from the departure city. If this rule is violated, the value for the destination city will be removed and an adequate message will be prompted to inform the user about the inconsistency.

System action

The element **system action** serves to define some system-initiated action, which can be triggered by a parameter. That means when the **trigger parameter** changes its value, the system action will be triggered. **System action** contains three sub elements – **trigger parameter, condition** and **action**. There can be one or more **trigger parameters** and **conditions** but only one **action**. After the system action is triggered, the conditions will be verified. If all conditions are fulfilled, the action will be performed.
The following is an example for a system action of the transaction “contact edition” for editing a saved contact:

```
<SystemAction>

  <TriggerParameter>editContactYesNo</TriggerParameter>

 <Condition>$contactName!= Null</Condition>

 <Condition>$editContactYesNo=="Yes"</Condition>

  <Action>

   <Prompt>The contact $contactName has been defined as the following:
  </Prompt>

   <Prompt>

     <Backend expr="@addressbook.getContact($contactName)"/>

   </Prompt>

   </Action>

</SystemAction>
```

This system action specifies that if the user utters that he/she wants to edit a contact, the system will invoke the backend application and prompt the definition of that contact automatically.

Backend method invocation

The element for backend method invocation is used in the postcondition specification. With this element the invocation of methods provided by the backend application will be specified. This element may contain the sub element – **result**, which describes the result of the method invocation. This **result** can be referred to in the later specification.

The following is an example of the postcondition of a transaction for searching for a contact in the address book:

```
<Postcondition>

<BackendExe expr="@addressbook.search($firstname,$lastname)">

 <Result id=”result” type= “Set”/>

</BackendExe>

<Prompt> @addressbook.output("searchAddressbook", result) </Prompt>

</Postcondition>
```

In the postcondition the backend application will be invoked, and the result will be stored in the result variable “result”. In the *result* element the variable is defined to have the type “Set”. Afterwards this result is passed to a function “output” for generating the adequate prompt.

Sometimes the result of the method invocation is not so critical, so the sub element “result” is optional to define.

5.2 Modeling Examples

This dissertation aims to propose a dialog model which can be applied to real industrial applications. For this purpose, a comprehensive analysis of real industrial applications and the feasibility of applying the enhanced dialog model to these applications has been carried out within the scope of this dissertation. Among these applications there are some relatively complex systems from the area of enterprise communications. In this section, I introduce two of these systems – HiPath Xpressions system and the HiPath CorporateConnect system of Siemens – and in particular, address how to specify their speech user interfaces with the enhanced frame-based dialog model.

HiPath Xpressions

HiPath Xpressions is a unified messaging system integrating services for voicemail, fax and email. The system can be accessed from any telephone and any networked computer. The user can access and manage any message with a telephone or a computer. This system has two kinds of interfaces – a web-based graphical user interface and a telephone-based voice user interface. The current telephone user interface is an IVR system based on DTMF input. The interface is menu-driven and system-initiated. Building a very comprehensive menu system for such a messaging system is unavoidable. Each possible interaction between the user and the system has to be covered by a menu. Constructing a flexible speech user interface based on the enhanced DIANE model is thus highly recommended in this case.

The functions of the Xpressions system can be grouped in different sub domains. The following are the essential functions provided by the user interface:

- Identify (login)
- Listen to the messages (Default new messages)
    - Output a message to a device
    - Go to previous message
    - Go to next message
    - Save message
    - Delete message
    - Repeat message playback
    - Repeat message header playback
    - Go directly to message without header
    - Pause message playback
    - Resume paused message playback
    - Reply to the played message
    - Forward the played message
    - Call message originator
- Send a message
    - Pause recording and re-record the message
    - Pause recording and delete the message
    - Pause recording and check the message
- Change answering options
    - Set greeting for external calls
    - Set greeting for internal calls

    - Set System greeting
    - Set answering machine functions (accept messages or greeting only)
    - Set the number for forwarding calls
    - Set the name of this mailbox
- Changing mailbox options
    - Set user prompt language
    - Set prompt level
    - Change password
    - Activate/deactivate the message notification
- Connection (call user: number or name)

The functions for changing answering options or mailbox options are independent of the dialog system state. So we can regard them as stateless functions. The message listening and recording functions are related to the current system state and thus stateful. However, this system state information can be saved in the backend application so that the front end – the user interface – can be regarded as stateless. It is always important to keep track of the message playback in the backend application, so that the system always knows which message is on play currently and which message is the next one. It is outside the scope of this thesis to develop a comprehensive speech user interface. Based on this thesis, a student project [Schilling, 2005] developed different speech user interfaces for real industrial applications such as the applications “Xpressions”, “CorporateConnect” and “ComAssistant” of Siemens. Here I explain the modeling process with two example functions. One is the function "identify". It is typical for many applications to request the users to login first before using the speech portal. Thus the successful execution of the identify function is a precondition to access other functions. In the following I show how this function can be modeled as a transaction in the speech user interface. Another function is “send message”. This function is special because during the sending process the user can request to play back the recorded message. If he/she is not satisfied with the message, he/she can request to re-record the message. So in this function there are several modeling challenges. I will show how to solve them subsequently.

Identify

The identify function will be modeled as a transaction. This transaction needs only to be executed if the user is not logged in yet. This is the precondition for this transaction. The user has to input his/her telephone number and password. These are parameters of this function. When all information is complete, the login function of the backend application (Xpressions) will be executed, and the execution result will then be prompted afterwards. The formal description with enhanced DIANEXML is included below:

```
<Transaction>

  <Name>identify</Name>

  <TrPrompt>Login to the application.</TrPrompt>

  <TrGrammar>identify.grm</TrGrammar>

  <Precondition>

    <Condition>@xpressions.isLoggedIn()==false</Condition>

    <Message>You are already logged in</Message>

  </Precondition>

  <TrParameter>

    <Name>telnumber</Name>

  </TrParameter>

  <TrParameter>

    <Name>password</Name>

  </TrParameter>

  <Postcondition>

     <BackendExe expr="@xpressions.login($telnumber,$password)" />

     <if cond="@xpressions.isLoggedIn()==true">

      <Prompt> You are successfully logged in. </Prompt>

     <else/>

      <Prompt> The id or password is wrong, login not successful.
      </Prompt>

     </if>

  </Postcondition>

</Transaction>
```

Send a message

The function “send message” enables the user to send a message to a contact. This function will be modeled as a transaction with parameters specifying content, receiver, and priority of the message, the information regarding whether the user wants to listen to the message before sending and whether he/she wants to re-record a new message. The condition for this transaction is that the user has already logged in. There are two system actions in this transaction. The first is that if the user expresses that he/she wants to listen to the message before sending, the system should play back the message. The second is that if the user utters that he/she wants to re-record the message the system should clear the current message content and be ready for recording a new message. To execute the transaction the message should be sent by the system and a sending confirmation should be prompted. The XML description is shown in the following:

```
<Transaction>

<Name>sendMessage</Name>

  <TrPrompt>send a message</TrPrompt>

  <TrGrammar>send.grm</TrGrammar>

  <TrGrammar>message.grm</TrGrammar>

  <Precondition>

    <Condition>%xpressions.isLoggedIn()==true</Condition>

    <Prompt>Please login first.</Prompt>

  </Precondition>

  <TrParameter>
    <name>message</name>

  </TrParameter>

  <TrParameter>

    <name>reciever</name>

  </TrParameter>

  <TrParameter>

    <name>priority</name>

  </TrParameter>

  <TrParameter>

    <name>listenYesNo</name>

  </TrParameter>

  <TrParameter>

    <name>newMessage</name>

  </TrParameter>

  <SystemAction>

    <TrParameter>listenYesNo</TrParameter>

    <Condition cond="$listenYesNo=='Yes'"/>

    <Action>

      <Clear name="listenYesNo"/>

      <Prompt> $message </Prompt>

    </Action>

  </SystemAction>

  <SystemAction>
```

```
    <TrParameter>newMessage</TrParameter>

    <Condition cond="$newMessage=='Yes'"/>

    <Action>

      <Clear name="message" />

      <Clear name="newMessge" />

    </action>

  </SystemAction>

  <Postcondition>

     <BackendExe expr="@xpressions.sendMessage($reciever,$message)"/>

     <Prompt>The message has been sent. </Prompt>

  </Postcondition>

</Transaction>
```

CorporateConnect

HiPath CorporateConnect is an enterprise mobility solution that allows employees to be reached at a single business number regardless of current location. This system provides mobile users with the ability to utilize the enterprise telephone network and to have full-feature access to functions such as callback and conferencing. To access these functions the mobile users can use a normal telephone or mobile phone. A telephone user interface is provided. This telephone user interface is menu-driven right now and thus restricted. For more complex functions the user has to use a web-based interface. The functions provided by the telephone user interface are listed below:

- Login
- Log on (register the remote phone)
- Log off (sign off the remote phone)
- Make call

- Change active profile
- Check message waiting
- Change default call routing
- Activate feature
    - Message waiting notification
    - Distinctive call routing
    - External/internal call routing
    - Callback destination routing
    - All call routing

These functions are relatively simple, thus I choose the function logon as example to illustrate the specification with the enhanced DIANEXML.

Logon

To use the one number service of CorporateConnect the user should first register the remote phone. This function is modeled as "logon" transaction in the interface. The conditions for executing this transaction are that the user must have been logged in and has not registered any remote phone yet. By registering, the user has the ability to change the default profile and give a number to which to redirect the call after logging off. This information is modeled with parameters. As a kind of assistance, the system first asks the user if he/she wants to change the default profile. So the information as to whether the user wants to change the default profile influences the value of the parameter “profile”. The following is the formal description in enhanced DIANEXML:

```
<Transaction>

  <Name>logon</Name>

  <Precondition>

    <Condition> @corporateconnect.isLoggedIn()==true </Condition>

    <Message> Please identify yourself. </Message>

  </Precondition>

  <Precondition>
    <Condition> @corporateconnect.isLoggedOn()==false </Condition>
    <Message> You are already logged on. </Message>
  </Precondition>
  <TrPrompt>Register a remote phone</TrPrompt>
  <TrGrammar>register.grm</TrGrammar>
  <TrGrammar>corporate.grm</TrGrammar>
  <TrParameter>
    <Name>changeProfileYesNo</Name>
  </TrParameter>
  <TrParameter>
    <Name>profile</Name>
    <InfBool>true</InfBool>
    <Infer>
     <if cond="($changeProfileYesNO=='no')">
      NOINPUT
     <else/>
      USERINPUT
    </Infer>
  </TrParameter>
  <TrParameter>
    <Name>redirectYesNo</Name>
  </TrParameter>
  <TrParameter>
```

```
    <Name>redirect</Name>

    <InfBool>true</InfBool>

    <Infer>

      <if cond="($redirectYesNO=='no')">

       NOINPUT

      <else/>

       USERINPUT

      </if>

    </Infer>

  </TrParameter>

  <Postcondition>

    <BackendExe expr="@corporateconnect.logon($profile,$redirect)"/>

    <if cond="@corporateconnect.isLoggedOn()==true">

      <Prompt> Your phone is registered successfully. </Prompt>

    <else/>

      <Prompt> Your request can't be processed right now, it will be
       cancelled.

      </Prompt>

    </if>

</Transaction>
```

5.3 Summary

This chapter has introduced an approach for implementing the enhanced frame-based dialog model. This approach has been adopted by Siemens to develop a new version of the dialog system DIANE. This dissertation focuses on the combination of speech applications, and thus it is beyond the scope of this dissertation to prove the power and generality of the frame-based model. However, I have shown some non-trivial examples in this chapter. A student project [Schilling, 2005] has addressed the issue of power and generality of the enhanced frame-based model. In the project, some non-trivial example applications have been developed completely, including the above-introduced Xpressions and Corporate Connect systems. The dialogs are proven to be flexible enough for this purpose. In addition, the project compared the enhanced frame-based model with the AGENDA dialog manager used in CMU communicator [Rudnicky & Xu, 2002] and assessed the enhanced frame-based model as powerful enough for various phenomena appearing in different non-trivial applications.

Consequently, the enhanced frame-based dialog model can be applied to various applications. So in the following chapters, I assume the applications to be combined into one integrated speech user interface are all specified with this model. This makes no actual restriction on the applications. The combination scheme is therefore general and covers various applications ranging from simple information-providing applications to sophisticated applications in the communication areas.

Chapter 6
Grammar Comparison

6.1 Introduction

In order to combine the speech user interfaces of two different applications to construct an integrated speech user interface for both applications which is usable and intelligent, an application-level combination such as those introduced in the approaches of Bui et al. [2005], Neto et al. [2003] and Seneff et al. [1999] no longer suffices. To solve the problems of task sharing (the same task appears in both applications) and information sharing (common information is used in both applications), it is necessary to dig into the next elementary level of a dialog specification – transactions and parameters.

Before solving the problems of identical transactions (task sharing) and identical parameters (information sharing), it is necessary to recognize and determine these identities first. In Chapter 4 and 5, I introduced the dialog specification elements. Some of these elements describe the required essential information for the transaction execution, and the others describe information which helps the system to construct a natural dialog to obtain this necessary information. Recalling the basic paradigm of the frame-based dialog modeling approach, an application is a set of frames, where each frame consists of a set of slots representing the required information to perform the frame. The natural language expressions for the frame and the slots are all defined in the corresponding grammars. For a more flexible dialog and a more formal dialog specification, different additional elements are introduced such as precondition, post-condition, constraints, prompts, etc.

Based on this analysis, we can say the essential elements in a dialog specification are the grammars and parameters, where the essential information of a parameter is again its grammar. All other specification elements serve to support the dialog system to construct a more natural dialog to obtain the essential information for a frame and execute the corresponding task in the backend application.

Without a common framework for the meaning of data such as the ontologies proposed for the semantic web, it is difficult to compare the arbitrary elements expressed in “strings” designed by different designers directly in order to find the overlaps. However, it is possible to use the properties of grammars to derive helpful information from the formal structure without having to clarify the meaning of strings. As the only useful semantic-carrying element, the grammar describes possible natural user utterances required to trigger a frame and fill in a parameter – actually what a frame or a parameter is in terms of natural language. Therefore, based on the comparison of grammars, the overlaps between transactions or parameters can be indicated and determined.

In this chapter, I discuss how to compare two grammars in detail. This is an essential foundation for the later combination algorithm. First, I recall the theoretical background for grammars and formal languages. Then I describe the use of context-free grammars in the context of spoken dialog systems. Afterwards I describe the theoretical considerations about comparison of the specific types of context-free grammars which are used in the language processing areas. Then I propose two possible approaches for comparing the specific context-free grammars used in the DIANE spoken dialog system. Finally I discuss some related works and summarize this chapter by highlighting the features of the two proposed comparison approaches.

6.2 Theoretical Background

As potential models for natural languages, formal grammars are classified by the Chomsky Hierarchy into four classes.

6.2.1 Chomsky Hierarchy

The Chomsky hierarchy defines four types of grammars and the equivalent automaton of the grammars.

The largest family of grammars in the Chomsky Hierarchy permits productions of the form $\alpha \to \beta$, where α and β are arbitrary strings of grammar symbols, with $\alpha \neq \varepsilon$. These grammars are known as *Semi-T hue*, *type 0*, *phrase structure* or *unrestricted* grammars. This type 0 language is exactly the set of languages accepted by a Turing machine.

Applying the restriction on productions $\alpha \to \beta$ of a phrase structure grammar that β is at least as long as α , the resulting grammar is called *context-sensitive* or *type 1* grammar and its language is a *context-sensitive* or *type 1* language. The set of languages defined by context-sensitive grammars is exactly the same as the set of languages accepted by a linear-bounded automaton.

The *type 2* or *context-free* grammar restricts the productions $\alpha \to \beta$ to $A \to \beta$, where *A* is one single non-terminal symbol. The language defined by a type 2 grammar is called a *type 2* or *context-free* language. The context-free grammar is equivalent to a push-down automaton.
If all productions of a context-free grammar are of the form $A \to wB$ or $A \to w$, where *A* and *B* are non-terminals and *w* is a string of terminals, then we say the grammar is right-linear. If all productions are of the form $A \to Bw$ or $A \to w$, we call it left-linear. A right- or left-linear grammar is called a *regular* or *Type 3* grammar. The language defined by *regular expressions* or the *regular grammar* is called *Type 3* language or *regular* language. The regular languages are precisely the ones accepted by finite automata.

Of the four types of formal grammars, regular expressions and context-free grammar have the widest practical use in different areas. Regular expressions particularly have served as useful tools in the design of lexical analyzers, the part of a compiler that groups characters into tokens – indivisible units such as variable names and keywords. A number of compiler-writing systems [Mason & Brown, 1990] automatically transform regular expressions into finite automata for use as lexical analyzers. Context-free grammar has been used widely in the specification of programming languages and even as part of natural languages. In addition, the corresponding pushdown automata have aided in the design of parsers, another key portion of a compiler, which given a grammar *G* and a word *w* can tell if $w \in L(G)$. Thanks to widespread knowledge of a variety of context-free-grammar-based techniques, efficient parsers can be designed relatively easily [Mason & Brown, 1990].

6.2.2 Context-free Grammars and Finite-state Automaton

Context-free grammars (CFGs) are a very important class of grammars because the formalism is powerful enough to describe most of the structure in natural languages and programming languages, and yet is restricted enough so that efficient parsers can be built to analyze sentences.
A context-free grammar G is a 4-tuple (Σ, N, P, S) where Σ is a finite set of terminals, called the alphabet, *N* is a finite set of non-terminals, including the start symbol *S*, and *P* is a finite set of rules, called productions, having the form $A \to D$ where A is an element of N and D is an element of $(\Sigma \cap N)^*$. The language of the grammar G –

L(G) – is the set of all the strings of non-terminals derivable from the start symbol *S* with the derivation rules *R*.

Though not as powerful as context-free grammars, finite-state automata are widely used in the specification of regular expressions such as the lexicon of a programming language.
A finite-state automaton is denoted by a 5-tuple $FSA = (K, \Sigma, \Delta, s, F)$ where K is a finite sets of states, Σ is a finite input alphabet, s in K is the initial state, $F \subseteq K$ is the set of final states, and Δ is the transition function mapping $K \times \Sigma$ to K. So that $\Delta(q, a)$ is a state for each state q and input symbol a. The language accepted by FSA, designated L (FSA), is the set of all string x where $\{x \mid \Delta(s, x) \text{ is in } F\}$.

Undecidable problems of Context-free Grammars
Given two different context-free grammars G1 and G2, the problem of determining whether $L(G1) \subseteq L(G2)$ is undecidable [Theorem 8.12, Hopcroft & Ullman, 1979], in addition to which, the problem of determining if the intersection of L(G1) and L(G2) - $L(G1) \bigcap L(G2)$ is empty is undecidable [Theorem 8.10, Hopcroft & Ullman, 1979]. It is also impossible to judge if the languages described by two context-free grammars are the same - $L(G1) = L(G2)$ [Theorem 8.12, Hopcroft & Ullman, 1979].

6.3 Context-free Grammars in Speech Processing

The grammars in spoken dialog systems serve two purposes: first, they define the natural language used formally in a certain domain and map different natural language user utterances to the representative semantics, which can be further used in the dialog management component; second, they are commonly used for language modeling to improve the quality of speech recognition [Allen et al., 2000; Bui et al., 2005; Dusan & Flanagan, 2000].
Among different types of formal models used in a natural-language processing system, the mostly widely adopted one is the context-free grammar, due to its power of expressing different linguistic structures existing in the natural language. Over the years, an augmentation of context-free grammar – so called “unification grammar” – has been established, which adds the notion of feature constraints to context-free grammar [McTear, 1998]. The adaptation of unification grammar is due to compactness and concision in the definition of the language. Though it is known that in the expressivity, unification grammar is more powerful than context-free grammar,

it is still quite arguable whether the additional expressive power that is gained by unification grammars is actually needed to describe natural languages [Moore, 1999]. So we regard context-free grammar as the most common formal model in speech processing to express and map the natural user utterances to semantics and provide a language model for better speech recognition. In the dialog systems introduced in Allen et al. [2000], Bui et al. [2005] and Dusan & Flanagan [2000], context-free grammars are adopted without any expressive restriction to the necessary natural language.

6.3.1 Approximations of Context-free Grammars

Despite the availability of extensive literature on the topic of efficient context-free parsing for large and very ambiguous grammars, context-free parsing still poses a serious problem in many practical applications such as real-time speech recognition. The human language user seems to process in linear time; humans understand longer sentences with no noticeable delay. This implies that context-free grammars are good and powerful for language interpretation, but are not likely models for human language processing. Therefore, there are different approaches in the academic world for approximating context-free grammars with finite-state devices, which are known to allow very efficient processing in linear time. In practice, these approaches solve the conflict between requirements of language modeling for recognition and of language analysis for sentence interpretation. Current recognition algorithms use the finite-state acceptor language models for computational efficiency. It is known that these models are inadequate for natural language interpretation, since they cannot express all relevant syntactic and semantic regularities. Context-free grammars can express many of those regularities, but are computationally less suitable for language modeling, because of the inherent cost of computing state transitions in their parsers. The approximation of context-free grammars with finite-state devices integrates these two techniques in a single system.
Going by the Chomsky Hierarchy, it is obvious that finite-state devices are not as powerful as context-free grammars. But interestingly, though not mentioned in the literature, I found that at least some of the constructions that cannot be treated with finite-state devices are also difficult for humans. For example, constructions involving center-embedding are very hard to process for humans, but are regarded as grammatical by linguisticians. In English particle verb constructions, the particle can either precede or follow the direct object ("put the book down" / "put down the book"). If the direct object contains a relative clause, and the particle follows the direct object, then the examples become very hard to understand (see Appendix 4 for examples).

If there is no restriction to the amount of center-embedding (recursion), it is equally impossible for finite-state devices to process these sentences. This suggests that finite-state devices could offer language models adequately accounting for the efficiency of human language processing.
Therefore, context-free grammars used in spoken-dialog applications often represent regular languages (which are equivalent to the languages modeled by finite-state automata), either by construction or as a result of a finite-state approximation of a more general context-free grammar.

6.3.2 Different Approximation Approaches

It would be perfect if the context-free grammar (generating actual regular language) could be used as a general form of specification, and an equivalent finite-state automaton could be transformed from it to be used in the recognition process. Unfortunately, there is no general algorithm that would map an arbitrary context-free grammar generating a regular language into a corresponding finite-state automaton. (See Theorem 8.15 in Hopcroft and Ullman [1979])
However, in the existing literature, a number of methods have been proposed for **approximating** a context-free language with a finite-state automaton. Nederhof [2000a] gives a good survey of different approximation methods.
Several of these methods can be categorized into two classes: one class of approaches constructs a pushdown automaton from the grammar, where the language accepted by the automaton is identical to the language generated by the grammar, and then approximates the pushdown automaton with a finite automaton [Johnson, 1998; Grimley Evans, 1997; Pereira & Wright, 1997]. Among them, there are two kinds of pushdown automaton approximation – subset and superset approximation. The subset approximations such as the one of Johnson [1998] reduce the infinite set of stacks in a pushdown automaton to a finite set leading to a finite automaton. The superset-approximations such as the one of Pereira and Wright [1997] build congruence classes of stack symbols and translate each congruence class to a unique state of a non-deterministic finite automaton.
Another superset approximation approach retains only the information about allowable terminals or pairs of adjacent parts of speech (cf. uni-gram, bi-grams, and tri-grams) [Stolcke & Segal, 1994).
A superset approximation based on recursive transition network is introduced in Nederhof [2000a]. The approach constructs a finite automaton for each non-terminal, and builds the complete recursive transaction network by collecting all finite automata of different non-terminals.

Many approximation approaches prove to be exact if the approximating context-free grammar is left-linear or right-linear [Grimley, 1997; Pereira & Wright, 1997; Nederhof, 2000a] Intuitively this can be explained by the fact that a left-linear or right-linear grammar is defined to be regular language and further equivalent to a finite-state automaton.

Furthermore, in Nederhof [2000a] it is proven that context-free grammars that are not self-embedding generate regular languages. According to Chomsky [1959b], a self-embedding grammar is defined as follows:

A grammar is self-embedding if there is some $A \in N$, such that $A \rightarrow^{*} \alpha A \beta$ for some $\alpha \neq \varepsilon$ and $\beta \neq \varepsilon$.

A grammar that is not self-embedding is defined to be *a strongly regular grammar*. [Nederhof, 2000b]. The proof is based on a constructive algorithm mapping a strongly regular grammar into an equivalent finite automaton [Nederhof, 2000a; Nederhof, 2000b]. This proof is not in conflict with the theorem [Theorem 8.15 in Hopcroft & Ullman, 1979], since the condition of strong regularity is a sufficient condition for the language to be regular, but is not a necessary condition. It means there are some grammars which are not strongly regular but which generate regular language. For such grammars, the approximations generate a larger language.

In practice, the finite automata approximation is normally applied to enhance the recognition accuracy. They are used as a frond-end filter to the real parser. So it is allowed that a certain percentage of ungrammatical input is recognized. Also it is allowed that "pathological" grammatical sentences are rejected that seldom occur in practice; an example are sentences requiring multiple levels of self-embedding. According to these practical considerations, most approximations are accepted according to their approximation quality. The most serious problem is actually the complexity of the construction of the automata from the compact representation for large grammars [Nederhof, 2000a].

6.4 Comparison of Context-free Grammars

6.4.1 Similarity of Context-free Grammars

My purpose in comparing grammars is to find out if there is any similarity between two context-free grammars used in spoken dialog system. There are different possible views on this similarity problem:

1. If a standard grammar library is introduced for dialog systems, where standard grammars such as date, time, number etc. are specified formally

and unambiguously, the involvement of the same standard grammar could indicate the similarity of two grammars.

2. A second possible similarity measurement can be based on the semantics of the grammars. Many grammars used in spoken dialog systems also serve to transform natural language phrases into semantic forms. For example, “January first two thousand six” will be transformed to “2006-01-01”. The similarity of two grammars can be assessed by comparing their semantic forms and respectively the value ranges of the semantic forms.
3. The third and the most general similarity measurement can be based on the languages produced by two grammars. Two grammars are similar if their languages are similar.

The first similarity view requires a standard grammar library, which cannot be assumed to be general in practice. The second similarity view assumes that each grammar exhibits a semantic transformation. In practice, however, many grammars appear not to have semantic forms such as the trigger grammars used in DIANE spoken dialog systems. So the most general and appropriate similarity measurement is based on the languages produced by the grammars.

There exist different similarity theories, which are well introduced in Lin [1998]. Adopting the general similarity theory to grammar comparison based on their languages, the similarity of two grammars G1 and G2 would be proportional to the ratio between common expressions existing in L(G1) and L(G2) and all expressions in L(G1) and L(G2).

However, the quantified similarity of two grammars is actually not relevant for me. The grammars used in spoken dialog systems serve for defining semantic structures and concepts of the natural language formally. Concretely, they are used for connecting natural user utterances with formal semantics and each grammar defines in general only one single meaning, the variations of the grammar just representing the manifold natural language representations of the same concept. So the grammar definition defines just different synonym values for a single semantic value. Therefore, and this is a very central observation for my approach, grammars can be assumed to define the same semantic, if there is at least one common expression in their languages.

