Hybrid Planning and Realization of Coherent Utterances for Multimodal Natural Language Dialogue Systems

by David Reitter

November 2004

a thesis submitted to
University College Dublin

for the degree of
Master of Science
in the Faculty of Science

based on research conducted at
MIT Media Lab Europe
and the
Department of Computer Science
University College Dublin,
Head of Department: Prof. Barry Smyth

supervised by Dr. Fred Cummins
internal evaluator: Henry McLoughlin
external evaluator: Prof. Robert Dale

Contents

1 Introduction **7**
- 1.1 Multimodal Interfaces . . . 7
  - 1.1.1 Human-human communication is multimodal . . . 7
  - 1.1.2 Human-computer interfaces lacks this kind of multimodality. . . 7
  - 1.1.3 Why? . . . 8
  - 1.1.4 A solution: adaptive mobile multimodal interfaces . . . 8
  - 1.1.5 Forms of multimodal communication . . . 10
  - 1.1.6 System components and processes addressed in this thesis . . . 11
- 1.2 Related Work . . . 12
  - 1.2.1 Static Multimedia versus Interactive Multimodality . . . 12
  - 1.2.2 Adding interaction modes to a system . . . 13
  - 1.2.3 Principled Generation . . . 14

2 System Overview **16**
- 2.1 A Virtual Personal Assistant . . . 16
- 2.2 Process flow in a dialogue system with multimodal generation . . . 17
- 2.3 Requirements for the generation task . . . 19
- 2.4 “UI on the Fly” . . . 21
  - 2.4.1 The generation process . . . 21
  - 2.4.2 Positioning UI on the Fly in Natural Language Generation . . . 24
  - 2.4.3 Requirements for the Dialogue Manager . . . 24
- 2.5 Summary: Contributions of this thesis . . . 25

3 Hard Constraints in Multimodal Functional Unification Grammar **27**
- 3.1 Tree structures in linguistic and visual interfaces . . . 28
- 3.2 Introduction to MUG . . . 29
  - 3.2.1 How grammars are used in generation . . . 29
  - 3.2.2 A restrictive blackboard architecture . . . 31
  - 3.2.3 What grammars specify . . . 32
  - 3.2.4 Attribute-value matrices in MUG . . . 34
  - 3.2.5 Designating constituents in AVMs . . . 39
  - 3.2.6 Grammar components are applied recursively to constituents . . . 40
  - 3.2.7 Structure sharing passes information . . . 43
  - 3.2.8 Grammar application algorithm . . . 46
  - 3.2.9 Functional expressions in MUG . . . 47
  - 3.2.10 Summary . . . 51

3.3 Semantic dialogue act representation in the Virtual Personal Assistant . 51
3.3.1 Types of dialogue acts in the Virtual Personal Assistant 51
3.3.2 Underspecification in the dialogue manager interface 53
3.4 The syntax of the MUG formalism 56
3.5 Conclusion 57

4 Soft Constraints: Trade-Off Decisions in a Fitness Function 59
4.1 Economy and Efficacy 60
4.2 Effort and efficacy in linguistics 62
4.3 Weighting the constraints 63
4.4 Situation profiles 64
4.5 The fitness function 65
4.6 A first evaluation of the fitness function 66
4.6.1 Experimental configuration 67
4.6.2 Results 69
4.6.3 Analysis 70
4.6.4 Conclusions 70

5 Coherence 73
5.1 Cross-modal coherence 73
5.1.1 Motivating cross-modal coherence 73
5.1.2 Examples of cross-modal coordination 76
5.2 Discourse coherence 79
5.2.1 Different aspects of discourse coherence 80
5.2.2 Centering Theory 81
5.2.3 Pronominalization rule 85
5.2.4 A parametric, evolving theory 85
5.2.5 Centering in MUG 87
5.3 Conclusion 89

6 Generation as a Constraint Optimization Problem 91
6.1 An efficient implementation 92
6.2 Formalizing the problem - the search tree 92
6.3 Depth-first backtracking search 94
6.4 Methods based on a heuristic function 95
6.4.1 Breadth first, best first and beam search 95
6.4.2 Branch & bound 96
6.4.3 Leaf ordering (local best first search) 97
6.4.4 Iterative deepening 97
6.5 Further methods 99
6.5.1 Variable reranking (or: minimum remaining values) 99
6.5.2 A* search. 100
6.5.3 Grammar compilation. 100
6.5.4 Obtaining an initial bound by choice classification. 100

6.5.5 Improving the fitness heuristic by regression. 101
6.6 Conclusion 101

7 The MUG Workbench: A Development Environment for Generation Grammars 102
7.1 Introduction 102
7.2 System overview 102
7.3 Debugging grammars 102
7.3.1 Inspecting variants of output: variants view 103
7.3.2 Tracing the steps of the generation algorithm: log view 105
7.4 Applications 105
7.5 Availability 106

8 Conclusion and Outlook 108
8.1 Content selection depends on further factors 108
8.2 Playground 109
8.3 Models based on empirical knowledge 109

Preface

There were three main factors attracting attention during the work behind this thesis.

A research lab determined to do unseen and off-beat things with multimodal human-computer interaction. *A research and development project* with European commercial, academic and charity partners who firmly planned to "tackle the holy grail of conversational speech interfaces" with some serious engineering. *And a thesis supervisor* with an active interest in phonetics, cognitive science and speech interfaces.

Me, the author, started out somewhere in between these forces, weighting the constraints announced by each of them, violating quite a few. Now, I hope to have something to offer to all of them.

My supervisor deserves my foremost thanks. Fred Cummins helped me formulate a coherent story that incorporates originally different strands of research. His guidance in balancing project and thesis work, and in finally preparing this thesis is greatly appreciated. Fred gave me the inspiration to understand the work from a greater context, which is probably what turned the process into a rewarding experience.

I would like to thank Robert Dale and Henry McLoughlin for serving as examiners of this thesis. Robert kindly invited me to spend time with his group at Macquarie University; so did Chris Schmandt at MIT, and I am grateful to both of them.

My colleagues at Media Lab Europe were Michael "Mick" Cody, Nick Hawes, John Kelleher, Eva Maguire and Erin Panttaja. I am thankful for their inspiring discussions. Jan Wielemaker and Vitor Santos Costa quickly repaired problems in the technical platforms needed to implement the generation algorithms. My colleagues in the FASiL project, including (but not limited to) Hans J.G.A. Dolfing and Kerry Robinson worked on the application context that made this work worthwhile.

This context was the Virtual Personal Assistant. Even though the system shown in this thesis ended up incorporating many more ideas, which aren't really needed by a personal information management system, the original inspiration stems from the Assistant as dialogue system with a spoken and graphical user interface. Both the European Commission and Media Lab Europe provided funding for it under the FASiL grant.

Andrea Chew made me forget my work. I am glad that happened, too.

Despite all the support I received in doing the job: any remaining mistakes are mine. One of them is already known. Jackie from Chapter 5, is indeed from South Africa. But the pets in her family are the works of phantasy.

Abstract

The output of multimodal human-computer interfaces is what this thesis is concerned with. Rather than hard-coding graphical and spoken representations, methods are introduced that plan and realize coherent output, appropriate to the situation and the device. The generation system expects a mode- and language-independent representation, as it can be supplied by the dialogue management component of a dialogue system. The generator then assembles mode-specific rendering instructions simultaneously for each mode with the aid of a unification-based functional grammar.

The approach proposed in this thesis abandons the canonical structure of pipelined planning and realization in natural language generation, in favor of hard constraints formulated in a grammar, and soft constraints that allow for the gradual adaptivity of the output. The grammar is constructed to ensure the coherence of output in different modalities, whose output is generated in a synchronized fashion rather than by separate, mode-specific generators. The soft constraints follow some of the Gricean maxims by incorporating two counteracting communicative goals: efficacy and efficiency. A fitness function encoding these goals takes into account situation- and user-specific factors, such as distractions in a single mode or the user's sensory impairments. The function leads to the selection of an appropriate output from the variety of potential outputs generated by the grammar. It is evaluated in a study with human subjects.

The thesis presents a unification based, hybrid grammar formalism which can combine pre-fabricated phrases and linguistically motivated grammar fragments, and an associated algorithm which integrates the formulation of grammars that lead to cross-modally coherent output. Methods are compared to efficiently implement a control strategy, combining hard and soft constraints as a constraint optimization problem.

The cross-modal coherence implemented by the grammar formalism is motivated by known phenomena, such as cross-modal priming, or alignment between interlocutors. To optimize discourse coherence, central ideas of Centering Theory are implemented using the grammar formalism.

Finally, novel methods and a ready-to-use implementation are introduced which allow user interface developers to inspect, maintain and extend grammars. The formalism and generation implementation is demonstrated with a grammar for a mobile, multimodal application, the Virtual Personal Assistant.

1 Introduction

1.1 Multimodal Interfaces

1.1.1 Human-human communication is multimodal

When we think about how humans communicate, speech comes to our minds. But there are more information channels that we use to transmit messages – gestures for example. The first thought might be a wildly gesticulating driver who is furious about not getting his right of way in jammed city traffic. However, gestures play a role when humans talk to each other directly, even if they are more subtle. We turn our upper bodies towards objects, when we talk about them. We raise our eyebrows in disbelief. We direct our eyes towards a corner of our field of view when we think hard. Whether that helps the thinking or not, it communicates a message. Body posture and facial expressions transmit signals, too, and they are coupled with speech.

Speech and gestures take place simultaneously or with a short delay. There is also a good deal of *coordination* on various linguistic levels between the modes. For example, deictic pronouns such as *this* are sometimes used in conjunction with pointing gestures. A recent study has found a correlation between shifts in body posture and rhetorical moves in dialogue (Cassell et al., 2001). Verbal communication between hard-of-hearing persons relies heavily on lip-observation, but even people without sensory impairment may find a phone conversation in a non-native language much harder to follow than talking to someone, when the full range of channels is available for communication. We speak of *coordinated multimodality.* In these cases, information in different modes is communicated mostly redundantly. However, given that, for example, discourse markers such as *while* or *and* can have very different rhetorical meanings, visual communication may help disambiguate speech. An ironic sentence paired with a sarcastic smile could serve as another example of multimodal communication that is complementary rather than redundant.

1.1.2 Human-computer interfaces lacks this kind of multimodality.

So, multimodal communication *per se* is no new idea in interaction among humans. One would think that human-computer interfaces would have picked up on the principle. After all, the communication modes are readily available: graphical user interfaces, lip-synced and whole-body avatars on color screens, spoken language and other sounds through speakers.

Some form of multimodality is quite common: keyboards, mice, sometimes even speech is used to interact with a desktop PC. Common output channels include the screen

and audio. However, user input is often not coordinated – it happens in sequence. The only helpful coordination of output in widespread user interfaces are *audio icons*, which are used to augment graphical user interfaces. However, unless they give simple, immediate feedback that some user input was inappropriate, these icons are purely redundant. They might catch a user's attention, but they would rarely communicate anything that is not clearly visible on the screen anyways.

1.1.3 Why?

There are several reasons for the lack of coordinated multimodality in human-computer interfaces. One is that full, complementary multimodality is not always appropriate. Consider the use of cell phones in restaurants. Cell phones are multimodal, as they can ring, vibrate, display things on the screen and just use the built-in speaker to transmit voice. If configured wrongly, full multimodal interaction is annoying to the environment (and its user). On another account, using the screen of a handheld computer (Personal Digitial Assistants) while on a bumpy bus ride is nerve-wracking, which means that multimodal devices may never rely on all modes being available to the user at all times. Other concerns include privacy, which is best exemplified by a voice-driven interface to an e-mail application, when used in a quiet, put populated train.

One solution to address these concerns would be to offer separate devices which offer interaction that is suitable to the task they are designed for, and for the usage situations. Multimodality may be configured. This is what can be found in today's cell phones, PDAs, and immobile devices such as desktop computers.

Unfortunately, this solution has its limitations, in particular in mobile situations. The screens of PDAs are too small for many tasks, in particular when it comes to inputting information. Voice-driven interfaces suffer from bad recognition, in particular in noisy environment or for large domains, for example when names are to be recognized. Generated computer-speech from a text-to-speech system (or in form of recorded prompts) tends to be lengthy and overly specific. Formulated in technical terms, the communication channels are noisy, unreliable and slow.

1.1.4 A solution: adaptive mobile multimodal interfaces

The problems of multimodal interfaces mentioned here are not a naturally given. Adaptivity is a solution that enables an interface to break through pre-defined interaction patterns. In our context, a multimodal interface may choose dynamically, which modes are used to interact with the user. Rather than just choosing one mode, this is much more of a gradual choice, changing the density of information in the channels. One means of such adaptation is to vary the length of the output. The more important information is displayed on the screen, and voice prompts are very short, if in a situation like the quiet train mentioned. It is worthwhile to remember that even a short voice prompt may improve the flow of interaction, given that the screen may not be perfectly visible in a bright environment.

Such choices are trade-off decisions. Several factors are to be considered, and among a range of possible options, the interaction is chosen that seems to be the best one in the given situation.

In short: coordinated multimodal output must respect the situation of the user.

In this thesis, I address questions in regard to dynamically generated, appropriate interfaces. While the techniques I present are not generally restricted in the modes, I focus on three particularly useful modes for the output. Firstly, computer-generated speech, to convey information intuitively, suitable for users ranging from the novice to the seasoned one. Secondly, I use natural language shown on a screen, which may help novice users. Thirdly, I use graphical user interface elements such as buttons, tables or text input field on a screen.

These modes can communicate not only much necessary information for mobile applications, they also complement each other. As mentioned, spoken input is difficult to recognize in case of a large search space, such as names from a corporate address book comprising thousands of entries, or similar-sounding names. With a screen, users can access the screen to pick the recipient of an e-mail address, but may make use of the voice mode to enter commands. In output, the pronunciation of foreign names is often difficult. A screen helps. In the case of referring to persons, e-mails or other entities relevant to the interaction, it is often less time-consuming and easier to understand, if these entities are highlighted on the screen, instead of verbally described. In contrast, it may also make sense to use redundant output – for example, in acoustically noisy environment with bright sunlight, where the computer speech is difficult to understand and the screen hard to read.

I formalize the idea of *what is appropriate* in a particular situation using basic communication principles. While we would like to convey as much information at a time, we also try to foresee how difficult it will be for our interlocutor to understand what we say. We try to be as useful and convenient as possible. Following these principles, we decide (in part) *what to include* in our message, and *how to convey it*.

The two dynamic models that influence these decisions describe the usage situation and the device properties. I formulate the models in a simple way, by looking at how useful output in a particular mode may be to the user, and how costly in terms of annoyance or privacy violation it may be.

It should be clear by now that a system must be flexible in order to adapt well. The need for flexibility has major consequences for the design process of a user interface (UI). The common design process for interfaces involving graphical UI elements (*widgets*) or natural language means that the output is handcrafted. The designer maintains fine-grained control over what the output is going to look like in the system. Flexibility is not wanted. There are reasons for this, which lie mainly in the consistency and the good user experience that is needed in a good interface.

In the approach put forward here, we still need a designer to ensure consistency and user experience. However, instead of designing the whole interface in one part, the interface is split up in several elements. Elements may consist of smaller elements. Wherever an element is used, there may be an alternative – this is where we allow the

system to make a situation-specific choice. In computer science, this design specification is called a *grammar*. The grammar is essential in *generating* output.

Where interfaces are generated on the fly, there, naturally, can be a good deal of change on the screen. For large screens, such as used in laptops, this would be highly demanding. We, in contrast, address devices that have limitations in their output channels, such as a small screen, or slow speech synthesis.[1] These *bottleneck devices* are particularly interesting candidates for adaptivity, as they allows us to work around the deficiencies of communication in the specific modes.

1.1.5 Forms of multimodal communication

Defining multimodal interaction, we need to differentiate between[2]

- *Mode*: How information is encoded – the same piece of information may be encoded in different modes. For example, the mass-medium radio uses speech and music as common modes. Many authors call this *modality*.
- *Channel*: This term refers to the different technological means used to deliver content to the user, often just for a single mode. For example, a channel can include the browsing platform, a specific navigation paradigm, a data exchange protocol such as HTTP and server-side technologies. In the radio example, the channel could consist of radio waves, a transmitter and a client-side receiver and headphones.

