From Single Word to Natural Dialogue

Bernsen, N. O. and Dybkjær, L.

Natural Interactive Systems Laboratory
University of Southern Denmark - Odense
Denmark

Abstract

Spoken language dialogue systems represent the peak of achievement in speech technologies in the 20th century and appear set to form the basis for the increasingly natural interactive systems to follow in the coming decades. This chapter first presents a model of the task-oriented spoken dialogue system, its multiple aspects and some of the remaining research challenges. In the context of this model, a first general model is presented of the complex tasks performed by dialogue managers in state-of-the-art spoken language dialogue systems. The dialogue management model is aimed to support best practice in spoken language dialogue systems development and evaluation.

Contents

Abstract 1
Contents 1
1. Introduction 2
2. Task-oriented Spoken Language Dialogue Systems 4
    2.1 Introduction 4
    2.2 Dialogue context 6
    2.3 Dialogue control 6
    2.4 Language 7
    2.5 Speech 7
    2.6 Performance and human factors 8
    2.7 Systems integration 8
    2.8 Conclusion 9
3. Managing the Dialogue 10
    3.1 Introduction 10
  Goal of the Dialogue Manager 11
    3.2 Efficient task performance 11
  System Varieties 12
    3.3 Multimodal systems including speech 12
    3.4 Multilingual systems 12
    3.5 Multi-task, multi-user systems 13
  Input Speech and Language 13
    3.6 Are the speech and language layers OK? 13
    3.7 Do the speech and language layers need support from the dialogue manager? 13
    3.8 Real-time requirements 15
  Getting the User's Meaning 15

3.9 Task complexity 15
3.10 Controlling user input 19
3.11 Who should have the initiative? 22
3.12 Input prediction/prior focus 24
3.13 Sub-task identification 24
3.14 Advanced linguistic processing 27
Communication 28
3.15 Domain communication 28
3.16 Meta-communication 29
3.17 Other forms of communication 33
3.18 Expression of meaning 33
3.19 Error loops and graceful degradation 34
3.20 Feedback 35
3.21 Closing the dialogue 37
History, Users, Implementation 38
3.22 Histories 38
3.23 Novice and expert users, user groups 39
3.24 Other relevant user properties 40
3.25 Implementation issues 41
4. Conclusion 43
Acknowledgements 44
5. References 44

1. Introduction

It is commonplace to say, and think, that everything to do with computers is moving very fast. Yet it is also a fact that the general field of computing is renowned for exaggerated claims and expectations about the swiftness of future progress. In the world of speech systems we have seen both. Thus, during the 1970s and 1980s repeated claims were made that near-perfect speech recognition had been achieved or was just around the corner. In 1960, promising recognition rates were reported for very small vocabulary (10 words), speaker-dependent, real-time recognition of isolated words [1]. Today, 40 years later, academic research in speech recognition is about to reach the end of the road, being replaced by steady progress through competitive industrial development [2]. Medium-sized vocabulary (+5.000 words), speaker-independent, real-time recognition of continuous (or spontaneous) speech has become commercial reality, and very large vocabulary (+60.000 words) spoken dictation systems which only need a minimum of speaker-dependent training can be purchased for about 100 Euros from companies such as IBM, Dragon Systems and Philips. Swift or not, this impressive progress has shifted the perspective in speech technology research dramatically. Today, robust, unlimited vocabulary, real-time speaker-independent continuous speech recognition is within reach and speech recognition technology has become a component technology which is finding its way into all sorts of interfaces to computer systems.

However, the coming-to-maturity of speech recognition is not the whole story of speech technology during the past four decades. By itself, speech recognition is a transformation of the acoustic signal into an uninterpreted string of words which may or may not make sense to a human but does not make any sense to the machine. This enables applications such as the coveted ‘phonetic typewriter’ [1] as well as spoken command applications in which the system executes in response to a spoken word or phrase rather than in response to the push of a button in the keyboard, mouse or otherwise. Among humans, speech is much more than that, of course. Speech is the primary modality for interactive exchange of information among people. Whilst hardly visible - even as a long-term goal - in 1960, the past 10 to 15 years have seen the emergence of a powerful form of interactive speech systems, i.e. task-oriented spoken language dialogue systems [3]. These systems not only recognise speech but understand speech, process what they have understood, and return spoken output to the user who may then continue the spoken interaction with the machine until the task has been accomplished. In their most versatile form, spoken language dialogue systems, or SLDSs, for short, incorporate speaker-independent, spontaneous speech recognition in real-time.

It is the task orientation which has made SLDSs possible at the present time. It is still too early to build fully conversational SLDSs which can undertake spoken interaction with humans the same way humans communicate with one another using speech-only - about virtually any topic, in free order, through free negotiation of initiative, using unrestricted vocabulary speech, and so on. However, a range of collaborative tasks are already being solved through speech-only dialogue with computers over the telephone or otherwise. One of the simplest possible examples is a system which asks if the user wants to receive a collect call. If the user accepts the call, the system connects the caller, and if the user refuses the call, the system informs the caller that the call was rejected [4]. A more complex task for which commercial solutions already exist, is train time-table information [5, 6, 7]. The user phones up the system to inquire, for instance, when there are trains from Paris Gare du Nord to Brussels on Thursday morning, and receives a spoken list of departures in return. As these examples show, task-oriented SLDSs constitute a powerful application paradigm for interactive speech technologies, which could be used for a virtually unlimited number of interactive user-system tasks. However, successful SLDSs remain difficult to build for reasons which often go beyond the purely theoretical and technical issues involved and which illustrate the general state of speech technology research at this point.

Even if capable of working as systems in their own right, speech recognisers are increasingly becoming system components. The same is true of speech generators which perform stand-alone tasks as text-to-speech systems but which are increasingly becoming system components as well. The SLDS is probably the most important technology which integrates speech-to-text, text-to-speech and various other components, such as natural language understanding and generation, but it is not the only one. Other integrated systems technologies incorporating some form of speech processing include speech translation systems, and multi-modal systems having speech as one of their input/output modalities, other modalities being, for instance, mouse pointing input gesture or output graphics such as information tables or a talking face. All of these integrated technologies represent a level of complexity which is comparatively new to the field of speech technology research. Together with the rapid increase in commercial exploitation of speech technology in general, those technologies have introduced an urgent need for system integration skills, human factors skills, general software engineering skills, and skills in creative contents creation to be added to the skills of groups which used to work on the basic component technologies. The field, in other words, is now faced with having to specialise software engineering best practice to speech technologies and to do so swiftly and efficiently. This re-orientation process has only just started. This is why the development of, for instance, task-oriented SLDSs remains fraught with home-grown solutions, lack of best practice methodologies and tools, ignorance about systems evaluation, lack of development platforms and standards etc. Only by solving problems such as these will it be possible to efficiently design and build task-oriented SLDSs which will achieve their ultimate purpose: to conduct smooth and effortless natural dialogue with their users during interactive task resolution.

Arguably, the methodical achievement of natural dialogue in task-oriented SLDSs is one of the most important challenges for speech technologies research at this point. It may be illuminating to view this challenge as a pointer into the future. Humans not only use speech for collaborative and interactive task resolution, they engage in spoken conversation about everything. Humans interact through speech in different languages. And, when interacting face-to-face through speech, humans communicate in many other ways in parallel: through lip movement, facial expression, gesture, and bodily posture, and the communication often makes use of objects which are present on-site and which themselves may have communicative contents, such as texts, maps, images etc. Task-oriented SLDSs, therefore, point all the way to systems which communicate with humans the same way in which humans communicate with one another. Task-oriented SLDSs conducting natural dialogue merely represent the first step towards the ultimate goal of speech technology research, which is that of integrating speech into fully natural interactive systems.

In the context just outlined, this article discusses progress towards the development of natural dialogue in task-oriented SLDSs. This is a large topic which comprises, at least, "last frontiers" in speech recognition, current challenges in speech generation, issues in natural language understanding and generation, dialogue management, human factors in SLDSs, system integration issues, and supporting tools, architectures and platforms. Rather than attempting to cover all of these aspects of SLDSs, we have chosen to focus on recent progress in the understanding of dialogue management. The dialogue manager is, in most cases, the core of the SLDS, which orchestrates everything the system does and refers to most events which occur in an SLDS. Moreover, dialogue management continues to pose challenges in research and development on which the following will attempt to shed some light. Before going into detail with dialogue management in Section 3, Section 2 outlines the system context in which a dialogue manager must operate. Section 4 concludes the chapter.

2. Task-oriented Spoken Language Dialogue Systems

2.1 Introduction

Speech recognition without speech understanding constitutes a broad and important, yet still fairly limited application area. As speech recognition improved sufficiently to enable recognition beyond the single word, it became interesting to post-process the recognised input string through adaptation of existing syntactic and semantic parsing techniques. The resulting understanding of spoken input pointed the way towards systems which could use their understanding of the input to help users carry out particular tasks through dialogue with the machine. However, the achievement of such systems required exploration of the challenges posed by sophisticated dialogue management. As long as only single-word utterances were recognised and no natural language understanding processing was involved, dialogue managers only had to match the recognised string against a set of possibilities and take the action associated with the best match. This process from speech recognition-only through inclusion of natural language understanding and dialogue management is reflected in the topics addressed in the DARPA Proceedings between 1989 and 1992 [8, 9, 10, 11]. On the output side, most SLDSs still use coded speech, i.e. pre-recorded words and phrases that are replayed to the user. For many languages, parametric speech (or fully synthetic speech) is still of insufficient quality for use in walk-up-and-use applications. More recently, the emerging field of dialogue engineering has come to include human factors issues as well as issues in systems integration.

Figure 1 shows a model of the elements which are, or could be, relevant to the design and construction of an SLDS. No existing system incorporates all those elements but all systems incorporate some of them. The elements may be viewed as abstract building blocks from which to build particular SLDSs.

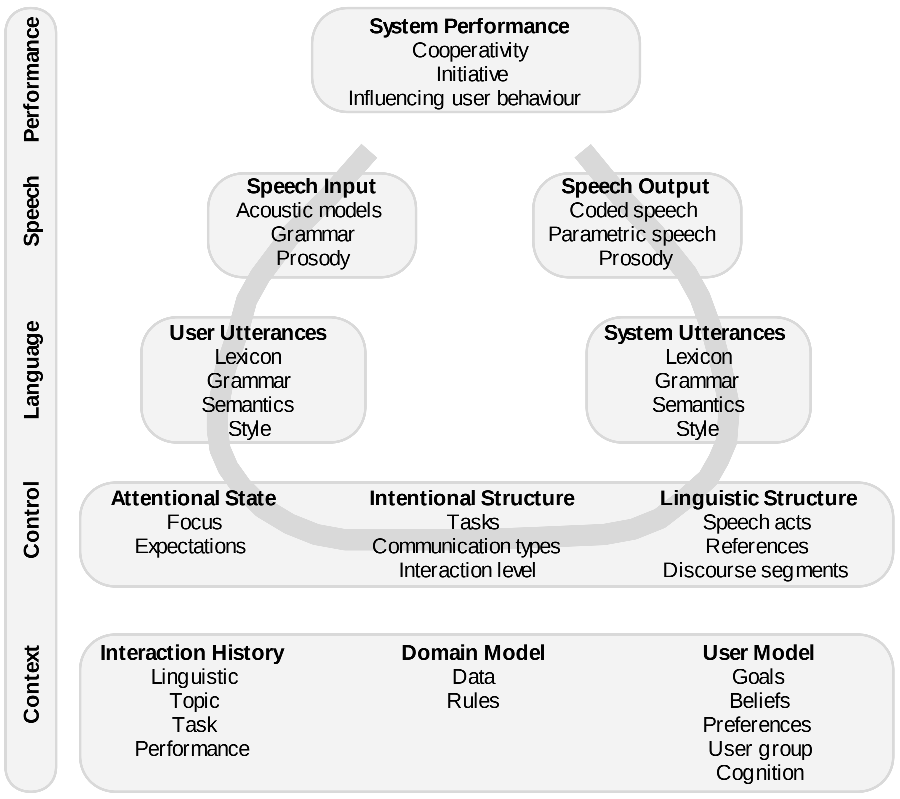

Figure 1. Elements for building SLDSs. Grey boxes reflect the logical architecture of SLDSs. The grey band shows overall information flow among the layers.

The elements are organised into five layers. At the bottom of the figure, the dialogue *context layer* includes aspects of the history of interaction, the domain model and the user model. At the level above the context layer, the interaction *control layer* includes (system) state of attention as well as the structures defined by the interlocutors' intentions and structural aspects of the linguistic exchanges. System control is largely based on structures at this level. The *language layer* describes the linguistic aspects of the interaction**.** Then follows the *acoustic layer* which includes the transformations between the speech signal and the symbolic expressions of language. Finally, the *performance layer* is a function of the other layers taken together and includes some general aspects of the system's behaviour.

The grey band in Figure 1 indicates the overall processing flow among elements – from input through control to output and performance - in a context defined by contextual elements. Developers often refer to elements in terms of the system components which implement these, such as the speech recogniser, the lexicon, the parser, the dialogue manager, the database, the language generator, the speech generator etc., system performance being replaced by an abstraction of the (physical) user and reflected in the human factors issues. Just as no existing SLDS includes all the elements of Figure 1, many SLDSs do not include all the components just mentioned. Very simple SLDSs, for instance, only include a recogniser with a simple parser, a basic dialogue manager and a speech generator.

We will now briefly present the five layers of Figure 1 to convey a sense of the context for which dialogue managers must be developed and in which they have to work. Section 3 will revisit this context from the particular point of view of the dialogue manager.

2.2 Dialogue context

Dialogue context is of crucial importance to language understanding and generation as well as to dialogue control and plays a central role in SLDS development. The dialogue context provides constraints on lexical selection, speech act interpretation, reference resolution, task execution and communication planning, system focus and expectations, the reasoning that the system must be able to perform and the utterances it should generate. Contextual constraints serve to remove ambiguity, facilitate search and inference, adapt to the user, and increase the information contents of utterances since the more context, the shorter the messages need to be [12]. Context specification is closely related to the particular task(s), application and users of the system. In a sense, each element of Figure 1 is part of the context of each other element.

The *interaction history* is primarily relevant to the local discourse and is built and used during interaction with the user. An interaction history is a selective record of information which has been exchanged during interaction. It is useful to distinguish between at least four types of interaction history as shown in Figure 1.

The *domain* of an SLDS is the aspect of the world about which the system can communicate. An SLDS often acts as front-end to some application, such as an email system or a database. The domain model captures the concepts relevant to the application in terms of data and rules. For instance, there may be rules for expanding relative date indications, such as "today", into absolute dates. The domain data may be used partly for providing information requested by the user and partly for checking user input. For example, the system may check if a flight route indicated by the user actually exists.

User modelling is particularly important in SLDS development. The better the system can take aspects such as user goals, beliefs, skills, preferences and cognition into account, the more co-operative the system can be [13]. The general fragility of current SLDSs means that they must be carefully crafted to fit the behaviour of their users. This remains a hard problem.