For example, a grammar defining key words for a transaction “flight reservation” would involve phrases such as “book a flight”, “make a flight reservation”, etc. Another grammar also defining key expressions for a transaction “flight reservation” may consist of phrases as “buy a flight ticket” “book a flight”, “reserve a flight”. Though they do not have identical expressions in their languages but rather only one common expression, they are similar, or we can even say they are identical. The different expressions are just synonyms for the same semantic, which is identified by the common expression “book a flight”.

Therefore, I define the similarity of grammars as the intersection of the languages defined by them. Two grammars are similar if and only if there is at least one common expression accepted by both grammars. That means if we can find a common expression in their languages, we can determine that the grammars are similar. Two grammars are different if and only if there is no common expression defined by them. It means if we cannot find any common expression in the defined languages, the grammars are different. This is possible only if the defined language is finite. If the language is not finite, it is necessary to approximate the language for determining the sameness of the grammars.

This similarity definition might not suffice for parameter grammars. In a parameter grammar, different parameter values are defined with natural language. It means the language defined by a parameter grammar is a set of synonyms of different values. Therefore, the probability for a common expression to exist in the languages of two parameter grammars is relatively high and can cause wrong decisions due to the ambiguity of the natural language. I discuss solely the solution for finding one common expression in two grammars, and in practice, this solution can be easily adapted to find enough common expressions sufficing a minimal threshold. For example, at least 5 common expressions are required, or, common expressions must comprise at least 10% of the all expressions defined in the grammar.

Using the symbol $Sim(A,B)$ to express that grammar A and B are similar and I define the similarity between two context-free grammars G1 and G2 as the following:

$$L(G1) \bigcap L(G2) \neq \phi \rightarrow Sim(G1, G2)$$

Two context free grammars are similar if there is at least one common expression existing in the languages defined by both grammars.

6.4.2 Theoretical Considerations of Comparing Context-free Grammars

According to Hopcroft & Ullman's [1979] Theorem 8.10, the problem of determining if the intersection of L(G1) and L(G2) is empty - $L(G1) \bigcap L(G2) = \phi$ - is undecidable. This means, without any approximation or restriction on the context-free grammars, it is impossible to answer the similarity question.

When considering different existing approaches approximating a context-free grammar with a finite-state automaton, the following idea is interesting: we could first

approximate a context-free grammar with a finite-state automaton or regular languages, and then compare the finite-state-automaton or regular languages. The intersection problem then turns into a solvable one.
Based on different approximations, different comparison mechanisms could be considered. I distinguish two categories of approximations: approximation with a finite-state automaton, and N-gram approximation.

Comparison based on finite-state automaton approximation

By constructing the pushdown automaton for a context-free grammar G, and further approximating the pushdown automaton with a finite automaton or based on recursive transition network technology, for each grammar G, a finite-state automaton FSA could be constructed approximating G.
Depending on the regularity of G and the methods applied for approximation, we can distinguish between superset approximation $L(FSA) \supset L(G)$, subset approximation $L(FSA) \subset L(G)$ and equivalent transformation $L(FSA) = L(G)$.
If G is defined as a strongly regular grammar (without self-embedding), which is also reasonable from the point of view of human language processing [Miller & Chomsky, 1963; Chomsky, 1963], we can construct an equivalent finite-state automaton for G [Nederhof, 2000a]. Given two finite-state automata FSA1 and FSA2 modeling strongly regular context-free grammars G1 and G2, it is feasible to build an automaton FSA so that $L(FSA) = L(FSA1) \bigcap L(FAS2)$. The construction of the intersection is to be found in formal language theory textbooks [Hopcroft & Ullman, 1979, p.59]. Further, it is decidable whether the language produced by a finite-state automaton is empty [Hopcroft & Ullman, 1979, p.63-64]. An algorithm based on the reach ability of states within an automaton starting from the start state is introduced by Hopcroft and Ullman [1979, p.63-64]. The automaton accepts an empty language if, and only if, no final state is within the set of reachable states.
So we can determine whether two strongly regular grammars are similar by the fact that the intersection automaton of the finite-state automata modeling these two grammars either does or does not produce an empty language.

Without any restriction on G1 and G2, instead of testing equivalence it is only possible to compute an approximating finite-state automaton FSA1 and FSA2.

If we have a superset approximation ($L(FSA) \supset L(G)$), the finite-state automaton intersection is sufficient but not necessary for the intersection of context-free grammars. The following consequence illustrates the soundness:

$L(FSA1) \bigcap L(FSA2) = \phi \rightarrow L(G1) \bigcap L(G2) = \phi$.

If the intersection of L(FSA1) and L(FSA2) is empty, we can be sure L(G1) and L(G2) have no overlaps. But conversely, if the intersection of L(FAS1) and L(FSA2) is not empty, we cannot tell whether the intersection comes from the original grammars or the imprecise approximation, so we cannot make any judgement about the intersection of L(G1) and L(G2).

If we have a subset approximation ($L(FSA) \subset L(G)$), the finite-state automaton intersection is necessary but not sufficient for the intersection of context-free grammars. The following conclusion from the intersection of L(FSA1) and L(FSA2) can be drawn:

$L(FSA1) \bigcap L(FSA2) \neq \phi \rightarrow L(G1) \bigcap L(G2) \neq \phi$

If the intersection of L(FSA1) and L(FSA2) is not empty, we can tell that L(G1) and L(G2) have overlaps with each other. But by contrast, if the intersection is empty, we cannot be sure if we have reduced the intersection of two grammars in the approximation, so we cannot judge the intersection of L(G1) and L(G2).

Interestingly, if we combine the superset and subset approximations, we get correspondingly sufficient conditions for empty and non-empty intersections of context-free grammars G1 and G2. Namely:

$L(SupersetFSA1) \bigcap L(SupersetFSA2) = \phi \rightarrow L(G1) \bigcap L(G2) = \phi$

$L(SubsetFSA1) \bigcap L(SubsetFSA2) \neq \phi \rightarrow L(G1) \bigcap L(G2) \neq \phi$

With such a combination, we could judge the intersection of two context-free grammars in most cases up to one, namely when the intersection of their supersets is non-empty but the intersection of their sub-sets is empty. Because it is non-transparent how an approximation actually changes the language [Nederhof, 2000b], at the moment we are not able to answer the intersection question in this case. Though this is an interesting point to be researched further, a deep investigation of this question would go beyond the scope of this dissertation. Second, presumably all approximations strive to be as close as possible to the original grammars; we could intuitively suppose the probability of this case is quite rare in practice.

Comparison based on N-gram Approximation

The N-gram approximation translates a context-free grammar G, with an alphabet E, into a list of phrases consisting of N symbols from E. The transformation is a pure “approximation” without the possibility of “equivalence”. Based on an N-gram approximation N1 and N2 of two context-free grammars G1 and G2, N1 and N2 can be compared with each other literally. The set N0 of identical N-grams can be easily found. With this consideration, the ratio R ($0 \leq R \leq 1$) of the size of N0 and summary of the sizes of N1 and N2 could give evidence for the similarity of L(G1) and L(G2). Applying a reasonable (better statistically counted) threshold for the relation R, a similarity between G1 and G2 could be derived from R. E.g. $R > 0.8 \rightarrow Sim(G1, G2)$. The challenge in this approach is to determine the right threshold for R, which might not be constant, but rather inconstant according to different grammars.

The similarity could be assessed by either of the comparison models introduced above. However, this thesis aims to find one approach which fits best for the grammar formalism used in DIANE.

Recalling the question, we want to know if two context-free grammars G1 and G2 are similar where this similarity is defined to be a yes-no answer to the question of whether the languages defined by both grammars contains at least one identical expression. Therefore, the comparison approach *A* should satisfy the following requirements:

Soundness: $A(G1, G2) = Yes \rightarrow Sim(G1, G2)$

Necessity: $Sim(G1, G2) \rightarrow A(G1, G2) = Yes$

Efficiency: I assess the efficiency of an algorithm by its time complexity, which is represented by the number of steps that the algorithm takes to solve the problem as a function of the size of the input.

In the next section, I will introduce the grammar formalism used in this system. I will also discuss the suitability of applying the different possible comparison approaches to this type of grammar according to the requirements.

6.5 Comparison of DIANE Grammars

For the context-free grammar (Σ, N, P, S) used in DIANE the following condition holds:

Let $N' = \{A \in N \mid A \rightarrow^{*} \alpha A \beta\}$

$\forall A \in N'[A \rightarrow^{*} \alpha A \beta \wedge \alpha \neq \varepsilon \wedge \beta = \varepsilon]$

What this actually means is that right recursion is the only way allowed in a DIANE grammar to express recursion in the corresponding language. This class of grammar is equal to the right-recursive grammar defined in Nederhof [2000a]. “Self-embedding” “cyclic” and “left recursion” [Nederhof, 2000a] is not allowed in DIANE grammar. Obviously, this class of grammar is strongly regular, as defined in Nederhof [2000a], and generates only regular languages.

In practice, the grammars used in other dialog systems [JSGF, 1998; Hunt & McGlashan, 2004] also exploit only the regular expressive power of a context-free grammar. So we can say that in general it is sufficient to consider context-free grammars used in different dialog management systems as strongly regular [Nederhof, 2000a]. The theoretical argument is that the expressive power of Chomsky-2 grammar is not needed in human-computer interaction (see Section 6.3). Therefore, the contributions in this section can be actually applied to all spoken dialog systems, and I just take DIANE grammars as example for discussion.

Strongly regular grammars can be modeled by finite-state automata [Nederhof, 2000a; Mohri & Nederhof, 2001], and they can also be approximated by N-gram models. Therefore, all theoretically-discussed approaches in the last section can be applied to DIANE grammars. I will discuss two comparison approaches for comparing two DIANE grammars.

6.5.1 Comparison of DIANE Grammars Based on Finite-state Modeling

Based on theories in the area of finite-state automata and context-free grammars, I propose a comparison approach which first models a context-free grammar with a finite-state automaton, then constructs the intersection automaton of the two generated constructed automata, and finally checks if the intersection automaton generates empty language. If the intersection finite-state automaton generates empty language, the compared grammars are different. Otherwise, they have at least one common expression, and are similar according to my definition.

Figure 6-1 illustrates this comparison algorithm:

Given two DIANE grammars G1 and G2
Run procedure FA (G1, G2)
Return Boolean Similarity of G1 and G2

Procedure FA:

```
Input: DIANE grammars G1 and G2
Output: Sim – the Boolean value representing Sim(G1,G2).

FSA1 = MAKE_FSA(G1)
FSA2 = MAKE_FSA(G2)
FSA=MAKE_INTERSECTION(FSA1,FSA2)
IF IS_EMPTY(FSA) THEN return false  ELSE return true
```

Figure 6-1 DIANE grammar comparison algorithm based on finite-state modeling

This algorithm contains three sub-procedures. They are:

- MAKE_FSA for constructing a finite-state automaton modeling a given DIANE grammar
- MAKE_INTERSECTION for constructing the intersection of two finite-state automata
- IS_EMPTY for determining if a given finite-state automaton produces an empty language.

There exist different approaches for each sub-procedure. In the following I briefly introduce a variant algorithm for each sub-procedure.

Finite-state modeling of DIANE grammar

I have referred to the algorithm introduced in Nederhof [2000a] for modeling a strongly regular grammar. This is applicable to strongly regular grammars, so I have just adapted it and taken only the part needed for non-recursive grammars and grammars with only right recursion (non-left-recursive and non-center-embedding). The following pseudo-code illustrates this procedure.

```
Procedure MAKE_FSA:
Input: DIANE grammar $G = (\Sigma, N, P, S)$
Output: An equivalent finite-state automaton $FSA = (K, \Sigma, \Delta, s, F)$
Let $K = \phi$, $\Delta = \phi$, $s = fresh\_state$, $f = fresh\_state$, $F = \{f\}$;
 $make\_fa(s, S, f)$
Pre-procedure for determining the sets of mutually recursive non-terminals:
Determine the set of all recursive non-terminals as:
        $\overline{N} = \{A \in N \mid \exists \alpha, \beta[A \rightarrow^{*} \alpha A \beta]$
Determine the partition N of $\overline{N}$ to be the sets of mutually recursive non-terminals:
        $N = \{N_1, N_2, ..., N_k\}$
```

$N_1 \cup N_2 \cup ... \cup N_k = \overline{N}$

$\forall i[N_i \neq \phi]$

$\forall i, j[i \neq j \Rightarrow N_i \cap N_j = \phi]$

And for all $A, B \in \overline{N}$ *:*

$\exists i[A \in N_i \wedge B \in N_i] \Leftrightarrow \exists \alpha_1, \alpha_2[A \to^* \alpha_1 B \wedge B \to^* \alpha_2 A]$

```
Procedure $make\_fa(q_0, \alpha, q_1)$:
  IF $\alpha = \varepsilon$
  THEN let $\Delta = \Delta \cup \{(q_0, \varepsilon, q_1)\}$
   ELSEIF $\alpha = a$, some $a \in \Sigma$
     THEN let $\Delta = \Delta \cup \{(q_0, a, q_1)\}$
ELSEIF $\alpha = X\beta$, some $X \in V, \beta \in V^*$ such that
THEN let $q = fresh\_state$;
          $make\_fa(q_0, X, q)$;
          $make\_fa(q, \beta, q_1)$;
ELSE let $A = \alpha$; ($\alpha$ must consist of a single non-terminal)
        IF there exists i such that $A \in N_i$
        THEN FOR EACH $B \in N_i$ do let $q_B = fresh\_state$ END FOR
                  FOR EACH $(C \to X_1 ... X_m) \in P$ such that $C \in N_i \wedge X_1, ..., X_m \notin N_i$
                  Do $make\_fa(q_C, X_1 ... X_m, q_1)$
                 END FOR
              FOR EACH $(C \to X_1 ... X_m D) \in P$ such that $C, D \in N_i \wedge X_1, ..., X_m \notin N_i$
                  Do $make\_fa(q_C, X_1 ... X_m, q_D)$
              END FOR
           END FOR
          Let $\Delta = \Delta \cup \{(q_0, \varepsilon, q_A)\}$
     ELSE for each $(A \to \beta) \in P$ do $make\_fa(q_0, \beta, q_1)$
     END IF
END IF
END Procedure
```

```
Procedure $fresh\_state()$:
  Create some object q such that $q \notin K$;
  Let $K = K \cup \{q\}$;
  Return q
END Procedure
```

Figure 6-2 Transformation from a strong regular grammar into an equivalent finite automaton
Source: Nederhof [2000a]

An example of constructing the equivalent finite-state automaton for a small grammar is given in Figure 6-3.

$S \rightarrow aA$ $\qquad \overline{N} = \{S, A, B\}$

$A \rightarrow BS$ $\qquad N = \{N_1, N_2\}$

$A \rightarrow bB$ $\qquad N_1 = \{S, A\}$

$B \rightarrow cB$ $\qquad N_2 = \{B\}$

$B \rightarrow d$

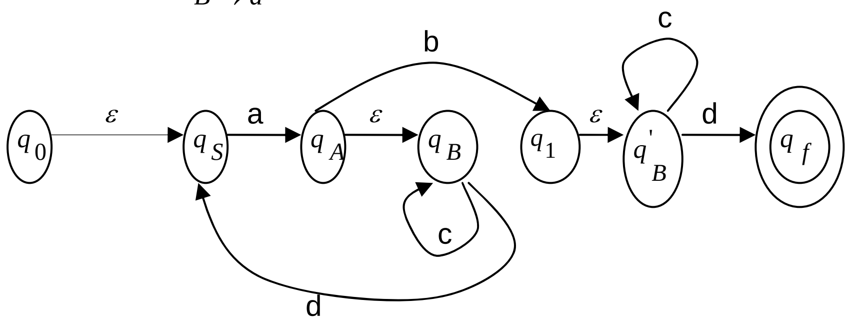


Figure 6-3 Application of the algorithm from Figure 6-2 on a small grammar

The states number of the resulting finite-state automaton could be exponential in the original grammar size, if the grammar is descended in all ways [Nederhof, 2000b]. But in Nederhof [2000a] a compact representation for finite-state automata has been proposed to avoid the exponential behavior and keep the complexity polynomial. It means given a grammar of size n, the approximation procedure may produce n^m states. Based on different empirical tests introduced in Nederhof [2000a], m can be assessed to be definitely greater than 1.

Intersection of finite-state automaton

The intersection of two deterministic finite-state automata can be constructed. The algorithm for constructing this intersection finite-state automaton is introduced in Hopcroft and Ullman [1979] . It is based on the idea of taking the Cartesian product of states and building the transitions appropriately. The following figure illustrates this algorithm.

Procedure MAKE_INTERSECTION

Input: Finite-state automata $FSA_1 = (K_1, \Sigma_1, \Delta_1, s_1, F_1)$ *and* $FSA_2 = (K_2, \Sigma_2, \Delta_2, s_2, F_2)$

Output: Finite-state automaton FSA as Intersection of FSA_1 *and* FSA_2 *so that* $L(FSA) = L(FSA_1) \bigcap L(FSA_2)$

$FSA = (K_1 \times K_2, \Sigma_1 \bigcap \Sigma_2, \Delta, [s_1, s_2], F_1 \times F_2)$

$K_1 \times K_2$ *- Cartesian product of states of* FSA_1 *and* FSA_2

$\Sigma_1 \bigcap \Sigma_2$ *- Intersection alphabet of both original automata*

$[s_1, s_2]$ *- Cartesian product of the initial states of both original automata*

$F_1 \times F_2$ *- Final states of the intersection automaton, which could be any Cartesian product of any valid final states of the original automata.*

For all q_1 *in* K_1*,* q_2 *in* K_2*, and a in* $\Sigma_1 \bigcap \Sigma_2$*,*

$\Delta([q_1, q_2], a) = [\Delta_1(q_1, a), \Delta(q_2, a)]$

Figure 6-4 Algorithm for determining the intersection of two finite-state automata

The size of the intersection automaton is equal to $|K_1| \times |K_2|$. So the complexity of this algorithm is $O(n^2)$.

Emptiness problem of deterministic finite-state automaton

The problem of whether the language produced by a finite-state automaton is empty is decidable [Hopcroft & Ullman, 1979, p63]. The algorithm introduced by Hopcroft and Ullman [1979] deletes all states that are not reachable by any input from the start state. If one or more final states remain, the language is non-empty. I illustrate this algorithm in the following figure:

```
Procedure IS_EMPTY
Input: finite-state automaton $FSA = (K, \Sigma, \Delta, s, F)$
Output: Boolean value indicating if the language produced by FSA is empty

Begin
  Let $R = \phi$ $L = \phi$
  Mark s
 $R = R \bigcup \{s\}$
Add s to the end of L
WHILE L is nonempty
```

```
    Choose one state q from the front of L
    IF for some input a, $q' = \Delta(q, a)$ is unmarked
    THEN Mark q'
        IF $q' \in F$ terminate the procedure and return true END IF
        Add q' to the end of L
        Add q' to R
        Remove q from L
    ELSE remove q from L
  END IF
END WHILE
IF $R \cap F \neq \phi$ return false ELSE return true END IF
```

Figure 6-5 Algorithm for determining the emptiness of a finite-state automaton

This algorithm is derived from the breadth-first search of graph traversal algorithms. In the worst case, $|\Sigma| \times |K|$ steps are needed for computation with this algorithm. So this procedure has the complexity of $O(n)$.

The complexity of the comparison algorithm can be calculated as the following:

For a given right-linear grammar of size n, the approximating finite-state automaton has a size of $n^m (m > 1)$.

Then the intersection procedure has a complexity of $O(n^{2m})$ with the original grammar size n.

The emptiness procedure has a linear complexity, so the summarized complexity of the whole comparison process can be assessed to be $O(n^{2m})$.

The number of states in a finite-state automaton modeling a grammar is obviously more than the number of productions in the grammar, so the complexity can be assessed to be greater than $O(n^2)$. This complexity is more or less acceptable in practice. In the next section, I introduce another algorithm considering the similarity problem from the beginning and thus working more efficiently.

6.5.2 Generation and Parsing-based Grammar Comparison

The algorithm based on finite-state automaton approximation introduced above is rather complicated. I investigate another simpler approach in this section. It is based on the consideration of how the common expression indicating the grammar similarity can be found most efficiently.

We can easily determine the intersection of two grammars G1 and G2 by parsing all expressions of L(G1) with the parser of G2 (P(G2)), or vice versa. If any generated expression is accepted by the parser, the two grammars accept some common expression, and thus have an intersection. Obviously, this basic algorithm is extremely inefficient for grammars which generate a large number of expressions and is not applicable to grammars which generate infinite language.

In this section, I propose a comparison approach based on generation and parsing, which solves the problems of large size and infinite grammars. This approach seems to be more efficient than the first comparison approach because of the fact that the grammars are compared at the beginning. The complexity spent in constructing "disjunctive" parts of finite-state automata can be therefore saved in this approach. The approach proposed in this section is inspired by two different types of production rules used for specifying context-free grammars. In the following, I first introduce the classification of production rules, and based on it I give the basic principle for the comparison.

Enumeration rules vs. derivation rules

A grammar defines its language using different production rules. With respect to the way of how a production rule is defined, production rules can be categorized into *enumeration rules* and *derivation rules*.

An *enumeration rule* $A \rightarrow \alpha$ provides possible expressions for a non-terminal symbol, where A is a non-terminal symbol, and α is a string of terminal symbols.

A *derivation rule* $A \rightarrow z$ gives the possible derivation ways for a non-terminal symbol, in the definition A is a non-terminal symbol, and z is a string of non-terminal and terminal symbols.

Possible expressions for a non-terminal symbol are enumerated by enumeration rules explicitly, whereas production rules define the way to calculate the expressions for a non-terminal symbol implicitly.

In practice, derivation rules are used for defining the recursive part of a grammar since it is impossible to enumerate infinite expressions. Also, for languages with certain semantic structures and/or large size, derivation rules are adopted, e.g. grammars for date and time. Otherwise, if the language to be defined has a relatively limited size, or there is no semantic structure inside the language which means that the expressions defined by the grammar are not decomposable, enumeration rules are used for defining the atomic expressions.

Obviously, each grammar must have enumeration rules, or else it would be impossible to determine any valid expressions generated by the grammar (assuming that there is no ε production allowed in the grammar).

Grammar comparison based on enumeration and derivation

The grammar comparison task can be split into two sub-tasks – comparison of atomic “sub-expression” (enumeration rules) and comparison of patterns (derivation rules). The idea is that the identical “sub-expressions” in two grammars can be first determined; then the potential identical expressions can be generated with the patterns defined in each grammar; finally the potential identical expressions can be parsed by the parsers of each grammar. If any expression is accepted by the other parser, a “common expression” has been found, and the grammars have any intersection. If no expression is accepted by the parser of the other grammar, there is no possible “common expression”, and the grammars have no intersection.

The following algorithm illustrates the process of my comparison algorithm:

```
Procedure FA:
Input: DIANE grammars G1 and G2
Output: Sim – the Boolean value representing the similarity of G1 and G2

Grammar G1’ = REDUCTION(G1)
Grammar G1”=APPROXIMATION(G1’)
L(G”)=GENERATION(G1”)
Return PARSE(L(G”), G2)

Procedure PARSE
Input: Language L(G1) and DIANE grammars G2
Output: a Boolean value indicating if there is any expression w in L(G1) so that $w \in L(G1)$

Parser P = Generate a parser for G2
FOR EACH $w \in L(G1)$ Do IF P(w) =true THEN return true ENDIF
End FOR
Return false
```

Figure 6-6 DIANE grammar comparison algorithm based on generation and parsing

This algorithm contains four sub-procedures. They are:
REDUCTION for reducing the grammar for potential grammar intersection
APPROXIMATION for handling the recursion in a grammar
GENERATION for generating all potential overlap expressions of both grammars
PARSE for parsing one grammar G1 - generated expressions with a parser for the grammar G2

In the following I give the details of each step.

6.5.2.1 Reduction of enumeration rules

Atomic “sub-expressions” are defined by enumeration rules in a grammar. Different enumeration rules for one “non-terminal” symbol refer to the same semantic category. E.g. the following grammar defines natural language for date:

$DATE \rightarrow DAY \quad MONTH$

$DAY \rightarrow First \mid Second \mid ... \mid Thirty - First$

$MONTH \rightarrow January \mid February \mid ... \mid December$

In this example, a “DAY” expression ranges from “first” to “thirty-first”. Suppose there is another grammar which is being compared with this grammar to determine their similarity, all enumerations for “DAY” can be reduced to one single representative instance, if different instances of “DAY” all correspond to the same semantic category in the other grammar. This is similar to different transitions between two states in a finite-state automaton. The transitions could be various, but the destination state is always the same. In addition, different values of the same semantic category cannot affect the state transition in a finite-state automaton. In other words, assuming that “first” and “second” correspond to an identical semantic category in another grammar G, if “first January” will not be accepted by G, “second January” will not be accepted either. So with respect to a single non-terminal, when comparing with another grammar G, different enumerations of it corresponding to a single semantic category of G will always produce the same similarity result. We could remove all “redundant” enumerations from “DATE” grammar, so that the language generation becomes much simpler. If no identical expression is found in G2, all enumeration rules of “DAY” will be removed from G1 before language generation. This is because that “DAY”-expressions are not in the vocabulary of G2.
I assume the grammars are all in Chomsky normal form [Hopcroft & Ullman, 1979, p. 94-96]. The following figure illustrates the reduction process:

```
Procedure REDUCTION (G1, G2)
Input: Context-free grammars G1, G2 in Chomsky normal form: $G_1 = (\Sigma_1, N_1, P_1, S_1)$ and
$G_2 = (\Sigma_2, N_2, P_2, S_2)$
Output: Reduced context-free grammar $G_1 = (\Sigma_1, N_1, P_1, S_1)$

BEGIN
  Let $W = \phi$
```

```
 FOR each $A \to a \in P_1$, where $A \in N_1$, $a \in \Sigma_1$ DO
 Let f = false
 FOR each $B \to b \in P_2$ DO
        IF a=b
        THEN
              f=true
              IF $(A,B) \in W$
              THEN remove $A \to a$ from $P_1$
              ELSE $W = W \bigcup \{(A,B)\}$
              END IF
     END IF
END FOR
IF f = false, THEN remove $A \to a$ from $P_1$ END IF
END FOR
END
```

Figure 6-7 Algorithm for reducing a context-free grammar

This algorithm has a complexity of $O(n^2)$ ($n_1 \times n_2$) in the worst case.

After reduction of different enumeration rules, the size of the grammar is in most practical cases more moderate, so all expressions can be generated relatively efficiently.

6.5.4.2 Approximation of recursive rules

The problem of infiniteness is not solved yet by the reduction of enumeration rules since the recursiveness of a grammar is caused by derivation rules. In this case, I propose to approximate the recursion with only finite loops. For flexibility, I introduce a recursion parameter j for the approximation process to indicate the unfolding levels of applications of rules. A recursion parameter j with value 1 means no recursive loop. The following figure describes my approximating algorithm. In this algorithm, recursive non-terminals are replaced by new non-recursive non-terminals, which define the finite (j-1) recursive loops.