We can distinguish several kinds of interaction according to their use of modes:

- *Uni-modal input/output:* The system offers one predefined particular input mode and one other output mode. *Example:* Speech In / Data Out systems that use speech mode as input and text, graphics as output mode.
- *Sequential multimodal interaction:* This is the simplest level of multimodal interaction. From a set of modes, the user may either choose a particular one in a specific dialogue state (user-directed multimodality), or, there is only one mode available for input and output, respectively (system-directed multimodality). The difference compared to uni-modal interaction is that during the communication sequence as a whole, more than one input/output mode is used. Inputs from modes are interpreted separately. (W3C, 2000)

  Example: Calling a call-center, where the caller needs to enter his ID number with the keys on his phone, and then talk to an agent. Here, the mode is determined by the system. In a window-based GUI, the user can switch windows by pressing a

[1] See also Kvale et al. (2001), who points out that the difference between an uncoordinated multimodal output and a sequential multimodal output is not clearly evident when the graphical display is static (output remains visible during times when speech is played).

[2] An earlier version of this section can be found in the *Research Report on Multimodal Fission and Fusion*, FASiL-Deliverable 5.1, MLE Tech. Report. Thanks to Nathalie Richardet who co-authored the report.

key or by using the mouse. Then the system response is given in one of the output modes. There, the user may decide which mode to use.

- *Uncoordinated simultaneous multimodal interaction:* Different modes are available in the same dialogue state but they are not coordinated: on the input side, several parallel input modes are active at the same time, but only one of the input channels is interpreted (e.g. the first input). On the output side, the output is made in different modes, but not synchronized.

  Example: A voice browser in a desktop environment could accept either keyboard input or spoken input in same dialog state; this browser can then output speech and graphics in one dialog state, but the two outputs are not synchronized. (W3C, 2000)

- *Coordinated simultaneous multimodal interaction:* Different modes can be used at the same time and their processes are coordinated (input interpretation process as well as the output rendering process). Here, we distinguish redundant and complementary combinations of modes: the same piece of information may be transmitted with different modes in parallel (redundant), or, different (but compatible) pieces of information can be conveyed (complementary).

In this thesis, we strive to achieve coordinated simultaneous multimodality, both in redundant and complementary combinations.

1.1.6 System components and processes addressed in this thesis

Interaction is a two-way process. While input and output may share commonalities, the computational processes used to understand input and those used to produce output are very different.

Multimodal systems need to integrate user input made in different modes. They look for simultaneously made input that is compatible in some way. Verbal input, for instance, may leave questions open, which are answered by a pointing gesture: *Show me more details about this!* may be accompanied by selecting a file symbol on the screen. The processes around *signal fusion* are generally well understood, with the pressing questions and main error sources lying in the recognition of the mode-specific input (i.e. automatic speech recognition, detection of facial expressions or gestures).

The output process commonly involves determining what the system's next move is (the *dialogue act*), distributing the output to the various available modes (*multimodal fission*) and producing a mode-specific representation of the dialogue act (*realization*). The last step depends on the particularities of the mode used: natural language utterances are composed in a way that seems to be very different from the way gestures are combined. This idea will be revised in this thesis: I present a formalism that can realize graphical output as well as natural language sentences. As a final step in the production of a system utterance, output is *rendered*, which involves lower-level, mode- and device hardware-oriented steps which are usually handled by specific software components, such as text-to-speech systems or graphical user interface browsers.

In this thesis, I will focus on the generation of multimodal output. This comprises multimodal fission and realization. An overview of the process and how it connects to the surrounding architecture is given in Chapter 2.

1.2 Related Work

Since Bolt's (1980) Put-That-There system introduced cross-modal coordination in multimodal user input, various projects have investigated multimodal input and output methods. In this section, I will present a general overview of recent work related to multimodal dialogue systems and natural language generation (NLG).

In classifying work, I turn to factors which are highly relevant to the underlying design decisions. The task of generating static multimedia documents has different requirements than the generation of interactive system outputs: the amount of content, the type of content and its perception and, last but not least, the efficiency requirements differ. Researchers have investigated a variety of modes to be combined. The chosen mode has an impact on the applicable domains and the way output is generated. Lastly, NLG is a field in its own right, addressing many of the issues encountered here in a general way.

In the past 15 years, several projects have attempted to generate multimodal interaction that provides for parameterization according to user group and usage situation. Only some simple ways of interaction have actually made their way into commercialized end-user products. As for complementary multimodal interaction, we find touch-screens combined with audio-feedback in interfaces of bank machines or public information terminals. Redundant multimodal interaction is a common configuration option in current desktop-environment personal computers, where messages that appear on the screen can be read out to make the devices accessible for visually impaired users.

Most of the close-to-operational multimodal services available today are mainly speech-centric system allowing basic speech-in-data-out interaction. Some of these systems explore this multimodal interaction on small devices as such as the iPAQ or the Palm. Only one close-to-operational system shown here proposes a more elaborate multimodal interaction, focusing on both the problems raised by the Fusion of simultaneous inputs and multimodal Fission.

1.2.1 Static Multimedia versus Interactive Multimodality

Two projects were among the early systems that combined graphics and automatically generated text according to the semantics of the content (rather than visual properties): COMET (Feiner & McKeown, 1998, COordinated Multimedia Explanation Testbed) and WIP (André et al., 1993, Wissensbasierte Informationspräsentation / Knowledge-based Information Presentation). These systems produces technical descriptions, as needed in manuals to operate a military radio transmitter (in the DARPA-funded COMET) or an Espresso machine or a lawn mover (in the German WIP). Other than the system presented in this thesis, both systems implemented a content planning module, which de-

termined a structured description of what to convey. Planning in WIP spans from large multimodal elements of the document down to single phrases in a text. Such rhetorical planning was also employed by Bateman et al. (2001). In COMET, a *media coordinator* then took on the job of multimodal fission: it distributed the elements of the semantics it received to mode-specific generators. Fission and realization were separate, and realization was specific to the mode. In WIP, mode-specific generators and presentation planners needed to communicate bi-directionally. The system presented in this thesis avoids the modularization, so that decisions taken in during fission and realization can influence each other. Both systems used a static domain knowledge base similar to the one presented in Chapter 3.

These systems generated static documents. Dynamic user interfaces represent a new, interactive communication mode. The application usually requires real-time response times. For this reason, deterministic techniques have been predominant, for example, transduction or structured input via XSLT. SmartKom (Wahlster, 2002) and the ongoing COMIC (Moore et al., 2004, Conversational Multimodal Interaction with Computers) are examples of such system.

The SmartKom produced a platform for *interactive* multimodal systems, with a focus on the development of self-explanatory and user adaptative interfaces. SmartKom's multimodal, natural language interface combines speech, gesture, and facial expressions for input and output, both in stationary and mobile contexts. In one of the SmartKom applications, users received help in navigating a database of TV programs and setting up a digital video recorder. COMIC demonstrated a system that attempts to consult the user on the choice of bathroom tiles. The generation component in COMIC is realized as a two-step process. *Fission* distributes content across modalities and generates mode-specific markup. A natural language realizer uses both linguistic realization and canned text, with a focus on the prosody in the output as a result of information structure. Other modes include a lip-synched avatar and visual actions on a screen, showing the various forms of tiles and a simulated 3D bathroom.

1.2.2 Adding interaction modes to a system

Multimodal dialogue has been gradually evolving from graphical interfaces, which were first augmented with audio icons[3] and then with voice elements. Examples include SALT[4], which is a set of voice navigation tags to be added to existing HTML documents (and others). On an operating system level, personal computers have been enhanced with selected speech interaction, for example Apple Mac OS X, which recognizes predefined voice commands and reads alerts aloud, either to catch the user's attention or as an accessibility feature. MATIS, a Dutch train timetable information service, approaches multimodality from the opposite direction (Kvale et al., 2001). It adds graphical output and

[3] Short, redundant sounds played to signal certain interface states such as questions, warnings or error message.

[4] Speech Application Language Tags, http://www.saltforum.org

pen input to what used to be a speech-only service. The system allows non-coordinated simultaneous multimodal interactions.

The advantage of such systems is that they can be developed on top of existing interaction infrastructure. However, as I have argued before, adaptivity is an important feature for multimodal systems, in particular in mobile contexts. Therefore, dynamic generation is necessary in order to delegate presentation related decisions to the system.

MUST, MATCH and QuickSet were projects that developed demonstrators which allow for multimodal interaction. All of these systems dealt with maps in some form. MUST (Boves & den Os, 2002, Multimodal and mUltilingual services for Small mobile Terminals) provided a tourist guide to Paris, MATCH (Johnston et al., 2002, Multimodal Access To City Help) one to New York City, and QuickSet was a geographical planning system for military operations (Cohen et al., 1997).

The preference for geo-data is not incidental. Users are not naturally inclined to use multimodal input: it depends on the particular task, and map tasks were the only task where experimental subjects freely choose to interact multimodally (Oviatt, 1999).

Lately, further modes have been added in the interaction. Embedded conversational agents use humanoid avatars to visualize gestures, such as in SmartKom, or animated human faces with synchronized lip-movements, as in COMIC or FASiL[5]. Reasons for the inclusion of such modes are usually to provide a richer user experience, add credibility to the system. On the input side, SmartKom attempts to analyze facial expressions in order to disambiguate speech input. The value for practical use is not necessarily a prime objective in such systems.

In the past 15 years, several projects have attempted to generate multimodal interaction that provides for parameterization according to user group and usage situation. Only some simple ways of interaction have actually made their way into commercialized end-user products. As for complementary multimodal interaction, we find touch-screens combined with audio-feedback in interfaces of bank machines or public information terminals. Redundant multimodal interaction is a common configuration option in current desktop-environment personal computers, where messages that appear on the screen can be read out to make the devices accessible for visually impaired users.

1.2.3 Principled Generation

The mode-specific generation of natural language and graphical user interfaces has often been addressed independently of the surrounding systems. Often, natural language generation takes on the task of conveying information, which is available in structured databases, in linguistic form. These systems usually generate a discourse, i.e. a structured sequence of sentence. An example is FoG (Bourbeau et al., 1990), which produces weather forecasts.

A step towards interactivity is taken by the ILEX system (O'Donnell et al., 2001; Dale et al., 1998), which takes the discourse context into account. This system generates

[5]Flexible and Adaptive Spoken and Multimodal Interfaces: http://www.fasil.co.uk

comparative descriptions of whiskeys or jewellery. The descriptions are not static, but depend on which museum exhibits the user has already seen.

Natural Language Generation (NLG) systems differ greatly in the degree of linguistic control they allow. They can be realized as templates with a slot filler mechanism, or as the inverse of parsing with a grammar, where differentiated syntactic and lexical choices are made (Reiter & Dale, 2000).

SURGE, a generation grammar for English (Elhadad & Robin, 1998), is an example of linguistically elaborate generation, implemented in Functional Unification Formalism (Elhadad & Robin, 1992, FUF), a grammar system that allows the encoding of hierarchical generation paradigms (grammar rules).

While a grammar similar to FUF is used in the approach brought forward in this thesis, I combine fine-grained realization with templates, to allow for adaptivity, while avoiding the linguistic pitfalls or many decisions and the computational woes of a large search space.

FoG, ILEX, and of course systems implemented in FUF are grammar-driven, with decision making taking place in a planning phase, or by the application of grammar rules. Recently, work has begun to modify the mechanism in which constraints[6] are formulated and applied in a grammar. ICONOCLAST (Bouayad-Agha et al., 2000) determined layout and style and allowed users to manipulate some of the system's constraints. Other systems go further: their decision-making process is guided by constraints acquired from a corpus (Knight & Hatzivassiloglou, 1995; Langkilde & Knight, 1998; Oberlander & Brew, 2000). Sidestepping linguistic decision-making, the generator may improve the fluency of its texts by choosing an ordering of sentences that has been most prevalent in its training corpus, rather than selecting defaults for unspecified input.

When we turn to the generation of graphical user interfaces, we find systems with very similar problems and similar solutions – SUPPLE, for example (Gajos & Weld, 2004). Although this system does not learn from a corpus, it has a variety of legitimate graphical user interface designs to choose from. There is no hard and fast rule about how to decide. Rather, the effect of a proposed output on the user and the future interaction is simulated and used.

These systems have in common that the systems do not only try to *satisfy* the constraints defined by a grammar. They try to optimize their solution, since some constraints may be violated – typically called *soft constraints*. This is where the system presented in the following chapters relates to these approaches: communicative decision-making is an optimization process.

[6] A constraint is a condition that needs to hold, usually in the relationship between two information units. A simple constraint in this context would be: if two sentences express a contrast, use a discourse connective like *however*.

2 System Overview

To generate output dynamically is a task which cannot function independently of the surrounding players of such a system. In this chapter, I will introduce the particular application and then the general architecture of a multimodal dialogue system. The remainder of this thesis will deal exclusively with the act of *output generation*.

2.1 A Virtual Personal Assistant

The *Virtual Personal Assistant* (VPA) is software that allows users to manage their e-mail, their personal contact database and their appointments on a small portable device (PDA). Commonly, this application domain is called *Personal Information Management* (PIM).

The VPA is the demonstrator of a two-year research effort among a consortium of several academic and industrial entities active in natural language processing, human-computer interface research and telecommunications, and two charities contributing to technology accessibility for people with vision and hearing impairments. FASiL, *Flexible and Adaptive Spoken and Multimodal Interfaces*, developed the VPA as a mobile PIM application.Languages supported are Portuguese, Swedish and English.

What distinguishes this application from a commonplace PDA, which already offers PIM? The main difference is the voice- and GUI-driven interaction model. The interface combines natural language input and output with a touch-screen. The interface is meant to adapt not only to the usage situation, but also to the needs of the user. This implies supporting deaf and hard-of-hearing users as well as users with slight or severe vision impairments.

The main research achievements of the FASiL project lie in improvements to speech understanding and multimodal output generation, and in the production of annotated spoken language and multimodal corpora in the three languages.

The UI on the Fly system realizes an output generation module for the FASiL VPA. While it could produce output for different modes, it was developed along the requirements of the VPA application. This mainly influenced the choice of modes (voice, screen), and dictated the domain of the prototype grammar. Elements of this grammar will be used in the discussion of the underlying principles in the remained of this thesis.

In the following section, I will describe where the generation component technically fits into a multimodal dialogue system like the VPA.

2.2 Process flow in a dialogue system with multimodal generation

To discuss the interplay of various components, I turn to a simple example, which represents a short multimodal interaction between the system and its user. Let's step through part of this dialogue.

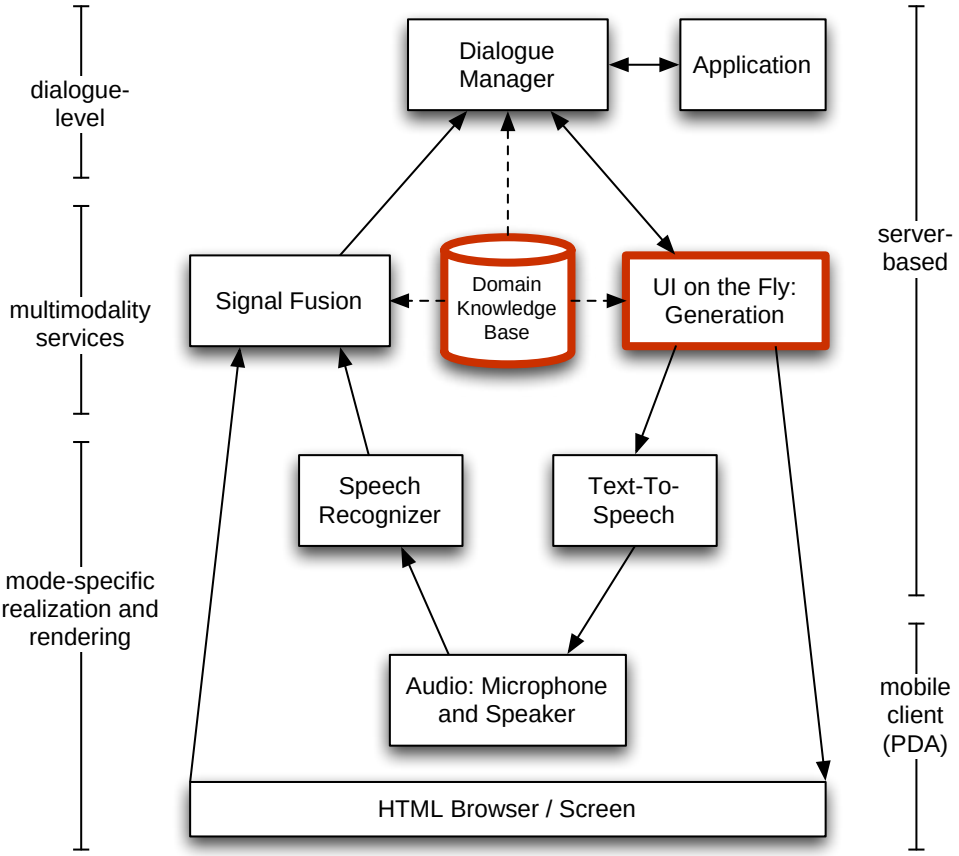


Figure 2.1: Information flow in a pipeline-based dialogue system with UI on the Fly.