2.3 Dialogue control

Despite the very real problems in designing grammars and parsing techniques for spoken input (see 2.4), it is probably in dialogue processing that current development of advanced SLDSs is furthest removed from theoretical and practical mastery in terms of best practice development and evaluation procedures, methods, tools, standards, and supporting concepts and theory. The main reason appears to be the comparative novelty and unexpected complexity of the problems encountered. The management by machine of spoken dialogue can only be investigated in running SLDSs or realistic simulations. Such investigations have only been possible during the last decade or so whereas research on speech recognition and generation, and on (written) linguistic input analysis and (written) language generation has a much longer history. Many individual aspects of *human-human* conversation have been investigated for decades but theoretical results have proven difficult to transfer to spoken human-machine interaction [3]. This is because the machine is a highly inferior partner in dialogue compared to human interlocutors.

Controlling the dialogue is a core function in SLDSs. Dialogue control determines what to expect from the user, how to interpret high-level input structures, which context elements to consult, what to output to the user, and generally when and how to do what. The nature of these control tasks implies that control has to operate on superordinate interaction structures and states. Following [14], the dialogue control layer distinguishes three types of superordinate interaction structure and state. The attentional state includes the entities in current interaction focus. The intentional structure addresses the multiple purposes of interaction, and the linguistic structure includes characterisation of high-level structures in the input and output discourse.

2.4 Language

The linguistic processing done by today's advanced SLDSs consists in (i) linguistic analysis of the input produced by the speech recogniser and (ii) generation of linguistic output to the speech generator from an underlying semantic representation. Natural language generation *per se* is often absent from current systems because the underlying output semantics, once chosen by the system, is directly linked to pre-designed system output phrases which simply have to be played to the user or passed through the speech synthesiser. As more advanced natural language generation becomes necessary, there does not seem to be any reason why SLDS designers could not draw upon existing knowledge about language generation in the natural language processing community. There are two major issues to take into account. It is important that the system's messages are clear and precise in the dialogue context. Moreover, as humans tend to model the system's output phrases, the system must be able to recognise its own output vocabulary and grammar.

The problems are more serious on the linguistic input side. Many SLDSs need some form of linguistic input analysis. However, spoken language behaves very differently from written language (text) and its behaviour remains poorly understood. This means that there is no easy way of transferring linguistic progress in written language understanding to the understanding of spoken input. Thus, there is still little consensus with respect to how to optimise grammars and parsing for SLDSs. Full written language parsing techniques do not work. What actually works, more or less, is commonly called 'robust parsing' but this term does not have any clear meaning at present apart from referring to less-than-full written language parsing. Other issues include: whether to use stand-alone grammar and lexicon(s) or build these into the speech recogniser; how to achieve spoken sub-language adequacy (lexicon and grammar) for language understanding and generation; whether to use morphology (declarative and principled, but slow processing) or a full-form lexicon (fast); how to integrate syntax and semantics; how to efficiently separate resources from the procedures which use them (modularity); how to add linguistic knowledge (grammar and vocabulary) to the system during or after development (extensibility); and how to build one shared grammar for analysis and generation (modularity) [15].

2.5 Speech

The speech layer concerns the relationship between the acoustic speech signal and a, possibly enriched, text (lexical string). This relationship is not a simple one. Speech includes a number of prosodic phenomena—such as stress, glottal stops, and intonation—which are only reflected in text in a simplistic manner. Conversely, words and their different spellings as we know them from text, do not have natural expressions in speech.

Speech recognition must cater for extra-linguistic noise and other phenomena, such as that the speech rate varies over time, the speech signal is mixed with environmental noise from other people speaking and may be of differing quality because of traffic, slamming doors etc., the pronunciation varies with the speaker, and speech from different participants may overlap, for instance with the system's utterances [16, 17].

Speech recognisers and speech generators still need improvement in many respects, including basic recognition rate in adverse conditions; coping with spoken language specificities, such as hesitations, repetitions of words or syllables, ill-formed phrases, incomplete sentences etc.; rejecting non-authorised words or interpreting them using the context of the sentence or dialogue; and dynamically adapting to the user's personal way of speaking (linguistic behaviour, own stereotypes etc.); voice output quality; and ability to handle input prosody and output prosody in concatenated pre-recorded speech or speech synthesis.

Meanwhile, the errors and misunderstandings that occur between user and system because of less-than-ideal speech recognition and generation can often be satisfactorily handled through spoken interaction. In fact, users can now speak to their computer systems in basically the same way as they speak to fellow humans, that is, by using continuous speaker independent speech, and they can understand the machine's spoken response without significant difficulty, especially when pre-recorded speech is being used. This means that, despite several remaining challenges, the speech layer is in place for practical use.

2.6 Performance and human factors

Any advanced SLDS incorporates many of the elements presented in Figure 1. Together, these elements determine the observable behaviour or performance of the system during interaction. The observable behaviour strongly influences the user's impression of, and satisfaction with, the system. Those element properties which contribute to observable system behaviour form part of the human factors aspect which is a fairly new discipline in SLDS design. Human factors cover all aspects of interactive system design which are related to the end-user's abilities, experience, goals, and organisational/cultural context [18]. Whilst the remit of this field is broad, theoretical and practical work tends to occupy a variety of small niches with few unifying approaches defining interactive system design on all of the dimensions noted.

However, along with the accelerating industrial exploitation of SLDSs it is becoming increasingly important to invest in human factors issues early in the development process in order to investigate what is needed to achieve a system which will satisfy the users' needs and expectations [19]. It is costly to partly redesign an implemented system which failed to take into account basic human factors issues and therefore was discarded by its users.

Human factors cut across the components of an SLDS. This means that human factors issues must be considered during development of any SLDS component as well as for the integrated system to ensure that the final SLDS will be adequate and satisfy user expectations to the extent possible. Human factors issues include, i.a., system cooperativity, dialogue initiative, dialogue structure, detection and correction of errors, user expectations and user background, user modelling, and how to address users and influence user behaviour. Depending on the task(s) solved, natural dialogue may sometimes also require the possibility for the user to draw upon additional modalities, such as haptic input or output graphics.

2.7 Systems integration

SLDS components must be integrated in two ways: (i) the individual components must be brought to work together, and (ii) the integrated components must somehow be connected to the world to allow users access the system. The inner circle of Figure 2 reflects (i) whereas the outer circle reflects (ii).

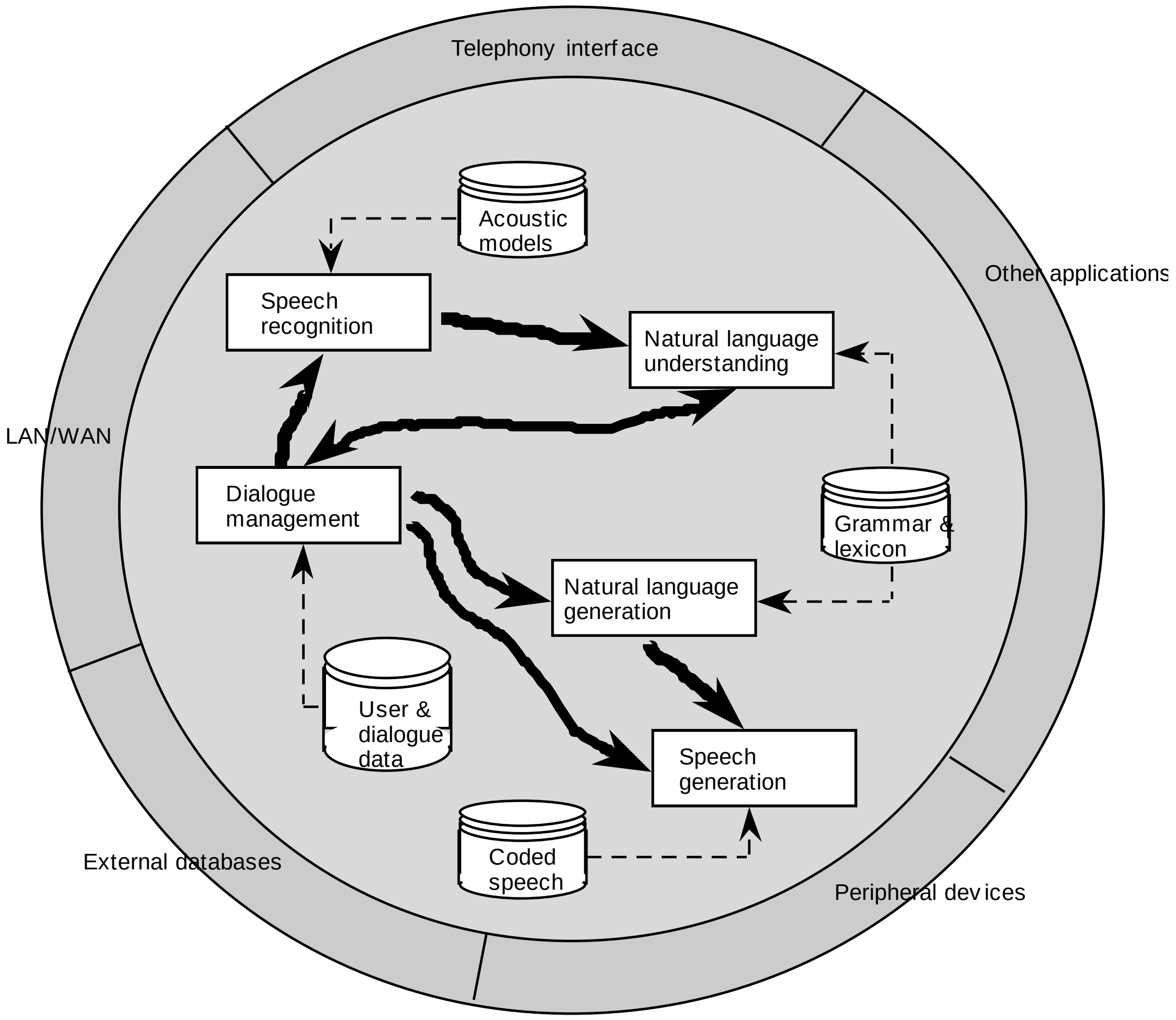

Figure 2. Functional and environmental systems integration of SLDSs.

The key components of SLDSs have been mentioned above. Individual components may integrate several smaller components. Any component—small or large—has to communicate with at least one other component. Various platforms are available which can help developers enable the communication in a relatively standardised way through Application Programming Interfaces (APIs).

The way in which the user accesses an SLDS differs from system to system. Some SLDS are only accessible over the telephone. Others are internet-based [20]. Some systems run locally on the user's machine and are accessed via microphone. There are systems of which a major part runs on a server and only a minor part runs on the user's machine. Some SLDS—client-based or client-server based—allow multimodal interaction. SLDSs may need access to external databases or collaborate with other applications. Thus, the environmental integration needed differs widely from one application to another and may be fairly complex.

2.8 Conclusion

It has emerged from the brief introduction to SLDSs above that all layers of Figure 1 as well as their integration into working SLDS pose a variety of unsolved problems. Arguably, the major challenges, including some thorny human factors issues, are to be found in the area of dialogue management. The next section will explore the complexity of dialogue management in detail.

3. Managing the Dialogue

3.1 Introduction

Dialogue management theory is still relatively uncharted territory. Current views on dialogue management tend to be heavily biased towards particular SLDSs, particular interactive tasks, and particular dialogue management solutions for those tasks. The reason is that few dialogue manager developers have extensive hands-on experience from developing different dialogue managers for a wider range of SLDSs and interactive tasks. Hands-on experience remains a major source of knowledge about dialogue management, given the fact that the generalisations that are required in the field cannot yet be found in the literature. If considered as general truths, those views may induce sub-optimal solutions to the development of SLDSs and dialogue managers which, by their nature, require a different approach. The presentation of issues in dialogue management below is based on analyses of a series of significantly different SLDSs from research and industry. The authors hope that this underlying diversity has contributed some amount of generality to the observations made. Generality is a precondition for achieving a theory of dialogue management which can support the development of dialogue managers for SLDSs.

Dialogue management is arguably the core functionality of spoken language dialogue systems (SLDSs). Figure 1 helps explain why this is the case. The figure shows the logical architecture of SLDSs as organised in a series of layers called performance, speech, language, control and context, respectively. A user interacting with an SLDS produces speech input (speech layer) and receives speech output (speech layer) from the system. In addition, the regular user might figure out more abstract properties of the system's behaviour, such as how cooperative it is during dialogue, the distribution of user and system dialogue initiative, or how the system attempts to influence the user's dialogue behaviour through its choice of words and in other ways (performance layer). The rest of the system's workings are hidden from inspection by the user. Figure 1 classifies these workings in terms of a series of headers, such as 'user utterances' or 'domain model', and elements subsumed by each header. The elements are high-level references to the functionality that may be present in today's SLDSs. Most of today's SLDSs do not have all of the functionalities referred to in Figure 1, but all SLDSs have some of the functionalities.

Dialogue management is primarily located in the control and context layers in Figure 1. When the user inputs an utterance to the system, the dialogue manager may receive a representation of the meaning of what the user said from the speech and language layers. It is the task of the dialogue manager to handle that representation properly, eventually producing an appropriate output utterance to the user. Depending on the complexity of the task which the system helps the user solve, the dialogue manager may have to make use of most or all of the elements listed in the control and context layers in Figure 1. For instance, to determine if the user has provided a new piece of information which the system needs in order to complete the task, the dialogue manager may have to check the history of sub-tasks performed so far (context layer: task history). Or, to decide if shortcuts in the dialogue can be made because this particular user does not need a certain piece of information or advice, the dialogue manager would check if the user belongs to the class of experienced users (context layer: user group).

A dialogue manager should be designed based on careful consideration of its task(s) and users. What actually happens is that the designer (or developer) designs a system based on both *design-time* and *run-time* considerations. Some design decisions only affect design-time work whereas many others affect run-time operation as well. The system's task, for instance, is fixed at design-time and cannot be modified at run-time. On the other hand, what the user actually inputs at some point, cannot be fully predicted at design-time and must be handled at run-time. The distinction between design-time and run-time considerations is inherent to the following account and we trust that the reader will be able to decide when a particular comment applies to design-time only or to run-time as well.

This section takes a systematic look at the many issues which may have to be dealt with in dialogue manager design. Some of the issues to be considered go beyond anything encountered in current industrial design of dialogue managers. In research prototype development, on the other hand, any of those issues may be encountered although no existing research project would seem to address all of them. This may soon change, however, given the rapid development of the field and, when this happens, it is in the nature of research progress that new projects will be addressing not only every single issue to be presented below but additional issues as well.

The following presentation has been sub-divided into 24 sections. The sections are structured under the following headings:

Goal of the Dialogue Manager

System Varieties

Input Speech and Language

Getting the User's Meaning

Communication

History, Users, Implementation

This structure roughly derives from analysis of a dialogue manager's main tasks and their ordering, and thus constitutes a partially ordered model of (possible) dialogue manager functionality. The model is based on in-depth analyses of the following dialogue manager exemplars: the Daimler-Chrysler Dialogue Manager [21], the dialogue manager in the Danish Dialogue System [3], the dialogue manager in Railtel/ARISE [22, 23], the dialogue manager in Verbmobil [24, 25 http://www.dfki.uni-sb.de/verbmobil/], and the dialogue manager in Waxholm [26, 27, http://www.speech.kth.se/waxholm/waxholm.html]. The model expands on the dialogue manager-related parts of Figure 1 and is summarised in Section 3.25 below. Let us assume that the model subsumes all existing and emerging dialogue managers for SLDSs, i.e., each of these dialogue managers represent a sub-set of the functionalities described in the model. Given that assumption, an obvious application of the model is to use it as a checklist during SLDS and dialogue manager design, to help identify the sub-set of functionalities needed in a particular case.