Procedure APPROXIMATION (G, j)

Input: Context-free grammars G: $G = (\Sigma, N, P, S)$ *and Integer j for recursive levels*

Output: Approximated non-recursive context-free grammar $G = (\Sigma, N, P, S)$

Pre-procedure for determining the sets of mutually recursive non-terminals:

Determine the set of all recursive non-terminals as:

$$\overline{R} = \{A \in R \mid \exists \alpha, \beta[A \rightarrow^{*} \alpha A \beta]$$

Determine the partition N of $\overline{N}$ *to be the sets of mutually recursive non-terminals:*

$$R = \{R_1, R_2, ..., R_k\}$$

$$R_1 \cup R_2 \cup ... \cup R_k = \overline{R}$$

$$\forall i[R_i \neq \phi]$$

$$\forall i, j[i \neq j \Rightarrow R_i \cap R_j = \phi]$$

And for all $A, B \in \overline{R}$ *:*

$$\exists i[A \in R_i \wedge B \in R_i] \Leftrightarrow \exists \alpha_1, \beta_1, \alpha_2, \beta_2[A \rightarrow^{*} \alpha_1 B \beta_1 \wedge B \rightarrow^{*} \alpha_2 A \beta_2]$$

```
BEGIN
  FOR each $1 \leq i \leq k$ DO
     Assign an ordering ( $A_1, A_2, ..A_i, A_{i+1}, ..., A_n$ ) to all recursive non-terminals of $R_i$ :
          $\forall A_i[A_i \rightarrow \alpha A_{i+1} \beta \in P]$
           $i < m \rightarrow A_i < A_m$
           FOR each $A \in R_i$ DO
                   FOR $1 \leq h \leq j$ DO
                   $N = N \cup \{A[h]\}$
         END FOR
   END FOR
    FOR each $A \rightarrow X_1 ... X_m \in P \wedge A \in R_i$
      FOR each $1 \leq h \leq j$ DO
          $P = P \cup \{A[h] \rightarrow X_1' ... X_m'\}$ Where
             $X_k' = X_k[h+1]$, if $X_k \in R_i \wedge A \geq X_k \wedge h < j$
                   $= X_k[h]$, if $X_k \in R_i \wedge A < X_k$
                   $= X_k$, otherwise
     END FOR
    Remove $A \rightarrow X_1 ... X_m$ from P
  END FOR
   FOR each $A[j] \rightarrow X_1 ... X_m \in P$ DO
           IF $X_{n(1 \leq n \leq m)} \in R_i$ THEN Remove $A[j] \rightarrow X_1 ... X_m$ from P
  END FOR
 FOR each $A \rightarrow X_1 ... X_m \in P \wedge A \notin R_i$ DO
```

```
            Remove $A \to X_1 ... X_m$ from P
            $P = P \cup \{A \to X_1' ... X_m'\}$ Where
                        $X_k' = X_k[1]$, if $X_k \in R_i$
                              $= X_k$, otherwise
   END FOR
   IF $S \in N_i$, $S = S[1]$
END
```

Figure 6-8 Algorithm for approximating a recursive context-free grammar with an arbitrary amount of recursion

For the sake of understanding, I give the following grammar as example:

$S \to aA$

$A \to bB$

$B \to cS$

$B \to c$

The mutually recursive non-terminals are $\{A, S, B\}$.

We can assign the order $\{S, A, B\}$ for these three non-terminals, so that $S < A < B$.

After approximation with the parameter j to be 2, the grammar becomes:

$S \to aA[1]$ $S[2] \to aA[2]$

$A[1] \to bB[1]$ $A[2] \to bB[2]$

$B[1] \to cS[2]$

$B[1] \to c$ $B[2] \to c$

Normally an approximation up to the first recursion level with a recursion parameter 2 is adopted. Obviously, the language is approximated to a smaller set. But we are aware of the fact that natural language used in spoken dialog systems is rather defined in a simple way, and recursive grammars are actually only used to define sequences of numbers or letters. So if the approximations do not have any overlaps in their language, it can be asserted that the recursion with more loops will not bring any overlaps either.

For example, let us assume that a grammar defines all possible lists of integer numbers. The one-loop approximation of it defines all lists of integer numbers with a length up to 2. If another grammar also defines integer lists, it will surely accept lists with 2 numbers as well. So in such a case, the algorithm terminates with the right result. If another grammar does not define integer lists, it accepts no expressions in the approximation grammar, and the result is also correct. The only “real” approximation in the algorithm is that the other grammar also defines integer lists but with a length of more than 2. In this case, the algorithm will not be able to find the actual overlap with 3 integers in a list due to the approximation. This problem can be solved by adopting a greater j (i.e. a recursion level up to 5 can be taken). The result will then be more precise, but the calculation would be less efficient. So there is always a trade-off here between precision and efficiency.

In real life, there are seldom such complicated situations, and as mentioned in Section 6.3 the natural language specification in a spoken dialog system is usually simple, so I recommend an approximation with a maximal recursion to the second level with awareness of the offset in very special cases.

According to my thorough experiments this approximation has not affected any comparison result.

6.5.4.3 Generation and Parsing

The generation and parsing steps are straightforward. Based on the reduced grammar G1, all possible identical strings are generated.

The following algorithm is used in my prototype for language generation:

```
Given the start symbol A, Grammar G
Run procedure p(S, G)
Return L(G)

Procedure P:
Input: symbol S, Grammar G
Output: RESULT – the set of all strings derivable from S.
Let R be all Rules in G with S at the left-hand-side.
FOR each rule r1 in R:
  Get all symbols in the right-hand-side of r1.
  FOR i=0;i<length(r1);i++;
    Symbol B = the i-th symbol in r1.
    IF B is a terminal symbol
    THEN IF RESULT is empty
          THEN
                Construct a new empty string STRING
                Add B to STRING
                Add STRING to RESULT
         ELSE add B to the end of each string in RESULT
         END IF
     ELSE run procedure P(B,G) = SUBSTRING
    IF RESULT is empty
    THEN
        RESULT = SUBSTRING
    ELSE
Construct $|RESULT| \times |SUBSTRING|$ new strings by adding each string in SUBSTRING to the end of each string in RESULT.
        RESULT = the set of all new constructed strings
     END IF
    END IF
  END FOR
END FOR
```

Figure 6-9 Algorithm for generating the language defined by a context-free grammar

The complexity of this algorithm is $O(n^2)$.

After potential expressions generation, each expression will be checked by the parser of the other grammar G2. The parser for a grammar can be automatically generated by a parser generator [Mason & Brown, 1990]. For parsing, CKY algorithm [Kasami, 1965] and Early's algorithm [Early, 1970] are most well-known. Both have a complexity of $O(n^2 w^3)$ with n to be the size of the grammar and w to be the size of the input string. Of course there are also more efficient parsing (but more restricted in the form of grammars) algorithms such as SLR, LALR [Taylor, 2000] with a complexity of $O(nw)$.

6.5.4.4 Complexity of the algorithm

The complexity of the whole algorithm can be calculated with the following form:

$O(n^2)$ *(Reduction)* + $O(n)$ *(recursion approximation)* + $O(n^2)$ *(generation)* + $O(n)$ *(Parsing) (with n to be the size of the grammar)*

Thus the complexity of this algorithm is $O(n^2)$, which can be more or less accepted in practice. Compared with the first algorithm based on finite-state modeling $O(n^{2m})$, this approach achieves certain efficiency.

However, it must be stated that the number of generated strings is not of concern in this theoretical complexity calculation, though they may influence the efficiency greatly. This is because after reduction, the number of generated strings becomes moderate in all practical cases I have evaluated. This comes from the fact that grammars used in spoken dialog systems tend to express relatively simply structured semantics. In other words, the semantics expressed in a grammar tend to be atomic, so that enumeration rules are mainly used in grammar specification. The grammars do not normally involve several semantics in one grammar category.

6.5.4.5 Soundness and completeness of the algorithm

Comparing grammar G1 and G2 with the algorithm described above, all strings S generated by the reduced grammar can be generated by the original grammar G1. If any string S is accepted by the parser of G2, S is in L(G2). Thus if the algorithm finds one expression w, so that $w \in L(G1) \wedge w \in L(G2)$, it can derive $L(G1) \bigcap L(G2) \neq \phi$. An extensive mathematical proof of the correctness of the algorithms introduced in this section would be superfluous and goes also far beyond the scope of this dissertation. Therefore, based on the assumption that all the algorithms work appropriately and are correct, I can declare that my algorithm based on generation and parsing is sound.

Despite the approximation of recursion, the reduction does not affect the state transition of the parser because only “identical transitions with respect to state transition” are removed from the original grammar. Thus the algorithm is complete in most cases except certain special situations with recursive grammars, which are not usual in natural language processing in spoken dialog systems. I abandon a complete mathematical proof for this issue for the same reason as that described above.

6.6 Summary

There are few contributions in the existing literature discussing the comparison of context-free grammars. My best efforts found only one approach introduced by Nederhof and Satta [2002] which describes an algorithm to determine if the intersection of two non-recursive context-free grammars G1 and G2 is empty. A finite-state automaton FSA1 is constructed for one context-free grammar G1. An inference mechanism is controlled by both the transitions of the finite-state automaton FSA1 and the rules of the other context-free grammar G2. The inference rules allow only transitions, which are also allowed by productions of the other context-free grammar. So if the inference mechanism can derive the final state of FSA1 from the initial state of FSA1 and the start symbol of G2, the intersection between L(G1) and L(G2) is not empty [Nederhof & Satta, 2002]. This is an interesting approach, though it only applies to non-recursive context-free grammars.

In this chapter two comparison approaches have been proposed. The finite-state-modeling-based approach uses the established models in formal language and methods for regular approximating of context-free grammars. The generation-and-parsing-based comparison approach uses mature parsing and generating techniques for context-free grammars. The heuristic of the second approach is the reduction and approximation of complicated grammars, which could otherwise generate thousands of expressions. After reduction the number of generated expressions is cut down enormously, according to my experiments.

Table 6-1 gives an overview of the criteria fulfilled by the two approaches provided in this section - the finite-state-automaton-based comparison approach (FSAC) and the generation-and-parsing-based comparison approach (GPC) - and also by the approach for parsing non-recursive context-free grammars introduced by Nederhof and Satta [2002] (NS). The complexity of the parsing algorithm is not mentioned in Nederhof and Satta [2002], but it has been mentioned that the size of inference items could be exponential to the grammar size. The column "right-recursion" indicates if grammars with right recursion are accepted by the approach.

	Soundness	**Completeness**	**Complexity**	**Right-Recursion**
FSAC	yes	Yes	$O(n^{2m})(m>1)$	Yes
GPC	yes	y/n	$O(n^2)$	Yes
NS	yes	Yes	unknown	No

Table 6-1 Criteria check of comparison approaches

Both FSAC and GPC approaches provide the right results in cases of assessing two grammars as similar.

The approach FSAC is also complete, meaning similar grammars are always assessed as similar by the algorithm.

The GPC approach does not give 100% completeness because of the approximation of recursions. Nevertheless, it suffices for applications in area of spoken dialog systems, because humans tend to produce natural utterances with much more simple structures for less cognitive loads. Both algorithms provide the right result in case of declaring two grammars as similar and both algorithms are capable of handling right-recursive grammars.

The FSAC has a higher complexity than GPC; the algorithm of FSAC is also more complicated and non-trivial than the one used in GPC. The main complexity of FSAC is the constructing of a finite-state automaton from a strongly regular context-free grammar, which is actually superfluous if two grammars have no common expressions at all. The greater efficiency achieved by GPC over FSAC comes from the early consideration of the sameness problem in the comparison approach. The FSAC has a bearing on higher complexity regardless of the question of whether two grammars are similar at all. Most grammars used in spoken dialog systems are either disjunctive or identical, so the high complexity is well avoided by reducing redundant enumerations in the GPC approach.

The NS approach is interesting but excludes recursive grammars, and so is not suited for many relevant use cases in practice.

Therefore, I propose to use the GPC approach for comparing grammars in dialog systems to determine functional or semantic overlaps in different dialog systems. I have also developed a prototype for empirical experiments. The comparison algorithm involved in the combining process will be described in the next chapter.

Based on thorough experiments on different grammars used in practical dialog systems, the GPC approach has been proven to be efficient in real-time and sufficient for different grammars.

Chapter 7

Combining Different Speech User Interfaces

An integrated speech user interface for different applications is often referred to as a "voice portal". A voice portal allowing for simultaneous access to different applications is supported by a multi-application/multi-domain dialog system. The process of endowing the corresponding dialog systems with the necessary information about the actual applications is normally referred to as dialog design. There exist different models for dialog design, which, however, only aim at single-domain/single-applications. To date, there is still no very efficient standard dialog design methodology for developing a "voice portal" for multiple applications. This chapter describes a novel methodology for constructing a multi-dialog system – a "voice portal" – for different applications, automatically or semi-automatically, based on existing single-domain dialog systems.

Different architectures for constructing a multi-dialog system and their advantages and disadvantages were introduced in Chapter 2. I decided to use the architecture based on integration at the level of dialog specifications for maximum reusability of existing resources (i.e. dialog specifications) and minimum complexity for designers to construct the multi-dialog system based on existing single-dialog systems.

Multi-application dialog systems face problems that have not occurred in the context of single-domain dialog systems. In Section 7.1 I introduce the concept of a multi-application dialog system, list some existing approaches to constructing multi-application dialog systems and declare my criteria for a good multi-application dialog system. Before I introduce my combination scheme, the requirements for the dialog systems that are to be combined together will be described in section 7.2. Based on the definition of dialog systems to be combined, I analyze different combination scenarios in Section 7.3. In Section 7.4, I elaborate my combination scheme. In Section 7.5, I explain the special issue of combing context specifications. In Section

7.6, I compare my approach with the existing approaches and summarize this chapter.

7.1 Multi-application Dialog System

A multi-application dialog system also referred to as a multi-domain dialog system is defined as a dialog system allowing the user to access a set of different applications simultaneously. Typical applications range from simple tasks such as operating a home device or booking a flight to more complex tasks such as intelligent traffic management or smart room-control.

Though dialog-modeling approaches have been extensively exploited in many research studies, they mostly concentrate on the development of a single-domain dialog system. Different open architectures of dialog systems supporting multiple applications/domains have been purposed, but few contributions have been made to the issue of how to deploy the multi-application supported (or application-transparent) dialog system to a set of different applications. In other words, it has not yet been exclusively discussed how the dialog system can be configured with the necessary information from different applications.

7.1.1 State of the Art

A voice portal enabling access to multiple applications was introduced in Nouza and Holada [2000]. Different applications are statically modeled as separate tree branches under the same root node. The dialogs between the user and the system are navigated in the tree in a menu-based style. The dialog specification is written in a script, which can be interpreted by the dialog manager. Though the dialog specification is decoupled from the dialog manager core, extending the system to new services is not trivial in the static architecture. The user cannot access different applications transparently. At any time there is only one branch of the tree which is active thus only one application is accessible. The user has to switch the application explicitly.

The open architecture of GALAXY-II [Seneff et al., 1999] developed in the context of a DARPA project addresses the issue of multiple applications. Different applications are controlled by different dialog managers individually and a meta-dialog manager is applied for managing all dialog managers. Simple domain extension is promised by this architecture by plugging a new application and its corresponding dialog manager into the existing system, though this architecture also suffers the disadvantage of requiring an explicit domain switch for accessing other passive applications.

With the appearance of multi-dialog systems, more research studies are striving to address the problem of easy portability of a dialog system to different domains. In Lin et al. [1998b], a multi-domain dialog system is introduced. The advantage of this system over the existing GALAXY-II architecture is the application of a single dialog manager in controlling different applications. Each application is described by a Task-Description-Table, which provides the necessary information about the application for the dialog manager. The dialog manager can host different Task-Description-Tables in runtime, so that different applications can be addressed. Domain extension means the implementation of the appropriate Task-Description-Table (TDT) and the integration of this TDT in the dialog manager. Domain switch is realized as activating the corresponding Task-Description-Table by the dialog manager. No meta-dialog manager is needed in this approach. However, at each time there is at most only one TDT active, so that only one application is accessible. The domain switch requires also an explicit command uttered by the user.

Very similar to the approach introduced by Lin et al. [1998b], an approach for dynamic multi-domain dialog processing is introduced by Pakucs [2003]. This proposed multi-domain dialog system focuses on dynamically extending the dialog systems with new applications. In this system, each domain is specified by a dialog specification, which is collected in a component named “dialog specification collection”. The dialog manager can interpret the content of the dialog specification collection in runtime and address the right dialog specification to enable the user to access the desired application. New domains can be easily plugged into the system at runtime by providing the corresponding dialog specification. A dynamic domain switch is enabled, but an explicit “switch command” such as “switch to the application A” from user is still necessary.

An advanced approach following a similar architecture as in Lin et al. [1998b] and Pakucs [2003] is introduced in Bui et al. [2005]. This approach brings different applications together into one dialog system by arranging existing dialog specifications of each application automatically into an application hierarchy. Based on the description provided by the dialog specification of each application, the similarity between two applications is calculated. According to the similarities, the applications are clustered in a binary tree with the most similar applications clustered under the same node in the tree. Given a user utterance, the similarity between the utterance and all applications are computed. If the difference between the highest similarity value and the next similarity value is beyond a predefined threshold, the desired application is determined to be the one with the highest similarity value. Otherwise the dialog system navigates along the binary tree with clarification dialogs until the intended application is reached. The outstanding part of this multi-application dialog system against existing systems is the transparent application switching. All applications are active at any time in the dialog, so the user can navigate the application hierarchy to access different applications with the help of the dialog system simultaneously and does not have to switch the active application explicitly by some predefined command.

An agent-based multi-application dialog system based on information state [Larsson & Traum , 2000] is introduced in Vrugt et al. [2004]. This system supports multiple applications by adopting an application-independent knowledge processing management system. Modular ontological descriptions for different applications are provided as dialog specifications. By integrating these descriptions, transparent access to different applications is enabled by the system. By using a common ontology space for all applications, the system supports reuse of knowledge of the same ontology type across applications.

Most existing multi-application dialog systems aim only to support more than one application within the same dialog system. Transparent domain switch [Bui et al., 2005] and information sharing across different applications [Vrugt et al., 2004] have been focused on in several research studies. A very important issue in multi-application dialog systems is the task sharing problem. This has not yet been solved by any study: it is addressed as future work in Bui et al. [2005].

7.1.2 Criteria

There are no other standard requirements for a multi-application dialog system than to support multiple applications in one dialog system. But a good multi-application dialog system should be more flexible and intelligent. From the existing research work, we can also see these trends. In this section, I analyze the important features of a multi-application dialog system with respect to the issue of “multiple applications.”

One important feature, which is also addressed by much recent research, is the *domain extensibility*. A multi-application dialog system should be able to be extended by new applications as simply as possible. It means that for the extension of new applications, no modification will be needed in the core of the dialog system. Only the description of the new application is needed, or, additionally, the corresponding dialog manager, if the dialog manager based multi-application dialog system is adopted.

As it acts as a speech user interface general to different applications, *transparent access to different applications* needs to be supported by the voice portal. This means the user can access any function of any application at any time simultaneously without having to name the right application. An explicit application switch may increase the cognitive load and decrease the flexibility and naturalness of a dialog system. Among many research studies, only the approach presented in Bui et al. [2005] achieves this feature.

Furthermore, a set of applications under the same voice portal is actually more than just a set of independent units. Rather, there can be cross-application functions or cross-application information. It means different applications may provide the same function for the user or share some common information. The voice portal should actually solve the *interoperability* problem between different underlying applications. Only in this way can the multi-dialog system act as a harmonic front-end to different applications. To my best knowledge, this issue has not been solved by any approaches yet and is addressed as future work in Bui et al. [2005].

In summary, a multi-application dialog system allows the user to access different applications within the same dialog system. The criteria for a good multi-application dialog system are domain extensibility, transparent access to different applications, interoperability of different applications in regards to the issues of task and information sharing. There is no existing approach which satisfies all these criteria. In the following sections of this chapter, my novel approach to solving all these issues will be introduced.

7.2 Requirements on Dialog Systems

Before I introduce the detailed combination scheme, I declare the requirements on the dialog systems adopted for the “voice portal” in this section.

I do not intend to combine dialog systems following different dialog models, but rather I aim to combine dialog systems written in the same language and following the same dialog model. In other words, different dialog systems to be combined with my approach are distinguished from each other only in the application specific parts – i.e. the dialog specifications for different applications. Since most spoken dialog systems strive to support as many applications as possible, this assumption does not make an obvious restriction on the combinable applications.

The spoken dialog system adopted must be application-independent. This means that it is possible to port the spoken dialog system to a new application without any modification in the core of the spoken dialog system. Despite the possible architecture of the meta-dialog manager, this can only be achieved by separating the domain-independent dialog management component from the domain-dependent knowledge. Porting the dialog system to a new application means exchanging the domain-dependent knowledge, which is often referred to as dialog specification for the domain/application.

Further, the spoken dialog system must be capable of supporting multiple tasks in one dialog in order to support different tasks from different applications in the voice portal.

The domain-dependent description such as that of the dialog specifications must be defined *formally* and *declaratively.* An example is VoiceXML [McGlashan et al., 2003] or DIANEXML [Block et al., 2004]. Because only declarative specifications can be compared and merged, while native coding e.g. program codes implementation is not appropriate for these issues. Further, the dialog specifications should be task-oriented. This means an application is modeled as a set of tasks. Only in this way could we analyze the application according to the task sharing and information sharing among them.

In Chapter 3, I analyzed different classes of dialog modeling approaches and argued that the frame-based dialog modeling approach is the most suitable one for the combination issue. In Chapter 4, I gave an example of a dialog modeling approach which can describe an application declaratively and formally.

The combination scheme introduced in the following sections is based on the assumption that a frame-based spoken dialog system is used to support a set of applications whose dialog specifications are defined formally and declaratively. My combination scheme aims to combine exactly this set of systems.

7.3 Combination Scenarios

To combine two applications, there are different scenarios owing to the different dependencies between these applications. In this section, I discuss how to compare two applications and define different relations of two applications.

7.3.1 Essential Application Information

There are many elements in the dialog specification of an application. However, not all of them represent the essential information of an application. Some elements are only used for dialog design, so they are not crucial in the process of determining the relations of two applications. In this subsection, I analyze which part of the dialog specification represents the essential information of an application.

An application can be defined by the following form according to the dialog modeling approach proposed in Chapter 4:

$A = < \{T_{1,} T_{2,} ... T_n\}, \{C_1, ..., C_m\} >$

$T_i = < ID, tg, PRE, PROMPT, P, CSTR, SYS, Post >$

$PRE = \{pre_1, ... pre_{n2}\}$

$PROMPT = \{prompt_1, ..., prompt_{n1}\}$

$CSTR = \{cstr_1, ..., cstr_{n3}\}$

$SYS = \{sys_1, ..., sys_{n4}\}$

$P = \{p_1, ..., p_{n5}\}$

$p_i = < PID_i, BOOL_i, PPROMPT_i, ig_i, dg_i, Infer_i >$

$BOOL_i = \{b_{i1}, ..., b_{in}\}$

$PPROMPT_i = \{pprompt_{i1}, ..., pprompt_{in1}\}$

$C_i = \{TID_1 ... TID_i, ..., TID_l\}$
$TID_i = ID, where\ T_i = < ID, tg, PRE, PROMPT, P, CSTR, SYS, Post > \in A$

A dialog specification representing an application may contain a context specification $\{C_1, ..., C_m\}$. The context specification can be regarded as an extra part of the dialog specification of an application, and I consider the combination issues of contexts and transactions separately. In the main combination scheme, I ignore the context specifications inside a dialog specification and consider an application in the form of a transaction specification $A' = \{T_{1,} T_{2,} ... T_n\}$. In section 7.5, I discuss the combination issue of context specification separately.

A (backend) application is modeled as a set of transactions (frames) in the view of a speech dialog system. Each transaction represents an atomic function provided by the backend application.

The essential part of a transaction is represented by its trigger grammar and parameters.

A trigger grammar defines all possible natural utterances which can indicate the corresponding transaction directly. For example, the trigger grammar for a transaction “flight reservation” would comprise phrases as “book a flight”, “reserve a flight”, “make a flight reservation”, etc. The trigger grammar of a transaction is functionally similar to the ID of a function, which can be used to refer to the transaction/function in a speech application/program unambiguously.

The parameters of a transaction represent all necessary information required in order to invoke the corresponding function in the backend application. For example, in order to book a flight, parameters such as “departure city”, “departure date”, “departure time”, “destination city”, etc. are required information for performing flight reservation transactions in the backend application.

The essential information of a parameter is again specified by its grammars – **initiative grammar (ig)** *and* **dialog grammar (dg)**. The initiative grammar serves to understand the parameter value if the user provides the input initiatively, while the dialog grammar is used to understand the user's answer when the system asks about a parameter in dialog.

In a user initiative utterance, where the **initiative grammar** is adopted, the system does not know which parameters will be addressed by certain utterances, so inputs for different parameters must be disambiguated from each other. For example, in the sentence "I want to fly **to** *Paris* **from** *Munich* **via** *Frankfurt*" for a flight reservation, "Paris", "Munich" and "Frankfurt" are all cities, but they are indicated by their prepositions to be "destination city" ("to"), "departure city" ("from") and "transfer city" ("via"). For this user initiative situation, initiative parameter grammars are used to interpret the sentences. Obviously, in any one transaction initiative grammars for different parameters are always unique. Otherwise, it is unrealistic to distinguish different parameter values from each other in one utterance.

The **dialog grammar** is used when the system asks the user about the value of a parameter in the dialog. In this situation, a certain parameter is focused, so this parameter must not be designated by the "indicator" any more. For example, when the system asks the user "where are you flying to?" the user can simply answer "Paris" without "to". Therefore, the dialog grammar is more relaxed than the initiative grammar. And, actually, the language defined by the dialog grammar is a part of the language defined by the initiative grammar without indicating words. In one transaction, different parameters may have overlapping dialog grammars. For example, dialog grammars for "departure city", "arrival city" and "transfer city" can all comprise different city names such as "Paris", "Munich", etc without any preposition such as "to", "from", "via", etc. The dialog grammar is quite similar to the data type used in program language. A data type tells if a parameter is of type integer or Boolean, etc and a dialog grammar tells whether a parameter is of type CITY, DATE or TIME, etc.

The dialog grammar defines the actual values of a parameter, whereas the initiative grammar tells us more about what a parameter actually is - the semantic concept of a parameter. So, the initiative grammar is more specific and representative for representing what a parameter is in the context of transaction. So I propose to use initiative grammars to represent the essential information of a parameter in a transaction and dialog grammars for representing the essential information (the value range) of a parameter in general sense.

The other elements such as PROMPT, PRE, CSTR, SYS etc. of a transaction serve to support the dialog management of a spoken dialog system. Nevertheless, these elements also provide some useful information about the function of a transaction. However, due to the ambiguity of natural languages and various possibilities of natural language expressing the same meaning, comparison of these elements gives only very weak evidence for the dependency of the corresponding transactions.

So an application can be regarded as a set of transactions indicated by its trigger grammars, and consists of a set of parameters. The parameters in the transactions can be further represented by their initiative grammars. Further, the dialog grammars of different parameters represent the value ranges of different parameters and the actual value needed to execute various transactions in an application respectively. Therefore, the dialog grammars can represent the essential information of an application, which is required to be input by the user. In fact, dialog grammars can be regarded as a part of the initiative grammars, but in order to represent the essential required user input of an application, I propose to consider the dialog grammars instead of initiative grammars.