User input: "Search my e-mail"

The user's speech input is recorded and processed by the first stage: an automatic speech recognition module (ASR). Usually, this module is specially tailored for the particular domain through language models. Systems differ in the way further processing is done. Some may replace pronouns and other references with descriptions of the entities

they refer to.[1] I call the module performing the transformation from an audio recording of speech to a meaning representation a *mode-specific recognizer*. Either way, a machine-readable, structured representation of the natural language input is produced. For example,

$$\begin{bmatrix} \text{command: start-search-interaction} \\ \text{database: e-mails} \end{bmatrix}$$

may describe the user input. This representation will differ in detail, however, it will be compatible input for the next stage, the dialogue manager.

The dialogue manager decides about *what to do next*. This component knows what was said before, and it knows which pieces of information need to be obtained from the user in order to execute the higher-level goal (e.g. searching for an e-mail). For instance, to search for an e-mail, we need to ask for a keyword. Such information is usually retrieved from a *knowledge base*, which encodes general, but domain-specific data, independent of the user. Another example of typical information found in a knowledge base would be that an e-mail consists of one or more recipients, a subject line, a body (text) and, optionally, some attached documents.

Usually, the dialogue manager will ask for all necessary information and then query the attached *application* to execute the task that the user has instructed it to do. An example for an application would be a database of documents (e-mails, for example) or a program that sends e-mail.

The dialogue manager will also react appropriately to faulty or misunderstood input, for example by restating the question it has asked before, in a slightly different way, or offering other help.

It should be noted that such a dialogue manager can be implemented with complete ignorance of the underlying tasks, application and the interaction modes and language (as in French, Chinese) that the human user employs. Therefore, the dialogue manager produces a language- and mode-independent representation of *what to say*, and also of the particular context. This context contains high-level goals that the user may have (the *task* of searching for something) and the intermediate action the system wants to perform (filling in a particular piece of information for the task). It also specifies the information already obtained about the task. I call the meaning (semantic) representation coming from a dialogue manager a *dialogue act*.

In our example, the dialogue management output could be a structured representation of *Ask the user for a keyword, we are in the process of searching for something, and the database we use is the e-mail database*. The next module deals with the question of *how to say it*. This is where *generation* joins the game.

Generation decides about the particular modes to be used (*multimodal fission*) and formulates a linguistic and graphical representation of the dialogue act. It takes into ac-

[1] For example, *Is it a new one?* may be replaced by a representation for *Is document number 672 a new e-mail?*

count the specifics of the language used, of the usage situation and the user's preferences. By *usage situation*, I mean the external context that that the user is in while operating the device. He or she may be in a busy restaurant, riding public transport, driving a car or simply be on his or her own at the workplace. The usage situation is detected with means external to the generation module. Various sensors such as microphones and a built-in camera may be used, and their result classified using a trained model. The details of situation detection will not be subject of further discussion.

The output of the generation module is a series of mode-specific, language-specific and situation-adapted representations, almost ready to be presented to the user. This could be a text with some prosody annotations, or HTML (Hyper-Text Markup Language) code to describe the visual display. In our example, this could be text for a short voice prompt: "Keyword?", and more elaborate text display, such as "Enter a keyword", alongside a field to allow for text entry, and a "Search" button.

The final step is again specific to the mode. Natural language, if supposed to be spoken, is processed with a text-to-speech (TTS) engine and output via speakers or headphones. Interactive graphics are usually displayed through a GUI manager (e.g. an HTML browser), which usually handles the immediate forms of interaction (for example visual feedback).

Now that the user has been presented with a question, he will make some input. For example, he will speak "Mary – and – birthday", then hit the "search" button on the screen.

This is where we enter the speech recognition (ASR) phase again. This time, however, we have additional input on the touch-screen. It needs to be analyzed as well by a mode-specific recognizer, and then combined with the speech input. This is handled by a process called *signal fusion*. The outcome of fusion is a mode-independent, language-independent representation.

Communication between the system's components is often handled with a blackboard (or multi-blackboard) architecture (as used in research systems such as VERBMOBIL (Wahlster, 2000) or SmartKom (Wahlster, 2003)), or with unidirectional pipelines. Figure 2.1 gives a broad account of how a dialogue system interacts with MUG.

2.3 Requirements for the generation task

Several requirements for the generation module can be identified.

The module generates output for single dialogue turns. There is no persistent dialogue-related information kept in the generation module. All information needed is given in the interface representation.

A central dialogue manager is in charge to determine which dialogue act should be rendered. This component is expected to be present in the dialogue system and is not described in this thesis. The generation module, however, will take individual decisions

about the details of what is output, if the original request cannot be fully honored, or if the original request is not fully specific. So, the interface between dialogue manager and generation has to support *shared responsibility*. I discuss the consequences of this in Chapter 3.

Adaptivity extends to the details of what is included in the system's output. The generator may choose to skip certain elements on the basis of external (situation-specific) factors. Adaptivity also influences the decision about which modes are used, and to what extent output is made in these modes. My approach to adaptivity is documented in Chapter 4.

System-directed and mixed-initiative systems are the prime targets for dialogue systems. A generation approach like the one described in this thesis can be used in a computational environments similar to the dialogue system described in the previous section. *System-directed* dialogue implies that the systems asks questions, the user replies. *Mixed-initiative* will allow for actions to be initiated by the user and by the system. Other systems, where the interaction is not directed at all would need to always offer a variety of interaction options to the user, in particular with menus on a screen, and adaptation would have to focus on different aspects. I do not address these latter systems.

Generation is a time-critical process, because it is one of the system components that can block the flow of user interaction. Work needs to be invested in coming up with an efficient process. I describe our methods and results in this area in Chapter 6.

Coherence is the property of dialogue (or discourse, in general) that renders it fluent. Good style in the natural language utterances and of the whole interface is vital to a positive user experience. I will address it in chapter 5.2.

Output in multiple languages will be generated. Linguistic information should be kept separate from algorithmic implementations. The grammar-based system described in Chapter 3 is language independent.

Interfaces are not developed by computational linguists. The requirement for multimodality carries a hidden consequence. Someone will need to specify the interfaces, and this person will not be the developer of the generation system. While large linguistically motivated generation systems depend on linguists for their grammars, my system provides tools that support designers and programmers. While the system presented here is far away from a graphical-based design tool, nonetheless the MUG Workbench, described in Chapter 7, is a step in this direction.

2.4 "UI on the Fly"

The generation system presented here is called *UI on the Fly*. As the name suggests, user interface (UI) is generated in real-time. This is a departure from common user interface design, where a designer decides about the details of the interaction. Because UI on the Fly is a dynamic generation system, many design choices are delegated to the system, which makes its decisions based on a variety of factors.

2.4.1 The generation process

Figure 2.2 explains where and how design choices are made in the system. The mode- and language-independent dialogue act representation introduced earlier is the starting point. First, an algorithm generates potential outputs. It does so with a *grammar*, which is a collection of small design specifications. The grammar sometimes offers many alternatives for a particular part of the output, which is why the output generation step can combine all the alternatives to many different variants of the final output. I call the grammar system *Multimodal Functional Unification Grammar*.[2]

Throughout this thesis, I will call these potential outputs *variants* – after all, they are variations of the same basic dialogue move given to the generation system.

The next step chooses one variant out of many: the one that is deemed the most appropriate one for the given usage situation and the used device. I call this step *ranking*. It is controlled by a *fitness function*, which encodes the goals that we would like to achieve. They are explained in detail in Chapter 4.

The best variant is a linguistic and visual representation of the output, which is processed and displayed by mode-specific renderers.

The devision between output generation and ranking, often called *generate & test*, is more a rhetorical move in this thesis rather than an implementation reality. As described in chapter 6, output generation and ranking are intertwined.

UI on the Fly has been instantiated for the PIM domain, with a focus on sending e-mails. Figure 2.3 shows some examples of generated output. Output a) is considered appropriate in the general situation, where both modes are available for interaction. Output b) is chosen for a case in which the user is driving a car, because we need to rely on the voice mode rather than screen output. Output c) shows a different dialogue turn, meant to disambiguate between two contacts with the same first name. The variant shown is generated for a cell phone, which explains the short screen output.

[2] The impatient reader may jump to Chapter 3 for a detailed justification as to why this grammar has such a highly scientific name

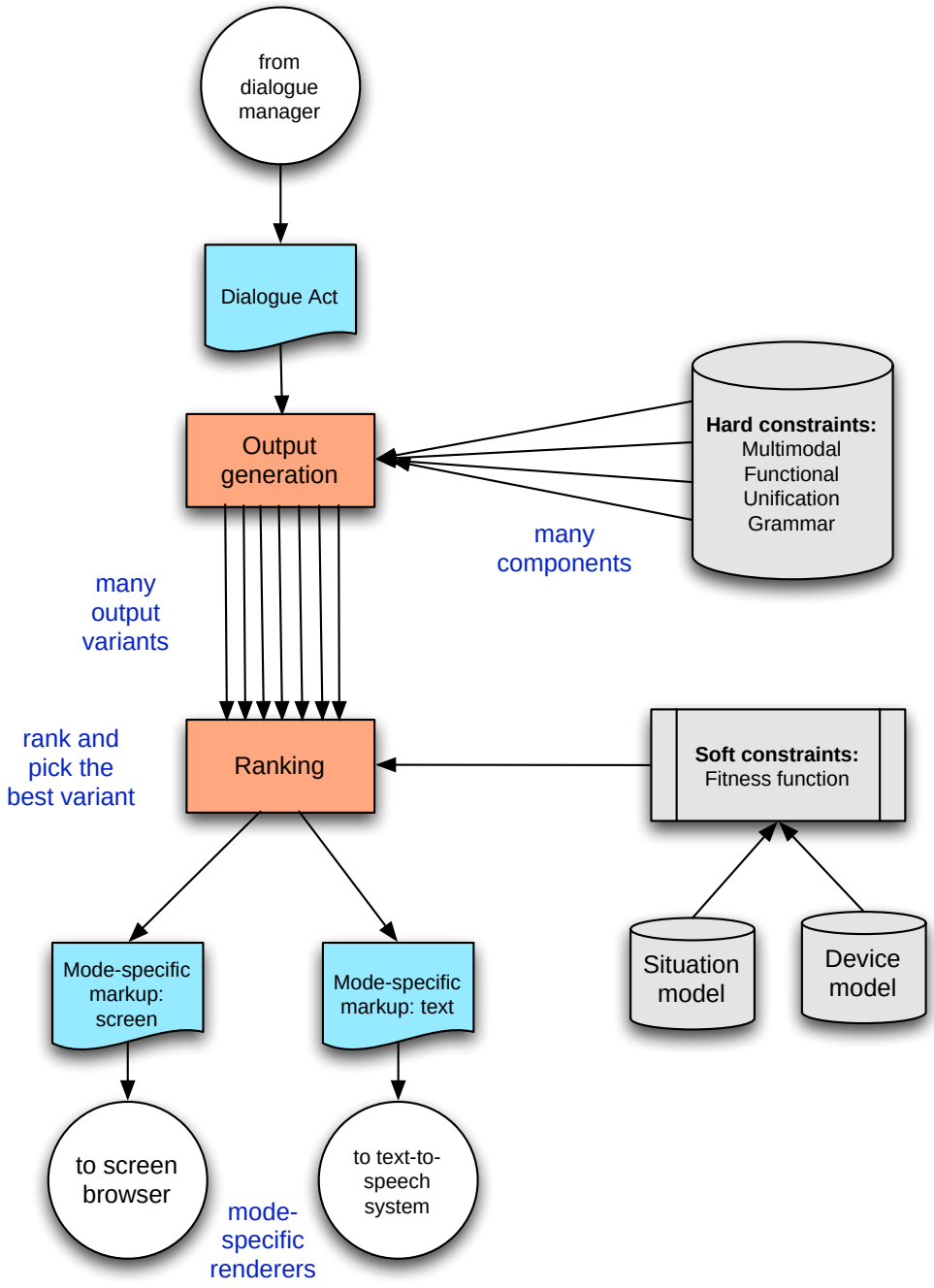


Figure 2.2: Simplified analysis of the generation process. First, multiple variants of the multimodal output are generated. Then, they are ranked using a fitness function. The best-scoring variants is then rendered.

(a) (b) (c)

Figure 2.3: Different situation- and device-adapted outputs, resulting from different dialogue acts. a) Voice: “Send the email?” (The screen output ’to the contact’ assumes the context of previous dialogue, talking about the recipient as a *contact*.) b) Voice: “Send the e-mail to Mick, yes or no?” c) Voice: “Who would you like to send the e-mail to? Jenna Templeton or Jenna Elfman?”

2.4.2 Positioning UI on the Fly in Natural Language Generation

Reiter's "consensus" architecture (Reiter, 1994) divides Natural Language Generation into three tasks:

- *Text planning and content determination.* The content of a message is chosen and organized into single propositions, which are usually realized in a tree structure reflecting the rhetorical units of the discourse.
- *Sentence Planning,* consisting of *aggregation,* where propositions are structured into clausal units, and *lexicalization,* where concepts in the knowledge base are translated into words or phrases according to a lexicon. The latter significantly influences syntactic structure, including constituent order.
- *Linguistic realization:* Morphological features are enforced, typographical schemata are realized. The final markup is generated.

Just like outlined in Section 2.4.1, the complete flow in generation assumes some form of database and a general communicative goal to start with, for example, a database with weather forecasts and the goal: *formulate a one-week forecast for Boston!*.

A common problem of natural language generation systems is that a pipeline between modules does not allow the system to revise choices taken in an early module, when the processing done by a later module shows that the output would be inappropriate. An example of this is the length of the final output. Reiter (2000) shows that pipelines with multiple outputs help, but that a revision-based architecture (i.e. without unidirectional pipelines) is superior. For this reason, UI on the Fly partially abandons the three-part approach used in many classical generation systems and outlined by Reiter. While I only implement some of the decision-making processes, it is clear that interdependencies between the modules suggest that their respective knowledge should influence each other's choices.

UI on the Fly addresses some of the aforementioned steps in a unified way: *Text planning and content determination* is mainly left to the dialogue management (DM) component, while UI on the Fly reserves the right to modify some of these decisions in case there are strong reasons to do so. My system handles *sentence planning* (for a single dialogue turn), even though there isn't much to do in the Virtual Personal Assistant domain. *Linguistic realization* is implemented fully by UI on the Fly, along with the realization of non-verbal elements, such as user interface widgets on the screen. The decision-making process that takes place in sentence planning and linguistic realization is called *micro-planning*.

2.4.3 Requirements for the Dialogue Manager

The generation approach in UI on the Fly is not suitable for all kinds of dialogue managers. An adaptive system like UI on the Fly needs information about *what to say* to

base its decisions on. Most of the data needed is fairly obvious, however. Facts about the current task are always present in the dialogue management component. Suppose a household management system is to backorder food that is missing from a user's refrigerator, the system will have a list of produce available when it asks for confirmation of the order. Or, if we are sending an e-mail message, the system will have information such as the recipients' e-mail addresses available in a structured form.

However, dialogue information is not necessarily structured: this mainly depends on the method of dialogue management. Current commercial and many dated research systems manage dialogue using finite states, for example in dialogue platforms such as the CSLU Toolkit (McTear, 1998). The state encodes information about the dialogue progress, which provides a simple, yet effective model. Usually, different states are used for short and longer prompts in a particular situations. There is typically a high number of states present. Models are easy to maintain without much linguistic or user interface know-how. However, any structured information is lost in the states. Such a dialogue system is difficult to attach to any natural language generation component, including the one developed in this thesis.

UI on the Fly expects the dialogue manager to have structured information available, such as a broad description of the task (e.g. "sending an e-mail"), a more fine-grained state giving the type of data we are manipulating (e.g. "adding an item to a list such as the recipients list of an e-mail", cf. Denecke (2000)). The dialogue manager needs to be aware of which facts about the current task have yet to be confirmed.

Such dialogue managers encode a dialogue *strategy* rather than a complete model of dialogues to expect (Denecke & Waibel, 1997; Papineni et al., 1990). Therefore, compared to state-based systems, their domains can be more easily extended and they can flexibly react to unforeseen situations.