Goal of the Dialogue Manager

3.2 Efficient task performance

The central goal of an SLDS is to enable the interactive task to be done *efficiently* and with a *maximum of usability.* This goal subsumes a number of sub-goals which are common in discussions of SLDSs and their components, such as robustness, flexibility, naturalness etc. As we will not be considering usability aspects in what follows (see [3, 28]), from the point of view of the present paper this goal reduces to efficient task performance. Ideally, efficient task performance means that the system takes the *steps that are necessary and sufficient to always provide appropriate output* at any given stage during the dialogue.

System Varieties

Our common idea of an SLDS is a *unimodal speech input/speech output* system which conducts a dialogue about a *single task* in a *single language* with a *single user* at a time. Such SLDSs are, indeed, the most common ones at the present time. But this should not blind us to the large space of alternative types of SLDSs, one of which might be just what is needed for a particular application.

3.3 Multimodal systems including speech

Appropriate SLDS output to the user is not always *spoken* output. For some tasks, the system's output may be system actions which the user can perceive, such as connecting the user with the person the user asked to speak to. Obviously, SLDSs may produce many other non-speech output actions in response to the user's input. Providing operator fall-back is one such action which we will come back to later.

Other non-speech output actions are due to the fact that the system provides output in modalities other than speech. Some tasks require the system to output information which is most effectively presented in, e.g., textual or pictorial form. Long lists of flight connections, for instance, are much more efficiently conveyed by text than by speech, and it is well-known that the contents of many pictures are virtually impossible to render through verbal means [29]. A bicycle repair guide, for instance, is not much worth to the novice without ample pictorial illustration. In such cases, the SLDS needs to be enhanced through other modalities of (output) information presentation, such as text or images presented on a screen. This is the case in the Waxholm system which also uses an animated graphical speaking face whose lip movements help the user disambiguate the system's synthetic speech and whose eye movements help the user focus on newly added information on the screen. Through its 'user speaks' input button, Waxholm also illustrates the fact that SLDSs do not need to take only speech as input. The user presses the button in order to make the system listen.

For many other tasks as well, an appropriate solution may be to combine spoken input with other input modalities, such as pointing gesture. So, speaking more generally, there is a fundamental question here about when to use speech input and/or speech output for a particular application, when not to use speech, and when to use speech in combination with other particular input/output modalities for information representation and exchange. The solution to this question about the functionalities, or roles, of modalities in particular cases not only depends on the task(s) to be solved by the system but on many other parameters as well, such as the users, the work environment etc. Having investigated the issue of speech functionality for a long time [30], we have developed a web-based tool called SMALTO to support early design decisions on when (not) to use speech for particular applications [31, http://www.elsnet.org/disc].

3.4 Multilingual systems

Appropriate SLDS output to the user is not always spoken output *in the same language* as the spoken input provided by the user. Spoken translation systems, such as Verbmobil, translate spoken input in one language into spoken output in another language. Spoken translation systems are basically different from standard SLDSs in that they do not conduct any dialogue with the user about the application domain: they do not provide the user with train departure times or help the user browse the web. Rather, they mediate dialogue on a particular task within a particular domain between users who speak different languages. At most, spoken translation systems conduct meta-communication dialogues with their users (see 3.16). Still, such systems share many issues of dialogue management with standard SLDSs.

Another obvious possibility is that an SLDS accepts spoken input in different languages for the same task(s) and responds in the language in which it is being addressed.

3.5 Multi-task, multi-user systems

In what follows, we focus on the design of an SLDS for a single task. Already today, systems are being built to handle several more or less independent tasks, such as consulting one's diary and answering email over the phone. Applications such as these serve to increase an already existing need for task and domain independent dialogue managers. Task and domain independent dialogue managers would facilitate the rapid prototyping of SLDSs for new tasks and new domains. We will return to this issue later (Section 3.25).

In what follows, we also focus on single caller (or user) – single system dialogue. An obvious next step is for several callers to conduct dialogue with the system to solve one and the same task. There are several real issues involved in creating smooth multi-user spoken dialogue with machines, such as that of handling simultaneous input from several users. These issues will not be discussed in this article.

Input Speech and Language

So far, in Sections 3.2 through 3.5, we have discussed SLDSs in general, their fundamental goals and varieties. From now on, we want to focus on dialogue management of "standard" task-oriented SLDSs, i.e. unimodal, monolingual, single-task and single-user SLDSs.

3.6 Are the speech and language layers OK?

Suppose that we have identified a certain task, T1, for which we consider building an SLDS. Suppose, in addition, that T1 is already known not to generate problems which cannot be solved by following best practice in developing the speech and language layers of the application (cf. Figure 1). In other words, our speech recogniser can be expected to come to possess the necessary robustness for the application, the vocabulary will not be infeasibly large, we are able to develop the grammar and parser required, language generation and speech generation of sufficient quality can be developed, etc.

3.7 Do the speech and language layers need support from the dialogue manager?

Things in this world often do not come for free. It may be that conditions have to be imposed on the dialogue manager in order to guarantee feasibility in the speech and language layers. The feasibility conditions may derive from, e.g., the need for real-time performance (see 3.8), task complexity (see 3.9), or achieving a sufficient recognition rate in a large-vocabulary system. The dialogue manager may thus have to actively support the processing done in the speech and language layers. It is worth noting, however, that many existing systems use no dialogue manager support for the speech and language layers. In the future, we will see many more forms of dialogue manager support for the speech and language layers than those listed below. We distinguish between dialogue manager support for *input prediction, input language processing control* and *output control.*

3.7.1 Input prediction

When doing input prediction, the dialogue manager uses knowledge about the dialogue to predict aspects of the next user input, thereby reducing the complexity which must be handled by the speech and/or language layers of the system. Input prediction may be possible if, for instance, the task has some structure to it (see 3.9 and 3.12). Thus, if the system knows that it is now going to ask the user a specific question, then the system can start making plausible guesses about, e.g., the vocabulary the user will be using in the following input utterance which will be a reply to the system's question to the user. Speech and/or language layer feasibility conditions could be that the dialogue manager must:

i. constrain the search space of the speech recogniser by constraining the set of words which are likely to occur in the next user utterance (sub-vocabulary prediction);
ii. constrain the search space of the keyword or phrase spotting component by delimiting its search space to the most probable keywords or phrases (keyword prediction);
iii. constrain the search space of the syntactic analysis component by narrowing the set of applicable grammar rules to a specific and probable subgrammar (subgrammar prediction);
iv. constrain the search space of the parser by narrowing the set of semantically meaningful units it should be working with (semantic prediction).

In other words, unless the dialogue manager performs successful input prediction based on some form of knowledge about the dialogue, the speech and/or language input processing components will, or may, perform too poorly for the application to be feasible (unless, of course, the problem can be circumvented in some other way).

3.7.2 Input language processing control

Input language processing control will become increasingly important as users' input utterances grow in length as well as in lexical, grammatical and linguistic context-determined complexity. Speech and/or language layer feasibility conditions could be that the dialogue manager must:

v. constrain the search space of the semantic analysis component by, e.g., delimiting the set of possible antecedents in cross-sentence anaphora resolution.

3.7.3 Output control

When doing output control, the dialogue manager follows principles or guidelines that may serve to reduce the complexity of certain aspects of system output, maximise output naturalness etc. Output control is an important means of controlling the user's dialogue behaviour (see 3.10). Speech and/or language layer feasibility conditions could be that the dialogue manager must:

vi. control the prosody of the spoken output based on control layer information about the message to be produced, such as speech act information;
vii. control the lexical variation of dialogue expressions to be produced by the language generation component;
viii. control the grammar of the dialogue utterances to be produced by the language generation component;
ix. control the style of the dialogue utterances to be produced by the language generation component.

3.8 Real-time requirements

It can be assumed that most or all SLDSs need to work in real-time or close-to-real-time, for such is the nature of spoken dialogue. In some cases of spoken human-human dialogue, we actually do tolerate waiting for a response from our dialogue partner, especially when we know that the partner has to search for information or do complex calculations or inferences before responding. Users may be expected to tolerate the same from machines but, generally speaking, close-to-real-time operation is a very important goal in SLDS design.

Getting the User's Meaning

3.9 Task complexity

Ideally, we would like to develop SLDSs which simply let the users speak their minds and then do whatever is necessary to complete the task. For some tasks, this is clearly feasible. Suppose, for instance, that our SLDS has to ask if the user wants to accept a collect call and, depending on the user's answer, simply routes or does not route the call to the user. In this case, the risk of letting the users speak their minds probably is as small as it ever gets. However, suppose that the task is about ordering VIP dinner arrangements at a large restaurant including date, time, duration, choice of rooms, many-course meals, special diets, wines, flowers, timing for speeches, entertainment, seating arrangements, payment arrangements etc. It is easy to imagine that some users will have quite a bit to say about that. They may have so much to say, in fact, that even the dialogue manager support to the speech and language layers described in 3.7 above will not be sufficient to guarantee that the system will always be able to provide appropriate output. Task complexity, therefore, probably is the single most important factor to consider for the dialogue manager developer.

Unfortunately, at this point, task complexity cannot be measured in any objective way. The reason is that task complexity is a function of several factors, including:

- the number of *pieces of information* which have to be communicated between the user and the system, such as departure airport, arrival airport, departure time, arrival time etc.;
- the number of *optional pieces* of information which could be, but do not have to be, communicated between the user and the system, such as whether or not the caller wants to book a return flight;
- whether or not the task itself has some *a priori structure* in terms of a full or partial ordering of its component sub-tasks. For instance, train schedules are often different on workdays and Sundays, which means that the system must know the date before it can offer particular departure times;
- whether or not the sub-tasks constituting the task are *independent.* For instance, a user may want to trade departure time for a cheap fare. In that case, the sub-task of fixing the departure time is not independent of the sub-task of determining if the caller has an interest in discount;
- whether the *flow of information* is one-way or two-way. One-way information flow is the simpler, such that, for instance, the system asks all the questions and the user provides all the answers;
- whether or not *negotiation* is involved;
- whether or not substantial *economic commitments* are being made during dialogue, such as the purchase of a flight ticket or the ordering of a bank transfer;

- whether or not *real-time updates* are important, such as information about delayed flights or impending strikes;
- etc.

In a maximum-complexity scenario, the task involves exchange of many necessary pieces of information, several optional loops in the dialogue structure, little or no a priori structure, several sub-task inter-dependencies, two-way flow of information, negotiation, security measures for safe-guarding the transactions made, and frequent real-time updates.

Based on task complexity factors such as the above, the following guidelines may be useful.

3.9.1 Volume of information

How many pieces of information must be exchanged between the user and the system to complete the task? In many cases, one or two pieces of information will suffice. The collect call SLDS above is a case in point: this system just needs one piece of information to know whether or not to transfer the call. The Operetta telephone directory assistance system is another example: it needs a person's name [32]. In other cases, exchange of something in the order of 4-6 pieces of information suffice to complete the task, as in the RailTel/ARISE train time-table information systems. In these systems, the user has to provide the system with just enough information about departure station, arrival station, date, and possibly time of day, that the system can compute the missing information to be offered to the user. It is feasible today with such systems to let the users speak their minds, have the system realise that one or two data points may be missing from a user's input, ask for these, and then compute the answer to the user's query. The Danish Dialogue System for flight ticket booking, on the other hand, requires so much information from the user that letting the users speak their minds may be counter-productive. The likelihood that the system fails to recognise and understand what the users say becomes too great. The same is true of the dinner arrangement system mentioned above. In such cases of relatively large task complexity (in terms of information volume), it may be necessary to adopt strategies for controlling the user's *input* so that it may stand a good chance of being recognised and understood by the system. Such strategies will be reviewed below (3.10). A task similar to that solved by the Danish Dialogue System has now been included in the US DARPA Communicator project [http://fofoca.mitre.org/]. It will be interesting to see if the project will follow input control strategies such as those described in Section 3.10.

Information optionality is another factor which contributes to increased task complexity in terms of information volume. Callers who ask about train time-tables get just that. But if the system can also book the train tickets, it must have the capability to exchange additional pieces of information, some of which are obligatory whereas others are optional: how many will travel; their age categories; will they need return tickets; will they all need return tickets or only some of them; will they need round-trip tickets; do they need information on the notions of green and red departures; etc. As, in such cases, letting users speak their minds may not be feasible, the challenge for the dialogue designer becomes that of eliciting all relevant pieces of information in as elegant a way as possible.

3.9.2 Ill-structured tasks

What is the task structure, if any? Some tasks are ill-structured or have no structure at all. Consider, for instance, an SLDS whose database contains all existing information on flight travel conditions and regulations for a certain airline company. This information tends to be quite comprehensive both because of the many practicalities involved in handling differently composed groups of travellers and their luggage, but also because of the legal ramifications of travelling which may surface if, e.g., the flight crashes. Some users may want many different individual pieces of information from the database whereas others may want only one piece. Which piece(s) of information a particular user wants is completely unpredictable. We call such user tasks *ill-structured tasks:* the user may want one, two or several pieces of information from the database, and the order in which the user may want the information is completely arbitrary as seen from the system's point of view. The system must be prepared for everything from the user all the time.

One way to try to reduce the complexity or large ill-structured tasks is to use, or invent, *domain structure.* That is, the domain may be decomposable into a number of sectors which themselves may be hierarchically decomposed, etc. So the system asks, for instance: "Do you want to know about travelling with infants, travelling with pets, travelling for the elderly and the handicapped, hand luggage, or luggage for storage?" And if the user says "Hand luggage", the system asks: "Do you want to know about volume of permitted luggage, electronic equipment, fragile items, or prohibited luggage?" Etc. In principle, any information hierarchy, however deep and/or broad, can be handled by SLDSs in this way. However, few users will accept to navigate through many hierarchical levels prompted by the system in order to find a single piece of information at the bottom of some deep domain hierarchy. Making the hierarchy more shallow will often make matters even worse. No user will accept to have to go through a shallow but broad hierarchy prompted by the system in order to find a single piece of information at the end of it. Just imagine a Danish train time table inquiry system which asks the user: "Do you want to go from Aabenrå, Aalborg, Aarhus ...", mentioning the country's 650 or so train stations in alphabetical order and in chunks of, say, ten at a time.