The following figures illustrate the process of extracting the essential form of a dialog specification:

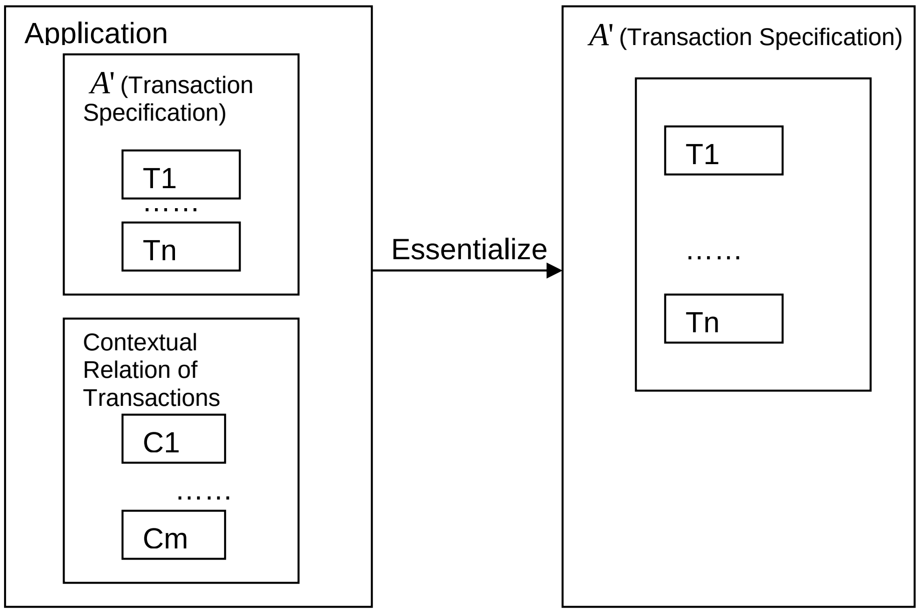
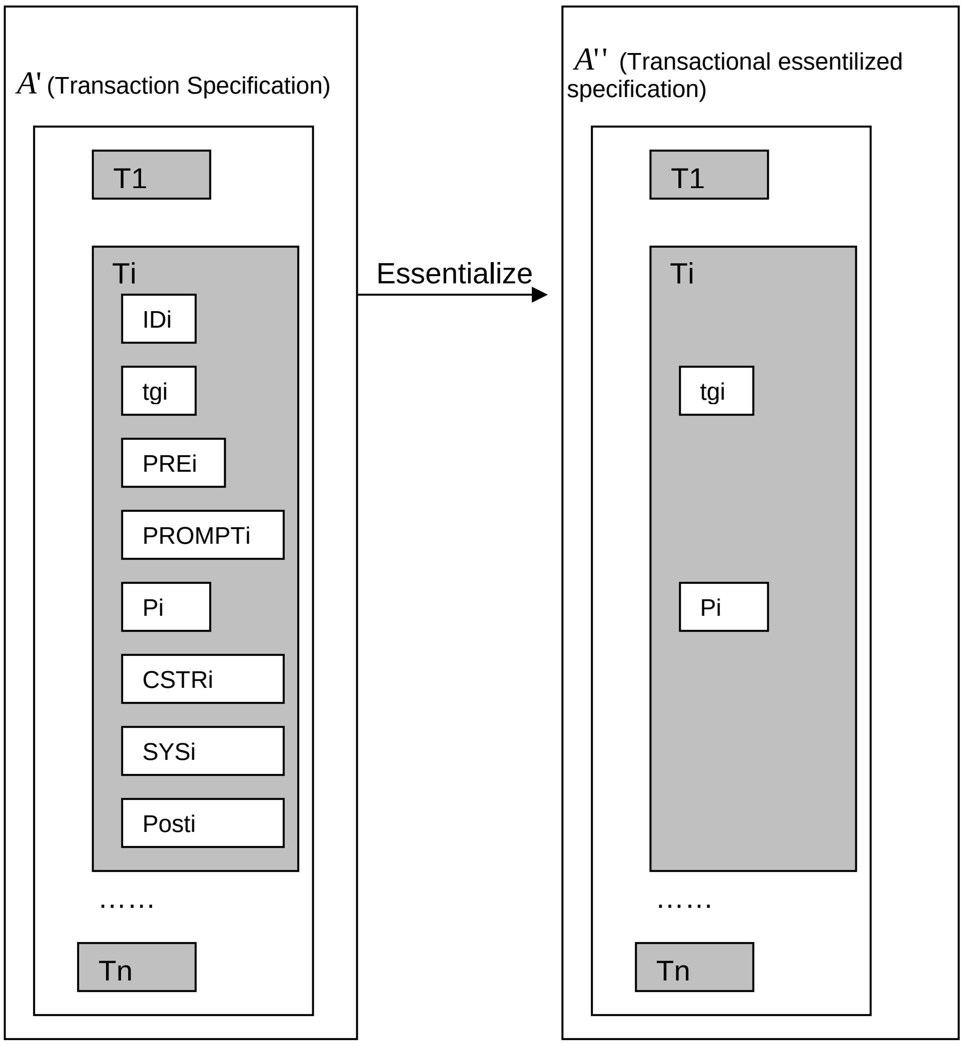

Application
A' (Transaction Specification)
T1
……
Tn
Contextual Relation of Transactions
C1
……
Cm
Essentialize
A' (Transaction Specification)
T1
……
Tn

Figure 7-1 Extraction of essential transaction specification from an application



A' (Transaction Specification)
T1
Ti
IDi
tgi
PREi
PROMPTi
Pi
CSTRi
SYSi
Posti
……
Tn
Essentialize
A'' (Transactional essentilized specification)
T1
Ti
tgi
Pi
……
Tn

Figure 7-2 Extraction of the essential information of a transaction

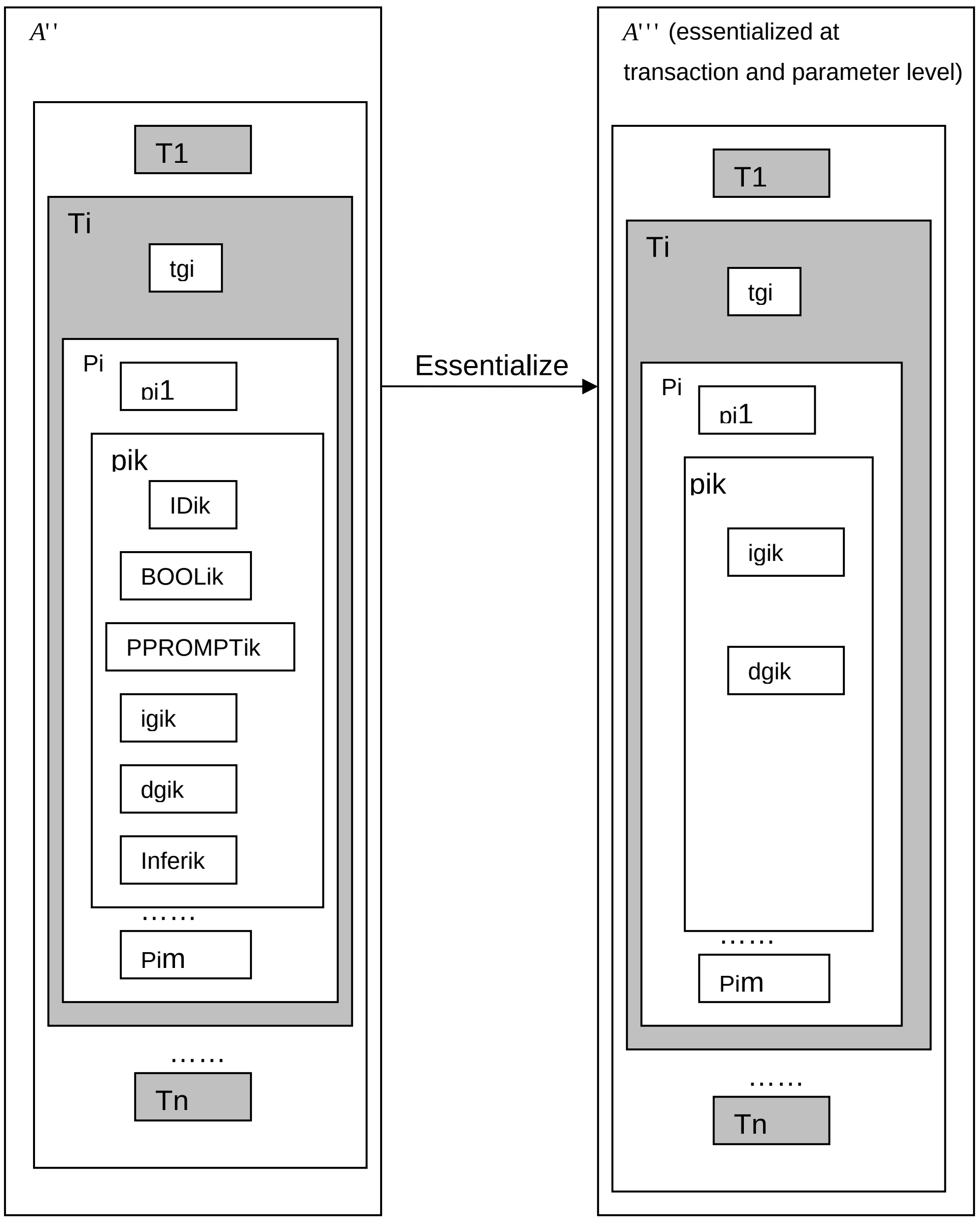


Figure 7-3 Extraction of the essential form of a parameter

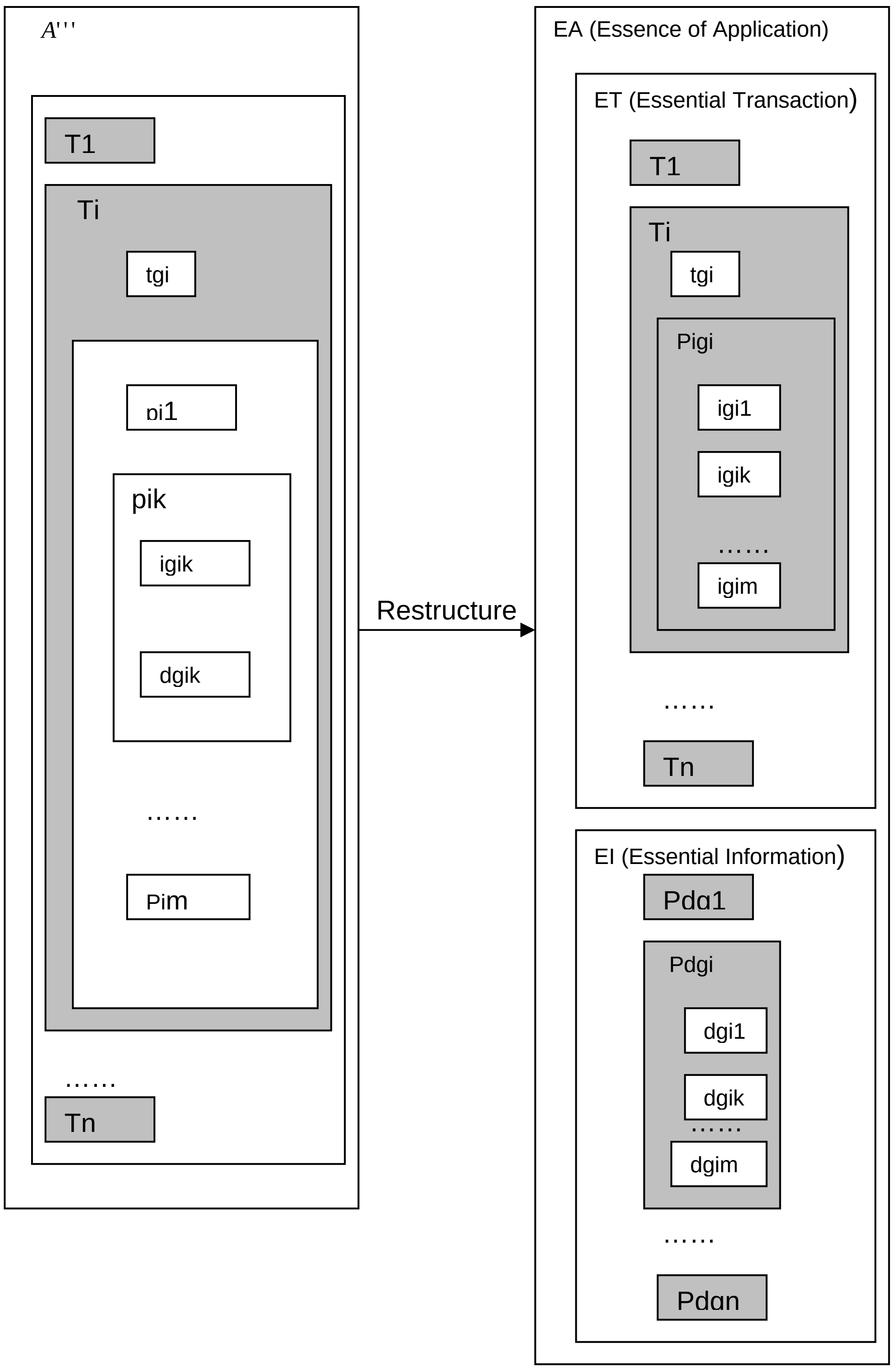


Figure 7-4 Restructuring of the essential form of an application into ET (Essential Transaction part) and EI (Essential information part)

More precisely, the essence of an application can be extracted into two parts – the essential transaction part and the essential information part. The transaction part consists of a set of transactions represented by their trigger grammars and initial parameter grammars, and the information part consists of a set of information items represented by the parameter dialog grammars:

$Essence(A) =< ET, EI >$

$ET = \{E(T_1),..., E(T_n)\}$

$E(T_i) =< tg_i, Pig_i > \quad Pig_i = \{ig_{i1},..., ig_{im}\} \ (1 \le i \le n)$

$EI = \{Pdg_1,..., Pdg_n\}$

$Pdg_i = \{dg_{i1},..., dg_{im}\} \ (1 \le i \le n)$

$(T_i = (ID_i, tg_i, ...P_i, ...Post) \in A, P_i = \{p_{i1},..., p_{im}\}, p_{ik} =< ID_{ik},..., ig_{ik}, dg_{ik}, Infer >)$

For the sake of understanding, let us consider a simple application example:
Assume an application consists of two transactions – “transport *dinnerware* from a *room* to another *room*” (TD) and “turn on the light in a *room*” (TL).
The first transaction TD will be modeled as the following:

- The trigger grammar for TD (TRANSPORT) defines phrases such as “transport”, “carry”, etc: $TRANSPORT = \{\text{"transport"}, \text{"carry"}, \text{etc.}\}$
- TD has three parameters – dinnerware, first room, second room.
- The initiative grammar and the dialog grammar (WARE) for the “dinnerware” are the same. They defines all transportable dinnerware such as “plate”, “bowl”, “fork”, etc: $WARE = \{\text{"plate"}, \text{"bowl"}, \text{"fork"}, \text{etc.}\}$
- The initiative grammar for the “first room” (FROMROOM1) defines possible expressions such as “from the kitchen”, “from the dining room”, etc. The corresponding dialog grammar (ROOM1) defines all possible values for the rooms such as “kitchen”, “dining room”:
  $FROMROOM1 = \{\text{"from the kitchen"}, \text{"from the dining room"}, \text{etc.}\}$
  $ROOM1 = \{\text{"kitchen"}, \text{"dining room"}, \text{etc.}\}$
- The initiative grammar for “the second room” (TOROOM2) defines possible expressions such as “to the kitchen”, “to the dining room”, etc. The corresponding dialog grammar (ROOM2) defines all possible values for the rooms such as “kitchen”, “dining room”.
  $TOROOM2 = \{\text{"to the kitchen"}, \text{"to the dining room"}, \text{etc.}\}$
  $ROOM2 = \{\text{"kitchen"}, \text{"dining room"}, \text{etc.}\}$

The second transaction TL will be modeled as the following:

- The trigger grammar for TL (TURNON) defines phrases such as “turn on the light”, “switch on the light”, etc:
  $TURNON = \{\text{"turn on the light"}, \text{"switch on the light"}, \text{etc.}\}$
- TD consists of one parameter – room.

- The initiative grammar of “room” (INROOM3) defines possible expressions such as “in the kitchen”, “in the dining room”, etc. The corresponding dialog grammar (ROOM3) defines all possible values for the rooms such as “kitchen”, “dining room”.

  $INROOM3 = \{\text{"in the kitchen", "in the dining room", etc.}\}$

  $ROOM3 = \{\text{"kitchen", "dining room", etc.}\}$

The essence of this application can be represented in the following structure:

$Essence(A) = < ET, EI >$

$$ET = \{< TRANSPORT, \{WARE, FROMROOM1, TOROOM2\} >,\\ < TURNON, \{INROOM3\} >\}$$

$EI = \{\{WARE, ROOM1, ROOM2\}, \{ROOM3\}\}$

7.3.2 Scenario Overview

When combining two different speech applications, their dependency can be determined by comparing their essential information.

Assume that two applications are expressed in the form introduced in the last section:

$Essence(A1) = < ET1, EI1 >$

$Essence(A2) = < ET2, EI2 >$

Comparison of these applications involves the comparison of the transaction parts and information parts.

Comparing two transaction parts with each other means comparing the essential information of each transaction T ($E(T) = < tg, Pig >$) in one application with each transaction in the other application. If there are similar transactions in two applications, the transactions parts are regarded as overlapping:

$$ET_1 \cap ET_2 \neq \phi \leftrightarrow \exists E(T_1) \in ET_1, E(T_2) \in ET_2 [E(T_1) \approx E(T_2)]$$

If two applications provide similar transactions in their speech user interfaces, it means the actual functions executed by the applications are similar. I refer to this scenario as functional overlap. Section 7.3.3 elaborates this scenario in detail.

Comparing two information parts with each other means comparing each dialog grammar dg1 in one application with each dialog grammar in the other application. If there are similar dialog grammars, the information parts are regarded as overlapping:

$$EI_1 \cap EI_2 \neq \phi \leftrightarrow \exists dg_1 \in EI_1, dg_2 \in EI_2 [dg_1 \approx dg_2]$$

If two applications have similar dialog grammars, it means the applications can probably share the information represented by the parameters with the similar dialog grammars. I refer to this scenario as semantic overlap. Section 7.3.4 elaborates the scenario of semantic overlap in detail.

If there is no dependency in both parts, two applications are disjunctive. This scenario will be introduced in Section 7.3.5.
Table 7-1 gives an overview of all possible scenarios when combing two applications:

	$ET1 \cap ET2 = \phi$	$ET1 \cap ET2 \neq \phi$
$EI1 \cap EI2 = \phi$	Disjunctive applications	Functional overlap
$EI1 \cap EI2 \neq \phi$	Semantic overlap	Functional overlap and Semantic overlap

Table 7-1 Combination scenario overview

For formal expression, I have adopted the symbol $\neq$ to represent that two transactions, parameters or grammars in the context of dialog modeling are not similar. The following convention is assumed in the formal expressions introduced in the rest of this thesis, when objects A and B are transactions, parameters or grammars:

$A \neq B \leftrightarrow \neg(A \approx B)$

7.3.3 Functional Overlap

If the transaction parts of two applications have overlap with each other, we say these applications have functional overlap. This overlap consists in similar transactions defined in both applications.
The functional overlap of two applications can be formally defined as the following:

A_1 and A_2 has functional overlap $\leftrightarrow$

$\exists T_1 \in A_1, T_2 \in A_2 [T_1 \approx T_2]$

The similarity of two transactions can be determined by the similarity of their trigger grammars, because trigger grammar is an indicator for the actual function provided by a transaction. The similarity of trigger grammars has been defined in Section 6. For example, a transaction with trigger grammars containing phrases such as “flight reservation”, “reserve a flight” and “book a flight”, and a transaction with trigger grammars specifying phrases such as “flight reservation” and “reserve an air ticket” are functionally similar – both provide a service of “flight reservation”. Based on the assumption that each trigger grammar always defines as many key phrases as possible to enable a flexible natural interaction, I state two transactions as functionally similar only when there is at least one common key expression defined in their trigger grammars.

$E(T_1) = < tg_1, Pig_1 > \quad Pig_1 = \{ig_{11}, ..., ig_{1n}\}$

$E(T_2) = < tg_2, Pig_2 > \quad Pig_2 = \{ig_{21}, ..., ig_{2m}\}$

$\mathrm{T}_1 \approx \mathrm{T}_2 \leftrightarrow tg_1 \approx tg_2$

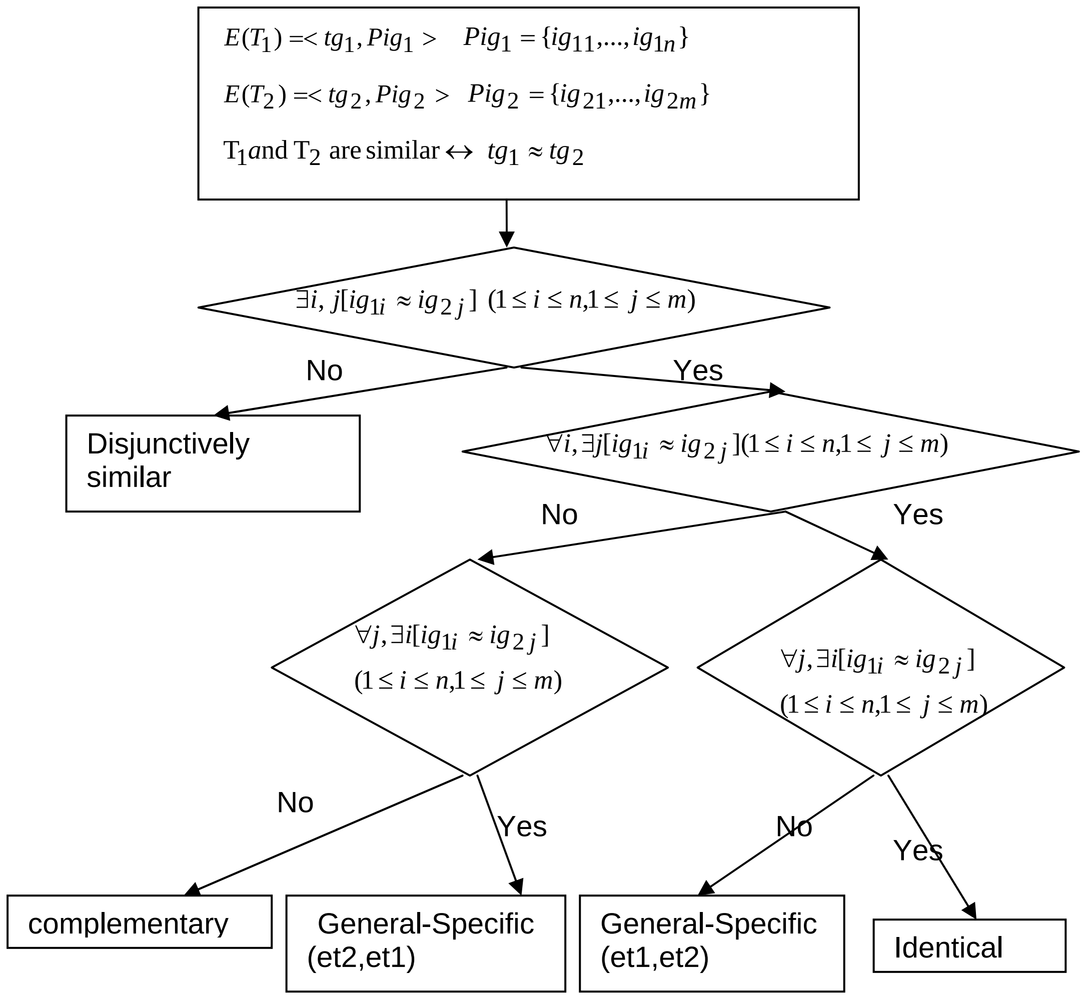
$tg_1 \approx tg_2 \leftrightarrow \exists w[w \in L(tg_1) \wedge w \in L(tg_2)]$

To determine the similarity of two transactions, only the trigger grammars are required. The similarity of trigger grammars is sufficient to indicate the fact that two transactions are similar. However, depending on the corresponding parameter sets represented by the initiative dialog grammars *Pig*, there are different kinds of "similarity".

Comparing two sets of initiative grammars means comparing each grammar in one set with each grammar in the other set. Two grammars are compared with respect to their similarity. Based on the introduction in Section 6, two initiative grammars are similar if they have at least one common expression as an intersection.

$ig_1 \approx ig_2 \leftrightarrow \exists w[w \in L(ig_1) \wedge w \in L(ig_2)]$

Based on the similarity comparison of each pair of initiative grammars, four different relations between two sets of initiative grammars can be drawn out. The following decision tree illustrates these relations as a complete logical case analysis.



$E(T_1) =< tg_1, Pig_1 > \quad Pig_1 = \{ig_{11},...,ig_{1n}\}$
$E(T_2) =< tg_2, Pig_2 > \quad Pig_2 = \{ig_{21},...,ig_{2m}\}$
T_1 and T_2 are similar $\leftrightarrow tg_1 \approx tg_2$
$\exists i, j[ig_{1i} \approx ig_{2j}]\ (1 \le i \le n, 1 \le j \le m)$
No
Yes
Disjunctively similar
$\forall i, \exists j[ig_{1i} \approx ig_{2j}](1 \le i \le n, 1 \le j \le m)$
No
Yes
$\forall j, \exists i[ig_{1i} \approx ig_{2j}]\ (1 \le i \le n, 1 \le j \le m)$
$\forall j, \exists i[ig_{1i} \approx ig_{2j}]\ (1 \le i \le n, 1 \le j \le m)$
No
Yes
No
Yes
complementary
General-Specific (et2,et1)
General-Specific (et1,et2)
Identical

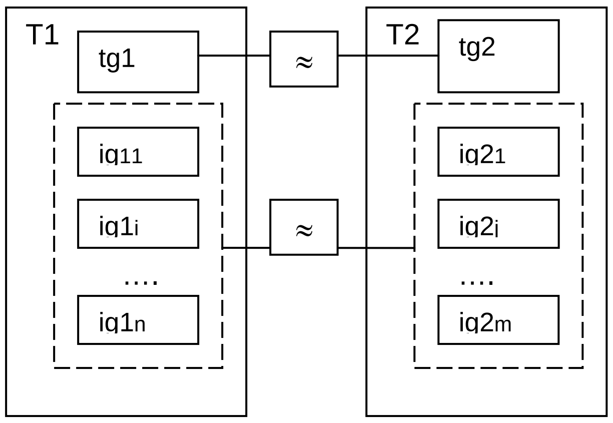
Figure 7-5 Different relations between similar transactions

In the following sections, I discuss each relation in detail.