2.5 Summary: Contributions of this thesis

This thesis makes contributions to three fields.

- *Multimodal Human-Computer Interaction*: I present a motivation and a combination of formalism and associated algorithm to address novel questions of *cross-modal coherence*. I present a reusable, grammar-based method to dynamically produce a multimodal user interface specification, adapting to a user's situation and device constraints. To my knowledge, this is the only system that addresses cross-modal coherence, adaptivity and reusability.

- *Natural Language Generation*: I present a hybrid grammar formalism which can combine pre-fabricated phrases and linguistically motivated grammar fragments. The generation system combines a grammar with soft constraints, which attempt to predict the effort a human interloctur undergoes in processing potential output, and balance it off against the benefit that the user might have in understanding the content conveyed. I introduce novel methods and a ready-to-use implemen-

tation to inspect, maintain and extend functional unification grammars. I discuss methods to optimize the search for an appropriate system output.

- *Discourse Coherence*: I present and discuss a unification-based implementation of a theory of local coherence, which can aid in the generation of such output that fits into its communicative context, i.e. is coherent.

3 Hard Constraints in Multimodal Functional Unification Grammar

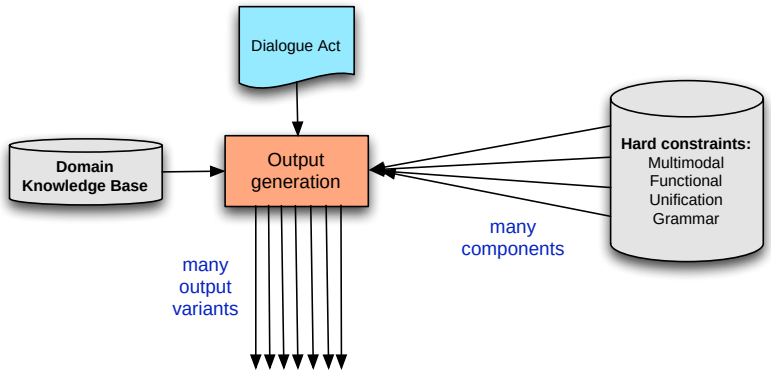


Figure 3.1: The realization process covered in this chapter.

UI on the Fly uses a central component to generate output: Multimodal Functional Unification Grammar (MUG). This is a grammar formalism that encodes the variety of possible outputs in an efficient, reusable way.

In this section, I will explain how MUG allows us to generate content. The task of the grammar is to encode hard constraints, which control the microplanning for a single dialogue act and the realization of content. In the Virtual Personal Assistant (VPA) application, the grammar is used to ensure coherence, make syntactic choices, generate referring expressions and select intra-sentential content. The grammar directly specifies how to realize the final mode-specific markup.

The key ingredients to generation are a *grammar formalism* (MUG) and an associated *grammar application algorithm*. The grammar formalism is used to encode the various possibilities to realize output in a multimodal user interface. The application algorithm tells us how to use the grammar to realize content. Both are language- and mode-independent. The formalism is based on Michael Elhadad's *Functional Unification Formalism* (FUF) (Elhadad & Robin, 1992). I will point out significant differences (and perhaps: improvements) as I explain how it works.

Grammar and application algorithm produce different alternative variants of the output. While all of them are designed to get their message across to the user, they differ in how suitable they are in the particular usage context. This is a gradual difference,

resulting from *soft constraints* specified in a *fitness function* used to score and rank each variant. This step will be dealt with later on in this thesis.

This chapter deals with the hard constraints in my generation approach. I introduce the grammar formalism and a general version of its associated generation algorithm. I present the VPA application as a an example of how the grammar can be put to use, and give examples of actual cross-modal coherence. I finish with a description of the syntax of grammar implementations.[1]

3.1 Tree structures in linguistic and visual interfaces

Anyone who has ever dealt with the linguistic analysis of sentences will know that tree structures have dominated large portions of linguistic theory. Figures 3.2[2] and 3.3 exemplify hierarchical structures on different levels.

The general idea is always the same: smaller parts of text combine, and the resulting larger text span assumes a role in the combination with other text spans. (Branches of such a tree structure are usually called *constituents*. The notion of a constituent will be further refined and applied to the context of MUG.)

Interestingly, this basic principle can be found in other representations of information as well. Graphical user interfaces often enforce a hierarchy of objects, as Figure 3.4 demonstrates. Larger constituents consist of smaller ones, and they are visually grouped. It is not surprising that the process of rendering an HTML (World Wide Web) page typically involves a tree-like specification and an even more elaborate rendering tree. The Document Object Model (DOM) is a widespread system to encode arbitraty documents. It uses tree structures.

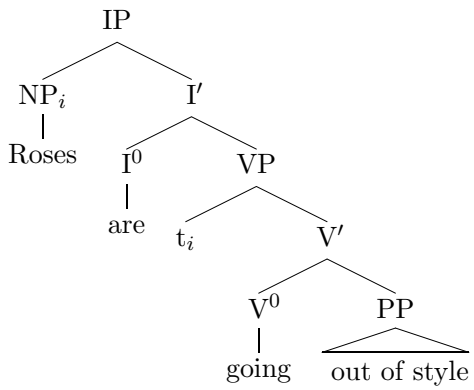
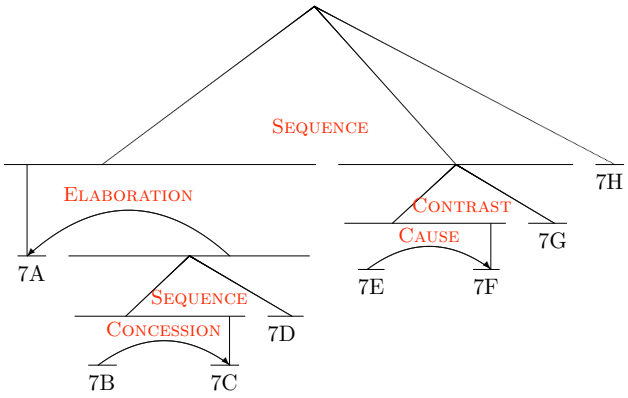


Figure 3.2: Syntax tree

[1] For a more compact account, see Reitter et al. (2004).

[2] This example and GB analysis is taken from Siskind/Dimitriadis, Documentation for qtree.



[Yesterday, the delegates chose their new representative.]^{7A} [Even though Smith received only 24 votes,]^{7B} [he accepted the election with a short speech.]^{7C} [Then the assembly applauded for three minutes.]^{7D} [Due to the upcoming caucus meeting,]^{7E} [the subsequent discussion was very short.]^{7F} [Nonetheless the most pressing questions could be resolved.]^{7G} [The meeting was closed at 7pm.]^{7H}

Figure 3.3: Rhetorical structure according to Rhetorical Structure Theory (Mann & Thompson, 1988), as analyzed in Reitter (2003a).

UI on the Fly uses tree structures to generate content, which is not at all remarkable. What is novel, however, is that there are no separate trees for the output in different modes. Instead, they are integrated. The output in the modes is structurally coordinated.

To describe which trees can be built up, UI on the Fly uses a *grammar* as mentioned before. The following sections introduce the grammar formalism.

3.2 Introduction to MUG

3.2.1 How grammars are used in generation

There are many forms of grammars in linguistics. Computer science, however, has a specific notion of a grammar. Grammars allow us to compose text that is made up of phrases. These parts of text, in turn, consist of smaller (lower-level) phrases, and so on, down to the word level (and further!). Such structures are found in languages, be it a programming language (as in *C++*, *Java*), or in a natural language (as in *French*).

Grammars contain rules, each of which *licenses* the realization of a part of the input structure. The grammar rule usually *requests* the application of further, lower-level, rules.[3]

[3]Compare this to the rules of a *production grammar*, which can look like this: $A \Rightarrow BC$. What this means, is that an element A is composed of elements B and C.

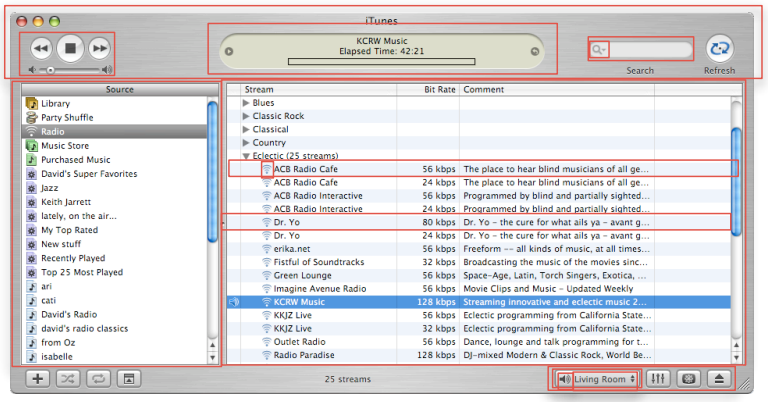


Figure 3.4: Parts of the tree structure in a contemporary graphical user interface. Some constituents are marked.

While it is beyond the scope of this thesis to introduce the general concept of grammars, two features of Multimodal Functional Unification Grammar (MUG) need to be emphasized, which distinguish it from grammars known in computer science:

MUG is a generation grammar. While grammars are usually used to *analyze* the structure of text, we use them to *produce* text.

MUG uses attribute-value matrices. Instead of atomic symbols such as A, B, C, which display no internal structure, we use richer categories. The data structure used to encode a category is an *attribute value matrix* (AVM). The structure shown in the next subsection is a first, simple example of an AVM. The following sections give an overview of AVMs. Because MUG does not use production rules, the notion of a *component* takes their place. A component is an AVM plus some additional information.[4] Unification is a function of two AVMs, which merges their information. AVMs and unification have been widely used throughout computational linguistics as a means to encode syntactic theories (most prominently, head-driven phrase structure grammar) and implement realization grammars (such as SURGE, Elhadad & Robin (1998)), as well as in multimodal applications, such as for signal fusion (Johnston, 1998). They have been discussed and examined at length. For a detailed discussion, see Carpenter (1992).

The nested attribute-value structures and unification are powerful principles that allow us to cover a broad range of planning tasks, including syntactic and lexical choices.

[4] In the following, the term *component* refers to its AVM wherever appropriate.

The declarative nature of the grammar allows us to easily add new ways to express a given semantic entity.

MUG does not use production rules. Usually, grammars encode a single rule or *production*, for example the context-free rule $A \Rightarrow BC$. What this means, is that any element of category A can be composed of elements of category B and C, or seen from a different angle, A produces B and C. Two or more components on the right hand side of the rule *produce* the one on the left hand side. In MUG, the same rule would be encoded in a different way:

$$\begin{bmatrix} \text{cat: A} \\ \text{first: } \begin{bmatrix} \text{cat: B} \end{bmatrix} \\ \text{second: } \begin{bmatrix} \text{cat: C} \end{bmatrix} \end{bmatrix}$$

In MUG, a rule encoded like this (plus some additional information) is called a *component*. The grammar consists of several components.[5]

Linear precedence is formulated explicitly in MUG. The traditional production rule given before implies that all elements expanded from B precede the elements expanded from C in the surface form of the described word (input or output). In MUG, the outputs resulting from B and C are explicitly concatenated:

$$\begin{bmatrix} \text{cat: A} \\ \text{text: concat}(\boxed{\text{T1}}, \boxed{\text{T2}}) \\ \text{first: } \begin{bmatrix} \text{cat: B} \\ \text{text: } \boxed{\text{T1}} \end{bmatrix} \\ \text{second: } \begin{bmatrix} \text{cat: C} \\ \text{text: } \boxed{\text{T2}} \end{bmatrix} \end{bmatrix}$$

3.2.2 A restrictive blackboard architecture

There are two mainstream architectures for systems that plan and realize natural dialogue. The *pipeline* model assumes a serial ordering of jobs, each carried out by a different module: for instance, text planning and content selection, then sentence planning, then linguistic realization. There are idiosyncratic interface representations between the modules, and while a single module may output more than one solution (usually as an n-best list), there is usually no backtracking across module boundaries. Target constraints can only be optimized in a single module. The *blackboard* architecture uses a central

[5] Note that the use of `cat` attributes here (`cat` stands for *category*) is simplified for illustrational purposes.

data exchange repository for all modules, which operate in parallel. This way, modules may add or revise information step by step (Galaxy Hub: Seneff et al. (1998), Verbmobil: Wahlster (2000)[6]). Synchronization problems and efficiency considerations are issues using the blackboard architecture.

Functional Unification Grammars make use of a restrictive blackboard architecture. That means that a single logic data structure as a central data repository is used. I call this data structure *the blackboard*. It is augmented during the generation process; information is monotonically added until a variant is generated (Figure 3.5). After the generation of one variant is finished, steps are undone on the blackboard, and different components from the grammar are chosen to generate another variant.

Even though we deal with a central blackboard that the grammar components can modify, the single components take only local scope: the constraints that the components state may only access the information space within a particular portion (a substructure) of the blackboard. Therefore, the grammar is *compositional* in Frege's sense (Frege, 1892).

Substructures are part of the large piece of information stored on on the blackboard. **A substructure that is to be or has been extended further with a grammar component is called a *constituent***. The application operation, which combines the copy of a component together with a constituent is called *unification*. It adds information to the constituent on the blackboard. Unification, constituents and components will be defined more formally subsequently (see Section 3.2.5).

3.2.3 What grammars specify

The grammar encodes partial user interface elements, groups of such elements or natural language phrases as *components*. A component specifies a set of constraints which define the cases in which it would apply, and also which daughter components are called. For example, a component could apply to all cases where we are giving out some pieces of information to the user. It could call further daughter components to realize the subject (e.g. *The email from Tony*) and the predicate of a sentence (e.g. *has been deleted*). It can state that the two text strings are concatenated as the result of the operation.

In contrast to a production grammar with rules such as $S' \rightarrow NP_{Subj}\ VP$, MUG uses only a single attribute-value matrix for each component to encode constraints and results of the operation. (See Sections 3.2.4, 3.2.6).

The components define constraints operating on the information represented in structures they combine with. The combination process is called *unification*. It ensures, that all constraints defined in the component are fulfilled, when it is applied to a constituent structure. More details about unification are given in the next section. However, it is useful to note that higher-level components encode the relationship between the information in the input and the lower-level components. Therefore, we can speak of a tree structure not unlike the one resulting from a production grammar. The constraints given

[6] SmartKom, (Wahlster, 2002), and COMIC (Moore et al., 2004) use the Verbmobil hub.

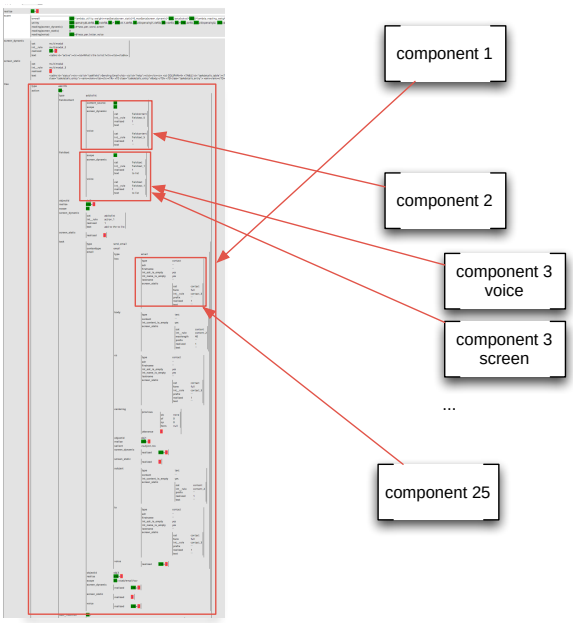


Figure 3.5: Schematic diagram showing that components (on the right) unify destructively with constituents on the blackboard (left). Where multiple components are unified with the same constituent, they realize output for the same meaning (described by the constituent) in different modes.

in the components allow grammar writers to place detailed restrictions on the cases when a component can be used to realize a portion of the original semantic input structure.

As in many grammars, there can be several competing components that can realize a portion of the input. This ambiguity leads to variety of realization options for the complete input. The logic of declaratively stated constraints in unification grammars has a valuable advantage. A declarative grammar separates the planning algorithm, formulated as set of soft and hard constraints, from the grammar application algorithm, which is a constraint satisfaction and optimization problem. In other words, the grammar does not depend on a specific order of execution. It only expects that the specified constraints are fulfilled.

In the following, I will motivate the use of AVMs, describe the application domain of my examples, describe grammars and the formalism used to encode grammars before detailing grammars further.