Another important problem about ill-structured tasks is that their vocabularies tend to differ considerably from one user to another. Contrary to, e.g., train time-table inquiries which can start uncontroversially from the "official" city names, or used car sales catalogues which can start from the car brands and production years, there is no "official" vocabulary, used by all or most users, for inquiring about pet transportation, high-volume luggage or staff support for the elderly. We are all familiar with this problem from the (non-electronic) yellow pages whose entries are often labelled differently from how we would have labelled them ourselves. SLDSs in fact offer an elegant solution to this problem, i.e. an electronic dictionary of equivalents. The user speaks his or her mind about the entry of interest. The system only has to spot a keyword in the user's input corresponding to an entry in its electronic dictionary, in order to direct the user to the right entry in its hierarchy. And the user does not have to worry about knowing or remembering the headings used to structure the hierarchy itself.

3.9.3 Well-structured tasks

Other tasks have some structure to them. Consider the flight ticket reservation task handled by the Danish Dialogue System. This task is a partially ordered one. It would normally make little sense, for instance, to ask for one of the morning departures until one has specified the date; and it normally makes little sense to expect the system to tell whether or not flights are fully booked on a certain date until one has indicated the itinerary. Moreover, ordinary users know that this is the case. Task structure is helpful if the task complexity makes it advisable to control the user's input (see 3.10).

It is important to note, however, that partial-order-only is what one is likely to find in most cases. Moreover, sub-task interdependencies may interfere with the designer's pre-conceived ideas about "the" task order. For instance, some users may want to know which departures are available at reduced fares before wanting to know about departure times, whereas others do not care about fare reductions at all. In this way, sub-task interdependencies may also introduce an element of negotiation into the dialogue. For instance, before accepting a certain departure, the caller may want to know if the subsequent departure is the cheaper of the two.

This makes the dialogue a two-way exchange where the caller and the system take turns in asking questions of one another and answering those questions. Both elements, negotiation and two-way information flow, complicate task model design.

Moreover, even the partial order that there is to the task, may be by default-only. If somebody simply wants to leave town as soon as possible, the itinerary matters less than the departure time of the first outbound flight which has a free seat.

3.9.4 Negotiation tasks

Does the task require substantial negotiation? Some task involve a relatively low volume of information and in addition have some structure to them. Still, they may be difficult to manage if they involve a considerable amount of negotiation. The Verbmobil meeting scheduling task is an example. Fixing a meeting simply requires fixing a date, a time or a time interval, and possibly a place, so the volume of information to be exchanged is relatively low. Unless one knows the date, it can be difficult to tell if one is free at a certain time, so the task has some structure to it. And for busy people, the date-time pair may matter more than the exact venue. The problem inherent to the Verbmobil task is that fixing meetings may require protracted, and ill-structured, negotiations of each sub-task. The outcome of a meeting date/time/venue negotiation is not a simple function of calendar availability and prior commitments but also depends on the importance of the meeting, the importance of the prior commitments, the possibilities of moving, cancelling or apologising other meetings, the professional and personal relationships between the interlocutors etc.

In addition, Verbmobil does nothing to impose structure on the (human-human) dialogue for which it provides translation support, allowing the dialogue to run freely wherever the interlocutors want it to go. In such cases, it makes little sense to judge the task complexity in terms of the volume of information to be exchanged or in terms of the task structure that is present, because the real problem lies in negotiating and eventually agreeing to meet. Had Verbmobil been a *human-machine* dialogue system product, the machine representing the diary of someone absent to the conversation, state-of-the-art engineering practice would probably have dictated much stronger system control of the dialogue, effectively turning meeting date/time/venue negotiation into a booking exercise (see 3.10).

Note also that negotiation is a natural human propensity. It seems likely that most SLDSs can be seen to potentially involve an element of negotiation. Verbmobil just involves a considerable amount of it! In the case of the Danish Dialogue System, for instance, the system may propose a list of morning departures and ask if the user wants one of them. If not, the joint search for a suitable departure time continues. So the reason why negotiation is less obvious or prominent in the dialogue conducted with the Danish Dialogue System when compared to the dialogue conducted with Verbmobil, is not that the task of the Danish system by itself excludes negotiation. Rather, the reason is that the system's dialogue behaviour has been designed to constrain the way in which negotiations are done.

3.9.5 Summary

When developing SLDSs we want to let the users speak their minds. However, if the task is a complex one, the SLDS project either has to be given up as a commercial undertaking, turned into a research project, such as Verbmobil, or requires some or all of the following strategies: (a) input prediction (see 3.7.1), (b) input language processing control (see 3.7.2), (c) output control (see 3.7.3), and (d) control of user input (see 3.10). To the extent that they are applicable, all of (a) through (d) are of course useful in any system but their importance grows with increasing task complexity. (d) subsumes (c) as shown in the next section. Obviously, (a) through (d) can also be used for very simple tasks, making dialogue engineering more easy to do for these tasks than if the users were permitted to speak their minds in a totally unconstrained fashion. In fact, as we shall see in Section 3.10, this language of ‘not permitting’ users certain things, ‘constraining them’ etc. is misleading. Elegant dialogue management solutions can often be found which do not in the least reduce the user’s sense of being engaged in natural spoken communication.

3.10 Controlling user input

The dialogue manager can do many things to control the user’s input in order to keep it within the system’s technical capabilities of recognition and understanding. Among task-oriented SLDSs, extreme lack of user input control may be illustrated by a system which simply tells the user about the task it can help solve and then invites the user to go ahead. For instance:

“Freddy ‘s Used Cars Saloon. How can I help you?”

Lack of user input control allows the user to produce *unconstrained input.* Clear signs of unconstrained user input are: very long input sentences; topics are being raised, or sub-tasks addressed, in any order; any number of topics is being addressed in a single user utterance. Even for comparatively simple tasks, the handling of unconstrained user input is a difficult challenge which most SLDS projects cannot afford to try to meet.

What user input control does is to impose, through a variety of explicit and implicit means, more or less strong constraints on what would otherwise have been unconstrained user input. Some of the user input control mechanisms available to the SLDS developer are:

3.10.1 Information on system capabilities

This section is about explicit information to users on what the system can and cannot do. For all but the simplest SLDSs, and for all but very special user populations, it is advisable that the system, up-front, clearly and succinctly tells the user things like what is its domain, what is its task(s) etc. This helps “tailoring” the user’s expectations, and hence the user’s input, to the knowledge the system actually has, thereby ultimately reducing the task of the dialogue manager. For instance, the following two systems both qualify in the generic sense as ferry time-table information systems (Ss): system S1 knows everything about the relevant, current and planned time-tables; S2 knows, in addition, about significant delays and is thus able to distinguish between planned and actual arrivals and departures. It is important to tell users whether the system they are about to use is an S1 or an S2.

System capability information does not have to be given to users through speech, of course. Waxholm, for instance, provides this information as text on the screen as well as in synthetic speech. If the information is given through speech, it is virtually always important that it is expressed briefly and clearly because users tend to loose attention very quickly when they have to listen to speech-only information.

It is often an empirical issue how much users need to be told about the system’s capabilities in order that their mental model of the system’s capabilities roughly match what the system actually can and cannot do. If the story is longer than a few facts, it is advisable to make it possible for regular users to skip the story. Also, those parts of the story which only relate to some optional loop in the dialogue, might be better presented at the start of that loop than up-front.

Brief on-line information on the system’s capabilities cannot be made redundant by any amount of paper information about the system.

3.10.2 Instructions on how to address the system

This section is about explicit instructions to users on how to address the system. By issuing appropriate instructions, the SLDS may strongly increase its control of the way it is being used. For instance, the Danish Dialogue System tells it users that it will not be able to understand them unless they answer the system's questions briefly and one at a time. If used at all, such operating instructions should be both very brief and eminently memorable. Otherwise, they will not work because too many users will forget the instructions immediately. Indications are that the quoted instruction from the Danish Dialogue System worked quite well. However, as part of the operating instructions for the Danish system, users were also instructed to use particular keywords when they wanted to initiate meta-communication (see 3.16). This worked less well because too many users forgot the keywords they were supposed to use.

3.10.3 Feedback on what the system understood

Feedback on what the system has understood from what the user just said helps ensure that, throughout the dialogue, the user is left in no doubt as to what the system has understood (see also 3.20). All SLDSs need to provide this form of (information) feedback. A user who is in doubt as to whether the system really did understand what the user just said, is liable to produce unwanted input.

3.10.4 Processing feedback

Processing feedback is about what the system is in the process of doing. When the system processes the information received from the user and hence may not be speaking for a while, processing feedback keeps the user informed on what is going on. (see also 3.20). Most SLDSs can benefit from this form of (processing) feedback. A user who is uncertain about what is going on inside the system, if anything, is liable to produce unwanted input.

3.10.5 Output control

The aim of output control (see also 3.7) is to "prime" the user through the vocabulary, grammar and style adopted by the system. There is ample evidence that this works very well for SLDSs. Humans are extremely good at (automatically, unconsciously) adapting their vocabulary, grammar and style to those of their partners in dialogue or conversation [33, 34]. Just think about how easily we adapt linguistically to small children, the hard of hearing, or foreigners with little mastery of our mother tongue. It is therefore extremely useful to make the system produce output which only uses the vocabulary and grammar which the system itself can recognise, parse and understand, and to make the system use a style of dialogue which induces the user to provide input which is terse and to the point. The unconscious adaptation performed by the users ensures that they still feel that they can speak their minds without feeling hampered by the system's requirements for recognisable vocabulary, simple grammar, and terse style.

A particular point to be aware of in this connection is that if the system's output to the user includes, e.g., typed text on the screen, then the textual output should be subjected to the same priming strategy as has been adopted for the system's spoken output. It is not helpful to carefully prime the user through the system's output speech and then undercut the purpose of the priming operation through a flourishing style of expression in the textual output.

3.10.6 Focused output and system initiative

If the system determines the course of the dialogue by having the initiative (see 3.11) all or most of the time, for instance through asking questions of the user or providing the user with instructions which the user has to carry out, a strong form of user input control becomes possible. The system can phrase its questions or instructions in a *focused* way so that, for each question, instruction etc., the user has to choose between a limited number of response options. If, for instance, the system asks a question which should be answered by a 'yes' or 'no', or by a name drawn from a limited set of proper names (of persons, weekdays, airports, train stations, streets, car brand names etc.), then it exerts a strong influence on the user's input. Dialogue managers using this approach may be able to handle even very-large-complexity tasks in terms of information volume (see 3.9.1).

Note that, in itself, system initiative is far from sufficient for this kind of input control to take place. If the system says, for instance, "ADAP Travels, can I help you?", it does, in principle, take the initiative by asking a question. However, the question is not focused at all and therefore does not restrict the user's input. An open or unfocused system question may therefore be viewed as a way of handing over the dialogue initiative to the user.

Note also that the method just described is better suited for some tasks than for others, in particular for tasks consisting of a series of independent pieces of information to be provided to the system by the user. Beyond those tasks, the strong form of input control involved will tend to give the dialogue a less-than-natural and rather mechanical quality. Moreover, some tasks do not lend themselves to system initiative-only, and large unstructured tasks cannot be handled through focused output combined with system initiative in a way which is acceptable to the users.

3.10.7 Textual material

The term 'textual material' designates information about the system in typed or hand-written form and presented graphically, such as on screen or on paper, or haptically, such as using Braille. Typically, this information tells users what the system can and cannot do and instructs them on how to interact with the system. For particular purposes, such as when the users are professionals and will be using the SLDS extensively in their work, strong user input control can be exerted through textual material which the user is expected to read when, or before, using the system. For obvious reasons, text-based system knowledge is difficult to rely on for walk-up-and-use systems unless the system, like Waxholm, includes a screen or some other text-displaying device. Users are not likely to have textual material on paper at hand when using the system: it is somewhere else, it has disappeared, or they never received it in the first place.

3.10.8 Barge-in

Barge-in means that users can speak to, and expect to be recognised and understood by, the system whenever they so wish, such as when the system itself is speaking or is processing recent input. Barge-in is not, in fact, an input control mechanism. Rather, it is something which comes in handy because full input control is impossible. In particular, it is impossible to prevent enough users from speaking when the system is not listening, no matter what means are being adopted for this purpose.

The likelihood of users speaking their minds "out of order" varies from one application and user group to another. In some applications, it may even be desirable that the users can speak freely among themselves whilst the system is processing the spoken input.

Still, barge-in technology is advisable for very many SLDSs, so that the system is able to recognise and process user input even if it arrives when the system is busy doing something other than just waiting for it. In Waxholm, the system does not listen when it speaks. However, the user may barge in by pressing a button to interrupt the system's speech. This is useful when, for instance, the user already feels sufficiently informed to get on with the task. For instance, users who are experts in using the application can use the button-based barge-in to skip the system's introduction. The Danish Dialogue System does not allow barge-in when the system speaks. This turned out to cause several transaction failures, i.e. dialogues in which the user did not get the result asked for. A typical case is one in which the system's feedback to the user (see 3.20) shows that the system has misunderstood the user, for instance by mistaking "Saturday" for "Sunday". During the system's subsequent output utterance, the user says, e.g.: "No, Saturday". The system's lack of reaction to what the user said is easily interpreted by the user as if the system had received the error message, which of course it hadn't, and couldn't have. As a result, the user will later receive a flight ticket which is valid for the wrong day.

3.11 Who should have the initiative?

Dialogue in which the initiative lies solely with the system was discussed as an input control mechanism in Section 3.10.6. This section generalises the discussion of initiative initiated in Section 3.10.6.

Dialogue management design takes place between two extremes. From the point of view of technical simplicity, one might perhaps wish that all SLDSs could conduct their transactions with users as a series of questions to which the users would have to answer 'yes' or 'no' and nothing else. Simpler still, 'yes' or 'no' could be replaced by filled pauses ("grunts") and unfilled pauses (silence), respectively, between the system's questions, and speech recognition could be replaced by grunt detection. From the point of view of natural dialogue, on the other hand, users should always be able to say exactly what they want to say, in the way they want to say it, and when they want to say it, without any restrictions being imposed by the system. Both extremes are unrealistic, of course. If task complexity is low in terms of, among other things, information volume and negotiation potential, then it is technically feasible today to allow the users to say what they want whilst still using some of the input control mechanisms discussed in Section 3.10. As task complexity grows in terms of information volume, negotiation potential and the other factors discussed in Section 3.9, it really begins to matter who has the initiative during the dialogue.

We may roughly distinguish three interaction modes, i.e. *system directed dialogue, mixed initiative dialogue,* and *user directed dialogue.* This distinction is a rough-and-ready one because initiative distribution among user and system is often a matter of degree depending upon how often which party is supposed to take the initiative. In some cases, it can even be difficult to classify an SLDS in terms of who has the initiative. If the system opens the dialogue by saying something like: "Welcome to service X, what do you want?" - it might be argued that the system has the initiative because the system is asking a question of the user. However, since the question asked by the system is a completely open one, one might as well say that the initiative is being handed over to the user. In other words, only focused questions clearly determine initiative (cf. 3.10.6). The same is true of other directive uses of language in which partner A tells partner B to do specific things, such as in instructional dialogues.

3.11.1 System directed dialogue

As long as the task is one in which the system requires a series of specific pieces of information from the user, the task may safely be designed as one in which the system preserves the initiative throughout by asking focused questions of the user. This system directed approach would work even for very-large-complexity tasks in terms of information volume and whether or not the tasks are well-structured. Note that, if the sub-tasks are not mutually independent or if several of them are optional, then system initiative may be threatened (cf. 3.9). Still, system directed dialogue is an effective strategy for reducing user input complexity and increasing user input predictability (cf. 3.7.1). In addition, system directed dialogue actually is relatively natural for some tasks.