7.3.3.1 Identical transactions

Similar transactions are identical if they consist of exactly the same number of parameters and their parameters are one-to-one similar. For example, System A provides a transaction et1 to search its address book for a contact with name and last name as parameters. System B provides a transaction et2 to search its corresponding address book for a contact with name and last name as parameters as well. We say et1 and et2 are identical. This relation can be defined by the following form:

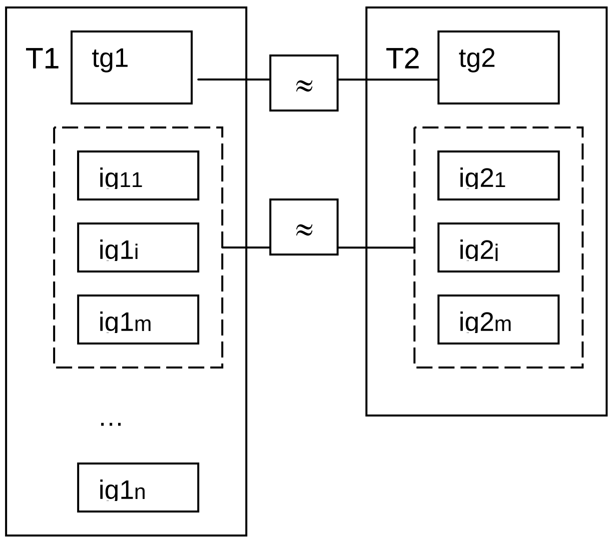
$$
\begin{aligned}
& tg_1 \approx tg_2 \wedge \\
& m = n \wedge \\
& \forall ig_2 \in Pig_2\ \exists ig_1 \in Pig_1 [ig_1 \approx ig_2] \wedge \\
& \forall ig_1 \in Pig_1\ \exists ig_2 \in Pig_2 [ig_1 \approx ig_2]
\end{aligned}
$$



7.3.3.2 General-specific transactions

Two similar transactions et1 and et2 have a general-specific relation, if transaction et1 is more specific than et2. This is the case, if et1 has more parameters and each parameter involved in et2 has a similar parameter in et1. For example, system A provides a transaction et1 for hotel reservation with parameters city, date, duration, room type and smoking room. And system b provides a transaction et2 for hotel reservation with parameters city, date, duration and room type. The transaction et1 is therefore more specific than et2, because it allows the user to specify his wish more specifically. This relation can be defined by the following form:

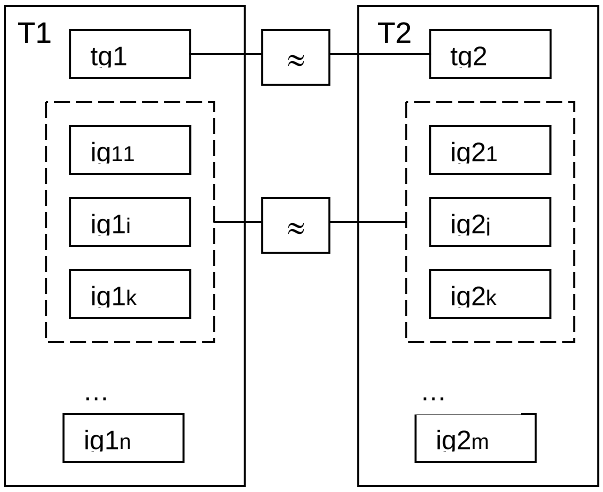
$$tg_1 \approx tg_2 \wedge$$
$$m < n \wedge$$
$$\forall ig_2 \in Pig_2 \, \exists ig_1 \in Pig_1 [ig_1 \approx ig_2]$$



7.3.3.3 Complementary transactions

Two similar transactions are complementary to each other if each transaction has some specific parameters which are not considered in the other one. For example, System A provides a transaction et1 for searching a contact in address book with “first name”, “last name” or “nickname” as parameters. System B provides a transaction et2 for searching a contact n address book with “first name”, “last name” or “telephone number” as parameters. The transactions et1 and et2 are therefore complementary, because et1 requires/allows the user to enter a nickname and et2 requires/allows the user to enter a telephone number for searching. The transactions et1 and et2 are functionally similar, but both have some specific parameters, which are not contained in the other transaction. No single one can replace the other one. This relation can be defined by the following form:

$$\exists ig_2 \in Pig_2, \forall ig_1 \in Pig_1[ig_1 \neq ig_2] \wedge$$
$$\exists ig_1 \in Pig_1, \forall ig_2 \in Pig_2[ig_1 \neq ig_2] \wedge$$
$$\exists ig_1 \in Pig_1, \exists ig_2 \in Pig_2[ig_1 \approx ig_2]$$

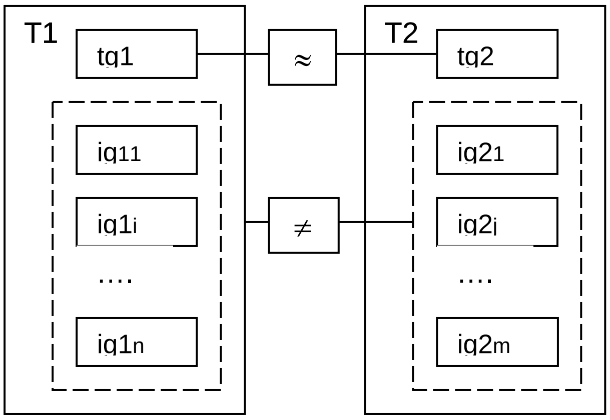


7.3.3.4 “Disjunctive similar transactions”

Two similar transactions are disjunctive if they have no similar parameters at all. For example, transaction et1 for “weather information” with parameter “city” and “date” and transaction et2 for “contact search” with parameter “first name” and “last name” are disjunctive. This case is however not realistic in practice, because two transactions are already judged as similar based on the comparison of their trigger grammars. It could only happen if the similarity of trigger grammars is caused by the ambiguity of natural languages. For example “book” as “something to read” is judged to be the same as “book” with the meaning of “reservation”. Particularly, such two transactions cannot be treated as similar, and they are independent from each other. This problem can be solved by involving the human designer, who features in the domain knowledge of both transactions/applications. In the next section, I propose an interactive combination tool for this kind of cooperation.

This relation can be defined by the following form:

$\forall ig_2 \in Pig_2, \forall ig_1 \in Pig_1 [ig_1 \neq ig_2]$



In summary, the similar transactions of two applications can be categorized into four types: identical transactions, general-specific transactions, complementary transactions and disjunctive similar transactions. The corresponding figures show intuitively that these scenarios include all possible functional overlapping scenarios.

7.3.4 Semantic Overlap

If the information parts (EI) of two applications have overlap with each other, we say these applications have semantic overlap. This overlap is due to the similar parameters required in both applications. In this consideration, each parameter is regarded as an individual object, not as an element of a transaction. The dependency between two parameters should thus be determined by the similarity of their dialog grammars. If the dialog grammars of two parameters are similar according to the definition in Section 6, the represented information is related, so two applications could share this information appropriately. For example, there is an application "hotel" and an application "flight". The application "hotel" provides a function "hotel reservation". The application "flight" provides a function "flight reservation". These two applications do not have any functional overlap with each other. However, the dialog grammar of the parameter “destination city” of the flight contains expressions such as “Munich”, “Berlin”, etc. The dialog grammar of the parameter “accommodation city” of the hotel contains city names such as “Berlin”, “Frankfurt”, etc. Based on the overlapping city “Berlin” in both grammars, the parameter “destination city” and the parameter “accommodation city” can be considered as

related and the applications can share the city information mutually. Similarly, the “arrival date” of the flight and the “hotel check in date” are similar and the “return date” of the flight is a semantic overlap with the “hotel check out date”.
The semantic overlap of two applications can be formally defined as the following:

$$A_1 \text{ and } A_2 \text{ has semantic overlap } \leftrightarrow$$
$$\exists dg_{ni} \in A_1, dg_{mj} \in A_2 [dg_{ni} \approx dg_{mj} \wedge E(T_n) \neq E(T_m)]$$

The expression $E(T_n) \neq E(t_m)$ means that the semantic overlap are not considered in case of functional overlaps. If two transactions are indicated as similar, their corresponding information items (the corresponding dialog grammars) will no longer be compared with respective to the semantic overlap.

7.3.5 Disjunctive Applications

Two applications are disjunctive if there are no overlaps in their transaction parts and information parts. It means that there is no functional overlap and no semantic overlap between two applications.
Based on the introductions in the last two sections about functional and semantic overlap, the following definition for disjunctive applications can be obtained:

$$A_1 \text{ and } A_2 \text{ are disjunctive } \leftrightarrow [\, Essence(A_1) = < ET_1, EI_1 >] \wedge [Essence(A_2) = < ET_2, EI_2 >]$$
$$\wedge [ET_1 \cap ET_2 = \phi] \wedge [EI_1 \cap EI_2 = \phi]$$

$$ET_1 \cap ET_2 = \phi \leftrightarrow \neg(\exists et_1 \in ET_1, et_2 \in ET_2 [et_1 \approx et_2])$$

$$EI_1 \cap EI_2 = \phi \leftrightarrow \neg(\exists dg_1 \in EI_1, dg_2 \in EI_2 [dg_1 \approx dg_2])$$

For example, the applications “intelligent home environment” and “remote access to information database” [Neto et al., 2003] are disjunctive applications. They do not provide any similar functions and or have any shared information with each other.

7.3.6 Scenarios Summary

In Section 7 above, all possible combination scenarios have been discussed. In order to demonstrate the completeness of these scenarios, the following table lists all scenarios together. The formal definitions show that all possible relations between two applications have been covered by these scenarios.

Scenario	**Definition**	**Example**
Disjunctive	$\forall et_i \in A_1, et_j \in A_2 [et_i \neq et_j] \wedge$ $\forall dg_k \in A_1, dg_m \in A_2 [dg_k \neq dg_m]$	intelligent home environment and remote database access

Functional overlap (FO)		$\exists et_i \in A_1, et_j \in A_2 [et_i \approx et_j] \leftrightarrow tg_i \approx tg$ $et_i = < tg_i, Pig_i >$ $et_j = < tg_j, Pig_j >$	contact search in A1 (CS1) and contact search in A2 (CS2)
	Identical Transactions	$\forall ig_1 \in Pig_i \exists ig_2 \in Pig_j [ig_1 \approx ig_2] \wedge$ $\forall ig_2 \in Pig_j \exists ig_1 \in Pig_i [ig_1 \approx ig_2] \wedge$	CS1 with parameter “last name” and “first name” CS2 with parameter “last name” and “first name”
	General-Specific Transactions	$\forall ig_1 \in Pig_i \exists ig_2 \in Pig_j [ig_1 \approx ig_2] \wedge$ $\exists ig_2 \in Pig_j \forall ig_1 \in Pig_i [ig_1 \neq ig_2]$	CS1 with parameter “last name” and “first name” CS2 with parameter “last name”, “first name” and “telephone number”
	Complementary transactions	$\exists ig_1 \in Pig_i, \exists ig_2 \in Pig_j [ig_1 \approx ig_2] \wedge$ $\exists ig_1 \in Pig_i, \exists ig_2 \in Pig_j [ig_1 \neq ig_2]$	CS1 with parameter “last name”, “first name” and “email address” CS2 with parameter “last name”, “first name” and “telephone number”
	Disjunctive similar transactions	$\forall ig_1 \in Pig_i, \forall ig_2 \in Pig_j [ig_1 \neq ig_2]$	Unrealistic scenario
Semantic Overlap		$\exists dg_i \in A_1, dg_j \in A_2 [dg_i \approx dg_j]$	“city” of hotel reservation and “destination” of flight reservation

Table 7-2 Scenarios for combining two speech applications

7.4 Combination Schemes

The combination approach proposed in this dissertation is inspired by the idea that different transactions from two dialog specifications can be merged together into one unified dialog specification:

$A_1 = \{T_{11}, T_{12}, ..., T_{1n}\}$ $A_2 = \{T_{21}, T_{22}, ..., T_{2n}\}$

$A_1 + A_2 = \{T_{11}, T_{12}, ..., T_{1n}, T_{21}, T_{22}, ..., T_{2n}\}$

A dialog system supporting A_1 and A_2 supports the unified dialog specification naturally. Obviously, this basic idea can only be applied to the disjunctive applications without any overlaps. Since there are no overlaps between different transactions, no other modification is needed for the merging process.

In more sophisticated combination scenarios as described in the last section, certain handling modifications to the transactions are required for constructing a useful unified dialog specification. These handling modifications cannot be performed completely automatically by the computer, and some assistance from the designer is required in the combination process. Therefore, I have constructed an interactive combination tool to enable a semi-automatic merging process.

In the next section, I elaborate this merging process.

7.4.1 Combination Process

Constructing an integrated speech user interface completely automatically is not possible, because the specific domain information is not provided, e.g. it is not known how two “price information” transactions should be merged together – simply keep only one and remove the other, or merge two transactions in a specific way. Neither is it known that the “destination city” of “flight reservation” (i.e. not the “departure city” of “flight reservation”) is the same as the “accommodation city” of “hotel reservation”, even when the semantic overlap can be assessed based on grammar comparison.

One way to solve this problem is to use a common knowledge basis for all applications such as the technology used in Semantic Web [Decker et al., 2000]. Using a semantic web, the computer can not only represent given contents but also understand them by mapping the contents to standard definitions in the semantic web. By connecting semantics used in the dialog specification to some standard defined in a semantic network, the identification of two terms can be formally assessed. However, building such a knowledge base for an enormous range of different applications tends to be a very complex research undertaking, which goes beyond the scope of this dissertation. Moreover, the dialog specification of an application already gives a formal and declarative description, except for the use of different words and meaning in the specification of grammars, and this is in fact due to the ambiguity of natural languages. So, I propose that the most efficient way of assessing the right relationship between two concepts described in natural languages is to let the human support the computer.

Therefore, I propose a semi-automatic combination process to involve the dialog designer, who holds domain specific knowledge, to support the computer with the sophisticated merging scenarios, and, further, suggest in the next section a combination tool to make this process as automatic as possible.

There are four main steps in the semi-automatic combination process. The following figure illustrates this process. In the next sections, I elaborate each step in detail.

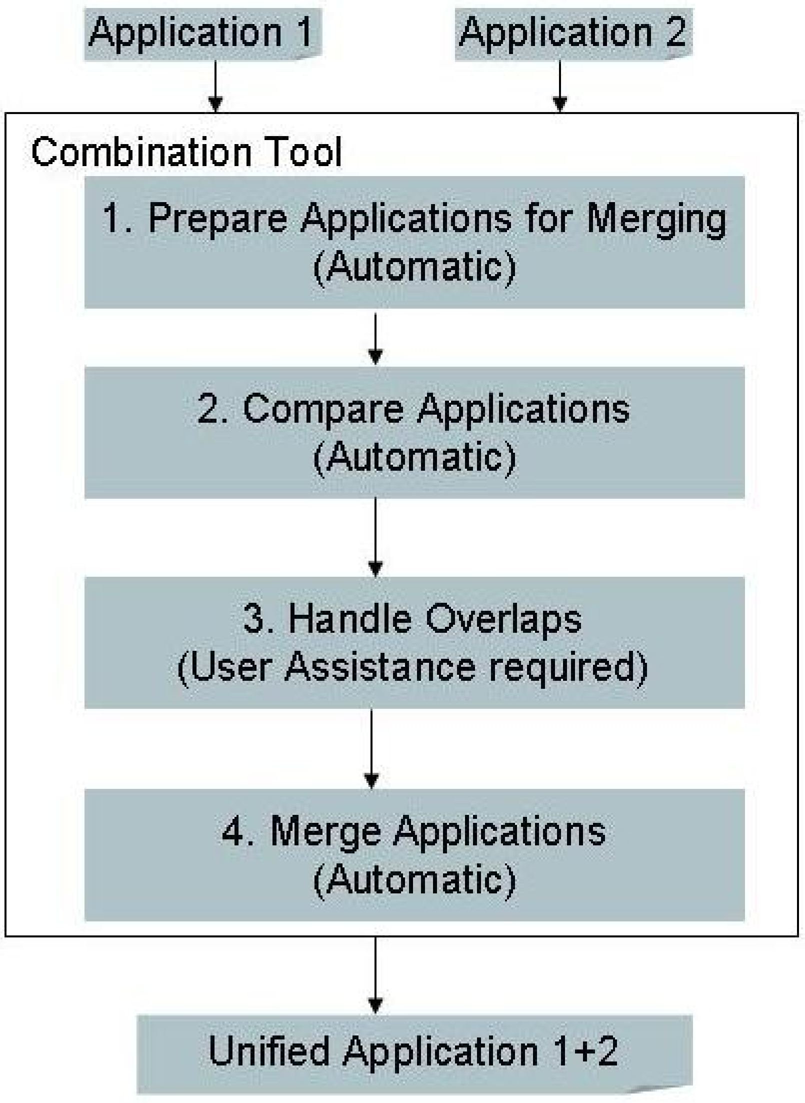

Application 1
Application 2
Combination Tool
1. Prepare Applications for Merging (Automatic)
2. Compare Applications (Automatic)
3. Handle Overlaps (User Assistance required)
4. Merge Applications (Automatic)
Unified Application 1+2

Figure 7-6 Combination process of the combination tool

7.4.1.1 Application preparation

Given two speech applications based on the same dialog model, the combination tool generates a unified dialog specification from their individual dialog specifications.
The dialog specifications will first be prepared by the tool for combination. This step is required for the uniqueness of identities of each transaction and parameter. Each object in the dialog application has an identity, which is unique in the dialog

specification. Without any preparation, merging two dialog specifications might destroy this uniqueness.
So in this step the tool adds the application identity as a prefix to the identity of each object in the dialog specification of this application. Different applications have different identities and the ID inside an application is guaranteed to be unique by the designer, so by this modification all IDs become unique in the unified dialog specification.
For example, in the applications “petStore” and “unifiedMessaging” there is a parameter called “name”. Obviously, in pet store, the “name” refers to a pet’s name and in unified messaging system the “name” refers to the user’s name. To distinguish one from the other in the unified dialog specification, the ID “name” in pet store application will be changed to “petStore_name” and the ID “name” in unified messaging system will be changed to “unifiedMessaging_name”.

7.4.1.2 Application Comparison

The different dependencies between two applications have been discussed in the last section. In order to handle these scenarios accordingly, the different scenarios must be recognized based on the grammar comparison. In the scenario section, the definition of each scenario has been elaborated, and by means of these definitions, it is not hard to determine different scenarios automatically.

Recognition of functional overlap

Different kind of functional overlaps were introduced in Section 7.3. Based on the existing definitions, in order to recognize the functional overlap of two applications, the trigger grammars of all transactions in two applications must first be compared with each other. If any similar trigger grammars are determined, the initiative grammars of the parameters belonging to the corresponding transactions are further compared with each other. Depending on whether the initiative grammars are completely identical, or in a super- and sub-set relation, or complementary, the similarity of the transactions and, respectively, the functional overlap are reported appropriately. This process is illustrated in the following algorithm:

```
Procedure FIND_FUNCTIONAL_OVERLAP (A1, A2)
Input: Applications A1 and A2,
A1=<{T11,T12,…T1n},{C11,…C1m}>
A2=<{T21,T22,…T2k},{C21,…C2l}>
```

```
Output: Set of functional overlaps S={s1,s2..,sn} si =<Type,ID1,ID2>

BEGIN
    S = make a new empty set;
    FOR each $1 \le i \le n$ DO
        IF $\exists j[tg_{2j} \approx tg_{1i}]$
        THEN
            IF $\exists ig_1 \in Pig_{1i}, \exists ig_2 \in Pig_{2j}[ig_1 \approx ig_2]$
            THEN
            IF ( $\begin{matrix}\forall ig_1 \in Pig_{1i}, \exists ig_2 \in Pig_{2j}[ig_1 \approx ig_2] \\ \wedge \forall ig_2 \in Pig_{2j}, \exists ig_1 \in Pig_{1i}[ig_1 \approx ig_2]\end{matrix}$ )=true
            THEN $s = <\text{"identical"}, ID_{1i}, ID_{2j}>$
                 $S = S \bigcup \{s\}$

            ELSEIF ( $\begin{matrix}\forall ig_1 \in Pig_{1i}, \exists ig_2 \in Pig_{2j}[ig_1 \approx ig_2] \\ \wedge \exists ig_2 \in Pig_{2j}, \forall ig_1 \in Pig_{1i}[ig_1 \ne ig_2]\end{matrix}$ )=true
                THEN $s = <\text{"general} - \text{specific"}, ID_{1i}, ID_{2j}>$
                     $S = S \bigcup \{s\}$

            ELSEIF ( $\begin{matrix}\forall ig_2 \in Pig_{2j}, \exists ig_1 \in Pig_{1i}[ig_1 \approx ig_2] \\ \wedge \exists ig_1 \in Pig_{1i}, \forall ig_2 \in Pig_{2j}[ig_1 \ne ig_2]\end{matrix}$ )=true
                THEN $s = <\text{"general} - \text{specific"}, ID_{2j}, ID_{1i}>$
                     $S = S \bigcup \{s\}$

            ELSE $s = <\text{"complementary"}, ID_{1i}, ID_{2j}>$
                 $S = S \bigcup \{s\}$
            ENDIF
          ENDIF
        ENDIF
    ENDFOR
END
```

Figure 7-7 Algorithm for finding functional overlaps of two speech applications

Recognition of semantic overlap

The semantic overlap has been introduced and precisely defined in Section 7.3.4. The semantic overlaps are recognized based on the comparison of dialog parameter grammars of different transactions in two applications. If two transactions are already accessed as similar, their parameters will not be considered in the semantic overlap scenarios. This recognition process is illustrated in the following Pseudo-code:

```
Procedure SEMANTIC_OVERLAP (A1, A2)
Input: Applications A1 and A2,
A1=<{T11,T12,…T1n},{C11,…C1m}>
A2=<{T21,T22,…T2k},{C21,…C2l}>
Output: Set of semantic overlaps O={o1,o2..,on} oi =<PID1,PID2>

BEGIN
    O = make new set
    S = FUNCTIONAOVERLAP(A1,A2)
    FOR each $1 \le i \le n$ DO
        IF $\forall <"don'tcare", ID_1, ID_2 > \in S[ID_{1i} \ne ID_1 \wedge ID_{1i} \ne ID_2]$
        THEN
            FOR each $1 \le q \le |P_{1i}|$ DO
            IF ( $\exists j(1 \le j \le k), g(1 \le g \le |P_{2j}|)$ $[dg_{1iq} \in Pdg_{1i}] \wedge [dg_{2jg} \in Pdg_{2j}] \wedge [dg_{1iq} \approx dg_{2jg}]$ )=true
            THEN $o =< PID_{1iq}, PIDg >$
                $O = O \bigcup \{o\}$
            ENDIF
        ENDIF
      ENDFOR
   RETURN O
END
```

Figure 7-8 Algorithm for finding semantic overlaps of two speech applications

7.4.1.3 Handling of Overlaps

The transactions and parameters are compared based on literal grammar comparison, which is not always exact due to the ambiguity of natural languages. Therefore the handling of different overlaps has to be interactive, to involve the dialog designer in bringing necessary domain specific knowledge into the combination process.

I propose a combination scheme for each scenario in the following.

Functional overlap - similar transaction

Two similar transactions provide the same abstract functionality to the user. Their differences become apparent to the user only in the details of a dialog. For example one transaction requires/allows more parameter input than another. The transactions are distinguished from each other only in their execution by different backend applications. These differences are not transparent to the user, so the user is not able to distinguish one from the other, and in the case of transaction ambiguity the user would not be able to address the right one. So, these transactions should be merged together into one. In addition, the differences should be shifted to the dialog modeling and execution parts of the merged transaction, so that the merged transaction represents the two similar transactions in the same way.
Therefore similar transactions from different applications will be merged together in the following way:

1. Only one single transaction will be generated from two transactions of two different applications, the new transaction will be given a new unique identity - ID_{new}.
2. The new trigger grammar for the merged transaction can be constructed as a unification of trigger grammars of two similar transactions, because both grammars define only synonyms for triggering a certain function.
3. Different prompts for the new transaction can be inherited from both original transactions. Regarding the constraints and system actions, it suffices to inherit the definition of constraints and system actions from any of two similar transactions. Because the function is the same, we can assume most prompts, constraints and system actions are similarly defined in two applications.
4. Precondition combination:
   If the transactions have any preconditions, the new merged transaction would be accessible if all preconditions for transaction t_1 are satisfied or all preconditions for transaction t_2 are satisfied. Therefore, all conditions of the same transactions are first connected together with logical “and”. Then the

conditions of different transactions are connected with logical “or”. The new condition is built up in this way. There is only one precondition for the new merged transaction. In order to maintain the necessary help messages in case of precondition violation, the new message for precondition violation will be constructed by combining original messages together, e.g. the transaction “check email” in Application A1 requires the user to be logged in to the system, and gives the information “You have to login before checking any email” if the precondition is not fulfilled. Likewise, another transaction “check email” in Application A2 requires that the user has provided his username, and gives the information “You have to provide your username first” if the precondition is not fulfilled. Merging these two “check email” transactions together with respect to the precondition handling, a new precondition will be generated consisting of a new condition and a new message. The new condition defines that the user has to be logged in or has provided his username. The new message will be constructed as “one of the following problems occurred – ‘you have to login before checking any email’ or ‘You have to provide your username first’”.

5. Post-conditions combination:

   The construction of the postcondition for the merged transaction depends on the preconditions of the original transactions. If the original transactions define some preconditions, in the postcondition these preconditions should be checked and the corresponding postcondition should be executed only when the precondition of the corresponding original transaction is satisfied.

   If the preconditions of both transactions are fulfilled, the combination tool is not able to determine how the merged transaction should be executed in different backend applications, because the same transaction may produce different results in different applications and it cannot be known in advance what result will arise in the applications. For constructing the postcondition part for this purpose, two solutions for automatic generation and a semi-automatic generation are possible with designer’s cooperation: execution only in one application, execution in both applications and manual customization of a new postcondition.

   Executing the transaction only in one application means that if both preconditions are fulfilled, only the postcondition of the chosen application will be performed accordingly. For example, the transaction “reserve a train ticket” in Application A1 and another transaction “reserve a train ticket” in Application A2 are to be combined together. In such a case, it suffices to execute the merged transaction in one application only – for example reserve the ticket in A1 and inform the user “your ticket has been reserved,”

Executing the transaction in both applications means combining the post-conditions of both transactions. So for execution of the merged transaction, the postconditions of both transactions will be called after each other; e.g. the transaction “login” of an email application and the transaction “login” of a voicemail application should be merged together and the transaction should be executed in both applications. At the end of the transaction, the login will be processed in the “email” application and the “voicemail” application and the user will be informed about the results by e.g. the following system prompts:

“In application email, login succeeded; in application voicemail, login succeeded.”

“In application email, login failed; in application voicemail, login succeeded.”

Manual specification of new postcondition for the merged transaction aims to produce a better natural system prompt to inform the user about the transaction execution result. The automatically constructed system prompts in the above section may sometimes confuse the user and thus cause bad usability. However, a more intelligent prompt based on the limited knowledge is not feasible. For this purpose the designer can be involved to construct a more intelligent post-condition for the merged transaction. In the above example, the designer could construct the post-condition to form a prompt “login succeeded” if the login in both applications succeeds, and a prompt “login failed” if the login in any application fails. In this way the single-sign-on feature is guaranteed for the user, and there will not occur a prompt such as “In application email, login succeeded; in application voicemail, *login succeeded*. In application email, login failed; in application voicemail, *login failed*.”

6. Parameter specification in the merged transaction

   There are different scenarios for functional overlaps, and they are distinguished from each other by their parameter specifications. So the construction for the new parameter specifications depends on the concrete kind of similarity.