3.2.4 Attribute-value matrices in MUG

We have seen that components encode constraints about which other components they combine with. To specify these constraints, MUG leverages the computational and model-theoretical advantages of *unification grammars*. These are based on data-structures called *attribute-value matrices* (AVM, or: feature structures), which encode and structure information, and an operation, *unification*, which merges AVMs, if they are *unifiable*.

AVMs represent a generic class of data structures, which are useful to represent constraints in a declarative formalism. From input to planning and realization, to the final output structure, the grammar uses AVMs, which makes it easier to understand. Because of AVMs, components are more reusable. Another advantage of the unification-based framework is that it forces us to clearly specify the scope of information. Local information allows for compositional construction. The declarative nature of the grammar allows us to easily add new ways to express a given semantic entity. The information that each component has access to is explicitly encapsulated by an AVM. The nested attribute-value structures and unification are powerful principles that allow us to cover a broad range of planning tasks, including syntactic and lexical choices.

The downside of unification grammars is that it is difficult to encode procedurally formulated constraints, for example constraints that search the best of several substructures according to a certain criterion. Syntactic theories such as HPSG suggest that such constraints don't regularly exist. Leaving the world of syntax, however, we find that theories of discourse coherence (as discussed later in this thesis), demand a ranking of AVM substructures that cannot be provided with plain unification.

I will defer the problems of particular applications until later and define AVMs in more detail at this point. An AVM describes linguistic data by means of a structure listing pieces of information. Each piece of information is characterized by an attribute and a value. Formally, we define an AVM as a set of attribute-value pairs, where each attribute is an atomic symbol and unique in the set. Each value can be atomic, a numeric value, a string, a list of values or an AVM. So:

$$\text{AVM}(A) \& < a, v >, < a, v' > \in A \rightarrow v = v'$$

If the AVM is *typed*, only attributes declared in its type may be present in the set. Their values must be subsumed by the type that was declared for the given attribute. A simple example demonstrates this:

$$\begin{bmatrix} \text{type: person} \\ \text{firstname: 'Tony'} \\ \text{lastname: 'Hawk'} \\ \text{age: 34} \end{bmatrix}$$

Suppose we define a type named *person*, as shown in Figure 3.6. This type declares five

attributes for the type: `firstname`, `lastname`, `age`, `birthplace` and `location`. The values associated with the two names are of type *string*, the one associated with `age` is an *integer*, and the two places are of type *place*.

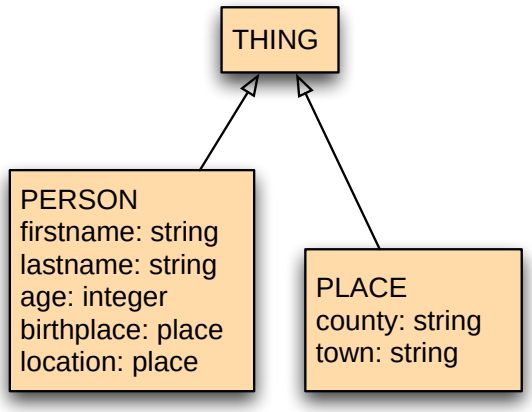


Figure 3.6: A simple type hierarchy with three types *person*, *location* and *thing*.

Recursion: Values used in AVMs may be more complex than shown in the previous example. Very often, they are AVMs themselves, in which case they are called *substructures*. Typical structures become relatively large. A small example for an AVM with a substructure in the the `location` attribute is

$$\begin{bmatrix} \text{type: person} \\ \text{firstname: 'Tony'} \\ \text{lastname: 'Hawk'} \\ \text{age: 34} \\ \text{location: } \begin{bmatrix} \text{zip: 92075} \end{bmatrix} \end{bmatrix}$$

AVMs are recursive. In order to identify a specific value within the AVM, an *attribute path* (or short: path) can be used. It gives a sequence of attributes, which uniquely identifies a position in such a nested AVM. For example, `location|zip` identifies the attribute whose value is 92075.[7]

Substructures are used to separate data. This is an advantage, as grammar components typically only deal with one substructure at a time, independently of each other. Where dependencies are wanted, data is explicitly shared across the structure with *variables*.

[7] In Chapter 5.2, the denotation val(`location|zip`) is used to refer directly to the value in formulas.

Up to now, AVMs bear a remarkable ressemblance with objects used in object-oriented formalisms. The difference to a, say, *Java* object lies in *under-specification*, in *structure sharing* and in an operation called *unification*. They will be explained in the following.

Structure sharing: Values used in AVMs may be shared. This means that they carry the same value, if they are eventually assigned one. If the value is not atomic, this sharing relationship persists. Variables, denoted in $\boxed{\text{boxes}}$, indicate structure sharing. For example, we can represent all people who live in the same town they were born in like this:

$$\begin{bmatrix} \text{type: person} \\ \text{birthplace: } \left[\text{town: } \boxed{1}\right] \\ \text{location: } \left[\text{town: } \boxed{1}\right] \end{bmatrix}$$

By using variable binding in grammar rules, grammar writers can specify equality of any set of values that occur in an AVM or in any of its substructures. If two values are bound to the same variable, they share their value rather than containing a copy of each others value. Usually, not just a single atomic symbol is shared, but whole sub-structures. For this reason, we often speak of *structure sharing*. The notion of structure sharing is realized in modern programming languages in form of *references*. Where a variable is used in the AVM notation, Java would use a reference to an object. This way, sharing is persistent and efficient.

The formal view of an AVM needs to be augmented to accomodate structure sharing. In addition to the set of attribute-value pairs, an AVM therefore contains a set of sets of attribute paths. The paths in each set point to substructures in the AVM sharing information.

Under-specification: Not all attributes declared must be assigned values. Thus, an AVM represents a set of items. For example, we can describe all 34-year-old people whose name is *Tony*:

$$\begin{bmatrix} \text{type: person} \\ \text{firstname: `Tony'} \\ \text{age: 34} \end{bmatrix}$$

If an attribute is missing, we say that is value is not *instantiated*. The same expression applies when an attribute is present, its value however is undefined and only bound via structure sharing to some other value. In this case, we also say that the variable is not instantiated.

Extension: Given an attribute-value matrix A, it can be *extended* by either adding an attribute-value pair to A, by instantiating existing variables to atomic symbols or other AVMs, or by extending the value in any of its attribute-value pairs.

Unification is a simple operation defined over pairs of AVMs (operator symbol: $\cup$). It builds on the fact that AVMs can be under-specified. Unification combines the information contained in both structures if they are consistent. Otherwise, unification fails. For example, we can unify the aforementioned *Tony* structure with one that represents all people from San Diego:

$$\begin{bmatrix} \text{type: person} \\ \text{firstname: 'Tony'} \\ \text{age: 34} \\ \text{location: } \begin{bmatrix} \text{county: 'San Diego'} \end{bmatrix} \end{bmatrix}$$
$$= \begin{bmatrix} \text{type: person} \\ \text{firstname: 'Tony'} \\ \text{age: 34} \end{bmatrix} \cup \begin{bmatrix} \text{type: person} \\ \text{location: } \begin{bmatrix} \text{county: 'San Diego'} \end{bmatrix} \end{bmatrix}$$

Note that unification only instantiates some values in the structures. It cannot change an atomic value, e.g. it could not change the value of the `age` attribute to 35. This is analogous to other logic formalisms, where variables are either unknown, instantiated or partially instantiated or constrained in some way.

Formally, the relation *unifiable* $\upsilon(A, B)$ holds between two attribute-value matrices A and B if there exists another feature structure which extends both A and B. The unification of two feature structures, $A \cup B$ is defined as the minimal attribute value matrix that extends both A and B. Unification is commutative:

$$A \cup B = B \cup A$$

$$\upsilon(A, B) \leftrightarrow \upsilon(B, A)$$

and it is associative:

$$(A \cup B) \cup C = A \cup (B \cup C)$$
$$\upsilon(\upsilon(A, B), C) \leftrightarrow \upsilon(A, \upsilon(B, C))$$

Consistency: In contrast to an object-oriented programming language, the grammar may only extend existing AVMs. It may not alter values in another way or remove attribute-value pairs from an AVM. Thus, AVMs monotonically become more specific.

Types: Like other object-oriented formalisms, unification-based grammars need to trade off the safety of strongly typed classes, where types are enforced, and the ease

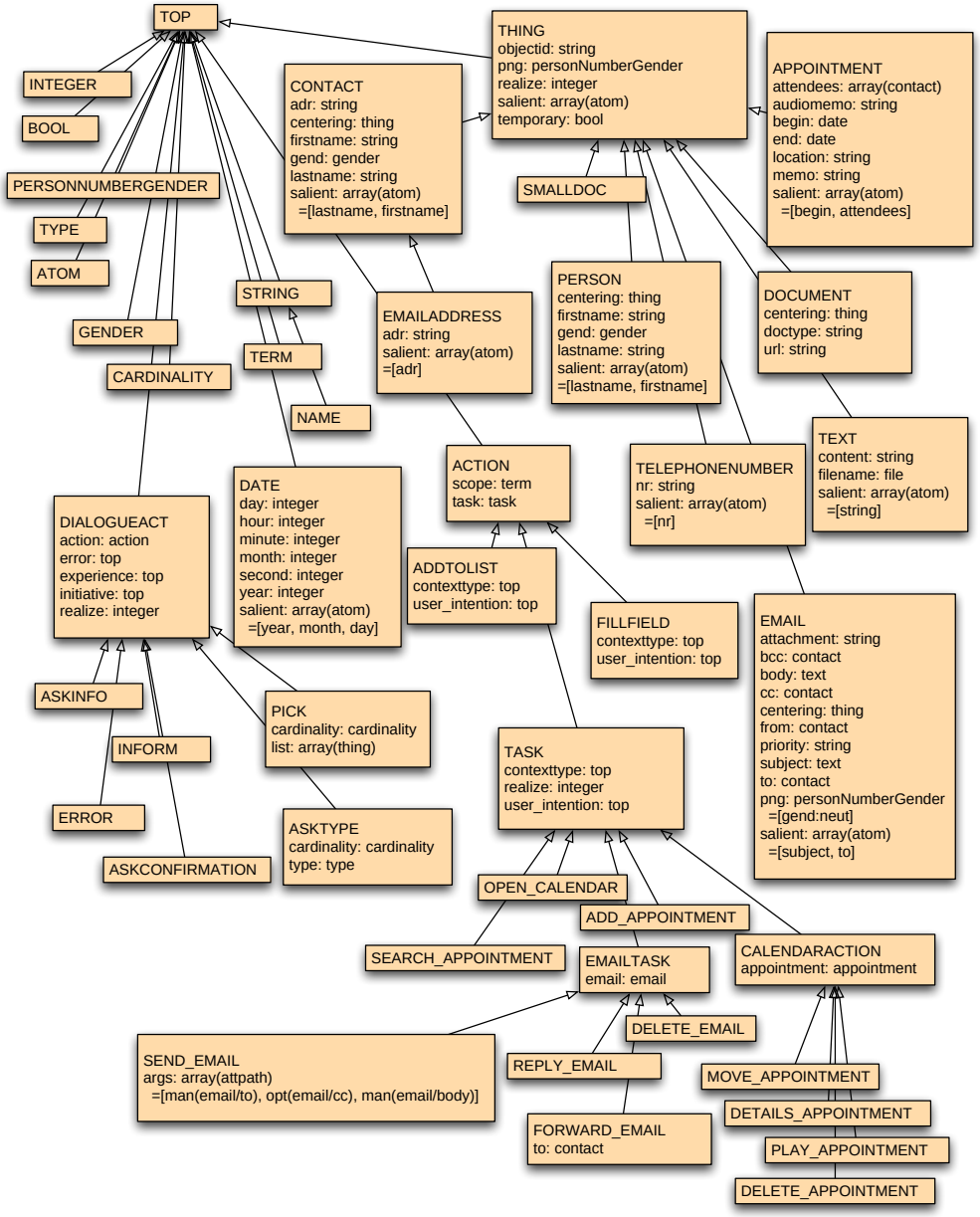


Figure 3.7: Knowledge base for the Virtual Personal Assistant

and prototyping advantages of free types. In common formalisms such as Head-Driven Phrase Structure Grammar or grammar platforms such as Carpenter's *Attribute Logic Engine*, feature structures are typed. Types are commonly defined in a hierarchical ontology, whereas types inherit features from their (subsuming) parent types. Each feature structure (that inclues every substructure) needs to have a type. Type declarations, however, put an additional hurdle in the way of developers. The issue is addressed with a type hierarchy that is used to issue non-fatal warnings in case of possibly ill-formed substructures in dialogue act input and grammar rules.

MUG uses *weakly typed* AVMs. It allows for types, but does not enforce them. That means, a structure may have a type. If it has, the MUG implementation will check the features used against those declared in the type hierarchy and issue a warning if there is a mismatch.

The type hierarchy may also be consulted by MUG functions (see Section 3.2.9), for example to realize a subsuming definite description in a disambiguation situation: *Which of these* contacts *would you like to send the e-mail to?*. In the unification based, late semantic fusion approach implemented for the Virtual Personal Assistant (VPA), the same ontology is used.

Figure 3.7 depicts the types used in the VPA domain.

The MUG formalism is a syntax to practically specify the grammars. It is described in Section 3.4. In the following discussion of how components combine in the grammar, the unification process plays a dominant role.

3.2.5 Designating constituents in AVMs

The following discussion explains the algorithm that forms the basis of the grammar formalism. Section 3.2.8 provides a concise account for the algorithm as pseudo-code, which may or may not be easier to understand than the plain-text description.

Recall that a Multimodal Functional Unification Grammar consists of a set of *components*. These components are analogous to the rules of a production grammar, and what follows here, is a discussion of the idea of declarative grammars, as they are instantiated in the grammar formalism MUG.

Each of the grammar components specifies a realization variant for a given partial semantic or syntactic representation. It does so elegantly in a single AVM, which is used to expand information on the blackboard. Initially, the blackboard contains the input, a dialogue act. Iteratively, components are drawn from the grammar to be unified with constituents on the blackboard. This adds information to the blackboard, until nothing is left to expand and all parts of the input are realized.

To explain which substructures on the blackboard are expanded, we need to define *constituents* formally. A substructure on the blackboard can be designated as a constituent for a specific mode m through an attribute called `cat` (for category). I call such a substructure an *m-constituent* (or short *constituent*). Values of m could, for example, be `screen` or `voice`, or the attribute could simply be represented as a variable Mode. The basic template for a constituent is

$$\begin{bmatrix} (...) \\ m \left[\text{cat } \textit{category}\right] \end{bmatrix}$$

To make such a structure a constituent, *category* must assume a value (in unification terms, we say it must be *instantiated*). This value specifies the constituent's category. Categories often play a linguistic role: e.g. *nominal phrase* or *sentence*. For MUG as a formalism, however, the value bears no further role beyond the fact that if the `cat` attribute is present and associated with some value, the surrounding structure becomes a constituent.

On the blackboard, m is always instantiated with one of the available modes. However, the same structure may be a `screen-`constituent and a `voice-`constituent at the same time, or just either of them. If it is a constituent for both modes, it will have two mode attributes, as shown here:

$$\begin{bmatrix} (...) \\ \text{screen} \left[\text{cat } \textbf{buttons}\right] \\ \text{voice} \left[\text{cat } \textbf{enumeration}\right] \end{bmatrix}$$

Any substructure on the blackboard can be designated (marked) as constituent. This is usually done by grammar components, when they are unified with larger constituents. Therefore, `cat` attributes are present in many components. Figure 3.5 illustrates this process schematically. Here, constituents on the blackboard (left) are marked with a red frame. The largest one is expand with component 1. This component designates three substructures as smaller constituents. Components 2 and 25 expand two of them, respectively and each of them is applied twice: once for each mode. These components work for all modes, and they would have a $\boxed{\text{Mode}}$ variable in their AVM instead of `voice` or `screen`. The third one (in the middle) is expanded by component 2 for the *voice* mode and by component 3 for the *screen mode*.

Figure 3.9 shows a component to illustrate the designation of constituents in detail. The component itself is of type PICK, so it could be used to expand constituents of this type, as the next subsection explains. The component designates several substructures as m-constituents. One of these substructures is highlighted in the figure: this constituent is of category DISAMBIGUATE. The other constituent that is designated by the component in the figure is of category UI-CHOOSEONE.

3.2.6 Grammar components are applied recursively to constituents

Now that we have defined what constituents are on the blackboard, we can describe more closely what happens to them. The short answer is that each m-constituent is expanded with the copy of a suitable component. Such a suitable component is one that unifies with the constituent, and for which all additional constraints (if any) are fulfilled.