From a dialogue engineering perspective, it may be tempting to claim that system directed dialogue is generally simpler to design and control than either user directed dialogue or mixed initiative dialogue, and therefore should be preferred whenever possible. This claim merits some words of caution. Firstly, it is not strictly *known* if the claim is true. It is quite possible that system directed dialogue, for all but a relatively small class of tasks, is *not* simpler to design and control than its alternatives because it needs ways of handling those users who do not let themselves be fully controlled but speak out of turn, initiate negotiation, ask unexpected questions etc. Secondly, products are already on the market which allow mixed initiative dialogue for relatively simple tasks, such as train time-table information [6]. And it is quite likely that users generally tend to prefer such systems because they let the users speak their minds to some extent.

3.11.2 Mixed initiative dialogue

In mixed initiative dialogue, any one of the participants may have – or take - the initiative. A typical case in which a mixed initiative approach is desirable is one in which the task is large in terms of information volume and both the user and the system need information from one another. Whilst asking its questions, the system must be prepared, sometimes or all of the time, that the user may ask a question in return instead of answering the system's own question. For instance, the user may want to know if a certain flight departure allows discount before deciding whether that departure is of any interest.

In practice, most SLDSs have to be mixed initiative systems in order to be able to handle user-initiated repair meta-communication. The user must have the possibility of telling the system, at any point during dialogue, that the system has misunderstood the user or that the user needs the system to repeat what it just said (see 3.16). Only systems which lack meta-communication altogether can avoid that. Conversely, even if the user has the initiative throughout the dialogue, the system must be able to take the initiative to do repair or clarification of what the user has said. When thinking about, or characterising, SLDSs, it may be useful, therefore, to distinguish between two issues: (a) who has the initiative in domain communication, and (b) who has the initiative in meta-communication (more in 3.16).

The need for mixed initiative dialogue in *domain communication* is a function of bi-directionality of the flow of information needed to complete the task, sub-task interdependencies, the potential for negotiation occurring during task completion, the wish for natural and unconstrained dialogue etc. (cf. 3.9). Ceteris paribus, mixed initiative dialogue is harder to control and predict than system directed dialogue.

3.11.3 User directed dialogue

In user directed dialogue, the user has the initiative all or most of the time. User directed dialogue is recommended for ill-structured tasks in which there is no way for the system to anticipate which parts of the task space the user wants to address on a particular occasion. The flight conditions information task (see 3.9) is a case in point, as is the email operation task (see 3.5). User directed dialogue is also useful for tasks with regular users who have the time to learn how to speak to the system to get the task done. However, user directed dialogue is harder to control and predict than system directed dialogue. For the time being, user directed dialogue is not recommended for walk-up-and-use users except for very simple tasks.

3.12 Input prediction/prior focus

In order to support the system's speech recognition, language processing and dialogue management tasks, the dialogue manager developer should investigate if selective prediction of the user's input is possible at any stage during the dialogue (see 3.7). This may be possible if, e.g., the system asks a series of questions each requesting specific pieces of information from the user. If the task has some structure to it, it may even be possible to use the structure to predict when the user is likely to ask questions of the system, thus facilitating mixed initiative (3.11) within a largely system directed dialogue. For instance, the Daimler-Chrysler dialogue manager and the Danish Dialogue System use various forms of input prediction. Another way of describing input prediction is to say that the dialogue manager establishes a (selective) *focus of attention* prior to the next user utterance.

Useful as it can be, input prediction may fail because the user does not behave as predicted. In that case, the dialogue manager must be able to initiate appropriate meta-communication (see 3.16). This is not necessarily easy to do in case of failed predictions because the system may not be aware that the cause of failed recognition or understanding was its failure to predict what the user said. Unless it relaxes/cancels the prediction, the risk is that the dialogue enters an error loop in which the system continues to fail to understand the user.

Input prediction can be achieved in many different ways. It may be useful to distinguish between the following two general approaches.

3.12.1 Knowledge-based input prediction

In *knowledge-based input prediction,* the dialogue manager uses a priori knowledge of the context to predict one or more characteristics of the user's next utterance. Note that the a priori nature of knowledge-based input prediction does not mean that implemented predictions should not be backed by data on actual user behaviour. It is always good practice to test the adopted knowledge-based input prediction strategy on user-system interaction data.

3.12.2 Statistical input prediction

In *statistical input prediction,* the dialogue manager uses corpus-based information on what to expect from the user. Given a corpus of user-system dialogues about the task(s) at hand, it may be possible to observe and use regularities in the corpus, such as that the presence of certain words in the user's input makes it likely that the user is in the process of addressing a specific subset of the topics handled by the system, or that the presence of dialogue acts DA5 and DA19 in the immediate dialogue history makes it likely that the user is expressing DA25. Waxholm uses the former approach, Verbmobil the latter.

3.13 Sub-task identification

It is a useful exercise for the dialogue manager developer to consider the development task from the particular point of view of the dialogue manager. The dialogue manager is deeply embedded in the SLDS, is out of direct contact with the user, and has to do its job based on what the speech and language layers deliver. This happens in the context of the task, the target user group, and whatever output and input control the dialogue manager may have imposed.

Basically, what the speech and language layers can deliver to the dialogue manager is some form of meaning representation. Sometimes the dialogue manager does not receive any meaning representation from the speech and language layers even though one was expected. But even if a meaning representation arrives, there is no guarantee that this representation adequately represents the contents of the message that was actually conveyed to the system by the user because the speech and language layers may have got the user's expressed meaning wrong. Still, whatever happens, the dialogue manager must be able to produce appropriate output to the user.

Current SLDSs exhibit different approaches to the creation of a meaning representation in the speech and language layers as well as to the nature of the meaning representation itself. An important point is the following: strictly speaking, the fact that a meaning representation arrives with the dialogue manager is not sufficient for the dialogue manager to carry on with the task. *First, the dialogue manager must identify to which sub-task(s), or topics, if any, that incoming meaning representation provides a contribution.* Only when it knows, or believes that it knows, that, can the dialogue manager proceed to sort out which contribution(s), if any, the incoming meaning representation provides to the sub-task (or those sub-tasks). In other words, many task-oriented SLDSs require the dialogue manager to do sub-task identification or topic identification.

The task solved by most SLDSs can be viewed as consisting in one or several sub-tasks or topics to be addressed by user and system. One or several of these sub-tasks have to be solved in order that the user and the system succeed in solving the task. Other sub-tasks may be optional, i.e. their solution is sometimes, but not always, required. An example class of optional sub-tasks is the meta-communication sub-tasks (see 3.16): if the dialogue proceeds smoothly, no meta-communication sub-tasks have to be solved.

Basically, dialogue managers can be built so as to be in one of two different situations with respect to sub-task identification. In the first case, the dialogue manager has good reason to assume that the user is addressing a particular domain sub-task; in the second case, the dialogue manager does not know which domain sub-task the user is addressing. In both cases, the dialogue manager must start from the semantic representations that arrive from the speech and language layers, look at the semantically meaningful units, and seek to figure out which sub-task the user is addressing.

3.13.1 Local focus

The dialogue manager may have good reason to assume that the user's utterance addresses a specific sub-task, such as that of providing the name of an employee in the organisation hosting the system. Depending on the task and the dialogue structure design, there can be many different reasons why the dialogue manager knows which sub-task the user is currently addressing: there may be only one sub-task, as in an extremely simple system; the task may be well-structured; the system just asked the user to provide input on that sub-task, etc. Generally speaking, this is a good situation for the dialogue manager to be in, as in the Danish Dialogue System. This system almost always knows, or has good reason to believe, that the user is either addressing a specific domain sub-task or has initiated meta-communication. Since it has good reason to believe which sub-task the user is addressing, the task of the dialogue manager reduces to that of finding out exactly what is the user's contribution to that sub-task (or one of those sub-tasks, if we count in the possibility that the user may have initiated meta-communication). In such cases, the system has a *local focus.*

The system may still be wrong, of course, and then it becomes the joint task of the system and the user to rectify the situation through meta-communication.

3.13.2 Global focus

The dialogue manager does not know which of several possible sub-tasks the user is addressing. The main reason why the dialogue manager does not know which sub-task the user is currently addressing, is that the dialogue manager has given the user the initiative, for instance by asking an open question in response to which the user may have addressed any number of possible sub-tasks. Alternatively, the user unexpectedly took the initiative. In such situations, the dialogue manager has to do sub-task identification, or topic identification, before it can start processing the user's specific contribution to the sub-task.

Sub-task identification is crucial in systems such as RailTel/ARISE, Waxholm, and Verbmobil. Waxholm uses probabilistic rules linking semantic input features with topics. Given the rules and a particular set of semantic features in the input, Waxholm infers which topic the user is actually addressing. For sub-task identification, Verbmobil uses weighted default rules to map from input syntactic information, keywords, and contextual information about which dialogue acts are likely to occur, into one or several dialogue acts belonging to an elaborate taxonomy of approximately 54 speech acts (or dialogue acts).

An important additional help in sub-task identification is support from a *global focus,* for instance when the dialogue manager knows that the task history (see 3.22) contains a set of not-yet-solved sub-tasks. These tasks are more likely to come into local focus than those which have been solved already. Another form of global focus can be derived from observation of dialogue phases. Most task-oriented dialogues unfold through three main phases, the introduction phase with greetings, system introductions etc., the main task-solving phase, and the closing phase with closing remarks, greetings etc. Sometimes it may be possible to break down the main task-solving phase into several phases as well. If the system knows which phase the dialogue is in at the moment, this knowledge can be used for sub-task identification support. Knowing *that* can be a hard problem, however, and this is a topic for ongoing research. For instance, the joint absence of certain discourse particles called topic-shift markers and the presence of certain dialogue acts may suggest that the user has not changed dialogue phase.

Generally speaking, the more possible sub-tasks the user might be addressing in a certain input utterance, the harder the sub-task identification problem becomes for the dialogue manager. When doing sub-task identification, the dialogue manager may follow one of two strategies. The simpler strategy is to try to identify one sub-task only in the user's input even if the user may have been addressing several sub-tasks, and continue the dialogue from there as in Waxholm. The more demanding strategy is to try to identify each single sub-task addressed by the user as in RailTel/ARISE. The latter strategy is more likely to work when task complexity is low in terms of volume of information.

3.13.3 After sub-task identification

Depending on what arrives from the speech and language layers, and provided that the dialogue manager has solved its sub-task identification task, the dialogue manager must now determine the users' specific contribution(s) to the sub-task(s) they are addressing (see 3.13). Following that, the dialogue manager must do one of five things as far as communication with the user is concerned:

(a) advance the domain communication (see 3.15) including the provision of feedback (see 3.20),

(b) initiate meta-communication with the user (see 3.16 and 3.19),

(c) initiate other forms of communication (see 3.17),

(d) switch to a fall-back human operator, or

(e) end the dialogue (see 3.21).

Advancing the domain communication means getting on with the task. Initiating meta-communication means starting a sub-dialogue (or, in this case, a meta-dialogue) with the user in order to get the user's meaning right before advancing the domain communication any further. No SLDS probably can do without capabilities for (a) and (b). If (b) fails repeatedly, some systems have the possibility of referring the user to a human operator (d). Otherwise, calling off the dialogue is the only possibility left (e).

In parallel with taking action vis-à-vis the user, the dialogue manager may at this stage take a smaller or larger series of internal processing steps which can be summarised as:

(f) updating the context representation (see 3.22) and

(g) providing support for the speech and language layers to assist their interpretation of the next user utterance (see 3.7).

3.14 Advanced linguistic processing

The determination of the user's contribution to a sub-task requires more than, to mention just one example, the processing of semantic feature structures. Processing of feature structures often implies value assignment to slots in a semantic frame even though these values cannot straightforwardly be derived from the user's input in all cases.

If the system has to decide on *every* contribution of a user to a sub-task, something which few systems do at present, advanced linguistic processing is needed. It may involve, among other things, cross-sentence co-reference resolution, ellipsis processing, and the processing of indirect dialogue acts. In nearly all systems, the processing of most of these phenomena is controlled and carried out by one of the natural language components - by the parser in the Daimler-Chrysler dialogue manager, by the semantic evaluation component in the Verbmobil speech translation system - but never without support from the dialogue manager.

3.14.1 Co-reference and ellipsis processing

In case of cross-sentence co-reference and ellipsis processing, the natural language system component in charge is supported by the dialogue manager which provides a representation of contextual information for the purpose of constraining the relevant search space. The contextual information is part of the dialogue history (see 3.22). Dialogue history information consists in one or several data structures that are being built up incrementally to represent one or more aspects of the preceding part of the dialogue. In principle, the more aspects of the preceding dialogue are being represented in the dialogue history, the more contextual information is available for supporting the processing done in the language layer, and the better performance can be expected from that layer. Still, co-reference resolution and ellipsis processing remain hard problems.

3.14.2 Processing of indirect dialogue acts

Advanced linguistic processing also includes the processing of indirect dialogue acts. In this case, the central problem for the system is to identify the "real" dialogue act performed by the user and disguised as a dialogue act of a different type. In contrast to direct dialogue acts, indirect dialogue acts cannot be determined on the basis of their surface form, which makes the frequently used keyword spotting techniques used for the identification of direct dialogue acts almost useless in such cases. Clearly, the processing of indirect dialogue acts calls for less surface oriented processing methods involving semantic and pragmatic information associated with input sequences. This is a hard problem.

Communication

3.15 Domain communication

The primary task of the dialogue manager is to advance the domain communication based on a representation of the meaning-in-task-context of the user's input (cf. 3.13 and 3.14). Let us assume that the dialogue manager has arrived at an interpretation of the user's most recent input and decided that the input actually did provide a contribution to the task. This means that the dialogue manager can now take steps towards advancing the domain communication with the user. Obviously, what to do in a particular case depends on the task and the sub-task context. The limiting case is that the dialogue manager simply decides that it has understood what the user said and takes overt action accordingly, such as connecting the caller to a user who has been understood to want to accept a collect call, replaying an email message, or displaying a map on the screen. Some other cases are:

3.15.1 More information needed

The dialogue manager inserts the user's input meaning into a slot in the task model, discovers that more information is needed from the user, and proceeds to elicit that information.

3.15.2 Database look-up

The dialogue manager looks up the answer to the user's question in the database containing the system's domain knowledge and sends the answer to the language and speech generation components (or to the screen, etc.).

3.15.3 Producing an answer

The dialogue manager inserts the user's input meaning into a slot in the task model, verifies that it has all the information needed to answer the user's query, and sends the answer to the language and speech generation components (or to the screen, etc.).

3.15.4 Making an inference

The dialogue manager makes an inference based on the user's input meaning, inserts the result into a slot in a database and proceeds with the next question. In Waxholm, for instance, the system completes the user's 'on Thursday' by inferring the appropriate date, and replaces qualitative time expressions, such as 'this morning', by well-defined time windows, such as '6 AM - 12 AM'. Verbmobil does inferencing over short sequences of user input, such as that a counterproposal (for a date, say) implies the rejection of a previous proposal; a new proposal (for a date, say) implies the acceptance of a (not incompatible but less specific) previous proposal; and a change of dialogue phase implies the acceptance of a previous proposal (for a time, say).