Identical transaction

The parameter specification of two identical transactions is exactly the same, so the new parameter specification can be inherited from any transaction.

$$p_i = p_{1i}$$

General-specific transactions

A specific transaction T_1 has more parameters than its general transaction T_2:

$$P_2 = \{p_1, p_2, ..., p_n\}$$

$$P_1 = \{p_1, p_2, ..., p_n, p_{n+1}, ..., p_m\}$$

The different parameters $\{p_{n+1}, ..., p_m\}$ are a particular specification, which is only supported by T_1, but not by T_2. For example, two transactions provide a hotel reservation service. One transaction can handle the user's request for a room with ocean view and the other cannot differentiate between one type of room and another. In this scenario, all mandatory information for a transaction is actually contained in the general transaction. The more specific transaction allows the user to make a more specific requirement. These more specific parameters are thus optional for executing the transactions. So the new parameter specification would contain all parameters of the general transactions in the way they are and also include the differentiated parameters in the more specific transactions as optional parameters. So the user *can* specify his/her wish more specifically, but does not have to do so.

$$P = \{p_1, p_2, ..., p_n, p'_{n+1}, ..., p'_m\}$$

$$p_i =< ID, \{BOOL_1, ..., BOOL_n\}\{PROMPT_1, ..., PROMPT_{n1}\}, ig, dg, Infer >$$

$$p_i' =< ID, \{BOOL_1, ..BOOL'_{opt}, ..., BOOL_n\}\{PROMPT_1, ..., PROMPT_{n1}\}, ig, dg, Infer >$$

$$BOOL'_{opt} = true$$

$BOOL_{opt}$ is the Boolean value in the parameter specification to define if a parameter is optional or mandatory. With a value "true", this element indicates that the corresponding parameter is optional, so the system does not require the information from the user but accept the user's initiative input for this parameter.

Complementary transactions

Two complementary transactions have common and different parameters. This is a common scenario for optional parameters. Optional parameters represent information which is not required but just supported by the backend application. Two complementary transactions provide the same essential function to the user, but they may allow the user to make different special wishes for the same function. For example, two transactions both provide restaurant information. One can categorize the restaurants according to the average menu price whereas the other one can distinguish different restaurants by their cooking style. In this situation, the new merged transaction can include all optional parameters and allow the user to make any specific requirements supported by any transaction. The parameter information can then be sent to the back end application accordingly. Depending on the designer's decision of merging post-conditions, the transaction will be executed appropriately. In these cases, the transactions would be executed by both systems.

Another possible scenario for complementary transactions is that the necessary information can be combined by different elements. Each combination suffices to execute the transaction in the backend application. For example, two transactions provide the user with the service of looking up a book. One transaction requires the user to provide the author's name and the title of the book, and the other requires the user to provide the ISBN of the book. For the first transaction, “author name” and “book title” are mandatory parameters. For the second transaction, “ISBN” is a mandatory parameter. In this situation, the user only has to provide one set of necessary information. This problem can be solved by first inheriting all different mandatory parameters into the new merged transaction. Depending on which group of parameters is addressed by the user first, the other group of parameters will all be inferred to a dummy value, so that the system will not ask the user to provide this information, which is no longer necessary. If the user has provided the information for “author name”, the system will then inference the “ISBN” to a dummy value, so the “ISBN” will not be asked for anymore, and the user is only asked to provide the ”title”. Similarly, if the user has provided the “ISBN”, the system will inference “author name” and “book title” to an arbitrary value, so the user will not be asked to provide this information.

The following algorithm illustrates the combination process for merging two similar transactions as described above:

$$T_1 = < ID_1, PROMPT_1, tg_1, \{pre_{11}, ... pre_{1m}\}, P_1, CSTR_1, SYS_1, Post_1 >$$

$$pre_{1i} = < condition_{1i}, message_{1i} >$$

$$P_1 = \{p_1, p_2, ..., p_n, p'_{n+1}, ..., p'_m\}$$

$$T_2 = < ID_2, PROMPT_2, tg_2, \{pre_{21}, ... pre_{2m}\}, P_2, CSTR_2, SYS_2, Post_2 >$$

$$pre_{2i} = < condition_{2i}, message_{2i} >$$

$$P_2 = \{p_1, p_2, ..., p_n, p_{n+1}, ..., p_k\}$$

$$T_1 \approx T_2 \leftrightarrow combine(T_1, T_2) = T_{new}$$

$$T_{new} = \{ID_{new}, tg_{new}, PROMPT_1, PRE_{new}, P_{new}, CSTR_1, SYS_1, Post_{new}\}$$

$$tg_{new} = tg_1 \cup tg_2$$

$$PRE_{new} = \{< condition, message >\}$$

$$condition = (condition_{11} \wedge ... \wedge condition_{1m}) \vee (condition_{21} \wedge ... \wedge condition_{2m})$$

$$\begin{aligned} message = &\text{"One of the following problems occurs :"} + message_{11} + \\ &\text{"} or\text{"} + ... + message_{1m} + \text{"} or\text{"} + message_{21} + \text{"} or\text{"} + ... + message_{2m} \end{aligned}$$

$$P_{new} = \{p_1, p_2, ..., p_n, p''_{n+1}, ..., p''_k, p'''_{n+1}, ..., p'''_m\}$$

$$p_i = < ID_i, \{BOOL_{i1}, ..., BOOL_{in}\}\{PROMPT_{i1}, ..., PROMPT_{in1}\}, ig_i, dg_i, Infer_i >$$

$$p'_i = < ID_i', \{BOOL'_{i1}, ..., BOOL'_{in}\}\{PROMPT'_{i1}, ..., PROMPT'_{in1}\}, ig'_i, dg'_i, Infer'_i >$$

$$\begin{aligned} P''_i &= p_i \qquad \text{if } \mathrm{BOOL}_{\mathrm{opt}} \text{ of } \mathrm{p_i} = true \\ &= < ID_i, \{BOOL_{i1}, ...\}\{PROMPT_{i1}, ..., PROMPT_{in1}\}, ig_i, dg_i, Infer_i'' > \quad \text{Otherwise} \end{aligned}$$

$$Infer''_i = Infer_i \cup if\,(\exists p \in \{p'''_{n+1}, ..., p'''_m\}[p \neq null] \wedge [p \neq \text{"}don'tcare\text{"}])\, p''_i = \text{"}don'tcare\text{"}$$

$$\begin{aligned} P'''_i &= p'_i \qquad \text{if } \mathrm{BOOL}_{\mathrm{opt}} \text{ of } \mathrm{p'_i} = true \\ &= < ID'_i, \{BOOL'_{i1}, ...\}\{PROMPT'_{i1}, ...\}, ig'_i, dg'_i, Infer_i''' > \quad \text{Otherwise} \end{aligned}$$

$$Infer'''_i = Infer'_i \cup if\,(\exists p \in \{p''_{n+1}, ..., p''_m\}[p \neq null] \wedge [p \neq don'tcare])\, p''_i = \text{"}don'tcare\text{"}$$

$$\begin{aligned} Post_{new} = IF\ &(Condition_{11} \wedge ... \wedge Condition_{1m}) == false \text{ THEN } Post_2 \\ &\text{ELSE IF } (Condition_{21} \wedge ... \wedge Condition_{2m}) == false \text{ THEN } Post_1 \\ &\text{ELSE ASK\_DESIGNER}(Post_1, Post_2, Post_1 + Post_2) \end{aligned}$$

Figure 7-9 Combination process for merging two similar transactions

This solution for parameter merging has a disadvantage, namely, if the user does not provide any initial information for one parameter of $\{p_{n+1}, ..., p_k\}$ or $\{p'_{n+1}, ..., p'_m\}$, the system will ask for the first unknown necessary parameter, and this would be one of $\{p_{n+1}, ..., p_k\}$. It is not possible to switch to the other possibility $\{p'_{n+1}, ..., p'_m\}$ afterwards. But this is acceptable; because we assume the systems all work well before combination, disabling the other possible option after the first system's question to one deviated parameter does not affect the usability and function of the transaction.

Similar parameters

Similar parameters might share the corresponding information provided by the user for them. But not all similar parameters have to share the information with each other. For example "city" for weather does not have to be the same as "departure city" for flight even when they both contain city names such as "Munich", "Berlin", etc. Therefore, the designer will be consulted as to whether the judged "information shared parameter pair" is valid.

In a confirmed case, the parameters should be inferred from each other in some correlation. The correlation of information-shared parameters can be specified by the inference rules of each parameter accordingly. The correlation could be a simple equation or have a more complicated form. In the following form, I adopt f(p) to express some calculation with p such as "p+2", "p-3", etc.

$$merge(p_1, p_2) \rightarrow Infer_1 = "p_1 = \mathrm{f}(p_2)" \wedge Infer_2 = "p_2 = \mathrm{f}(p_1)"$$

In DIANE, all parameter values in a dialog session are continuously stored in the engine's memory. So any parameter value of executed transactions can be accessed again later in the same dialog session. This memory of the dialog engine is "public storage", where the values of different parameters are stored and can be accessed by different transactions.

Due to the limited domain knowledge of the combination tool, it is not capable of determining the exact parameter correlation. The designer must assist the combination tool by telling it the right correlation. If it is just a simple equation, then the combination tool can automatically generate the corresponding inference rule, e.g. the city of a hotel reservation would be inferred by the destination city of a flight reservation, and vice versa.

If the correlation is more sophisticated, the designer has to write the inference rule manually, e.g. the time of the transaction "wake up call" could be related to the departure time of a flight in the morning. The two "times" are not identical but are related to each other. It is normal to get up 2 hours earlier to prepare for a trip, but in reality the exact time difference may vary from user to user. Such an inference rule is always domain-specific, and thus must be specified by the designer.

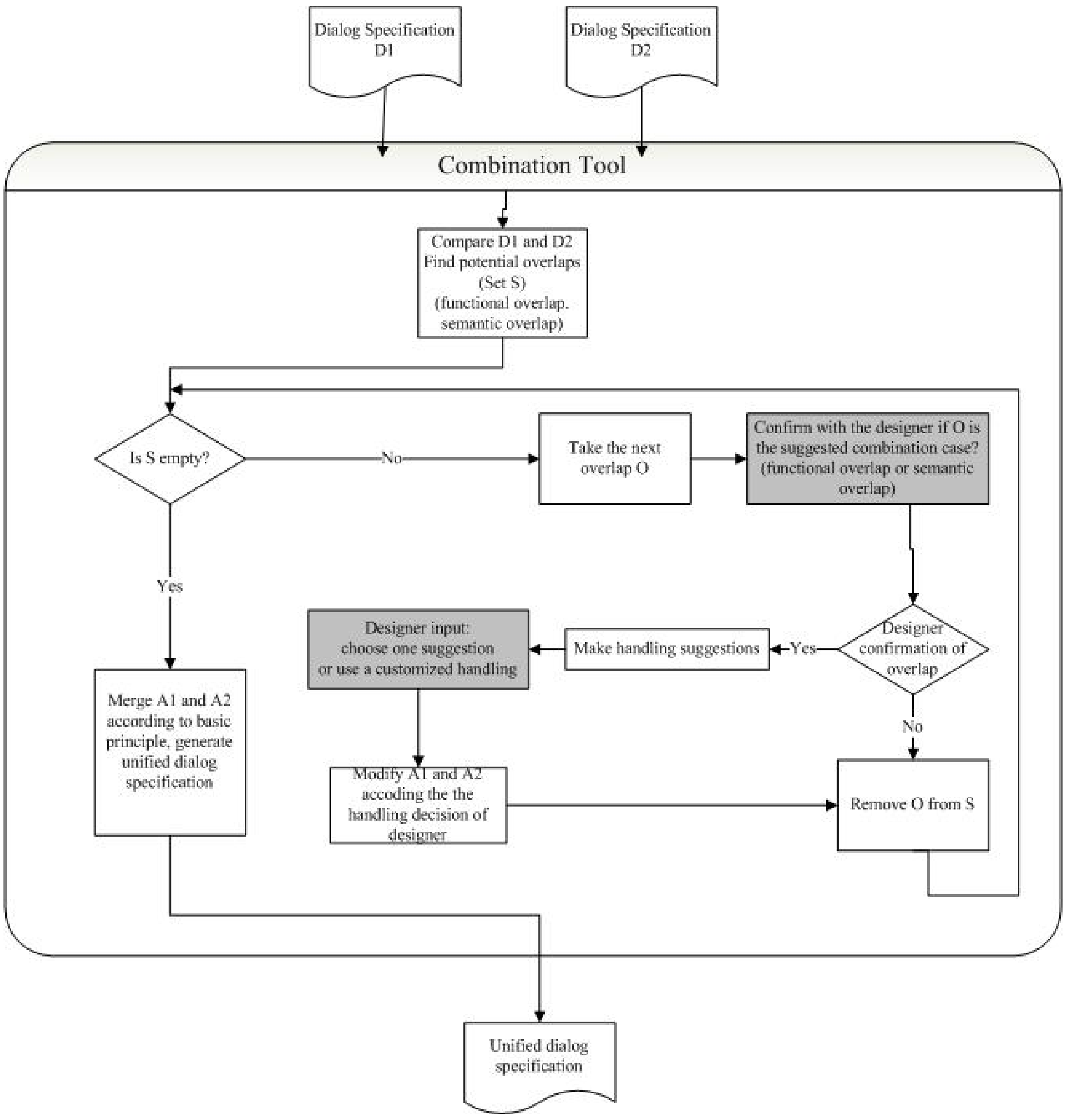
With the general algorithm and inference mechanism, the combination tool can handle the simple information-sharing cases almost automatically, based on the designer's guidance. Due to the limited domain knowledge, which is unavoidable as long as no standard definition of language meaning exists and there is no common basic reference information layer for all applications, the best I can offer is a customization in more sophisticated information-sharing cases.

Applications merging and new application construction

After solving all overlaps the applications can be combined, thanks to the basic combination principle. Disjunctive transactions will all be one-to-one written in the unified transaction file. Overlapped transactions will be merged and constructed according to the handling in Step 3. Information sharing will be implemented by parameter inferences.

7.4.2 Combination Tool

For the interactive combination process introduced in last section, a prototype for an interactive combination tool has been developed. The following figure illustrates the control flow of the combination tool:

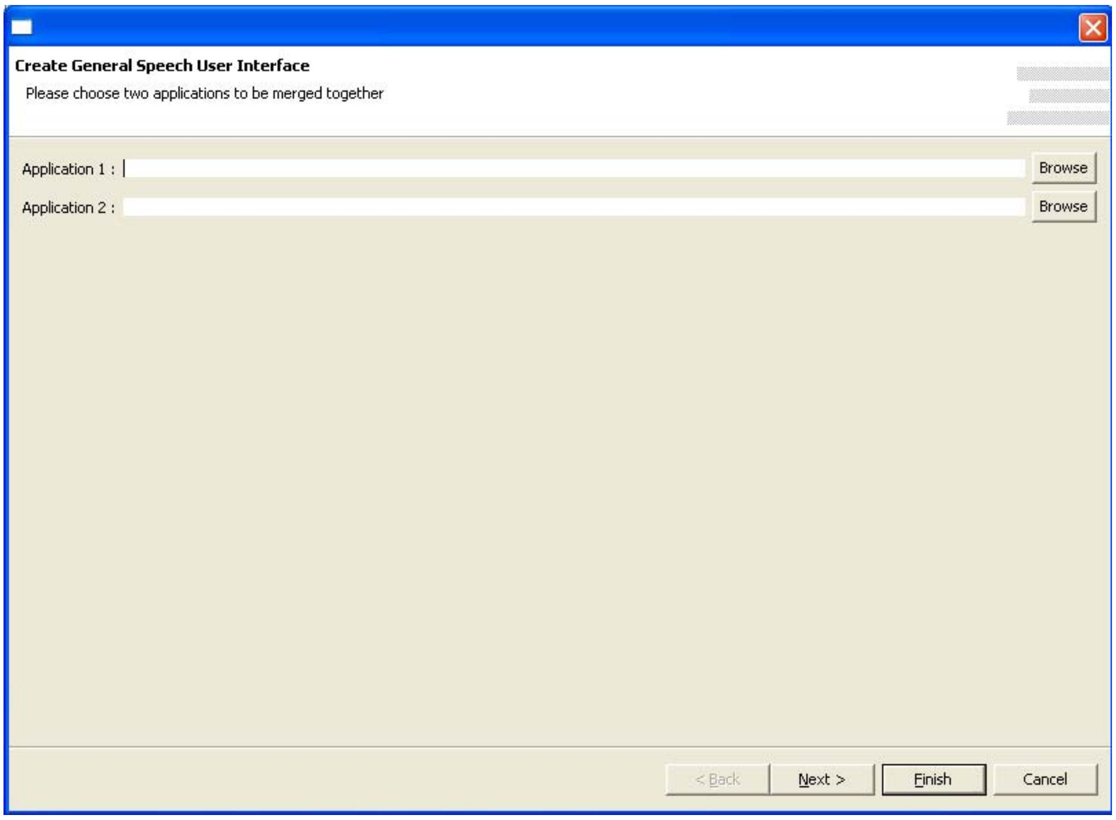


Dialog Specification D1
Dialog Specification D2
Combination Tool
Compare D1 and D2 Find potential overlaps (Set S) (functional overlap, semantic overlap)
Is S empty?
No
Take the next overlap O
Confirm with the designer if O is the suggested combination case? (functional overlap or semantic overlap)
Yes
Designer confirmation of overlap
Make handling suggestions
Designer input: choose one suggestion or use a customized handling
Merge A1 and A2 according to basic principle, generate unified dialog specification
Modify A1 and A2 accoding the the handling decision of designer
Remove O from S
Unified dialog specification

Figure 7-10 The interactive combination tool

White boxes refer to the steps performed by the combination tool automatically. Grey boxes indicate steps requiring designer's corporation. The combination tool works as follows:

1. The combination tool obtains two dialog specifications D1 and D2 as input. Figure 7-11 shows the screen shots of the implemented combination tool for this step:


Create General Speech User Interface

Please choose two applications to be merged together

Application 1 :

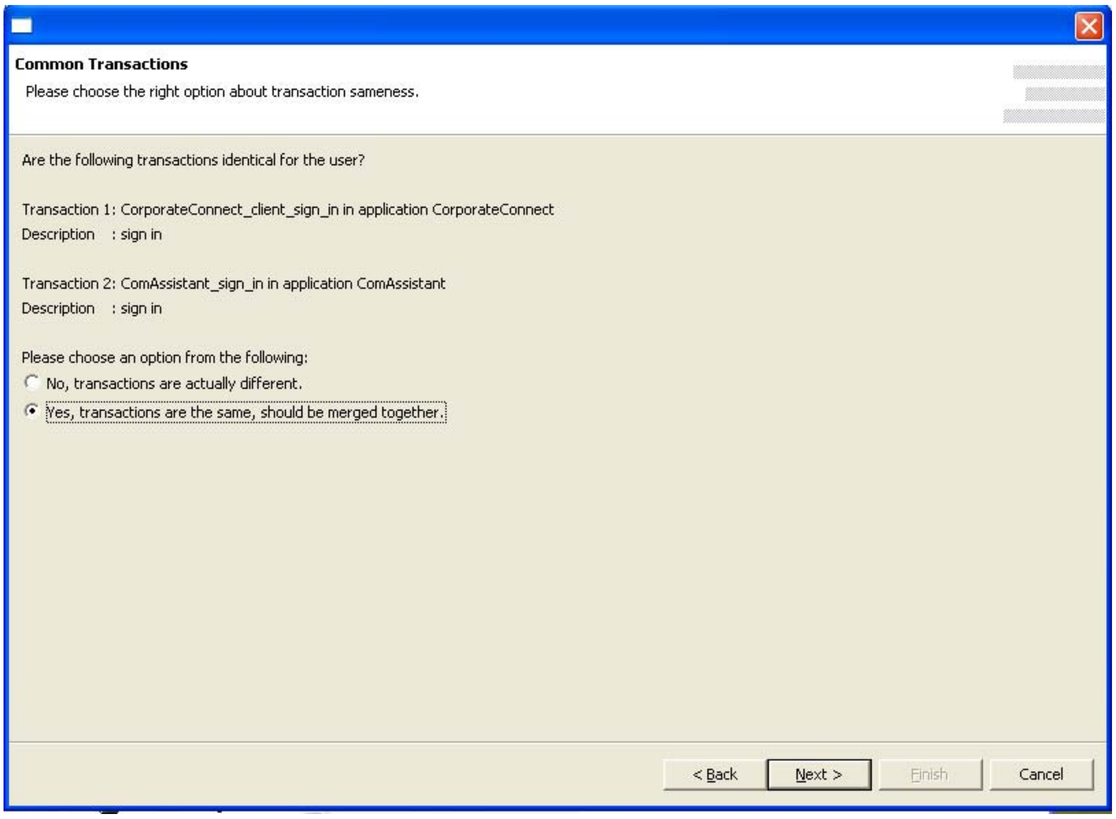
Application 2 :

Figure 7-11 Input dialog of the combination tool

2. The tool compares D1 and D2, finds the set of overlaps S which has to be handled. The overlaps which occur in the combination scenarios are functional overlap, and semantic overlap.

3. If the set S is not empty, then go to Step 4. If the set S is empty, it means that there are no overlaps between two speech applications or the overlaps have all been handled appropriately, so go to Step 7.

4. The tool takes the next overlap O in the set S and asks the designer for a confirmation if the judged overlap is valid. If the designer declines the judgment, then this overlap is removed from Set S, and the next move is to go to Step 3. Otherwise the designer confirms the tool's judgment, then goes to Step 5. I tested the combination tool with the case of combining two communication systems – ComAssistatnt and CorporateConnect – developed as HiPath applications by Siemens. These two applications have a set of overlaps. One is that both applications have transactions for logging in. Figure 7-12 shows the screen shots of this tool for Step 4 when combining these two applications:



Common Transactions

Please choose the right option about transaction sameness.

Are the following transactions identical for the user?

Transaction 1: CorporateConnect_client_sign_in in application CorporateConnect
Description : sign in

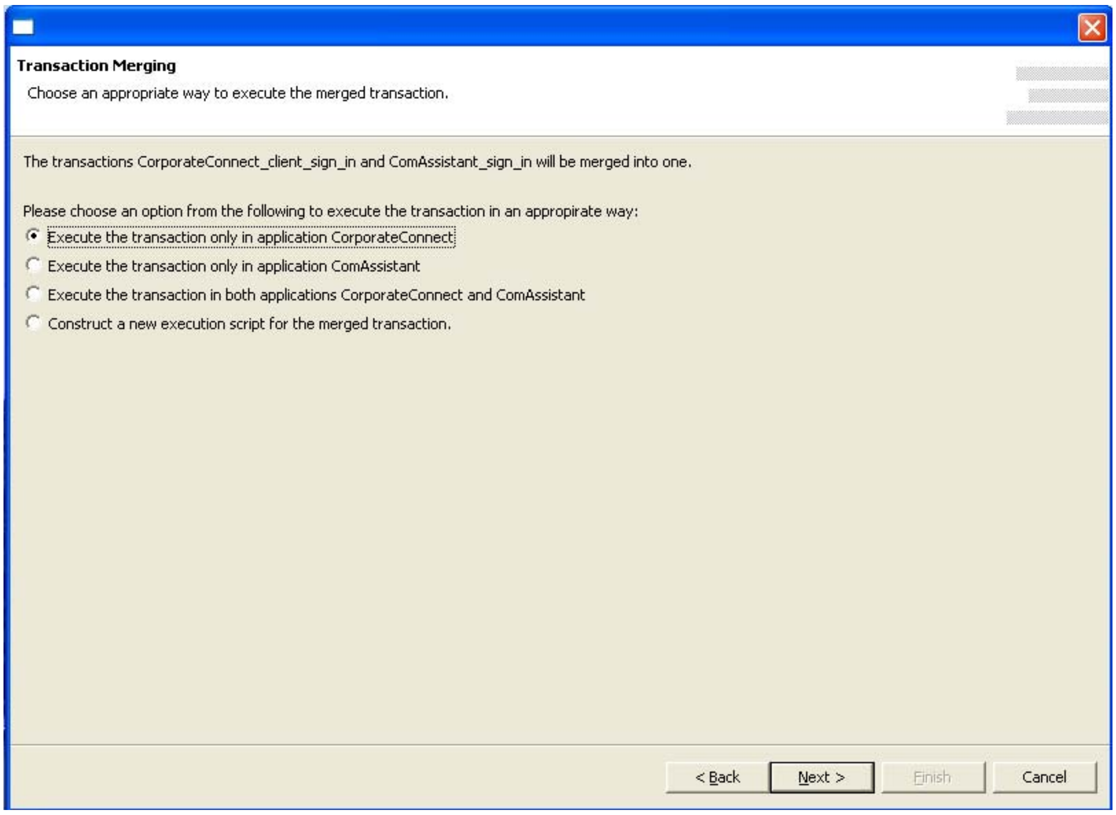
Transaction 2: ComAssistant_sign_in in application ComAssistant
Description : sign in

Please choose an option from the following:

No, transactions are actually different. [✓ Yes, transactions are the same, should be merged together.]

Figure 7-12 Overlap confirmation dialog of the combination tool

5. The tool makes several handling suggestions for the overlap O. The designer can either choose one suggestion to solve the overlap, or input a customized modification. Figure 7-13 shows the screen shots of the combination tool for suggesting the designer should handle the overlap mentioned above in Step 4.



Transaction Merging

Choose an appropriate way to execute the merged transaction.

The transactions CorporateConnect_client_sign_in and ComAssistant_sign_in will be merged into one.

Please choose an option from the following to execute the transaction in an appropirate way:

- [✓ Execute the transaction only in application CorporateConnect]
- Execute the transaction only in application ComAssistant
- Execute the transaction in both applications CorporateConnect and ComAssistant
- Construct a new execution script for the merged transaction.

Figure 7-13 Overlap handling dialog of the combination tool

6. The decision in Step 5 will be applied to D1 and D2. The overlap O will be removed from the set S, then the next move is to go back to Step 3.
7. A unified dialog specification for both applications will be generated according to the basic combination principle, since all overlaps have been handled in the previous steps.

In this design the tool cooperates with the designer in order to generate an intelligent speech user interface with moderate efforts from the designer side.

This combination tool has been developed as a prototype which has been proven to be fully operational. Different applications have been combined with this tool. There are complex communication systems developed by Siemens such as ComAssistant, CorporateConnect and Xpressions, test applications developed by ourselves such as coffee machine control, light control and also information-providing systems such as geographic information about Asia and football match results. The combination results are fully operational in the DIANE dialog system. Based on the generated dialog specification, the DIANE dialog system can provide an integrated speech user interface for the user to access the applications directly.

7.5 Combination Issues of Context Specifications

The combination issues of context specification are discussed separately because most frame-based dialog systems do not contain the concept of context specification. "Context" as an element for dialog specification is somewhat lately introduced in this dissertation. Compared with transactions or parameters, "context" is not a necessary element that must be specified in the dialog specification. It is rather a very good assistance element, with which the dialog between the user and the system can be constructed more intelligently.

In this section I discuss the combination issue with respective to the element "context". In Chapter 4, the helpful specification element context and the corresponding dialog management mechanism were introduced. With the help of context specification, the user can express his intention related to the current dialog context. For example, in an application the user can "delete an Email", "delete an SMS", "read an Email", etc. With a "context" element, "delete an Email" and "read an Email" are supposed to be specified in the same context, whereas the transaction "delete an SMS" will be in a different context from the other two transactions. After listening an Email by saying "please read the first new email", the user can delete this email by saying "delete it" without having to specify "it" to be an Email or an SMS. This is obvious for the human; however for the computer, without context specification it is ambiguous.

There are different issues for using the context scheme in an integrated speech user interface which combines two speech applications with different contexts. In the following I explain these issues and their solutions accordingly.