Such constraints may state that the component only applies for a specific mode m. If a component applies to any mode, we use the variable [Mode] as an attribute name. An simple example of this is given in Figure 3.8, a more complicated one follows in Figure 3.9.

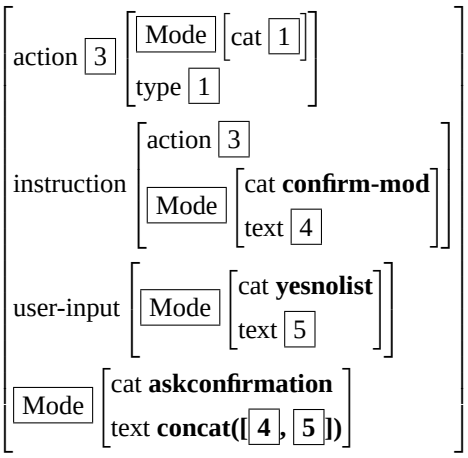


Figure 3.8: A MUG component that handles the confirmation of tasks or user input. The mode given in variable [Mode] may be *voice* or *screen*. Category and Type in the *action* branch are unified, which means that for each type that may occur in action, the grammar will provide a lexical form.

After the expansion is completed (i.e. the unification is performed), the result is written directly to the blackboard, replacing the original constituent. And because grammar components designate further structures as constituents, there is more to be expanded – the algorithm is recursive.

With this in mind, one could now give a natural language account the ongoings in Figure 3.9, the following would be a start:

> In any output mode, we can produce output for a PICK situation using the following ingredients: a) the output for an instruction and b) the output for some user-interaction field. The resulting texts from a) and b) are simply concatenated to form the end result. To realize the instruction, we invoke, for the mode that the whole component is called, a component of category DISAMBIGUATE, giving it the original quantity (how many elements does the user select?) and the current action (what is done with the element that the user picks?). We also give it the type of element, that is supposed to be picked. (...)

Three AVMs shown in this chapter exemplify some of the steps carried out. The component in Figure 3.10 is used to expand the highlighted constituent in Figure 3.9,

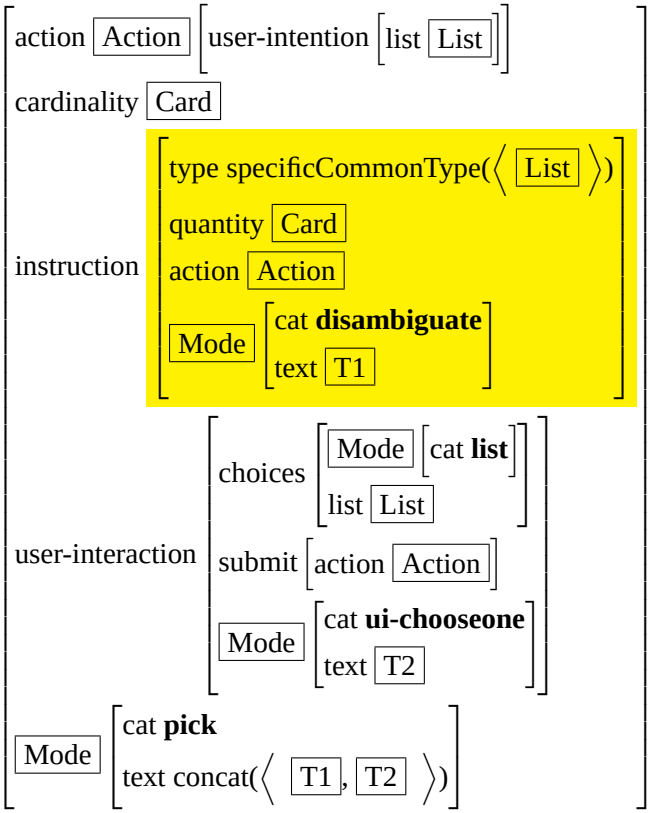


Figure 3.9: Component of category PICK illustrating the use of structure sharing and the notion of a constituent. The marked substructure is one of the constituents. It will be expanded with component of category DISAMBIGUATE, one of which is shown in Figure 3.10.

resulting in the AVM shown in Figure 3.11. Here, the original dialogue act is ignored and not contained in the AVMs shown. It could be unified with the large structure at a later time: the order of unifications does not matter, as unification is an associative operation.

Each component deals with a separate semantic entity

Constituents on the blackboard often stem from part of the original dialogue act that was put on the blackboard in the beginning. A different use case might provide a good example here: A substructure describing a task such as *delete-email* may be a screen-constituent. It may contain a further substructure, describing which e-mail is to be deleted. This substructure may be a constituent, too. In other cases, constituents may

not stem from the original dialogue act, but be completely produced by components (by way of unification).

In this example, there may be a grammar component to translate the task *delete-email* into natural language, for example by saying: *remove*. This component may, in turn, instantiate the `cat` attribute of the email substructure, forcing it to be expanded by another grammar component. The result *John's e-mail* is then concatenated: *remove John's e-mail*.

Figure 3.8 gives an example of another component. This one concatenates a confirmation prompt with a a list offering *yes* and *no* choices. The YESNOLIST is realized by other components, in order to distinguish between the different modes here. On the screen, buttons would appear, while the speech output would simply contain the question *Yes or no?*, or simply nothing at all, in which case the YESNOLIST component would realize an empty string. The upper component shown in the figure is not concerned with the choices on the lower level.

So, this declarative grammar can be ambiguous: there may be several competing components of the same categories in the grammar, which are all unifiable with some constituent. This ambiguity is needed to generate a variety of outputs from the same input. Each output will still be faithful to the original input. However, only one variant will be optimally adapted to the given situation, user, and device (see Chapter 4).

Marking substructures as constituents is the way for components to designate parts of their own AVM for further expansion through the grammar. This designation (for a mode m) takes place by instantiating the special attribute mentioned earlier, `cat`, which occurs within an AVM that is a value associated with another special attribute named after m. In other words, whenever a substructure has an attribute path `m|cat` with an instantiated value, it is a constituent.[8] (In an AVM, the variable m is denoted as Mode. The two writing variations are equivalent.)

Components are grouped in *categories*, which are used to index them for efficient retrieval. The value of the `cat` attribute of a constituent gives the component that the constituent can be expanded with. In Figure 3.9, a constituent whose category is DISAMBIGUATE is marked. But apart from indexation, component categories do not add to or restrict the expressiveness of the formalism.

3.2.7 Structure sharing passes information

The structure sharing mentioned before provides an essential mechanism to pass data between the different components involved in realizing an utterance. As mentioned previously, variables are denoted as numbers or names, placed in boxes: 1, Mode.[9]

Already shown in Figure 3.8, structure sharing is also evident in Figure 3.9. This component has a range of variables, among them T1, T2.

[8] The designation of constituents with the `cat` feature takes the place of FUF's (Elhadad & Robin, 1992) `cset` attribute, which FUF uses to explicitly give a list of constituents.

[9] The name of a variable is of no importance to the grammar. In this thesis, numbers are used just to simplify reading.

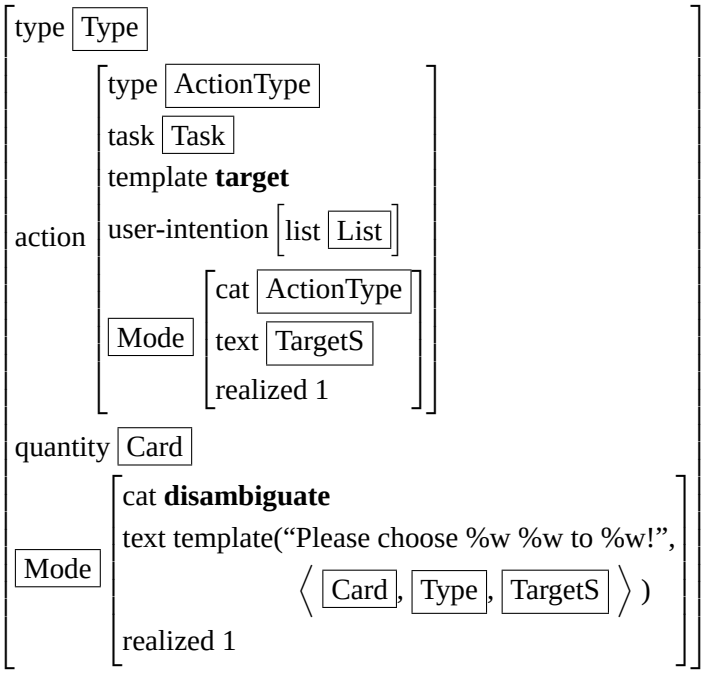


Figure 3.10: Component of category DISAMBIGUATE. A more complex one, similar to this, is used in the *Virtual Personal Assistant* application.

T1 is used in attribute path `instruction|Mode|text` and in the *concat* function in path `Mode|text`. It states that the values in positions where T1 occurs, must always be the same. This applies to T2 similarly. The function *concat* simply concatenates the texts in its list argument. It is described in Section 3.2.9.

Recall that the mode is often replaced by a variable Mode to generalize a component, which can plan or realize output in all modes. The value of the Mode variable can only be instantiated to one of the modes.

Note that in Figure 3.11, the variable Action is used for structure sharing, while, in attribute path `instruction|action`, the value of this variable (an AVM) is shown at the same time. The same happens for the variables Type and T1. This notational convention (attribute, variable, value) is known from other unification-based formalisms.

Structure sharing is essential for the grammar. It is used to pass down information from the higher levels (semantic dialogue act) to lower-level components that generate parts of possibly mode-specific output. If a grammar rule states that two sub-structures are shared, this means that they will be persistently shared: they are synchronized. This means that they are not copied, and sharing does not come with a price tag: there is no computational (or space-related) expense for shared structures.

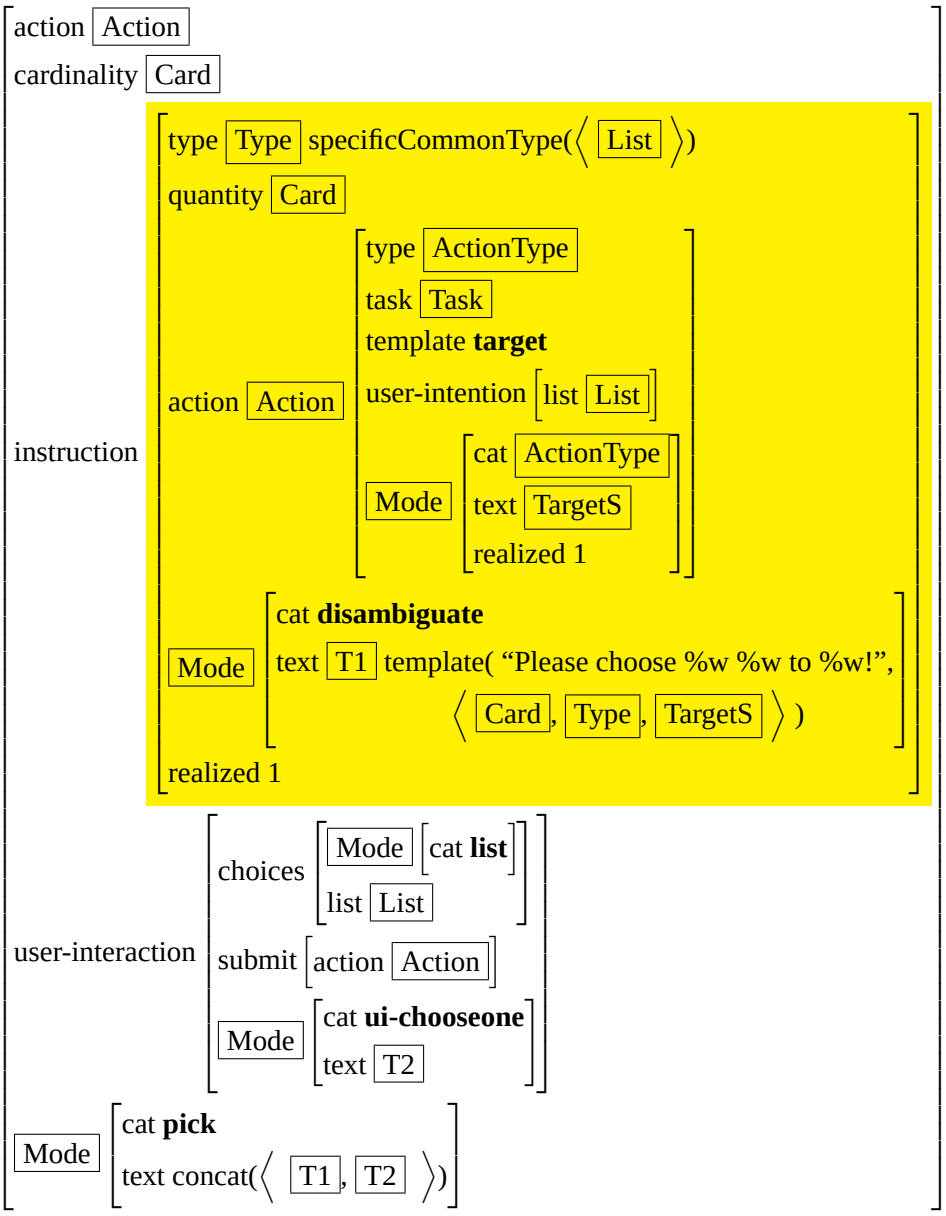


Figure 3.11: When the DISAMBIGUATE component from Figure 3.10 is applied to the appropriate constituent (marked in Figure 3.9), this structure results. The structure here has not been unified with the dialogue act input, to illustrate the variable bindings and shorten the representation. If it were, variables such as Card and Action would be instantiated. The variables in the text template are instantiated by further components (not shown here).

3.2.8 Grammar application algorithm

The dialogue manager is assumed to provide a dialogue act representation as AVM DialogueAct. The input structure is then framed, so that it is designated as a constituent of category *multimodal* for all modes. Here, it is assumed that two modes (screen and voice) are known to the system (Figure 3.12 shows the general scheme, Figures 3.15 and 3.16 an example before and after the framing.

$$\begin{bmatrix} \text{screen} \left[\text{cat } \textbf{multimodal}\right] \\ \text{voice} \left[\text{cat } \textbf{multimodal}\right] \\ \text{tree } \boxed{\text{DialogueAct}} \end{bmatrix}$$

Figure 3.12: The frame used to initiate the grammar application process. DialogueAct is replaced with the input dialogue act.

The input structure is then copied to the blackboard und the grammar application begins. The central application algorithm ensures two principles:

1. All components that realize a particular constituent (i.e. semantic entity!) must unify with each other and the entity representation.

2. All m-constituents must be realized, i.e. unified with a component for mode m from the grammar. Components may designate further constituents, thereby calling for more (lower-level) components to be instantiated.

Principle (1) ensures cross-modal coherence via the restrictions put in place by the components. Components that are cross-modally incoherent should not unify, therefore their combination cannot be used by the grammar application algorithm. This extends Elhadad's uni-modal FUF. (See Chapter 5.2 for a motivation of cross-modal coherence.)

Principle (2) allows the components to cause recursion. As mentioned before, constituents are designated by an instantiated `cat` attribute within the mode-specific branch of a substructure. This branch is always the one within an attribute that is named after a mode.

Unification is destructive: it updates the blackboard. Note that there may be competing grammar components, which creates choice points. The algorithm is non-deterministic and produces, for example via backtracking, a set of final structures. Unifications are undone if another solution is explored.

Figure 3.13 shows the algorithm that is applied to the blackboard in order to expand constituents recursively. APPLY essentially identifies all un-expanded constituents on the blackboard (F) and calls EXPAND to expand each of them. Expansion can, through unification, designate further constituents on the blackboard. The previous sections in this chapter have given a plain text description of what is happening here. The

APPLY(F) :

Repeat until no more calls to EXPAND can be made: For each mode $m \in \mathbb{M}$
and for each substructure F' in F
if there is an attribute-value pair $< m, s > \in F'$,
if there is an attribute-value pair $<$ cat, $c > \in s$ (for some category c)
and no attribute-value pair $<$ expanded, true $> \in s$,
$F' \leftarrow$ EXPAND(F').

EXPAND(X):
Select a component C from $\mathbb{G}$
for which all constraints C_C are fulfilled and
for which $\upsilon(C_F, X)$ holds (unifiable)
$X \leftarrow X \cup C_F$
$X \leftarrow X \cup \{<$ expanded, true $>\}$ (and update F)
return X

Figure 3.13: The application algorithm for MUG grammars.

attribute-value pair <expanded, true> has been left out, as it is used only as a bookkeeping mechanism to ensure that the algorithm does not expand anything twice.