It is important to note that such domain-based inferences abound in human-human conversation. Without thinking about it, human speakers expect their interlocutors to make those inferences. The dialogue manager has no way of replicating the sophistication of human inferencing during conversation and dialogue. Most current systems are able to process only relatively simple inferences. The dialogue manager developer should focus on enabling all and only those inferences that are strictly necessary for the application to work successfully in the large majority of exchanges with users. Even that can be a hard task. For instance, should the system be able to perform addition of small numbers or not? Simple as this may appear, it would add a whole new chapter to the vocabulary, grammar and rules of inference that the system would have to master. In travel booking applications, for instance, some users would naturally say things like "two adults and two children"; or, in travel information applications, some users may want to know about the 'previous' or the 'following' departure given what the system has already told them. The developer has to decide how important the system's capability of understanding such phrases is to the successful working of the application in real life. In many cases, making an informed decision will require empirical investigation of actual user behaviour.

Finally, through control of user input (see 3.10), the developer must try to prevent the user from requiring the system to do inferences that are not strictly needed for the application or which are too complex to implement.

3.15.5 More constraints needed

The dialogue manager discovers that the user's input meaning is likely to make the system produce too much output information and produces a request to the user to provide further constraints on the desired output.

3.15.6 Inconsistent input

The dialogue manager discovers that the user's input meaning is inconsistent with the database information, infeasible given the database, inconsistent with the task history, etc. The system may reply, e.g., "There is no train from Munich to Frankfurt at 3.10 PM ....", or "The 9 o-clock flight is fully booked already ....".

3.15.7 Language translation

The dialogue manager translates the user's input meaning into another language.

3.15.8 Summary

Among the options above, the first four ones and the last one illustrate straightforward progression with the task. The two penultimate options illustrate domain sub-dialogues.

Quite often, the system will, in fact, do something more than just advancing the domain communication as exemplified above. As part of advancing the domain communication, the system may provide feedback to the user to enable the user make sure that what the user just said has been understood correctly (see 3.20).

3.16 Meta-communication

Meta-communication, although secondary to domain communication, is crucial to proper dialogue management. Meta-communication is often complex and potentially difficult to design. In meta-communication design, it is useful to think in terms of distinctions between (a) *system-initiated* and *user-initiated* meta-communication and (b) *repair* and *clarification* meta-communication. These are all rather different from each other in terms of the issues they raise, and distinction between them gives a convenient breakdown of what otherwise tends to become a tangled subject. In general, one of the partners in the dialogue initiates meta-communication because that partner has the impression that something went wrong and has to be corrected.

Note that we do not include system feedback under meta-communication. Some authors do that and there does not seem to be any deep issue involved here one way or the other. We treat feedback as a separate form of system-to-user communication in 3.20. Primarily for the user, feedback (from the system) is the most important means for discovering that something went wrong and has to be corrected.

3.16.1 System-initiated repair meta-communication

System-initiated repair meta-communication is needed whenever the system has reason to believe that it did not understand the user's meaning. Such cases include, i.a.:

- nothing arrived for the dialogue manager to process although input meaning was expected from the user. In order to provide appropriate output in such cases, the dialogue manager must get the user to input the meaning once more. It is worth noting that this can be done in many different ways, from simply saying, e.g., "Sorry, I did not understand" or "Please repeat" to asking the user to speak louder or more distinctly. The more the system knows about the probable cause of its failing to understand the user, the more precise its repair meta-communication can be. Any such gain in precision increases the likelihood that the system will understand the user the next time around and thus avoid error loops (see 3.19);
- something arrived for the dialogue manager to process but what arrived was meaningless in the task context. For instance, the user may be perceived as responding 'London' to a question about departure date. In order to provide appropriate output in such cases, the dialogue manager may have to ask the user to input the meaning once more. However, as the system actually did receive some meaning representation, it should preferably tell the user what it did receive and that this was not appropriate in the task context. This is done by, e.g., Waxholm and the Danish Dialogue System. For instance, if the Danish Dialogue System has understood the user to want to fly from Aalborg to Karup, it will tell the user that there are no flight connections between these two airports. Another approach is taken in RailTel/ARISE. These systems would take the user's 'London' to indicate a change to the point of departure or arrival (see below, this section). Verbmobil uses a statistical technique to perform a form of constraint relaxation in case of a contextually inconsistent user input dialogue act (cf. 3.14).

A core problem in repair meta-communication design is that the user input that elicits system-initiated repair may have many different causes. The dialogue manager often has difficulty diagnosing the actual cause. The closer the dialogue manager can get to correctly inferring the cause, the more informative repair meta-communication it can produce, and the more likely it becomes that the user will provide comprehensible and relevant input in the next turn.

3.16.2 System-initiated clarification meta-communication

System-initiated clarification meta-communication is needed whenever the system has reason to believe that it actually did understand the user's meaning which, however, left some kind of uncertainty as to what the system should produce in response. Such cases include, i.a.:

- a representation of the user's meaning arrived with a note from the speech and/or language processing layers that they did not have any strong confidence in the correctness of what was passed on to the dialogue manager. The best approach for the dialogue manager to take in such cases probably is to get the user to input the meaning once more rather than to continue the dialogue on the basis of dubious information, which may easily lead to a need for more substantial meta-communication later on. Alternatively, as the system actually did receive some meaning representation, it might instead tell the user what it received and ask for the user's confirmation;
- a representation of the user's meaning arrived which was either inherently inconsistent or inconsistent with previous user input. In cases of inherent inconsistency which the system judges on the basis of its own domain representation, the system could make the possibilities clear to the user and ask which possibility the user prefers, for instance by pointing out that 'Thursday 9th' may be either 'Thursday 8th' or 'Friday 9th'. Cases of

inconsistency with previous user input are much more diverse, and different response strategies may have to be used depending on the circumstances;

- a representation of the user's meaning arrived which was (semantically) ambiguous or underspecified. For instance, the user asks to be connected to Mr. Jack Jones and two gentlemen with that name happen to work in the organisation; or the user wants to depart at "ten o-clock", which could be either AM or PM. In such cases, the system must ask the user for information that can help resolve the ambiguity. The more precisely this can be done, the better. For instance, if the system believes that the user said either 'Hamburg' or 'Hanover', it should tell the user just that instead of broadly asking the user to repeat. Experience indicates that it is dangerous for the system to try to resolve ambiguities on its own by selecting what the system (i.e. the designer at design-time) feels is generally the most likely interpretation. The designer may think, for instance, that people are more likely to go on a flight at ten AM than at ten PM and may therefore assign the default interpretation "ten AM" to users' "ten o-clock". If this approach of interpretation by default is followed, it is strongly advised to explicitly ask the user for verification through a "yes/no" feedback question (see 3.20).

Whilst user meaning inconsistency and (semantic) ambiguity are probably the most common and currently relevant illustrations of the need for system clarification meta-communication, others are possible, such as when the user provides the system with an irresolvable anaphor. In this case, the system should make the possible referents clear to the user and ask which of them the user has in mind.

As the above examples illustrate, system-initiated clarification meta-communication is often a 'must' in dialogue manager design.

In general, the design of system clarification meta-communication tends to be difficult, and the developer should be prepared to spend considerable effort on reducing the amount of system clarification meta-communication needed in the application. This is done by controlling the user's input and by providing cooperative system output. However, as so often is the case in systems design, this piece of advice should be counter-balanced by another. Speaking generally, users tend to loose attention very quickly when the system speaks. It is therefore no solution to let the system instruct the user at length on what the system really means, or wants, on every occasion where there is a risk that the user might go on to say something which is ambiguous or (contextually) inconsistent. In other words, careful prevention of user behaviour which requires system-initiated clarification meta-communication should be complemented by careful system clarification meta-communication design. One point worth noting is that, for a large class of system-initiated clarifications, yes/no questions can be used.

3.16.3 User-initiated repair meta-communication

User-initiated repair meta-communication is needed whenever the system has demonstrated to the user that is has misunderstood the user's intended meaning. It also sometimes happens that users change their minds during the dialogue, whereupon they have to go through the same procedures as when they have been misunderstood by the system. In such cases, the user must make clear to the system what the right input is. Finally, users sometimes fail to get what the system just said. In this case, they have to ask the system to repeat just as when the system fails to get what the user just said. These three (or two) kinds of user repair meta-communication are mandatory in many systems.

User-initiated repair meta-communication can be designed in several different ways.

(a) *Uncontrolled repair input.* Ideally, we would like the users to just speak their minds whenever they have been misunderstood by the system, changed their minds with respect to what to ask of, or tell, the system, or failed to get what the system just said. Some systems do that, such as Waxholm, but with varying success, the problem being that users may initiate repair in very many different ways, from "No, Sandhamn!" to "Wait a minute. I didn't say that. I said Sandhamn!"

(b) *Repair keywords.* Other systems require the user to use specifically designed keywords, again with varying success. In the Danish Dialogue System, users are asked to use the keyword 'change' whenever they have been misunderstood by the system or changed their minds, and to use the keyword 'repeat' whenever they failed to get what the system just said. Keywords are simpler for the system to handle than unrestricted user speech. The problem is that users sometimes fail to remember the keywords they are supposed to use. The more keywords users have to remember, the higher the risk that they forget them. For walk-up-and-use systems, something in the order of two to three keywords would seem to be the maximum users can be expected to remember.

(c) *Erasing.* A third approach is used in RailTel/ARISE. This approach is similar to using an eraser: one erases what was there and writes something new in its place. For instance, if the system gets 'Frankfurt to Hanover' instead of 'Frankfurt to Hamburg', the user simply has to repeat 'Frankfurt to Hamburg' until the system has received the message. No specific repair meta-communication keywords or meta-dialogues are needed. The system is continuously prepared to revise its representation of the user's input based on the user's latest utterance. Whilst this solution may work well for low-complexity tasks, it will not work for tasks involving selective input prediction (see 3.12) and may be difficult to keep track of in high-complexity tasks.

3.16.4 User-initiated clarification meta-communication

User -initiated clarification meta-communication is probably the most difficult challenge for the meta-communication designer. Just like the user, the system may output, or appear to the user to output, inconsistent or ambiguous utterances, or use terms which the user is not familiar with. In human-human conversation, these problems are easily addressed by asking questions such as: "What do you mean by <u>green departure</u>?" or "Do you mean <u>scheduled</u> arrival time or <u>expected</u> arrival time?" Unfortunately, most current SLDSs are not being designed to handle such questions at all. The reasons are (a) that this is difficult to do and, often more importantly, (b) that the system developers have not discovered such potential problems in the first place. If they had, they might have tried to avoid them in their design of the system's dialogue behaviour, i.e. through user input control. Thus, they would have made the system explain the notion of a green departure before the user was likely to ask what it is, and they would have made the system explicitly announce when it was speaking about scheduled arrivals and when is was speaking about expected arrivals. In general, this is one possible strategy to follow by the dialogue manager developer: to remove in advance all possible ambiguities, inconsistencies and terms unknown to users, rather than to try to make the system handle questions from users about these things. We have developed a tool in support of co-operative system dialogue design [35, http://www.elsnet.org/disc]. Part of the purpose of this tool is to avoid situations in which users feel compelled to initiate clarification meta-communication.

There is an obvious alternative to the strategy recommended above of generally trying to prevent the occurrence of user-initiated clarification meta-communication. The alternative is to strengthen the system's ability to handle user-initiated clarification meta-communication. The nature of the task is an important factor in determining which strategy to follow or emphasise. Consider, for instance, users inquiring about some sort of yellow pages commodity, such as electric guitars or used cars. Both domains are relatively complex. In addition, the inquiring users are likely to differ widely in their knowledge about electric guitars and cars. A flight ticket reservation system may be able to address its domain *almost* without using terms which are unknown to its users, whoever these may be. Not so with a used cars information system. As soon as the system mentions ABS brakes, racing tyres or split back seats, some users will be wondering what the system is talking about. In other words, there seems to be a large class of potential SLDSs which can hardly start talking before they appear to speak gibberish to some of their intended users. In such cases, the dialogue manager developers have better prepare for significant user-initiated clarification meta-communication. It is no practical option for the system to explain all the domain terms it is using as it goes along. This would be intolerable for users who are knowledgeable about the domain in question.

3.16.5 Summary

To summarise, the dialogue manager is several steps removed from direct contact with the user. As a result, the dialogue manager may fail to get the user's meaning or it may get it wrong. Therefore, both the system and the user need to be able to initiate repair meta-communication. Even at low levels of task complexity, users are able to express themselves in ways that are inconsistent or ambiguous. The system needs clarification meta-communication to handle those user utterances. In some tasks, user clarification meta-communication should be prevented rather than allowed. In other tasks, user clarification meta-communication plays a large role in the communication between user and system.

3.17 Other forms of communication

Domain communication including feedback (see 3.20) and meta-communication are not the only forms of communication that may take place between an SLDS and its users. Thus, the domain-independent opening of the dialogue by some form of greeting is neither domain communication nor meta-communication. The same applies to the closing of the dialogue (see 3.21). These formalities may also be used in the opening and closing of sub-dialogues. Another example is system time-out questions, such as "Are you still there?", which may be used when the user has not provided input within a certain time limit.

If the SLDS's task-delimitation is not entirely natural and intuitive to users (cf. 3.10.1), users are likely to sometimes step outside the system's unexpectedly limited conception of the task. By the system's definition, the communication then ceases to be domain communication. For some tasks, users' out-of-domain communication may happen too often for comfort for the dialogue manager developer who may therefore want to do something about it. Thus, Waxholm is sometimes able to discover that the user's input meaning is outside the domain handled by the system. This is a relatively sophisticated thing to do because the system must be able to understand out-of-domain terms. Still, this approach may be worth considering in cases where users may have reason to expect that the system is able to handle certain sub-tasks which the system is actually unable to deal with. When the users address those sub-tasks, the system will tell them that, unfortunately, it cannot help them. In the Verbmobil meeting scheduling task, users are prone to produce *reasons* for their unavailability on certain dates or times. Verbmobil, however, whilst being unable to understand such reasons, nevertheless classifies them and represents them in the topic history (see 3.22).

3.18 Expression of meaning

Once the system has decided what to say to the user, this meaning representation must be turned into an appropriately expressed output utterance. In many cases, this is being done directly by the dialogue manager. Having done its internal processing jobs, the dialogue manager may take one of the following approaches, among others:

3.18.1 Pre-recorded utterances

The dialogue manager selects a stored audio utterance and causes it to be played to the user by sending a message to the player.

3.18.2 Concatenation of pre-recorded words and phrases

The dialogue manager concatenates the output utterance from stored audio expressions or phrases and causes it to be played to the user by sending a message to the player.

3.18.3 Filling in a template used by a synthesiser

The dialogue manager selects or fills an output sentence template and causes it to be synthesised to the user.