Basic Principle

Context Specification

The contextual relations between different transactions do not change because of the merging. So after merging the transactions of two applications together, the contexts defined in each application should be retained in the new application.

Concerning the normal work routine of a user, he/she always tends to execute the transactions in the same application successively. So for each application, a new context will be introduced to connect the transactions in the same application contextually. This new context contains all transactions in the corresponding application. This basic principle can be applied to the disjunctive combination scenario.

$A_1 = < \{T_{11}, ...T_{1n}\}, \{C_{11}, ...C_{1n}\} >$

$A_2 =< \{T_{21},...T_{2n}\},\{C_{21},...C_{2n}\} >$

A_1 and A_2 are disjunctive $\rightarrow A_1 + A_2 =< \{....\},\{C_{11},...,C_{1n},C_{21},...C_{2n},C_{A1},C_{A2}\} >$

$C_{A1} = \{TID_{11},...TID_{1n}\}$

$C_{A2} = \{TID_{21},...TID_{2n}\}$

Context specification in case of functional overlap

If there are any similar transactions in two applications, which are merged together in the integrated speech user interface, the contexts containing these transactions should be modified accordingly, because the original transactions do not exist anymore.

After merging two similar transactions T1 and T2 into a unified transaction T3, the occurrence of T1 and T2 in all contexts should be replaced by T3.

$T_1 \approx T_2 \rightarrow A_1 + A_2 =< \{....\},\{C'_{11},...,C'_{1n},C'_{21},...C'_{2n},C'_{A1},C'_{A2}\} >$

$C'_i = C_i \text{ if } TID_1 \notin C_i \wedge TID_2 \notin C_i$

$\quad = C_i - \{\mathrm{TID}_1, \mathrm{TID}_2\} + \{\mathrm{TID}_3\} \text{ if } TID_1 \in C_i \vee TID_2 \in C_i$

Adapting the context specification for the integrated speech application, the dialog system can address the right transaction more efficiently. Through the introduction of a new context for each application, we simulate the fact that the user normally prefers to access transactions inside one application, rather than execute cross-application transactions.

For example, an application A1 consisting of transactions “activate SMS notification of lottery win” (ASN), “deactivate SMS notification of lottery win”, “buy a lottery ticket” (BL) and “search a contact” (SC) and an application consisting of transactions “activate SMS notification of email reception” (ASE), “deactivate SMS notification of email reception”, “check email”(CE), “search a contact” (SC) and “edit a contact” (EC) should be combined together. In Application A1, the transactions related to the lottery are grouped into the same context. In Application A2, the transactions related to email are grouped into the same context and the transactions related to contact are grouped into the same context. The original context specifications of both applications are defined as following:

$A_1 =< \{ASN, DSN, BL, SC\},\{\{"ASN","DSN","BL"\}\} >$
$A_2 =< \{ASE, DSE, CE, SC, EC\},\{\{"ASE","DSE","CE"\},\{"SC","EC"\}\} >$

When combining A1 and A2 together, two transactions are found to be similar – SC in A1 and SC in A2. So these two transactions are merged together into a new transaction SC1.

The context specification of the new merged application will be generated into the following according to the algorithm introduced above:

$A_1 + A_2 = < \{ASN, DSN, BL, SC1, ASE, DSE, CE, EC\}, \{C_1, C_2, C_3, C_4, C_5\} >$

$C_1 = \{"ASN", "DSN", "BL"\}$

$C_2 = \{"SC1", "EC"\}$

$C_3 = \{"ASE", "DSE", "CE"\}$

$C_4 = \{"ASN", "DSN", "BL", "SC1"\}$

$C_5 = \{"ASE", "DSE", "CE", "SC1", "EC"\}$

The context C1 is inherited from application A1. The contexts C2 and C3 are inherited from Application A2. The context C4 is constructed to bind the transactions in Application A1 in one group. The context C5 is constructed to bind the transactions in Application A2 in one group. The overlapped transactions “SC” in both applications are replaced by the new ID “SC1” in the contexts of the merged application.

With the help of this context specification, the new version of the DIANE System which has been created based on the concepts of this thesis and which supports the enhanced frame-based model can engage in a conversation such as the following:

S: Welcome to the system, how can I help you?
U: I want to buy a lottery ticket …
…. (Dialog for buying lottery ticket)
S: Your lottery ticket has been bought.
U: Notify me by SMS.
(*Two transactions are triggered by the utterance “ASB” and “ASE”, due to the active contexts {C1, C4}, the transaction ASB is chosen without a clarification dialog*)
S: The SMS notification for lottery win has been activated.
U: Activate the SMS notification for email reception.
U: Edit a contact.
… (Dialog for contact editing)
U: Deactivate the SMS notification.
(*Two transactions are triggered by the utterance “DSB” and “DSE”, due to the active contexts {C5}, the transaction DSE is chosen without further clarification, the user stays in the same application A2*)
S: The SMS notification for email reception has been deactivated.

7.6 Summary

The combination scheme proposed in this chapter can generate a unified dialog specification based on the declarative specifications of two applications. This integrated speech user interface enables speech access to both applications. The construction of this multi-dialog system is automatic in simple cases and is semi-automatically supported by a combination tool, which allows for customization by a dialog designer. More speech applications can be combined together by applying the same process iteratively to add another speech application to the merged application. The tool has proven to be fully operational on different non-trivial realistic applications. Compared with existing multi-domain or multi-application dialog systems, my approach merges different dialog specifications at the functional layer and thus enables transparent access to different applications with a normal frame-based dialog system. The approach also solves the task-sharing and information-sharing problems, which, due to my best knowledge, are not addressed in other research studies yet.

The following table compares my approach – Frame-based Multi-application Dialog System (FMDS) - with two existing approaches for constructing multi-application dialog systems, which are particularly representative of research focusing on multi-application dialog systems. One is the meta-dialog manager based Multi-domain Dialog System (MDMDS) [Seneff & Polifroni, 1999]. The other one is the Application-based Multi-application Dialog System (AMDS) [Bui et al., 2005].

	Automatic Construction	**Transparent Access**	**Task sharing**	**Information sharing**
FMDS	Y/N	Yes	Yes	Yes
MDMDS	No	No	No	No
AMDS	Yes	Yes	No	No

Table 7-3 Comparison of different multi-application dialog systems

The automatic construction represents the process to construct a multi-application dialog system from several single-domain dialog systems automatically without changing the existing infrastructures. My frame-based approach (FMDS) handles simple cases without task and information sharing completely automatically. Only in more sophisticated scenarios is the involvement of the designer required. The meta-dialog manager in MDMDS must be extended to support a new dialog manager, so this approach does not support any automatic construction. The AMDS proposed in

Bui et al. [2005] constructs an application hierarchy automatically, but the corresponding scenarios handled here can also be handled automatically in the FMDS approach.
The transparent access to different applications is supported by both FMDS and AMDS. An explicit domain switch is required in MDMDS.
The task-sharing and information-sharing problems are so far only considered and solved in FMDS.

In summary, the combination scheme proposed in this chapter is novel owing to the idea of constructing a multi-application dialog system by combining the dialog specifications of different applications at the frame/function layer.

Chapter 8

Conclusion

This chapter first introduces other scientific work related to this dissertation. It then discusses interesting issues that surfaced while writing this dissertation and suggests directions for future work. The last section concludes the thesis.

8.1 Related Work

This dissertation aims to propose a methodology for constructing an integrated speech user interface by combining existing dialog specifications. The methodology works even better if the interfaces of the applications are constructed in a way such that they can be combined easily. As far as I know, this perspective – combining speech user interfaces – has not been studied by anyone yet. The existing research on multi-domain speech dialog systems has been reported further above in Chapter 7.1.1 and does not really address the combination of pre-existing service building blocks. However, there is a similar research trend in the area of Web services where complex services are composed from smaller building blocks. So I give an overview of research work in the Web services composition area in the following section. Further, another issue addressed by this thesis is the comparison of semantics. The proposed solution here is to use the grammar as indicator for semantics of different functions and information. There is a general research trend called "semantic web" for constructing a common and standard semantic foundation for various websites. I will give an overview of semantic web in the second sub-section.

8.1.1 Web Service Composition

I have discussed how to combine different speech user interfaces in this thesis. This composition aspect has not been discussed much in the area of dialog systems, but in the area of web services the composition aspect has been increasingly in focus in the last few years. Some ideas in this thesis are inspired by the solutions in the web service areas. In the following, I give an overview of general web service concepts and web service composition approaches.

A web service is defined as a software application identified by a URI (Universal Resource Identifier), whose interfaces and binding are capable of being defined and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols. SOAP (Simple Object Access Protocol) [Gudgin, et al., 2003], WSDL(Web service Definition Language) [Christensen et. al, 2001] and UDDI (Universal Description, Discovery and Integration) [UDDI, 2001], which are based on XML and Internet technology, constitute the core standards of Web services. SOAP defines a simple XML based protocol for exchange of structured information in a decentralized and distributed environment. A SOAP message, which encapsulates information on the encoding rule for expressing application-defined data types and the invocation of remote procedure calls and their response in a so-called SOAP envelope, is used to access Web services by users in a platform-independent and language-neutral approach. WSDL provides an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described in an abstract fashion, and then bound to a concrete network protocol and message format to define an endpoint. UDDI is a set of definitions for a services registry where information on businesses, organizations, available Web services and technical interfaces for accessing services are stored, including the definition for services of publishing and discovery of information. SOAP, WSDL and UDDI have gained wide support and consensus, and serve as fundamental cornerstone of Web services technology.
The key to Web services is on-the-fly software creation through the use of loosely coupled, reusable software components. Applications are assembled from a set of appropriate Web services and no longer written manually. Seamless composition of Web services has enormous potential in streamlining business-to-business transactions or in enterprise application integration.
Compositions of Web services are created in many different ways. Many compositions are created manually by the service provider by taking simple Web-accessible programs, such as a form-validation program or database lookup program, and composing these programs using typical procedural programming constructs such as if-then-else, sequence or while-loop. A number of software systems are available to facilitate manual composition of Web services. Such programs including a diversity of workflow tools [Van der Alast & Woflan, 1999] enable a user to manually specify a composition of Web services to perform some task. Most recently, technologies have been proposed that use some form of semantic markup of Web

services in order to automatically compose Web services to perform some desired task.

Manual composition supported by flow specification languages

To support the manual composition of Web services, different Web service composition languages such as BPEL4WS [Curbera et al., 2002], WSFL [Leymann, 2001], and BPML [Arkin, 2002] have been proposed by different major software vendors like IBM, Microsoft and Sun Microsystems. They help to specify the workflow of different Web services. Here I explain the BPEL4WS scheme as an example.

BPEL4WS (Business Process Execution Language for Web Service) is an XML-based language for the formal specification of business processes and business interaction protocols. To model the composition of different Web services, it specifies the roles of each of the partners (Web services) and the logical flow of the message exchanges. BPEL4WS binds Web services into cohesive units encapsulated in activities. An activity is either a primitive activity or structured activity. The major primitive activities include:

- invoke – invoking an operation of external Web services
- receive – waiting for the messages from external source
- replay – used together with receive to replay results to external source
- assign – the assignment of values to variables

To represent complex control structures, structured activities are incorporated. Five structured activities are defined including:

- Sequence – defines the sequence of execution of activities
- Pick – used for making a choice of process based on conditions
- While – supports the repeated execution of activities
- Switch – supports conditional behavior
- Flow – used for parallel execution of activities

The structured activities can be nested and combined in arbitrary ways. By this means, the composition of different Web services can be specified with BPEL4WS accordingly. An example composing customer, flight and hotel services together with BPEL4WS and WSDL can be found in Srivastava and Koehler [2003].

BPEL4WS provides a specific language in addition to the existing Web service specification languages, so that it is possible to describe the composition of different Web services in a formal way. It considers Web services as the minimal composition unit and aims at a combination at the structural, not the semantic, layer. The composition is sequential. The problem solved by BPEL4WS is how to use a set of

Web services to provide a new function. It constructs a set of Web services as a result.

In comparison, the combination scheme proposed in this thesis analyzes the dialog specifications in the next smallest building blocks – transactions, parameters – and aims at a combination at the semantic, not the structural, layer. It splits the dialog specifications of different applications into smaller building blocks and reconstructs these building blocks into an integrated dialog specification. The composition is parallel. The goal is not to provide any new functions, but to bring different functions provided by different applications together in parallel. The problem solved by this thesis is how to solve the conflicts between different applications to provide an integrated speech user interface. The conflicts are after all the redundant functions provided by more applications. The combination results of this thesis are not a set of dialog specifications but one exact integrated dialog specification. This dialog specification solves all conflicts between different applications and represents domains and functions of all these applications.

In summary, BPEL4S aims at a structural, sequential composition of Web Services. My combination scheme aims at a semantic, parallel composition of dialog specifications. Furthermore, Web services cannot actually be compared with the dialog specification; they are rather similar to the next smallest building blocks of a dialog specification – transactions. However, both propositions confront one common problem. That is, how to determine the exact semantic of a Web service/transaction. The semantic is the function these elements exactly provide. BPEL4S counts on the comparison of input and output of Web services. My thesis is based on the comparison of grammars which represent how a user expresses the transaction. This is actually the semantic of a transaction expressed in natural language. Again, we see the difference of a structural composition (BPEL4S) and a semantic merging (my combination scheme).

Automatic Composition of Semantic Web Services

The automatic composition of semantic web services can be regarded as a solution involving two parts – semantic markup of the content and capabilities of Web services based on pre-agreed ontologies and AI planning for automatic composition of semantically well-described web services.

OWL-S [Martin et al., 2004], formerly DAML-S [Ankolekar et al., 2002], is the usually used ontology for web services. OWL-S defines **Service class** to model web services with the properties *presents*, *describedBy* and *supports*. The properties in turn have classes *ServiceProfile*, *ServiceModel* and *ServiceGrounding* as their respective ranges.

- The *ServiceProfile* gives a high-level description of the service that can be used to advertise its features and used by clients to select and locate the service from registries. The most important information it contains are the inputs, outputs, preconditions and postconditions of the service.
- The *ServiceModel* is a detailed description of the service in which it is modeled as a process. This description is further sub-divided into a process model, which describes the sub-components of the service and a process control model, which provides a runtime framework to monitor the execution of the service. In the process model description of a composite process, the sub-processes dependencies and interactions can be expressed by Sequences, Split, Unordered, etc.
- The *ServiceGrounding* provides the binding level information of how a client can access the service, e.g. by using SOAP or JAVA RMI.

The services capabilities annotated in OWL-S are now machine-understandable. So, given a goal description, an appropriate plan for composing corresponding web services to achieve the goal can be constructed based on AI planning techniques [McIlraith & Son, 2002]

Compared to the purpose of this dissertation, web service composition aims more at constructing a composite service on-the-fly, based on declarative description statically or dynamically and manually or automatically. The objective is to construct a new service. In this dissertation the main goal is to combine different speech user interfaces into a unified speech-enabled access layer. To construct new functions based on existing ones is not the goal of this thesis. Rather a harmonized access layer facilitating reusability of existing individual speech user interfaces and interoperability between different applications has been achieved here. Nevertheless, the idea of describing services more specifically and semantically with pre- and postconditions (as in the OWL service profiles), and various composition constructions introduced in web service composition approaches, inspired many ideas in this thesis.

8.1.2 Semantic Web

Semantic web knowledge has been already introduced as a pre-agreed knowledge base for describing the semantic content and capabilities of different web services. This has already been discussed in the last section. Here I refer to semantic web as a related work in the area of common knowledge bases for general semantic understanding. In combining different speech user interfaces, one crucial problem I have met is comparing different functions and finding out their relations. However, without a general common knowledge base for all applications, the relations cannot be determined mechanically. This dissertation proposes a heuristic solution based on comparison of grammars which define natural languages. In this section, I discuss the use of semantic web for providing a common framework that allows data to be shared and reused across application, enterprise and community boundaries.

For the semantic web to function, computers must have access to structured collections of information and sets of inference rules that they can use to conduct automated reasoning. For this purpose, two technologies are adopted in the Semantic Web - eXtensible Markup Language (XML) and the Resource Description Framework (RDF). XML allows users to add arbitrary structure to their documents but says nothing about what the structures mean. Meaning is expressed by RDF, which encodes it in sets of triples, each triple being rather like the subject, predicate and object of an elementary sentence. These triples can be written using XML tags. In RDF, a document makes assertions that particular things (people, Web pages or whatever) have properties (such as "is a sister of," "is the author of") with certain values (another person, another Web page). This structure turns out to be a natural way to describe the vast majority of the data processed by machines. Subject and object are each identified by a Universal Resource Identifier (URI), just as used in a link on a Web page. (URLs, Uniform Resource Locators, are the most common type of URI.) The predicates are also identified by URIs, which enables anyone to define a new concept, a new predicate, just by defining a URI for it somewhere on the Web.

The triples of RDF form webs of information about related things. Because RDF uses URIs to encode this information in a document, the URIs ensure that concepts are not just words in a document but are tied to a unique definition that everyone can find on the Web. For example, imagine that we have access to a variety of databases with information about people, including their addresses. If we want to find people living in a specific zip code area, we need to know which fields in each database represent names and which represent zip codes. RDF can specify that "(field 5 in database A) (is a field of type) (zip code)," using URIs rather than phrases for each term.

However, two databases may use different identifiers for what is in fact the same concept, such as *zip code*. A program that wants to compare or combine information across the two databases has to know that these two terms are being used to mean the same thing. Ideally, the program must have a way to discover such common meanings for whatever databases it encounters.

A solution to this problem is provided by the third basic component of the Semantic Web, collections of information called ontologies. In philosophy, an ontology is a theory about the nature of existence, of what types of things exist; ontology as a discipline studies such theories. Artificial-intelligence and Web researchers have co-opted the term for their own jargon, and for them an ontology is a document or file that formally defines the relations among terms. The most typical kind of ontology for the Web has a taxonomy and a set of inference rules.

The taxonomy defines classes of objects and relations among them. For example, an *address* may be defined as a type of *location*, and *city codes* may be defined to apply only to *locations*, and so on.

With ontology pages on the Web, solutions to terminology (and other) problems begin to emerge. The meaning of terms or XML codes used on a Web page can be defined by pointers from the page to an ontology. However, the same problems as before now arise if one points to an ontology that defines *addresses* as containing a *zip code* and the other point to one that uses *postal code*. This kind of confusion can be resolved if ontologies (or other Web services) provide equivalence relations: one or both of the ontologies may contain the information that the zip code in the first ontology is equivalent to the postal code in the second ontology.

So the ontology as a shared conceptualization based on the semantic proximity of terms in a specific is the proposed solution for information-comparing in Semantic Web. Transferring this philosophy to the comparing issue discussed in this thesis means that a general ontology for different domains and applications must be constructed first. And in the dialog specification, each function and each parameter must be identified by a unique URI, which points to some term in a defined ontology. Also, different ontologies defined by different designers must be connected with each other so that there are no two different concepts in two different ontologies which are not related but are meant to be the same object. Assuming the existence of such a large ontology base, the comparison of two applications can be based on the comparison of the ontology concepts behind their functions and parameters. So we do not need to compare the natural language, which is sometimes ambiguous, anymore.

However, whether it is realistic to build such a network of ontologies for various speech applications remains an open question, which can be researched further. Considering the unlimited domains in practice, this plan is rather unrealistic. But if the goal is to combine applications in a certain domain such as travel planning, this proposal becomes considerable.

8.2 Discussion

This dissertation addresses issues in the area of user interaction. In this area, the usability – the user acceptance of the integrated speech user interface and the designer acceptance of the interactive combination tool is a key issue. This issue should be considered in the context of this dissertation. However, due to the enormous complexity of addressing this issue, I leave this point to future research based on this dissertation.

8.2.1 Usability of Integrated Speech Application User Interfaces

In an integrated speech user interface for different applications, the usability regarding accessing different applications is an open question. In this dissertation, I assume that an integrated user interface without application separation results has better usability than one requiring the user to switch the application explicitly. However, this proposition should be proved based on a comprehensive usability evaluation. Such an evaluation should consider the following points:

- What are the criteria for a good integrated speech user interface?

  Besides the criteria for a good normal single-domain speech user interface, the special criteria particular to an integrated speech user interface for multiple applications should be discussed. It is an open question as to what the best way to integrate different applications into a speech user interface is. One possibility is proposed in this dissertation – merging all overlapping transactions and sharing common information in different applications, with the application boundary being transparent to the user. Another possibility has been proposed by many existing multi-domain dialog systems (e.g. [Seneff et. al., 1999]) – the applications remain separate from each other and no interoperability is enabled, and the user has to remember different applications and switch the domain explicitly. These two approaches stand for two extreme directions – no interoperability and maximum interoperability. There are also other possibilities in between these two extremes. For example, the applications can remain separate from each other but can share the common information in the integrated speech user interface. How we can compare these different approaches depends first on what criteria are important.
- How can the usability of an integrated speech user interface be empirically tested?

What kind of dialogs is more interesting for a user – crossing different applications or concentrating on a single application? What kind of tasks should be designed to indicate the usability of a speech user interface? These are all challenging questions which would be interesting to research in more detail.

- How to assess the usability based on empirical tests?

  There are different evaluation benchmarks for a speech user interface, such as number of turns for accomplishing a task. Which of them are suitable for judging the usability of an integrated speech user interface is an open question, which should be addressed in detail.

To provide such a comprehensive usability test and evaluation is far beyond the scope of this dissertation. Nevertheless, this dissertation provides the necessary foundations for such an evaluation. The algorithm provided enables the construction of an integrated speech user interface with maximum interoperability. With little modification, an integrated speech user interface with only information sharing and explicit domain switch can be constructed as well. Therefore, this dissertation can be regarded as a necessary precondition for such a comprehensive usability research of integrated speech user interfaces for multiple applications.

8.2.2 Evaluation of the Combination Algorithm

The combination algorithm has been evaluated over certain non-trivial industry applications and has been proven to work well in these scenarios However, to prove the complexity and correctness of the combination algorithm, a systematic methodology is required. This comprehensive evaluation framework should consider the following points:

- What kind of applications should be combined with the algorithm?

  Obviously, not all applications are suitable to have a speech access layer. And, it is still an open question which kind of applications should own a speech user interface. Only by being based on a systematical analysis of different applications can the general applicability of the combination algorithm to all different applications be proven.

- Can all speech-suited applications be combined with the combination algorithm?

  The applications suited to have a speech user interface range from thousands to unlimited. How to prove that the combination algorithm can be applied to all these applications remains a challenging open question.

- The usability of the combination tool

  The combination tool is an interactive tool involving the designer in a combination process. It is an open question whether it is good to involve the designer in all uncertain decision points, or rather whether it is preferred by the designer to use some default setting.

- The correctness of the combination tool

  How can it be proved that the combination tool works correctly? Two possibilities exists theoretically – judgment based on the constructed speech user interfaces, or proof of the correctness of each sub-step in the combination tool.

Due to the scope of this dissertation, such a comprehensive evaluation cannot be finished.

However, based on certain non-trivial example applications, the combination tool has been verified as able to construct a flexible integrated speech user interface with moderate complexity.

The full operational feature of the combination tool can be highlighted by the case of combing the speech user interfaces of two Siemens HiPath applications – ComAssistant and CorporateConnect. In a student project [Schilling, 2005], speech user interfaces for both applications were developed with the extended frame-based dialog model introduced in this thesis. Due to the complexity of these applications, the expressive power of the extended dialog model has been assessed as enough for real speech-suited applications [Schilling, 2005]. There are both functional and semantic overlaps between these two applications. Both of them provide function – log in, call management, etc. In both application there is a parameter called “context”, which represents the user’s communication profile. A “context” parameter can be “work place”, “vacation” or “on the way” etc.

With the combination tool, it is possible to construct an integrated speech user interface for these two applications in a few minutes. The constructed speech user interface can be launched in the DIANE dialog system directly. An example dialog between the user and the integrated speech user interface is shown below:

System (S): Welcome! Please state your user ID.

User (U): One two three four five.

S: Let me repeat that: your user ID is one two three four five, is that correct?

U: Yes.

S: Please say your password!

U: One two three four.

S: Your password is one two three four, is that correct?

U: Yes.

S: Hello Mrs. Williams, welcome to ComAssistant and CorporateConnect! What can I help you?

(Two "log in" transactions of both applications have been merged into one, and the user is logged in to both systems in one log in process.)

U: Call one one one.

S: Calling one one one ….

(The "call" transactions of both applications have been merged, the function is provided by ComAssistant only.)

U: Change my profile.

("Change profile" is a transaction of CorporateConnect.)

S: To which profile do you want to change?

U: To office.

S: Your profile has been changed to **office**.

U: What is my active context now?

("Context query" is a transaction provided by ComAssistant.)

S: Your active context is **office**.

("Profile" in CorporateConnect and "context" in ComAssistant refer to the same semantic, are therefore sharing the same value in the integrated speech user interface. For this, a corresponding inference rule is constructed automatically.)

….

Although the verification based on case studies is convincing, a more comprehensive evaluation would extend this dissertation very well.

8.3 Conclusion

This dissertation was motivated to address the issue of combining different speech user interfaces to construct an integrated speech user interface.

Along with this motivation, different approaches for modeling applications to develop their speech user interfaces have been discussed. In particular, different dialog modeling approaches have been compared as regards to their suitability to be combined together. Based on the analysis result that no existing dialog-modeling approach is actually best tailored to the combination issue, a new frame-based dialog model has been introduced. This dialog model extends the existing frame-based models to describe different dependencies in applications declaratively. This model improves the general power of frame-based approaches. In addition, the generality and power of this model has been verified in a student project [Schilling, 2005].

Based on the enhanced frame-based dialog model, a comprehensive combination scheme for combining different speech applications following the enhanced model has been proposed as the emphasis of this dissertation. Different relations between two applications have been elaborated in detail and a corresponding solution for each combination scenario has been proposed. Based on non-trivial empirical tests the combination scheme has been verified.

During the analysis of application relations, a side issue of comparing context-free grammars has surfaced as an interesting and challenging theme, which had to be solved in order to finish the combination task. Therefore, the theoretical background of grammars and formal languages has been taken into account in an in-depth study. Further, two different comparison algorithms – finite-state modeling based comparison and generation-parsing based comparison – have been introduced and compared accordingly. The generation-parsing based comparison has been proposed as a more simple and efficient algorithm. Later, in the combination tool, this algorithm has been implemented and proven to be acceptable in runtime regarding its efficiency.

Based on the proposed approaches in this thesis, a flexible integrated speech user interface for different applications can be constructed automatically or semi-automatically based on the existing dialog specification of each application. This integrated speech user interface not only enables transparent speech access to multiple applications but also supports maximum interoperability among different applications.