By means of structure sharing and instantiation of variables, the grammar provides a string or a functional expression returning a string for each mode in the paths $m|text$, which can be read out.

APPLY cannot be applied to be plain dialogue act that is given as input. We first need to mark the whole structure as a constituent for all modes (*framing*). The constituent category is always MULTIMODAL, i.e. components of this category are needed in the grammar to serve as an entry point. Such an entry component is given as example in Section 3.4.

3.2.9 Functional expressions in MUG

After the grammar application, functional expressions are evaluated. A simple example for such an expression is *concat(['the ', 'person'])*, which evaluates so *'the person'*.

Functional expressions extend the power of MUG components. They are terms used as feature values. Expressions are evaluated *after* the grammar has been applied. The evaluation process itself may trigger further grammar application for some functions.

Why these expressions, if we already have functional components? one might ask. The reason for their existence is that most of the functions can only be evaluated once all or some their arguments are instantiated. For MUG components, their order of application is not guaranteed: it is not generally guaranteed that one component is unified

with substructures on the blackboard before another one.[10] For functional expressions, it is guaranteed that they are evaluated after the grammar application. Before an expression is evaluated, all of its arguments are evaluated.

The second important argument for the functions is that their implementation may specify any algorithm – it would be illusionary to think that a practical generation/realization grammar can ressort to purely unification-based methods. The string manipulation functions (capitalization, concatenation and truncation/summarization) as outlined below are prime examples for this. Consequently, functional expressions are known in other unification formalisms, with *concat* being usually present (Pollard & Sag, 1994). The implementation of the MUG engine allows grammar writers to implement further functions. These functions are listed in the following.

capitalize(Text) Return a copy of Text with the first letter capitalized.

concat(List) Return a string containing the concatenation of all objects in List. List may only contain strings.

foreach(List, Template, OutTemplate) A copy of each element from List is unified with Template. For each of the objects, OutTemplate is instantiated and added to the return list. Template and OutTemplate are both AVMs, and they should share at least part of their structures through variables. Foreach applies the grammar application algorithm to its result, so that the templates may designate new constituents. This is used to realize, for example, a list of referring expressions to contacts all in the same way.

multiply(List, Template) Return a list, which contains a copy of each element from List, unified with a copy of Template. The grammar is applied to the result.

template(Template, List) Template is a string containing normal text and placeholders of the form %w. For each placeholder, the next element from List is inserted in the text. The following example is from the *Virtual Personal Assistant* grammar, which implements the DISAMBIGUATE component and its daughters in a more complex way than shown in previous figures:[11]

[10] The logic of unification grammars implies that each feature path is the equivalent of a constraint. All constraints hold at the same time for the data that is described by a feature structure. Therefore, there is no order in which the constraints are checked.

[11] For a description of the syntax, see Section 3.4

$$\begin{bmatrix} \text{template } \textbf{disambiguate} \\ \text{version } \textbf{long} \\ \text{determiner } \textbf{one} \\ \text{action } \begin{bmatrix} \text{task } \textbf{send-email} \\ \text{field } \textbf{to} \end{bmatrix} \\ \text{type-np } \textbf{contact} \\ \boxed{\text{Mode}} \begin{bmatrix} \text{cat } \textbf{template-mod} \\ \text{text template}(\text{ “Please choose \%w \%w to \%w.”}, \langle \text{ ‘one’, ‘contact’, ‘send the email to’ } \rangle) \end{bmatrix} \end{bmatrix}$$

will be evaluated to:

$$\begin{bmatrix} \text{template } \textbf{disambiguate} \\ \text{version } \textbf{long} \\ \text{determiner } \textbf{one} \\ \text{action } \begin{bmatrix} \text{task } \textbf{send-email} \\ \text{field } \textbf{to} \end{bmatrix} \\ \text{type-np } \textbf{contact} \\ \boxed{\text{Mode}} \begin{bmatrix} \text{cat } \textbf{template-mod} \\ \text{text “Please choose one contact to send the email to.”} \end{bmatrix} \end{bmatrix}$$

Of course, the arguments to the template functions are variables in real-life MUG components. For example, the following three MUG components show how a string

can be output in both available modes, or optionally left empty. They copy the common *String* and a mode-specific prefix *P* into the template.

```
component(realize_string, screen, AVM, string_mod_1) :-

    AVM === [       string: String,
                    screen: [
                                cat: realize_string,
                                realized: 1,
                                prefix: P,
                                text: template('~w<P>~w</P>',
                                               [P,String])
                            ]
             ].

component(realize_string, voice, AVM, string_mod_2) :-
    AVM === [   string: String,
                voice: [
                        cat: realize_string,
                        realized: 1,
                        prefix: P,
                        text: template('<PROMPT xml:lang=
                                       "en-GB">~w ~w
                                       </PROMPT>',[P,String])
                       ]
           ].

component(realize_string, Mode, AVM, string_mod_empty) :-
    AVM === [   string: String,
                voice: [
                        cat: realize_string,
                        realized: 0,
                        text: ''
                       ]
           ].
```

trim(Text) Return a copy of text with leading spaces removed.

unifyEach(ListOfAVMs, AVM) For each element of ListOfAVMs, make a copy of AVM and unify the copy with the element. The elements in ListOfFDs will be changed as a side-effect of this operation, AVM is not affected.

specificCommonType(ListOfAVMs) Returns the most specific type that is common to all elements of the list ListOfAVMs. The knowledge base (see Figure 3.7) is queried to derive information about the types. The result, for example, is *Contact* if ListOfAVMs is a list containing elements of types *Contact* and *Emailaddress*.

summarize(Length, Text) Returns a summary of the Text, approximately Length characters long. Return Text if the text is shorter than Length. (How the summary is done is not defined; the current implementation, however, takes words from the beginning and then end and puts three dots in between.)

3.2.10 Summary

To sum up the steps introduces in this chapter, the general generation process for MUG grammars is defined as follows:

1. Augment the input representation (Figures 3.14, 3.15 and Section 3.3.1) with default values and unique object-ids defined in the knowledge base (cf. Appendix 3.7). This step typically adds information that is type-specific – for example, salience information about typical ways to identify an object of a certain type, as shown in Figure 3.15[12]. The object-ids may be used to keep references to a database (cf. KPML, Matthiessen & Bateman (1991)).
2. encapsulate the dialogue act representation in a feature structure, which calls the component MULTIMODAL for each mode.
3. apply the grammar (Section 3.2.8)
4. evaluate functional expressions (Section 3.2.9)
5. for each mode m, read mode-specific text (or markup) from feature paths.

3.3 Semantic dialogue act representation in the Virtual Personal Assistant

3.3.1 Types of dialogue acts in the Virtual Personal Assistant

Although the semantic input is independent of mode (as in screen, voice) and language (as in English), the input semantics are domain-specific. I will use the VPA system as a case study of such a representation.

The input identifies a general *dialogue act*. For the VPA domain, the following dialogue acts were used.

- ASKCONFIRMATION: The systems asks to validate or confirm certain data, either regarding a whole task or just some single slot that is to be filled to achieve the task. The scope of the confirmation is given in a *scope* attribute. For example, such a dialogue act is represented by the sentence *Send the e-mail now to Michael Bennett?*

[12] For example, an e-mail is typically referred to by realizing its *subject line* attribute and, secondly, its sender or recipient.

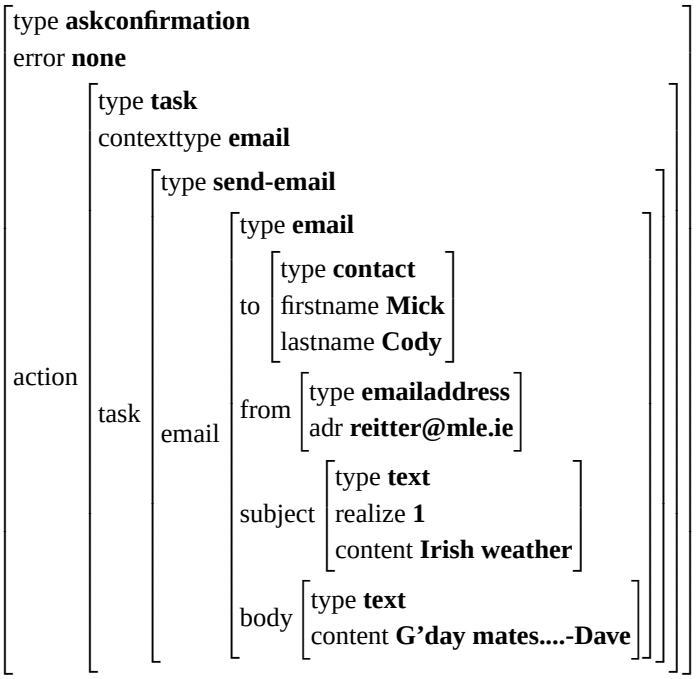


Figure 3.14: Input representation: confirmation of sending of an email

- ASKINFO: The system asks for information, such as the recipient of an e-mail address or a string to be searched for. For example, *To whom should the e-mail be sent?*
- PICK: A choice needs to be made from a set, as it occurs in the disambiguation of input. For example, *Do you mean Jenny Cleary or Jenny Langley?*
- INFORM: Convey information to the user, such as *the e-mail has been sent*.

The input structure, an example of which is shown in 3.14 is compositional. As opposed to common semantics formalisms, the representation does not employ explicit quantifiers (such as *every, some* or *most*), as they were not found to be needed in the personal-information management tasks that were implemented. The input-AVM taken from the *Virtual Personal Assistant* and shown in Figure 3.14 specifies the type of act in progress (askconfirmation), and the details of the interaction type. It then specifies the details of the current action, in this case, the email that the user is sending.

3.3.2 Underspecification in the dialogue manager interface

The overall specification of the dialogue act is understood to be mandatory for the generation component. However, generation may decide to leave out some details of the dialogue act – possibly, because their realization turns out to be too long to be spoken or to fit on the screen. Typically, this can have adverse effects on the dialogue: certain information would never be conveyed. Therefore, a mechanism was put in place to allow the dialogue manager to direct the sentence planning task of including and excluding semantic detail.

An attribute `realize` allows the dialogue manager to indicate the need to realize a certain portion of the meaning, as shown in Figure 3.14. The generation component is not obliged to obey this attribute, which is why it is evaluated in conjunction with a soft constraint. This `realize` attribute is compared, during scoring, with the actual mode-specific realization: if a semantic object was not realized even though the dialogue manager asked for it, the variant score will incur a penalty. Take for example Figure 3.14, path `action|task|email|subject|realize`, whose value of 1 indicates that the e-mail's subject line *Irish weather* should be realized in the output. The recipient, however, does not carry such an attribute: it is underspecified. Therefore, realization is fully optional and only carries a default benefit.

Fully mandatory realization specifications are not used in the input. However, grammar components often use a `realized` attribute to force the realization of certain lower-level constituents, usually in syntactic contexts that mandate full realization. This attribute could be instantiated in the input structure as well. The `realized` attribute is specific to the mode, so it is always included in the entity in an attribute path `m|realized`. The attribute makes the semantics *recoverable*, feeding back to the dialogue management component and informing it about the portions of the dialogue that ended up being realized in the output.

Input to the system may be underspecified, mainly in terms of the `realize` attribute, which may be missing completely.[13] This should be seen as the default case, where planning decisions are left to the generation system. However, in many stages during an interaction, the dialogue manager will see the need to confirm portions of the task information. In particular, this will be information that was input by the user in spoken form where the recognition reliability is limited. Such information may be confirmed "on the fly", i.e. in a prompt asking for the next input. For example does *Please enter the subject of the e-mail to Tim Morgan!* confirm the recipient of the e-mail elegantly and efficiently. A dialogue system with hard-coded output and dialogue strategies would have to prompt *To Tim Morgan, please say no if this isn't right* and wait a few seconds, before moving on the *Please enter the subject line!*. The flexible, underspecified input format eliminates the need for these prompts.

[13] As a side remark, Langkilde & Knight (1998) go further in their NITROGEN system: even semantic attributes such as `plural` may be underspecified, so that the generation system chooses an appropriate variant. In NITROGEN, appropriateness is defined according to statistical data extracted from a corpus. In MUG, it is defined according to a fitness function.

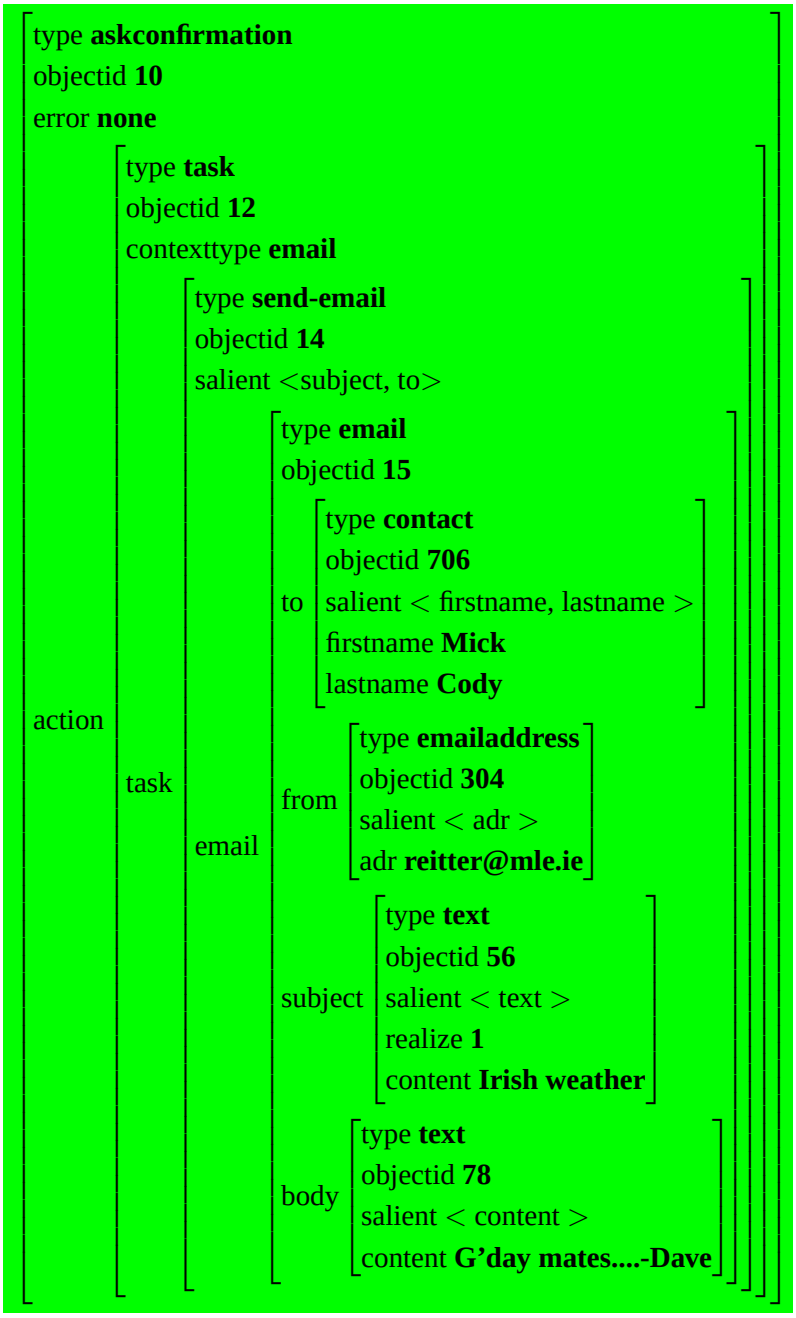


Figure 3.15: Input AVM from Figure 3.14 after augmentation with unique object IDs and default information from the knowledge base.