3.18.4 Producing meaning

A more sophisticated approach is to have the dialogue manager produce the *what,* or the meaning, of the intended output and then have the output language layer determine the *how,* or the form of words to use, in the output. In this approach, the how is often co-determined by accompanying constraints from the dialogue manager's control and context layers, such as that the output should be a question marked by rising pitch at the end of the spoken utterance.

3.18.5 Summary

The first two options are closely related and are both used in, e.g., the Danish Dialogue System. Waxholm and the Daimler-Chrysler dialogue manager use the third option. This option is compatible with relatively advanced use of control layer information for, e.g., determining the prosody of the spoken output. This can also be done in the first approach but is difficult to do in the second approach because of the difficulty of controlling intonation in concatenated pre-recorded speech.

3.19 Error loops and graceful degradation

An important issue for consideration by the dialogue management developer is the possibility that the user simply repeats the utterance which caused the system to initiate repair meta-communication. The system may have already asked the user to speak louder or to speak more distinctly but, in many such cases, the system will be in exactly the same uncomprehending situation as before. The system may try once more to get out of this potentially infinite loop but, evidently, this cannot go on forever. In such cases, the system might either choose to fall back on a human operator or close the dialogue. To avoid that, a better strategy is in many cases for the system to attempt to carry on by changing the level of interaction into a simpler one, thereby creating a 'graceful degradation' of the (domain or meta-) communication with the user [36]. Depending on the problem at hand and the sophistication of the dialogue manager, this can be done in many different ways, including:

3.19.1 Focused questions

The user may be asked focused questions one at a time instead of being allowed to continue to provide one-shot input which may be too lengthy or otherwise too complex for the system to understand. For instance, the system goes from saying "Which information do you need?" to saying "From where do you want to travel?"

3.19.2 Asking for re-phrasing

The user may be asked to re-phrase the input or to express it more briefly, for instance when the user's answer to a focused question is still not being understood.

3.19.3 Asking for a complete sentence

The user may be asked to produce a complete sentence rather than grammatically incomplete input, as in Waxholm.

3.19.4 Yes/no questions

The user may be asked to answer a crucial question by 'yes' or 'no'.

3.19.5 Spelling

The user may be asked to spell a crucial word, such as a person name or a destination.

3.19.6 Summary

It is important to note that the levels of interaction/graceful degradation approach can be used not only in the attempt to get out of error loops but also in combination with system-initiated clarification meta-communication (cf. 3.16.2). So, generalising, whenever the system is uncertain about the user's meaning, graceful degradation may be considered. The Daimler-Chrysler dialogue manager standardly accepts three repetitions of a failed user turn before applying the graceful degradation approach. This seems reasonable.

3.20 Feedback

System feedback to users is essential to successful dialogue management. In order to be clear about what system feedback involves, it is convenient to distinguish between two kinds of feedback, *information feedback* and *process feedback.*

3.20.1 Information feedback

The user must have the opportunity to verify that the system has understood the user's input correctly. In general, the user should receive feedback on each piece of information which has been input to the system. The feedback needs not be communicated through speech. Waxholm, for instance, provides a textual representation on the screen of what the system has recognised as well as of the system's output response. The important thing is that the user can perceive the feedback and verify if what the system did was what the user intended the system to do by providing a certain input. So the system's feedback may consist in presenting a particular map on the screen, or a table packed with information of some kind or other, or in playing a certain voice mail which it believes that the user has asked for. In many cases, however, the feedback will be speech produced by the system. Imagine the following dialogue:

Dialogue 1

S1: "ADAP Travels, can I help you?"

U1: "When is the first morning train from Frankfurt to Hamburg tomorrow morning?"

S2: "5.35 AM".

U2: "Many thanks. Goodbye."

S3: "Goodbye".

The conversation looks OK, doesn't it? However, the problem is that the system misrecognised 'Hamburg' as 'Hanover'! Dialogue 1, therefore, is a transaction failure. Moreover, the transaction failure is in this case a result of sloppy dialogue design. A better solution would be:

Dialogue 2

S1: "ADAP Travels, can I help you?"

U1: "When is the first morning train from Frankfurt to Hamburg tomorrow morning?"

S2: "The first train from Frankfurt to Hanover on 3rd May 1999 leaves at 5.35 AM".

U2: [Initiates repair meta-communication.]

S3: ....

The system feedback in the example above is relatively straightforward to design. One thing to consider is whether the system should repeat *twice* some or all of its core information to the user, such as in saying "... 5.35 AM, 5.35 AM". However, the feedback strategy in Dialogue 2 cannot always be trusted. What may happen is the following transaction failure:

Dialogue 3

S1: "ADAP Travels, can I help you?"

U1: "When is the first morning train from Frankfurt to Hamburg tomorrow morning?"

S2: "The first train from Frankfurt to Hanover on 3rd May 1998 leaves at 5.35 AM".

U2: "Thank you."

S3: "Do you need any other information?"

U3: ....

In Dialogue 3, the user simply ignores the system's feedback and happily walks away with what is likely to be false information, just as in Dialogue 1. If the users turn out to be too prone to make the error illustrated in Dialogue 3, the more burdensome strategy shown in Dialogue 4 might be considered:

Dialogue 4

S1: "ADAP Travels, can I help you?"

U1: "When is the first morning train from Frankfurt to Hamburg tomorrow morning?"

S2: "You want to go from Frankfurt to Hanover tomorrow morning?"

U2: [Initiates repair meta-communication.]

S3: ....

Experience has shown that the strategy illustrated in Dialogue 4 is more robust than the strategy illustrated in Dialogue 3 [7]. The price to be paid for adopting the Dialogue 4 strategy is that the user and the system have to spend more dialogue turns to solve the task. The Dialogue 4 strategy can be considered an early step in graceful degradation (see 3.19).

The four dialogues above may suffice to illustrate the subtleties that dialogue designers have to face. The amount and nature of the feedback the system should give to the user also depends on factors such as the cost and risk involved in the user-system transaction. Feedback on important bank transfers or costly travels are obviously more critical than feedback on which email the system should be reading to the user next. Current opinion probably is that the dialogue manager developer should prefer the safer among the two most relevant feedback options. Even travel information, if the user gets it wrong, can have serious consequences for that user.

For important transactions, an additional safeguard is to give the user a full summary of the agreed transaction at the end of the dialogue, preceded by a request that the user listens to it carefully. If this request is not there, the user who has already ignored crucial feedback once, may do so again. The additional challenge for the dialogue designer in this case is, of course, to decide what the system should do if the user discovers the error only when listening to the summarising feedback. One solution is that the system goes through the core information item by item asking yes/no questions of the user until the error(s) have been found and corrected, followed by summarising feedback once again.

3.20.2 Process feedback

SLDS dialogue manager developers may also consider to provide process feedback. Process feedback is meant to keep the user informed that the system is "still in the loop", i.e. that it has not gone down but is busy processing information. Otherwise, the user may, e.g., believe that the system has crashed and decide to hang up, wonder what is going on and start asking questions, or believe that the system is waiting to receive information and start inputting information which the system does not want to have. All of these user initiatives are, or can be, serious for the smooth proceeding of the dialogue.

Process feedback in SLDSs is still at an early stage. It is quite possible for today's dialogue manager designers to come up with new, ingenious ways of providing the needed feedback on what the system is up to when it does not speak to the user and is not waiting for the user to speak. The best process feedback need not be spoken words or phrases but, perhaps, grunts or ehm's, tones, melodies, or appropriate earcons. Waxholm tells the user, for instance, "I am looking for boats to Sandhamn.", thereby combining information feedback and process feedback. In addition, Waxholm uses its screen to tell the user to wait whilst the system is working.

3.21 Closing the dialogue

Depending on the task, the system's closing of the dialogue may be either a trivial matter, an unpleasant necessity, or a stage to gain increased efficiency of user-system interaction.

Closing the dialogue by saying, e.g. "Thank you. Good bye." is a trivial matter when the task has been solved and the user does not need to continue the interaction with the system. Users often hang up without waiting for the system's farewell.

In some cases, however, when the user has solved a task, the dialogue manager should be prepared for the possibility that the user may want to solve another task without interrupting the dialogue. This may warrant asking the user if the user wants to solve another task. Only if the user answers in the negative should the system close the dialogue.

Closing the dialogue is a dire necessity when the system has spent its bag of tricks to overcome repeated error loops (see 3.19) and failed, or when the system hands over the dialogue to a human operator. In the former case, the system might ask the user to try again.

History, Users, Implementation

3.22 Histories

As soon as task complexity in terms of information volume exceeds one piece of information, the dialogue manager may have to keep track of the history of the interaction. *Dialogue history* is a term which covers a number of different types of dialogue records which share the function of incrementally building a dialogue context for the dialogue manager to use or put at the disposal of the language and speech layers (see 3.7, 3.14). Note that a dialogue history is *not* a log file of the interaction but a dedicated representation serving some dialogue management purpose. Note also that a 'dialogue history' may be a record of some aspect of the entire (past) dialogue or it may be a record only of part of the dialogue, such as a record which only preserves the two most recent dialogue turns. In principle, a dialogue history may even be a record of several dialogues whether or not separated by hang-ups. This may be useful for building performance histories (see below) and might be useful for other purposes as well. It is useful to distinguish between several different types of dialogue history.

3.22.1 Task history

Most applications need a *task history,* i.e. a record of which parts of the task have been completed so far. The task history enables the system to:

- focus its output to the user on the sub-tasks which remain to be completed;
- avoid redundant interaction;
- have a global focus (cf. 3.13); and
- enable the user to selectively correct what the system has misunderstood without having to start all over with the task.

The task history does not have to preserve any other information about the preceding dialogue, such as how the user expressed certain things, or in which order the sub-tasks were resolved. If an output screen is available, the task history may be displayed to the user. If the user modifies some task parameter, such as altering the departure time, it becomes necessary to remove all dependent constraints from the task history.

3.22.2 Topic history

A *topic history* is in principle more complex than a task history. It is a record of the topics which have come up so far during the dialogue and possibly in which order they have come up. Even low-complexity systems can benefit from partial or full topic histories, for instance for detecting when a miscommunication loop has occurred (cf. 3.19) which requires the system to change its dialogue strategy, or for allowing users to do input repair arbitrarily far back into the preceding dialogue. Another thing which a topic history does is to build a context representation during dialogue, which can be much more detailed than the context built through the task history. In spoken translation systems, such as Verbmobil, the context representation provided by the topic history is necessary to constrain the system's translation task.

3.22.3 Linguistic history

A *linguistic history* builds yet another kind of context for what is currently happening in the dialogue. The linguistic history preserves the actual linguistic utterances themselves (the surface language) and their order, and is used for advanced linguistic processing purposes (see 3.14). Preserving the linguistic history helps the system interpret certain expressions in the user’s current input, such as co-references. For instance, if the user says: “I cannot come for a meeting on the Monday”, then the system may have to go back through one or more of the user’s previous utterances to find out which date ‘the Monday’ is. The Daimler-Chrysler dialogue manager and Verbmobil, for instance, use linguistic history for co-reference resolution. Compared to the task history and the topic history, a linguistic history is a relatively sophisticated thing to include into one’s dialogue manager at present.

3.22.4 Performance history

A *performance history* is rather different from any of the above histories. The system would build a performance history in order to keep track of, or spot relevant phenomena in, the users’ behaviour during dialogue. So a performance history is not about the task itself but about how the user handles the task in dialogue with the system. For instance, if the system has already had to resolve several miscommunication loops during dialogue with a particular user, it might be advisable to connect that user with a human operator rather than continue the agony. One way or another, performance histories contribute to building models of users, whether during a single dialogue or during a series of dialogues with a particular user.

3.22.5 Summary

Future systems solving high-complexity tasks are likely to include both a task history, a topic history and a linguistic history, as is the case in Verbmobil. For increased usability and adaptivity, they may need a performance history as well.

3.23 Novice and expert users, user groups

The discussion above has focused on the central importance of the task to the dialogue manager developer. However, developers also have to take a close look at the intended users of the application as part of designing the dialogue manager.

An important issue common to many different dialogue management tasks is the difference between novice and expert users. In most cases, this is a *difference continuum,* of course, rather than an either/or matter. Furthermore, it may sometimes be important to the dialogue manager developer that there are, in fact, two different distinctions between novice and expert users. In particular, someone may be an expert in the domain of the application but a novice in using the system itself. Depending on for which of four user groups (system expert/domain expert, system expert/domain novice, system novice/domain expert, system novice/domain novice) the system is to be developed, the dialogue manager may have to be designed in different ways.

3.23.1 Domain and system experts

If the target user group is domain and system experts only, the developer may be able to impose strict task performance order, a relatively large number of mandatory command keywords, etc., and support use of the system through written instructions, all of which makes the dialogue manager design much easier to do.

3.23.2 System novices

If the target group is walk-up-and-use users who can be expected to be novices in using the system, a much more user-tailored design is required.

3.23.3 Domain and system novices

The need for elaborate, user-tailored design increases even further if the system novices are also domain novices, so that any domain technicality either has to be removed or explained at an appropriate point during dialogue. For instance, even though virtually every user comes close to being a domain expert in travel time-table information, many users do not know what a ‘green departure’ is and therefore have to be told.

3.23.4 Other user groups

Depending on the task and the domain, the dialogue manager developer(s) may have to consider user groups other than novices and experts, such as the visually impaired, users speaking different languages, or users whose dialects or accents create particular problems of recognition and understanding. In the case of dialects or accents, performance history information might suggest that the dialogue manager makes use of the graceful degradation approach (cf. 3.19).

3.23.5 Mixing user groups

The ‘downside’ of doing elaborate dialogue design for walk-up-and-use users can be that (system) expert users rightly experience that their interaction with the system becomes less efficient than it might have been had the system included special shortcuts for expert interaction.

Given the relative simplicity of current SLDSs, users may quickly become (system) experts, which means that the short-cut issue is a very real one for the dialogue manager developer to consider. The Danish Dialogue System, for instance, allows (system) expert users to by-pass the system’s introduction and avoid getting definitions of green departures and the like. Waxholm allows its users to achieve the same thing through its barge-in button. The acceptance of unrestricted user input in RailTel/ARISE means that experienced users are able to succinctly provide all the necessary information in one utterance. Novice users who may be less familiar with the system’s exact information requirements, may provide some of the information needed and be guided by the system in order to provide the remaining information.

3.24 Other relevant user properties

If the task to be solved by the dialogue manager is above a certain (low) level of complexity, the dialogue manager designer is likely to need real data from user interactions with a real or simulated system in order to get the design right at design-time. Important phenomena to look for in this data include:

3.24.1 Standard goals

What are the users’ standard goal(s) in the task domain? If the users tend to have specific standard goals they want to achieve in dialogue with the system, there is a problem if the system is only being designed to help achieve some, but not all, of these goals. Strict user input control (see 3.10) may be a solution - but do not count on it to work in all possible circumstances! Deep-seated user goals can be difficult or impossible to control. Of course, another solution is to increase the task and domain coverage of the system.

3.24.2 User beliefs

Do the users tend to demonstrate that they have specific beliefs about the task and the domain, which may create communication problems? It does not matter whether these user beliefs are true or false. If they tend to be significantly present, they must be addressed in the way the dialogue is being designed.