Bibliography

[1] Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L. and Stent, A.: "**An architecture for a generic dialog shell**", *In: Natural Language Engineering, Vol. 6(3-4), pp 213-228*, Cambridge University Press, 2000

[2] Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L. and A. Stent: "**Towards conversational human-computer interaction**", *In: AI Magazine, Vol. 22(4): pp 27-37*, 2001

[3] Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McDermott, McIlraith, S., Narayanan, S., Paolucci, M., Payne, T. and Sycara, K.: "**DAML-S: web service description for the semantic web**", *In*: *Proceedings of the 1st International Semantic Web Conference (ISWC)*, 2002

[4] Austin, H., Oerder, M. and Steinbiss, V.: "**The Philips automatic train timetable information system"**, *In: Speech Communication, Vol. 17: pp 249-262*, 1995

[5] Arkin, A.: "**Business process modeling language (BPML)**", http://www.bpmi.org/ , 2002

[6] Block, H.U., Caspari, R. and Schachtl, S. "**Callable manuals - access to product documentation via voice**", *In: Wahlster, W. (ed.) Special Journal Issue Conversational User Interface. it - Information Technology 46 (2004) 6, München, Oldenbourg Wissenschaftsverlag (ISSN 1611-2776): 299-304*, Munich, 2004

[7] Bohus, D. and Rudnicky, A. I.: "**RavenClaw: Dialog management using hierarchical task decomposition and an expectation agenda**", *In: Proceedings of EUROSPEECH 2003, pp 597-600*, Geneva, 2003

[8] Bui, T. H., Zwiers, J., Nijholt, A., and Poel, M.: “**Generic dialog modeling for multi-application dialog systems**”, *In: Proceedings of 2nd Joint Workshop on Multimodal Interaction and Related Machine Learning Algorithms,* Edinburgh, UK, 2005

[9] Caspari R.: “**Dokumentation zum SUIBuilder für ViCA**“, Version 1.1, *Siemens AG, CT IC 5*, 2003

[10] Catizone, R., Setzer, A. and Wilks, Y.: “**Multimodal dialog management in the COMIC project**“, *In: Proceedings of EACL 2003 Workshop on Dialog Systems: interaction, adaptation, and styles of management*, 2003

[11] Chomsky, N. “**Formal properties of grammars**”, *In: R. Duncan Luce, Robert R. Bush, and Eugene Galanter, editors, Handbook of Mathematical Psychology; Volume II, pp 323-418,* John Wiley, 1963

[12] Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S.: “**Web services description language (WSDL) 1.1**”, W3C, http://www.w3.org/TR/wsdl, 15 March 2001

[13] Chu-Carroll, J.: “**Form-based reasoning for mixed-initiative dialog management in information-query systems**”, *In: Proceedings of EUROSPEECH 99, pp 1519-1522*, Budapest,1999

[14] Cover, T.M. and Thomas, J.A: “**Elements of information theory**”, *Wiley series in telecommunications*, Wiley New York, 1991

[15] Curbera, F., Goland, Y., Klein, J., Leyman, F., Roller, D., Thatte, S. and Weerawarana, S.: “**Business process execution language for web services (BPEL4WS) 1.0**”, http://www.ibm.com/developerworks/library/ws-bpel, Aug. 2002

[16] Decker, S., van Harmelen, F., Broekstra, J., Erdmann, M., Fensel, D., Horrocks, I., Klein, M. and Melnik, S.: “**The semantic web: The roles of xml and rdf**”, *In: IEEE Expert, 15(3)*, 2000

[17] Doest, T., Hugo, Moll, M., Bos, R., Van de Burgt, S. and Nijholt, A,.: "**Language engineering in dialog systems**", *In*: *Computers in Engineering Symposium*, *pp 68–79,* Houston, TX, 1996

[18] Dusan, S. and Flanagan, J.: "**An adaptive dialog system using multimodal language acquisition**", *In: Proceedings of the International CLASS Workshop: Natural, Intelligent and Effective Interaction in Multimodal Dialog Systems,* Copenhagen, Denmark, 2002

[19] Earley, J. C.: "**An efficient context-free parsing algorithm**", Communications of the ACM*, 13(2), pp 94--102*, February 1970

[20] Grimley E. E.: "**Approximating context-free grammars with a finite-state calculus**", In: *Proceedings of the 35th Annual Meting of the Association for Computational Linguistics, Proceedings of the Conference, pp 452--459*, Spain, 1997

[21] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J. and Nielsen, F.: "**SOAP version 1.2 part 1: Messaging framework**", *W3C Recommendation*, http://www.w3.org/TR/soap12-part1/, 2003

[22] Hopcroft, J. E. and Ullman, J. D.: "**Introduction to automata theory, languages, and computation"**, *Addison-Wesley*, 1979

[23] Hunt, A. and McGlashan, S.: "**Speech recognition grammar specification version 1.0**", *W3C*, http://www.w3.org/TR/2004/REC-speech-grammar-20040316/, 2004

[24] Hsu, W. T., Wang, H. M. and Lin, Y. C.: "**The design of a multi-domain Chinese dialog system**", *In: International Symposium on Chinese Spoken Language Processing*, Taiwan, 2002

[25] Johnson, M.: "**Finite-state approximation of constraint-based grammars using left-corner grammar transforms**", *COLING-ACL, pp 619-623,* 1998

[26] JSGF, "**Java speech grammar format specification 1.0**", http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/, 1998

[27] Kasami T.: “**An efficient recognition and syntax algorithm for context-free languages**” *Technical Report AF-CRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA*, 1965

[28] Larsson, S., Cooper, R. and Ericsson, S.: ”**Menu2dialog**”, *In: IJCAI Workshop on Knowledge and Reasoning in Practical Dialog Systems*, 2001

[29] Larsson, S. and Traum, D.: “**Information state and dialog management in the TRINDI dialog move engine toolkit**”, *Natural Language Engineering, 6(3-4), pp 323--340*, 2000

[30] Lemon, O., Bracy, A., Gruenstein, A. and Peters, S.: “**The WITAS multi-modal dialog system**“, *In: Proceedings of EUROSPEECH 2001*, 2001

[31] Leymann F.: “**Web service flow language (WSFL1.0)** “, http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf , 2001

[32] Lin, D.: “**An information-theoretic definition of similarity”**, *In: Proceedings of the 15th International Conf. on Machine Learning, pp 296—304*, Morgan Kaufmann, San Francisco, CA, 1998

[33] Lin, Y. C., Chiang, T. H., Wang, H. M., Peng, C. M. and Chang, C. H.: ”**The design of a multi-domain mandarin Chinese spoken dialog system**” *In: Proceedings ICSLP1998*, *pp 41-44*, 1998

[34] Lin, B., Wang, H. and Lee, L.: “**Consistent dialog across concurrent topics based on an expert system model**”, *In Proceedings of EUROSPEECH 99,* Budapest, 1999

[35] Litman, D. and Allen, J.: “**A plan recognition model for subdialogs in conversation**”, *In: Cognitive Science, Vol. 11: pp 163-200*, 1987

[36] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N. and Sycara, K.: “**Owl-s: Semantic markup for web services**”, http://www.daml.org/services/owl-s/1.1/overview/, 2004

[37] Mason, T. and Brown, D.: “**Lex & Yacc**”. *O'Reilly and Associates, Inc*., CA, 1990

[38] McGlashan, S., Burnett, D., Carter, J., Tryphonas, S., Ferrans, J., Hunt, A., Lucas, B. and Porter, B.: ”**Voice extensible markup language (voicexml) version 2.0**”, *Technical report, World Wide Web Consortium (W3C)*, http://www.w3.org/TR/voicexml20/ , Feb. 2003

[39] McIlraith, S. and Son, T.C.: “**Adapting golog for composition of semantic web services**”, *In: Proceedings of 8th International Conference on Principles of Knowledge Representation and Reasoning, pp 482-493*, 2002

[40] McTear, M.: “**Spoken dialog technology: Enabling the conversational interface**”, *In: ACM Computing Surveys, Vol. 34(1): pp 90 – 169*, 2002

[41] Miller, G. and Chomsky, N.: “**Finitary models of language users**”, *In: R. Luce, R. Bush, and E. Galanter, editors, Handbook of Mathematical Psychology. Volume 2. John Wiley*, 1963

[42] Mohri, M. and Nederhof, M. J.: “**Regular approximation of context-free grammars through transformation**“, *In: Robustness in Language and Speech Technology. pp 153-163, Kluwer Academic Publishers*, 2001

[43] Moore, R. C.: "**Using natural-language knowledge sources in speech recognition**”, *In: Computational Models of Speech Pattern Processing, Ponting (ed.), Springer-Verlag, pp 304-327*, 1999

[44] Nederhof, M. J.: “**Practical experiments with regular approximation of context-free languages**”, *Computational Linguistics, 26(1), pp 17-44*, March 2000a

[45] Nederhof, M. J.: “**Regular approximation of CFLs: a grammatical view**”, *In: H. Bunt and A. Nijholt, editors, Advances in Probabilistic and other Parsing Technologies, chapter 12, pp 221-241*, Kluwer Academic Publishers, 2000b

[46] Nederhof, M. J. and Satta, G.: “**Parsing non-recursive context-free grammars**”, *In: 40th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, pp 112-119*, USA, 2002

[47] Neto, J. P., Mamede, N. J., Cassaca, R. and Oliveira, L. C.: “**The development of a multi-purpose spoken dialog system**”, *In: Proceedings of EUROSPEECH 2003,* 2003

[48] Nouza, J. and Holada, M. “**A voice-operated multi-domain telephone information system**”, *In: Proceedings Acoustics, Speech, and Signal Processing*, 2000

[49] O’Neil, I. M. and McTear, M. F.: ”**Object-oriented modelling of spoken language dialog systems**”, *In: Natural Language Engineering, Vol. 6(3-4): pp 341-362*, Cambridge University Press, 2000

[50] Pakucs, B.: “**Towards dynamic multi-domain dialog processing**”, *In: Proceedings of EUROSPEECH 2003*, Geneva, 2003

[51] Paul, M., Frederick, C., Stuart, A., Eric, B. and Nicole, Y.: “**SpeechActs: a spoken language framework**”, *In: IEEE Computer Vol. 29(7): pp 33-40*, 1996

[52] Pereira, F.C.N. and Wright, R.N..: "**Finite-state approximation of phrase-structure grammars**", *In: Emmanuel Roche and Yves Schabes, editors, FiniteState Language Processing*, *MIT Press, Cambridge, pp 149—173*, 1997

[53] Polifroni, J. and Chung, G.: ”**Promoting portability in dialog management**”, *In: Proceedings ICSLP, pp 2721-2724*, Denver, CO, 2002

[54] Sanford, D.L.: “**A theory of dialog structures to help manage human-computer interaction**”, *In: IEEE Transactions on systems, man and cybernetics, Vol. 18(4): pp 567-574*, 1988

[55] Schilling, A.: ”**Modeling of speech dialog applications**“, *Student Project*, 2005

[56] Seneff, S.: “**Discourse and dialog modeling in the galaxy system**”, *In: Proceedings of Dialog System and Discourse Analysis Workshop,* Taiwan, 1997

[57] Seneff S., Goddeau, D., Pao C. and Polifroni, J.: “**Multimodal discourse modelling in a multi-user multi-domain environment**”, *In: Proceedings of ICSLP’96,* USA*,* 1996

[58] Seneff, S., Lau, R. and Polifroni, J.: “**Organization, communication, and control in the galaxy-II conversational system**”, *In: Proceedings of EUROSPEECH 99, pp 1271-1274*, Budapest, Hungary, 1999

[59] Sheshagiri, M., Desjardins, M. and Finin, T.: ”**A planner for composing service described in daml-s**”, *In: Proceedings Workshop on Planning for Web Services, International Conference on Automated Planning and Scheduling*, Trento, 2003

[60] Sparks, R., Meiskey, L. and Brunner, H.: “**An object-oriented approach to dialog management in spoken language systems**”, *In: Proceedings of the SIGCHI conference on Human factors in computing systems*, 1994

[61] **SpeechObjects specification version 1.0**, http://www.w3.org/TR/2000/NOTE-speechobjects-20001114/#speechobjects, 2000

[62] Srivastava, B. and Koehler, J.: “**Web service composition - current solutions and open problems**”, *In: Proceedings of International Conference on Automated Planning and Scheduling 2003*, 2003

[63] Strum, J., den Os, E. and Boves, L.: “**Dialog management in the Dutch ARISE train timetable information system**”, *In: Proceedings of EUROSPEECH 99*, 1999

[64] Stolcke, A. and Segal, J.:” **Precise n-gram probabilities from stochastic context-free grammars**”, *In: Proceedings of the 32th Annual Meeting of the Association for Computational Linguistics pp 74-79*, 1994

[65] Taylor, C. R.: "**LL parsing, LR parsing, complexity, and automata**", *ACM SIGCSE Bulletin, Volume 34, Issue 4, pp 71-75,* 2000

[66] Teppo, A., and Vuorimaa, P.: "**Speech interface implementation for XML browser**", *International Conference on Auditory Display, Espoo, Finland*, 2001

[67] Traum, D., Andersen, C., Chong, Y., Josyula, D., O'Donovan-Anderson, M., Okamoto, Y., Purang, K. and Perlis, D.: "**Representations of dialog state for domain and task independent meta-dialog**", *In: Electronic Transactions on Artificial Intelligence, Vol. 3: pp 125--152*, 1999a

[68] Traum, D., Bos, J., Cooper, R., Larsson, S., Lewin, I., Matheson, C. and Poesio, M.: "**A model of dialog moves and information state revision**", *Technical report, TRINDI Deliverable D2.1*, 1999b

[69] Traum, D. and Larsson, S.: "**The information state approach to dialog management**", *In: Jan van Kuppevelt and Ronnie Smith, editors, Current and New Directions in Discourse and Dialog,* Kluwer, 2003

[70] UDDI. "**Universal description, discovery, and integration (uddi v2.0)**". http://www.uddi.org/ , October 2001

[71] Vrugt, J. and Protele, T.: "**Application-independent knowledge-processing in a task-oriented speech-dialog-system**", *it – Information Technology, 46(6), pp 306-314*, 2004

[72] Wang, K.: "**An event driven model for dialog systems**", *In: Proceedings of ICSLP 1998,* Sydney, 1998

[73] Wu, D., Sirin, E., Parsia, B., Hendler, J., Nau, D.: "**Automatic web services composition using SHOP2**", *In: Proceedings of Planning for Web Services Workshop in ICAPS 2003*, 2003

[74] Wu, X., Zheng, F. and Wu, W.: "**Hybrid dialog management approach for a flight spoken dialog system**", *In: Proceedings of the First International Conference on Machine Learning an Cybernetics,* Beijing, 2002

[75] Wyard, P. J., Simons, A. D., Appelby, S., Kaneen, E., Williams, S. H. and Preston, K. R.: ”**Spoken language systems – beyond prompt and response**”, *In: BT Technology Journal, Vol. 14(1): pp 187-205,* 1996

Appendices

Appendix 1: DTD Definition of DIANEXML

Transaction DTD Definition

```
<!ENTITY % parameterDTD SYSTEM "Parameter.dtd">

%parameterDTD;

<!ENTITY % boolean "#PCDATA">

<!-- Elements -->

<!ELEMENT Transaction

  (Name

  ,ExeFunc

  ,ExePrompt?

  ,TrPrompt?

  ,TrConfirmBool?

  ,TrConfirmPrompt?

  ,TrGrammar*

  ,TrParameter*

  ,TrAddCode*

  ,(CstrFunc | ConfirmFunc)

  ,CstrPrompt?

  ,SemLessGrammar*

  ,SemLessStartGrammar*

  )>

<!ATTLIST Transaction uri %uri; #IMPLIED>
```

```
<!ELEMENT ExeFunc (#PCDATA)>
<!ELEMENT CstrFunc (#PCDATA)>
<!ELEMENT ExePrompt (%Prompt;)>
<!ELEMENT CstrPrompt (%Prompt;)>
<!ELEMENT ConfirmFunc (%Prompt;)>
<!ELEMENT TrPrompt (%Prompt;)>
<!ELEMENT TrConfirmPrompt (%Prompt;)>
<!ELEMENT TrConfirmBool (%boolean;)>
<!ELEMENT TrGrammar (%grammar;)>
<!ATTLIST TrGrammar uri %uri; #IMPLIED>
<!ELEMENT SemLessGrammar (%grammar;)>
<!ATTLIST SemessGrammar uri %uri; #IMPLIED>
<!ELEMENT SemLessStartGrammar (%grammar;)>
<!ATTLIST SemLessStartGrammar uri %uri; #IMPLIED>
<!ELEMENT TrParameter
 (Name
 ,RecBool?
 ,OptBool?
 ,DefaultVal?
 ,InfBool?
 ,AlwaysConfBool?
 ,UserConfBool?
 ,ConfIfInfBool?
 ,ConfIfDefBool?
 ,InterruptBool?
 ,TestAllBool?
 ,OutOfTaskBool?
 ,CopyPermittedBool?
  )>
<!ELEMENT RecBool (%boolean;)>
<!ELEMENT OptBool (%boolean;)>
<!ELEMENT InfBool (%boolean;)>
<!ELEMENT AlwaysConfBool (%boolean;)>
<!ELEMENT UserConfBool (%boolean;)>
<!ELEMENT ConfIfInfBool (%boolean;)>
<!ELEMENT ConfIfDefBool (%boolean;)>
<!ELEMENT InterruptBool (%boolean;)>
<!ELEMENT TestAllBool (%boolean;)>
<!ELEMENT OutOfTaskBool (%boolean;)>
<!ELEMENT CopyPermittedBool (%boolean;)>
<!ELEMENT DefaultVal (#PCDATA)>
<!ELEMENT TrAddCode (#PCDATA)>
```

Transaction Set DTD Definition

```
<!-- Entities -->
<!ENTITY % TransactionDef SYSTEM "Transaction.dtd">
%TransactionDef;
<!ELEMENT TransactionL
        (Transaction*
        ,SemLessStartGrammar*
        ,SemLessGrammar*
)>
```

Parameter DTD Definition

```
<!-- Entities -->

<!ENTITY % Prompt "#PCDATA">

<!ENTITY % uri "CDATA">

<!ENTITY % grammar "#PCDATA">

<!ENTITY % name "#PCDATA">

<!ELEMENT Parameter

  (Name

  ,PmType?

  ,IGrammar?

  ,DGrammar?

  ,PmPrompt?

  ,PmRekPrompt?

  ,PmRekDiscardPrompt?

  ,PmConfirmPrompt?

  ,PmHelpFunction?

  ,PmHelpPrompt?

  ,PmInferredPrompt?

  )>

<!ELEMENT Name (%name;)>

<!ELEMENT PmType  (#PCDATA)>

<!ELEMENT IGrammar (%grammar;)>

<!ATTLIST IGrammar uri %uri; #IMPLIED>

<!ELEMENT DGrammar (%grammar;)>
```

```
<!ATTLIST DGrammar uri %uri; #IMPLIED>

<!ELEMENT PmPrompt (%Prompt;)>

<!ELEMENT PmRekPrompt (%Prompt;)>

<!ELEMENT PmRekDiscardPrompt (%Prompt;)>

<!ELEMENT PmConfirmPrompt (%Prompt;)>

<!ELEMENT PmHelpPrompt (%Prompt;)>

<!ELEMENT PmInferredPrompt (%Prompt;)>

<!ELEMENT PmHelpFunction (#PCDATA)>
```

Parameter Set DTD Definition

```
<!-- Entities -->

<!ENTITY % ParDef SYSTEM "Parameter.dtd">

%ParDef;

<!ELEMENT ParameterL (Parameter*)>
```

DIANE Grammar Formalism

```
Grammar  ::= Grammar_Element*

Grammar_Element ::= Production | LexiconEntry | Code_Element | Comment_Element

Code_Element ::= “{:“ <Java Code> “:}“

Comment_Element ::= “/*“ <Comment>  “*/“

Production ::= <Nonterminal> “=“ Alternatives “;“

Alternatives ::= Right_Hand_Side (“|“ Right_Hand_Side)*

Right_Hand_Side ::= Symbol* (Code_Element)

Symbol ::= (<Terminal> | <Nonterminal>) (“:“ <IndexSymbol>)

LexiconEntry ::= “/“ <Terminal> “=“ <Terminal> “;“
```

<Terminal>, <IndexSymbol> = Alphanumeric ASCII string

<Nonterminal> = Alphanumeric ASCII string starting with $

Appendix 2: DTD Definition of Enhanced DIANEXML

The enhanced DIANEXML extends only the transaction part of the whole DIANEXML. The added elements are emphasized.

```
<!ENTITY % parameterDTD SYSTEM "Parameter.dtd">

%parameterDTD;

<!ENTITY % boolean "#PCDATA">

<!ENTITY % expression "CDATA">

<!ENTITY % executable.content

        " Assign|Clear|If|Prompt|BackendExe ">

<!ENTITY %if.attrs "cond %expression; #REQUIRED">

<!-- elements for system prompt -->

<!ELEMENT Prompt (#PCDATA)>

<!-- assign value to a parameter -->

<!ELEMENT Assign EMPTY>

<!ATTLIST Assign

    Name CDATA #REQUIRED

    Expr CDATA #REQUIRED >

<!-- backend operation access -->

<!ELEMENT BackendExe

   (Output?

   )>

<!ATTLIST BackendExe

    expr %expression #REQUIRED >
```

```
<!ELEMENT Output EMPTY >

<!ATTLIST Output

   id CDATA #REQUIRED

   type CDATA #REQUIRED >

<!-- delete parameter value -->

<!ELEMENT Clear EMPTY>

<!ATTLIST Clear

   Namelist CDATA #IMPLIED>

<!-- if elseif else statement -->

<!ELEMENT If (#PCDATA|%executable.content;|elseif|else)*>

<!ATTLIST If %if.attrs;>

<!ELEMENT Elseif EMPTY>

<!ATTLIST Eleseif %if.attrs;>

<!ELEMENT Else EMPTY>


<!ELEMENT Transaction

  (Name

  ,TrPrompt?

  ,TrConfirmBool?

  ,TrConfirmPrompt?

  ,TrGrammar*

  ,TrParameter*

  ,Precondition?

  ,Context?

  ,Constraint?

  ,SystemAction?
```

```
 ,Postcondition?
 ,SemLessGrammar*
 ,SemLessStartGrammar*
 )>
<!ELEMENT Precondition
  (Condition, Message?)>
<!ELEMENT Condition (#PCDATA)>
<!ELEMENT Message (#PCDATA)>
<!ELEMENT Context (#PCDATA)>
<!ELEMENT Constraint
 (TriggerParameter*
 ,Condition*
 ,VoilatAction)>
<!ELEMENT TriggerParameter (#PCDATA)>
<!ELEMENT VoilateAction (%Executable.content;)>
<!ELEMENT SystemAction
  (TriggerParameter*
  ,Condition*
  ,Action)>
<!ELEMENT Action (%Executable.content;)>
<!ELEMENT Postcondition (%Executable.context;)>
<!ATTLIST Transaction uri %uri; #IMPLIED>
<!ELEMENT ConfirmFunc (%Prompt;)>
<!ELEMENT TrPrompt (%Prompt;)>
<!ELEMENT TrConfirmPrompt (%Prompt;)>
<!ELEMENT TrConfirmBool (%boolean;)>
```

```
<!ELEMENT TrGrammar (%grammar;)>

<!ATTLIST TrGrammar uri %uri; #IMPLIED>

<!ELEMENT SemLessGrammar (%grammar;)>

<!ATTLIST SemessGrammar uri %uri; #IMPLIED>

<!ELEMENT SemLessStartGrammar (%grammar;)>

<!ATTLIST SemLessStartGrammar uri %uri; #IMPLIED>

<!ELEMENT TrParameter

 (Name

 ,Value?

 ,RecBool?

 ,OptBool?

 ,DefaultVal?

 ,InfBool?

 ,AlwaysConfBool?

 ,UserConfBool?

 ,ConfIfInfBool?

 ,ConfIfDefBool?

 ,InterruptBool?

 ,TestAllBool?

 ,OutOfTaskBool?

 ,CopyPermittedBool?

  )>


<!ELEMENT RecBool (%boolean;)>

<!ELEMENT OptBool (%boolean;)>

<!ELEMENT InfBool (%boolean;)>
```

<!ELEMENT AlwaysConfBool (%boolean;)>

<!ELEMENT UserConfBool (%boolean;)>

<!ELEMENT ConfIfInfBool (%boolean;)>

<!ELEMENT ConfIfDefBool (%boolean;)>

<!ELEMENT InterruptBool (%boolean;)>

<!ELEMENT TestAllBool (%boolean;)>

<!ELEMENT OutOfTaskBool (%boolean;)>

<!ELEMENT CopyPermittedBool (%boolean;)>

<!ELEMENT DefaultVal (#PCDATA)>

<!ELEMENT Value (%expression|If)>

Appendix 3: BNF Definition of Script Language for DIANEXML

Expression	::=	ConditionalExpression
	|	ConditionalSimpleExpression
	|	ValueExpression
BackendExpression	::=	"%" <CLASS_NAME> "." <METHOD_NAME> "(" Argument ")"
Argument	::=	ValueExpression ("," ValueExpression)*
ConditionalExpression	::=	ConditionalSimpleExpression "&&" ConditionalSimpleExpression
	|	ConditionalSimpleExpression "||" ConditionalSimpleExpression
	|	"!" ConditionalSimpleExpression
ConditionalSimpleExpression	::=	RelationalExpression
	|	ConditionalExpression
	|	BooleanLiteral
RelationalExpression	::=	ValueExpression "==" ValueExpression
	|	ValueExpression "!=" ValueExpression
	|	ValueExpression "<" ValueExpression
	|	ValueExpression ">" ValueExpression
	|	ValueExpression "<=" ValueExpression
	|	ValueExpression ">=" ValueExpression
ValueExpression	::=	Literal
	|	BackendExpression
	|	Parameter

	| Variable
Parameter	::= "$" <PARAMETER>
Variable	::= "?" <VARIABLE>
Literal	::= StringLiteral
	| FloatLiteral
	| IntegerLiteral
	| CharLiteral
	| StringLiteral
	| BooleanLiteral
	| "null"
BooleanLiteral	::= "true"
	| "false"
StringLiteral	::= <STRING_LITERTAL>
FloatLiteral	::= <FLOAT_LITERTAL>
IntegerLiteral	::= <INTEGER_LITERAL>
CharLiteral	::= <CHARCTER_LITERTAL>

Appendix 4: Examples of Deeply Nested Sentences

a) I called the man who put the book that you told me about down up.

b) The man who the boy who the students recognized pointed out is a friend of mine.

c) The man the boy the students recognized pointed out is a friend of mine.

d) The rat the cat the dog chased bit ate the cheese.

Curriculum Vitae

Dongyi Song was born on November 22, 1979 in Sichuan, China. She attended primary school there from 1986 to 1992, and high-school from 1992 to1998. In February 1998 she came to Germany.

She entered the Technological University of Dresden in October 1998, studying Computer Science with a minor in business studies. During her studies, she gained a variety of practical experience from industrial internships – she was twice in a software company in Sindelfingen working there as an internal software developer, she served as a student assistant for programming at the university, and she also worked for Siemens in Munich doing website development. In June 2003 she came to Munich to write her Diploma thesis at Siemens. The thesis was titled "Sprachsteuerung für die Verwaltung von Kommunikationsprofilen" ("Speech User Interface for the Management of Communication Profiles"). Supervised by Prof. Dr. Heinrich Hußmann from the University of Munich and Mr. Jürgen Trotzke from Siemens, she finished her Diploma thesis in December 2003. In January 2004 she received her diploma.

In January 2004, Dongyi started work on her thesis "Combining Speech User Interfaces of Different Applications", supervised by Prof. Dr. Heinrich Hußmann from the University of Munich, and financially and administratively supported by Siemens AG in Munich. Her research interests include modeling and developing speech user interfaces, dialog models, dialog systems and all software engineering techniques and technologies in general.