3.24.3 User preferences

Do the users tend to have specific preferences which should be taken into account when designing the dialogue? These may be preferences with respect to, for instance, dialogue sub-task order.

3.24.4 Cognitive loads

Will the dialogue, as designed, tend to impose any strong *cognitive loads* on users during task performance? If this is the case, the design may have to be changed lest the cognitive load makes the users behave in undesirable ways during dialogue. One way to increase users' cognitive load is to ask them to remember to use specific keywords in their interaction with the system; another, to use a feedback strategy which is not sufficiently heavy-handed so that users need to concentrate harder than they are used to doing in order not to risk ignoring the feedback.

3.24.5 Response packages

Other cognitive properties of users that the dialogue manager developer should be aware of include the *response package* phenomenon. For instance, users seem to store some pieces of information together, such as "from A to B". Asking them *from where* they want to go therefore tends to elicit the entire response package. If this is the case, then the dialogue manager should make sure that the user input prediction enables the speech and language layers to process the entire response package.

3.25 Implementation issues

The issue of dialogue management can be addressed at several different levels of abstraction. In this article, we mostly ignore low-level implementation issues such as programming languages, hardware platforms, software platforms, generic software architecture, database formats, query languages, data structures which can be used in the different system modules, etc. Generic software architectures for dialogue management are still at an early stage, and low-level implementation issues can be dealt with in different ways with little to distinguish between these in terms of efficiency, adequacy etc. Good development tools would appear to be more relevant at this point. A survey of existing dialogue management tools is provided in [37].

3.25.1 Architecture and modularity

There is no standard architecture for dialogue managers, their modularity, or the information flow between modules. Current dialogue manager architectures differ, among many other things, with respect to their degree of domain and task independence. The functionality reviewed above may be implemented in any number of modules, the modularity may be completely domain and task dependent or relatively domain and task independent, and the dialogue manager may be directly communicating with virtually every other module in an SLDS or it may itself be a module which communicates only indirectly with other modules through a central processing module. As to the individual modules, Finite State Machines may be used for dialogue interaction modelling, and semantic frames may be used for task modelling. However, other approaches are possible and the ones just mentioned are among the simplest approaches.

3.25.2 Main task of the dialogue manager

If there is a central task which characterises the dialogue manager as a *manager,* it is the task of deciding how to produce appropriate output to the user in view of the dialogue context and the user's most recent input as received from the speech and language layers.

Basically, what the dialogue manager does in order to interpret user input and produce appropriate output to the user is to:

- use the knowledge of the current dialogue context and local and global focus of attention it may possess to:
- *map from the semantically significant units in the user's most recent input (if any), as conveyed by the speech and language layers, onto the sub-task(s) (if any) addressed by the user;*
- *analyse the user's specific sub-task contribution(s) (if any);*
- use the user's sub-task contribution to:
- *execute a series of preparatory actions (consistency checking, input verification, input completion, history checking, database retrieval, etc.) usually leading to:*
- *the generation of output to the user, either by the dialogue manager itself or through output language and speech layers.*

The dialogue management activities just described were discussed in Sections 3.13 - 3.21 and 3.23 - 3.24 above. The analysis of the user's specific sub-task contribution is sometimes called 'dialogue parsing' and may involve constraints from most of the elements in the speech input, language input, context and control layers in Figure 1. In order to execute one or more actions that will eventually lead to the generation of output to the user, the dialogue manager may use, e.g., an AI dynamic planning approach as in Verbmobil, a Finite State Machine for dialogue parsing as in Verbmobil, an Augmented Transition Network as in Waxholm, or, rather similarly, decide to follow a particular branch in a node-and-arc dialogue graph as in the Danish Dialogue System.

As the dialogue manager generates its output to the user, it must also:

- *change or update its representation of the current dialogue context; and*
- *generate whatever constraint-based support it may provide to the speech and language layers.*

These dialogue management activities were described in Sections 3.7, 3.10, 3.12, and 3.22 above.

At a high level of abstraction, what the dialogue manager has to do thus is to apply sets of decision - action rules, possibly complemented by statistical techniques, to get from *(context + user input)* to *(preparatory actions + output generation + context revision + speech and language layer support).* For simple tasks, this may reduce to the execution of a transformation from, e.g. (user input keywords from the speech layer) to (minimum preparatory actions + dialogue manager-based output generation including repair meta-communication) without the use of (input or output) language layers, context representations and speech and language layer support. Different dialogue managers might be represented in terms of which increased-complexity properties they add to this simple model of a dialogue manager. Waxholm, for instance, adds semantic input parsing; a statistical topic spotter ranging over input keywords; out-of-domain input spotting; two-level dialogue segmentation into topics and their individual sub-structures; preparatory actions, such as dialogue parsing including consultation of the topic history, and database operations including temporal inferencing; user-initiated repair meta-communication; a three-phased dialogue structure of introduction, main phase, and closing; an output speech layer; advanced multimodal output; and topic history updating.

3.25.3 Order of output to the user

As for the generation of output to the user, a plausible default priority ordering could be:

i. if system-initiated repair or clarification meta-communication is needed, then the system should take the initiative and produce it as a matter of priority;
ii. even if (i) is not the case, the user may have initiated meta-communication. If so, the system should respond to it;
iii. if neither (i) nor (ii) is the case, the system should respond to any contribution to domain communication that the user may have made; and
iv. then the system should take the domain initiative.

In other words, in communication with the user, meta-communication has priority over domain communication; system-initiated meta-communication has priority over user-initiated meta-communication; user domain contributions have priority over the system's taking the next step in domain communication. Note that feedback from the system may be involved at all levels (cf. 3.20). Note also that the above default priority ordering (i) through (iv) is *not* an ordering of the system states involved. As far as efficient processing by the dialogue manager is concerned, the most efficient ordering seems to be to start from the default assumption that the user has made a contribution to the domain communication. Only if this is not the case should (i), (ii) and (iv) above be considered.

3.25.4 Task and domain independence

The fact that dialogue management, as considered above, is task-oriented, does not preclude the development of (relatively) task independent and domain independent dialogue managers. Task and domain independence is always *independence in some respect or other,* and it is important to specify that respect (or those respects) in order to state a precise claim. Even then, the domain or task independence is likely to be limited or relative. For instance, a dialogue manager may be task independent with respect to some, possibly large, class of information retrieval tasks but may not be easily adapted to all kinds of information retrieval tasks, or to negotiation tasks. Modular-architecture, domain and task independent dialogue managers are highly desirable, for several reasons (cf. 3.5). For instance, such dialogue managers may integrate a selection of the dialogue management techniques described above whilst keeping the task model description and the dialogue model description as separate modules. These dialogue managers are likely to work for all tasks and domains for which this particular combination of dialogue management techniques is appropriate. The Daimler-Chrysler dialogue manager and the RailTel/ARISE dialogue manager are cases in point.

Much more could be done, however, to build increasingly general dialogue managers. To mention just a few examples, it would be extremely useful to have access to a generalised meta-communication dialogue manager component, or to a domain independent typology of dialogue acts.

4. Conclusion

Spoken language dialogue systems represent the peak of achievement in speech technologies in the 20th century and appear set to form the basis for the increasingly natural interactive systems to follow in the coming decades. This chapter has presented a first general model of the complex tasks performed by dialogue managers in state-of-the-art spoken language dialogue systems. The model is a generalisation of the theory of dialogue management in [3] and aims to support best practice in spoken language dialogue systems development and evaluation. To provide adequate context, dialogue management was situated in the context of the processing performed by the spoken language dialogue system as a whole. So far, the presented dialogue management model has been used to systematically generate a full set of criteria for dialogue manager evaluation. First results are presented in [38].

Acknowledgements

The work was carried out in the EU Esprit Long-Term Concerted Action DISC, Grant No. 24823, on Spoken Language Dialogue Systems and Components: Best practice in development and evaluation [http://www.elsnet.org/disc]. The support is gratefully acknowledged. We would also like to thank the DISC partners Jan van Kuppevelt and Uli Heid who also analysed some of the exemplar dialogue managers (see 3.1). This generated heated theoretical discussions and greatly improved our understanding of the intricacies of natural language processing in SLDSs.

5. References

[1] Fatehchand R.: Machine Recognition of Spoken Words. Advances in Computers. Academic Press 1960, 193-229.

[2] Sharman, R.: Commercial Viability will Drive Speech Research. Elsnews 8.1, 1999, 5.

[3] Bernsen, N. O., Dybkjær, H. and Dybkjær, L.: Designing Interactive Speech Systems. From First Ideas to User Testing. Springer Verlag 1998.

[4] Bossemeyer, R. W. and Schwab, E. C.: Automated alternate billing services at Ameritech: Speech recognition and the human interface. Speech Technology Magazine 5, 3, 1991, 24-30.

[5] Aust, H., Oerder, M., Seide, F. and Steinbiss, V.: The Philips automatic train timetable information system. Speech Communication 17, 1995, 249-262.

[6] Peng, J.-C. and Vital, F.: Der sprechende Fahrplan. Output 10, 1996.

[7] Sturm, J., den Os, E. and Boves, L.: Issues in Spoken Dialogue Systems: Experiences with the Dutch ARISE System. Proceedings of ESCA Workshop on Interactive Dialogue in Multi-Modal Systems. Kloster Irsee, Germany, 1999, 1-4.

[8] DARPA: Speech and Natural Language. Proceedings of a Workshop. San Mateo, CA, Morgan Kaufmann, 1989.

[9] DARPA: Speech and Natural Language. Proceedings of a Workshop held at Hidden Valley, Pennsylvania. San Mateo, CA, Morgan Kaufmann, 1990.

[10] DARPA: Speech and Natural Language. Proceedings of a Workshop. San Mateo, CA, Morgan Kaufmann, 1991.

[11] DARPA. Proceedings of the Speech and Natural Language Workshop. San Mateo, CA, Morgan Kaufmann, 1992.

[12] Iwanska, L.: Summary of the IJCAI-95 Workshop on Context in Natural Language Processing, Montreal, Canada, 1995.

[13] Gasterland, T., Godfrey, P. and Minker, J.: An Overview of Cooperative Answering. Journal of Intelligent Information Systems, 1, 1992, 123-157.

[14] Grosz, B. J. and Sidner, C. L.: Attention, Intentions, and the Structure of Discourse. Computational Linguistics 12, 3, 1986, 175-204.

[15] Heid, U., van Kuppevelt, J., Chase, L., Paroubek, P. and Lamel, L.: Working Paper on Natural Language Understanding and Generation Current Practice. DISC Deliverable D1.4, 1998.

[16] Baggia, P., Gerbino, E., Giachin, E. and Rullent, C.: Spontaneous speech phenomena in naive-user interactions. In Proceedings of TWLT8, 8th Twente Workshop on Speech and Language Engineering, Enschede, The Netherlands, 1994, 37-45.

[17] Waibel, A.: Interactive Translation of Conversational Speech. IEEE Computer, 1996, 41-48.

[18] del Galdo, E. M. and Nielsen, J.: International User Interfaces. New York: Wiley, 1996.

[19] Thomson, D. L. and Wisowaty, J. L.: User Confusion in Natural Language Services. Proceedings of ESCA Workshop on Interactive Dialogue in Multi-Modal Systems. Kloster Irsee, Germany, 1999, 189-196.

[20] Wyard, P. J. and Churcher, G. E.: The MUeSLI Multimodal 3D Retail System. Proceedings of ESCA Workshop on Interactive Dialogue in Multi-Modal Systems. Kloster Irsee, Germany, 1999, 17-20.

[21] Heisterkamp, P. and McGlashan, S.: Units of Dialogue Management: an example. Proceedings of ICSLP'96, Philadelphia, 1996, 200-203.

[22] Lamel, L., Bennacef, S., Bonneau-Maynard, H., Rosset, S. and Gauvain, J. L.: Recent Developments in Spoken Language Systems for Information Retrieval. In Proceedings of the ESCA Workshop on Spoken Dialogue Systems, Vigsø, Denmark, 1995, 17-20.

[23] den Os, E., Boves, L., Lamel. L. and Baggia, P.: Overview of the ARISE project. Proceedings of Eurospeech, Budapest, 1999, 1527-1530.

[24] Bub, T. and Schwinn, J.: Verbmobil: The Evolution of a Complex Large Speech-to-Speech Translation System. DFKI GmbH Kaiserslautern. Proceedings of ICSLP'96, Philadelphia 1996, 2371-2374.

[25] Alexandersson, J., Reithinger, N. and Maier, E.: Insights into the Dialogue Processing of Verbmobil. Proceedings of the Fifth Conference on Applied Natural Language Processing, ANLP '97, Washington, DC 1997, 33-40.

[26] Bertenstam, J. Blomberg, M., Carlson, R., Elenius, K, Granström, B., Gustafson, J., Hunnicutt, S., Högberg, J., Lindell, R., Neovius, L., de Serpa-Leitao, A., Nord, L. and Ström, N.: The Waxholm system - a progress report. Proceedings of ESCA Workshop on Spoken Dialogue Systems, Vigsø, 1995, 81-84.

[27] Carlson, R.: The Dialog Component in the Waxholm System. Proceedings of the Twente Workshop on Language Technology (TWLT11) Dialogue Management in Natural Language Systems, University of Twente, the Netherlands, 1996, 209-218.

[28] Failenschmid, K., Williams, D., Dybkjær, L. and Bernsen, N. O.: Draft Proposal on Best Practice Methods and Procedures in Human Factors. DISC Deliverable D3.6, 1999.

[29] Bernsen, N. O.: Why are Analogue Graphics and Natural Language both Needed in HCI? In F. Paterno (Ed.): Interactive Systems: Design, Specification, and Verification. Focus on Computer Graphics. Springer Verlag 1995, 235-51.

[30] Bernsen, N. O.: Towards a tool for predicting speech functionality. Speech Communication 23, 1997, 181-210.

[31] Bernsen, N. O. and Luz, S.: SMALTO: Speech Functionality Advisory Tool. DISC Deliverable D2.9, 1999.

[32] Fraser, N. M., Salmon, B. and Thomas, T.: Call Routing by Name Recognition: Field Trial Results for the Operetta(TM) System. IVTTA'96, NJ, USA, 1996.

[33] Zoltan-Ford, E.: How to get people to say and type what computers can understand. International Journal of Man-Machine Studies 34, 1991, 527-547.

[34] Amalberti, R., Carbonell, N. and Falzon, P.: User representations of computer systems in human-computer speech interaction. International Journal of Man-Machine Studies, 38, 1993, 547-566.

[35] Dybkjær, L.: CODIAL , a Tool in Support of Cooperative Dialogue Design. DISC Deliverable D2.8, 1999.

[36] Heisterkamp, P.: Ambiguity and uncertainty in spoken dialogue. In Proceedings of Eurospeech'93, Berlin, 1993, 1657-1660.

[37] Luz, S.: State-of-the-Art Survey of Dialogue Management Tools. DISC Deliverable D2.7a, 1999.

[38] Bernsen, N. O. and Dybkjær L.: Evaluation of Spoken Language Dialogue Systems. In Luperfoy, S. (Ed.): Automatic Spoken Dialogue Systems. MIT Press, 1999.