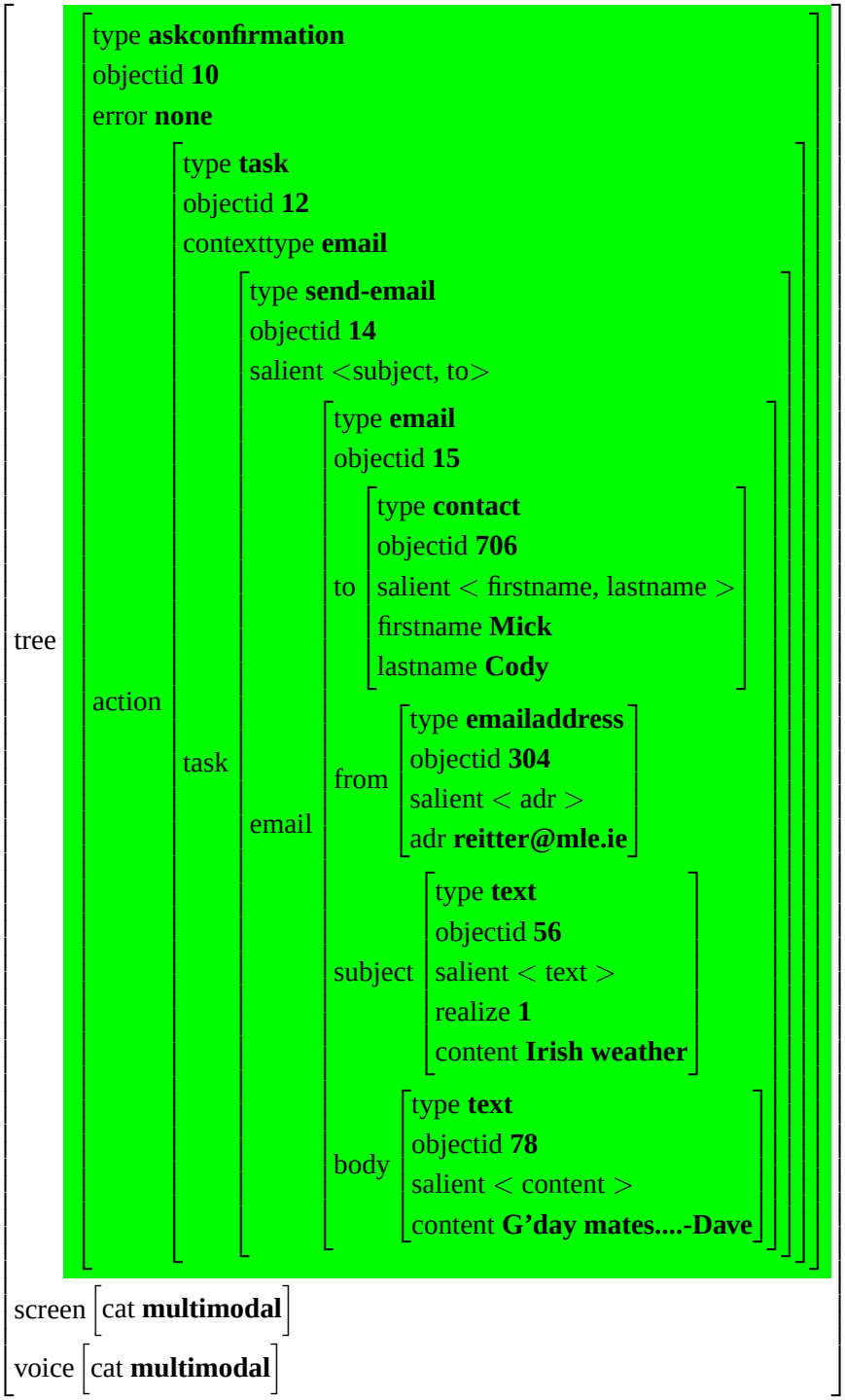


Figure 3.16: Input AVM as shown in Figure 3.15, but after framing, before it is unified with three MULTIMODAL components, one for each mode. The highlighted portion stems from the input dialogue act. With the framing completed, the whole structure is marked as a constituent now.

3.4 The syntax of the MUG formalism

The graphical representation used in this chapter has a computer-readable equivalent. This section gives a brief account of the syntax used to specify MUG components.

The MUG specification language is close to Prolog syntax. One or more grammar files contain components, which are usually made up of one big AVM (additional disjunctive or conjunctive unifications are allowed). Variables can be named and always start with an upper-case letter. The grammar writer is allowed to add detailed comments about components or their parts. The grammar formalism implements structure sharing within the AVM and other constraints associated with a component by using named variables.

In the file containing the MUG each component is specified according the following syntax:

```
component(<ComponentCategory>, Mode, AVM, <ComponentName>) :-

          Mode=<Mode>,                % optional
          AVM === <FunctionalDescription>
          , <additional constraint>
          (...)
          .
```

`AVM` stands for the AVM of the component. Note that the component must be finished with a period. <`ComponentCategory`> is redundant, because it stands for the value of the mode-specific `cat` attribute, which is already present in the AVM. It is noted for indexation and better readability. <`Mode`> is the mode for which the component is written (e.g. screen_dynamic or voice). The line must be left out if the component is mode-independent. <`ComponentName`> represents a unique identifier of the component, which is used in debugging grammars. Note that these data do not add to the expressivity of the grammar, with the exception of the Mode constraint. However, additional constraints formulated in Prolog may operate on variables. (In practice, negated constraints and also constraints pertaining to a global device-profile are introduced in this way.)

In a MUG component, all capitalized words represent variables. `Mode` for example is a variable, which may (and should) be used in the AVM. Variables are also used for structure sharing (see below).

An example for a valid component specification is:

```
component(multimodal, Mode, AVM, multimodal_1) :-
    AVM === [
              tree:[ Mode: [ text: ModeRep ] ],
              Mode:[ cat:multimodal, text: ModeRep ]
            ].
```

In fact, this is the component that is present in every MUG it is the initial component from which realization starts. Figure 3.16 shows the MUG representation of the structure shown passed on to this component.

Further constraints may be added to components, such as constraints about the instantiation state of a variable (*given(X)*). These can be used to constrain the applicability of a component to cases where the input semantics specify (or leave out) certain information. For example, we may specify that a contact object is only realized using the e-mail address field, if a name for the contact is not available.

In the following, a formal account of the syntax is given:

An AVM is denoted with square brackets `[ ]`. It contains a comma-separated list of $1..n$ attribute-value pairs in the form of `Attribute:Value`. Each Attribute must be an atom, which means, it must start with a lower case letter (a-z). It can also be a variable (in certain circumstances).

Each value can be either atomic or an AVM. If it is atomic, it may be

- a string, enclosed in single quotation marks (`'...'`), or an atom. Such an atom is a single word beginning with a lower-case letter (`a-z`).
- an integer or a floating-point number (e.g. `65`),
- a comma-separated list of values, which has to be enclosed in square brackets (`[...]`).
- A variable, which is expressed as a single word that begins with an upper-case letter (A-Z).

The order of the attribute-value pairs in an AVM is not significant. Lists and AVMs are discerned by the fact that AVMs contain only attribute-value pairs and lists contain only values.

3.5 Conclusion

MUG implements a formalism for ambiguous, unification-based grammars. It is compositional, as grammar components depend only on other components that they invoke. The ingredients to this formalism have been described in this chapter, and an examples of the practical implementation of the *Virtual Personal Assistant* application have been given.

Compared to the VPA implementation, the figures in this chapter are abbreviated. Snapshots of the full attribute value matrices on the blackboard would not fit on a single page here. This fact makes it clear that MUG deals with very large structures on the blackboard (as do similar formalisms). Essentially, MUG stores all information or constraints collected during the course of realization. Nothing is deleted. While this sometimes makes it difficult to describe and understand a particular information state on the MUG blackboard, it represents no problem for an actual implementation which does not copy information unless needed. During grammar development, appropriate filtering mechanisms need to be employed in order to keep an overview over this turmoil of data structures (see Chapter 7).

What is novel about this grammar is the departure from mode-specific generators as used in other multimodal dialogue systems. Here, we realize the output for the different modes in parallel. We enforce that realization components, that realize the same meaning in different modes, need to be compatible: they need to unify. A motivation of this is given in a subsequent chapter on Coherence (that is, cross-modal coherence, in Chapter 5). With the grammar, we have stated the *hard constraints*, which always need to be fulfilled in a particular realization. Often, dialogue systems encode as many hard constraints as possible to be enforced at an early stage, as to reduce search space. MUG, in contrast, leaves some choices to *soft constraints*. To apply the soft constraints, we need to define them (next Chapter) and also define a suitable control strategy (Chapter 6).

4 Soft Constraints: Trade-Off Decisions in a Fitness Function

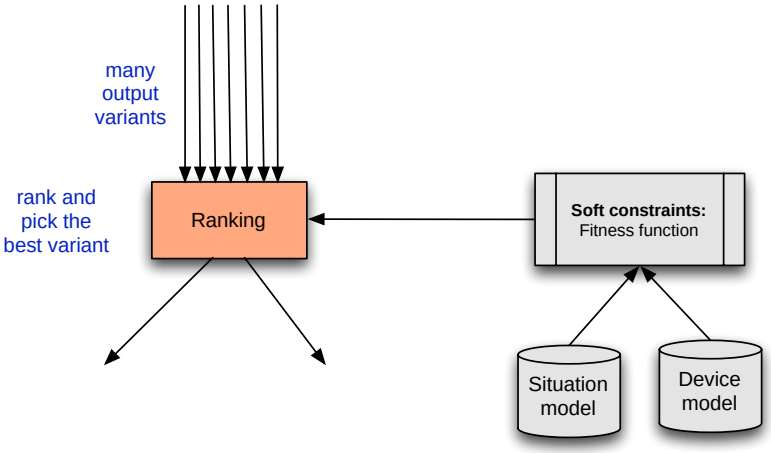


Figure 4.1: The ranking process covered in this chapter.

Human communication has many registers: there are usually many ways to express what we want to say. Under the same circumstances, a range of grammatically acceptable utterances would be truthful or not truthful, thus express the same meaning. However, only few of these seemingly equivalent utterances would actually be uttered by a speaker in a certain situation.

Undoubtedly, there are many factors involved in explaining a speaker's choices, apart from the grammatical and lexical framework a particular language imposes on communication. An important factor is our knowledge of the recipient of the communication: *who is listening and what are the communicative circumstances?* And: *what can the person be expected to understand?* Furthermore, using a term from theoretical linguistics, what *performance* can be expected of someone in processing our communication?

In this discussion, one certainly idealizes the speaker. We assume that the speaker is a considerate one, as his communications are concise, yet understandable. This judgment takes the hearer's *communicative situation* into account. In the context of a multimodal dialogue system, we can probably assume that the speaker (i.e. the system) tries to make optimal choices in terms of style and appropriateness.

To do so, the system adapts to the situation and the device by means of *situation and device models*. It does this gradually. This design decision was inspired by the characteristics especially of situations, which seem gradual: for example, an environment is not either silent or very noisy – there are many degrees of noise in between.

While the previous chapter has dealt with means that *a)* determine how to realize an utterance given the hard constraints posed by grammaticality and *b)* to some extent, coherence, this chapter develops the idea of soft constraints. These are goals we need attempt to reach, and we try to do as well as we can.

For the linguistically basic questions covered by the hard constraints, it is trivial to judge whether the choices they make are acceptable. The soft constraints are more novel for a natural language generation system. The generation system is evaluated in this chapter with respect to the judgements implemented by the soft constraints: a user study was conducted in order to find out whether the soft constraints can lead to increased perceived efficiency and the actual reliability of the communications in different situations.

4.1 Economy and Efficacy

In the following, we will address solutions to a dilemma that is often seen in communication: while we intend to deliver a message, doing so does not come for free. We endeavor to keep our own effort low, as well as our interlocutor's effort. H.P. Grice has most prominently formulated the principles of rational and cooperative linguistic behavior (1975, cited in Dale & Reiter (1995)). His *Maxim of Quality* postulates

- Make your contribution as informative as is required (for the current purposes of the exchange)!
- Do not make your contribution more informative than is required!

In relation to that, Grice's *Maxim of Manner* states the goal *Be perspicuous*, with a specialization *Be brief (avoid unnecessary prolixy)*. From a naïve point of view, these maxims would postulate communication to be minimal. No unnecessary information should be contained, no information repeated.

The linguistic and psycholinguistic evidence seems to contradict this idea. Consider a situation where a speaker points at a lone cow on a meadow and says, "Look, the cow is chewing daisies!". Clearly, the pointing gesture has a communicative function – namely to direct the addressee's attention to the cow, or, in other terms, to make the cow a salient entity, so that the following verbal utterance is coherent. Even if a deictic referring expression, such as "that cow" is used, it would seem unnatural for the speaker not to point to the cow, or to gesture in another way towards it. This clearly suggests that the gesture is not redundant. For the maxim of quantity, the term "information" would need to encompass communicative functions in general, i.e. denote *efficacy*.

The communicative situation influences the information gain as well as the effort. Consider Figure 2.3. While a) might be appropriate in situations where the interlocutor (user) is visually distracted, b) is more appropriate if the user can focus on the device.

Even if the efficacy of a potential utterance in a particular communicative situation is taken into account, a speaker may still deem pure redundancy "necessary". For example, it can be employed to make the signal more robust, i.e. to ensure that the message is understood. Dale & Reiter (1995) point out that psycholinguistic evidence shows that even unnecessary detail is employed in referring expressions. For example, a large, black dog, which could have been uniquely identified as *the large dog* or as *the black dog*, was referred to as *the large black dog* by the majority of speakers (cf. Levelt (1989)).

UI on the Fly consists of more than the MUG. While the MUG defines what is a faithful representation of the original dialogue intention, we still need to reduce the search space further – after all, MUG finds a number of faithful variants of the output (typically over 100). The communicative principles introduced in this chapter allow us to decide which variant to present. We encode the principles as *soft constraints*.

The approach brought forward in this thesis assumes a trade-off view for communicative choices. In particular, the selection of content underlies this trade-off in our particular implementation. In our NLG approach, we follow the goals of *predicted effort* and *efficacy* as outlined in the following.

- *Predicted (cognitive or interactive) effort:* Signals should respect the cognitive effort required of the recipient. The message should be easy to understand. As we are constructing an NLG system rather than a theory of natural language production, we ignore the production effort.
  - *Discourse coherence.* A prominent theory of local discourse coherence, Centering (Grosz et al., 1995), favors utterance-to-utterance transitions which pose the least possible changes as to which discourse entity is in focus, and which ones are suggested to be in focus in the next utterance.
  - *Cross-modal coherence.* Mode-specific signals are coordinated across modes. There is consistency on the lexical level, which relates to psycholinguistic evidence of cross-modal priming, where, for instance, a visual stimulus can influence lexical access for verbal signals.
  - *Utterance brevity.* Signals should communicate information as efficiently as possible.
- *Efficacy:* The message should contain all necessary information for the communicative intent. In a dialogue situation, information should be given to advance the discourse towards the overall dialogue goal.

The two goals are mutually contradictory: while *efficacy* attempts to express as much content as possible, *effort* tries to keep the communication brief. Because *efficacy* differentiates between important and less important pieces of information, we end up

with output that tries to say only the important things on the screen and in the synthesized speech.

There are different ways to implement the trade-off. An important NLG-wide problem to address is that some of these goals operate on data that is not available during the generation process until very late (Reiter & Dale (2000), Oberlander & Brew (2000)). This applies to the *brevity* goal, which refers to the utterance length. In MUG, utterance length depends on a lot of late decisions that occur at the leaves of the constituent tree. A typical example for such a decision would be whether to include the subject line in a nominal phrase that refers to an e-mail.

The coherence goals are realized in MUG as hard constraints, which lead to early decisions in the generation process. The goals of predicted cognitive effort and efficacy are realized as soft constraints, which can be violated.

4.2 Effort and efficacy in linguistics

The idea of economy is not new in linguistics. Zipf (1949) argued for quantitative balancing effects in a variety of fields, stipulating that effort consideration is universal to human behavior. *Synergetic linguistics* assumes that language systems use self-regulation to keep a range of parameters (word frequency, word length, polysemy) in certain "harmonic" relationships to each other. Often, the object of synergetic linguistics is language change, which is induced by a number of speakers.

The objective here is to describe communicative choices in a particular situation. An influential idea for the generation system presented in this thesis sees conflicting goals at the hart of linguistic decision-making. Phonology provided the first application area of such rules. The movement turned out to be very successful, with Optimality Theory (Prince & Smolensky, 1993) being the best-known framework.

Several authors, in particular Martinet (1952), Lindblom (1998), Flemming (2001) developed the idea of conflicting goals. Flemming examines a specific phonetic problem, where the (F2) frequencies in a vowel/consonant pair (V/C) adapt reciprocally. While, in such a pair, both V and C are assumed to have an underlying *target frequency* $F2$, speakers usually reach these target frequencies in neither phoneme; they "undershoot". Instead, the frequencies of both phonemes, $F2(C)$ and $F2(V)$ assimilate. Flemming analyzes the phenomenon using two basic constraints: *Don't deviate from targets!* with an associated violation cost proportional to the deviation, and *Minimise articulator velocity (effort)!*. Constraints are weighted rather than ranked as in Optimality Theory, so that multiple violations of one constraint can overrule the other contraint, even if the latter constraint carries more weight. Analogous to language-specific constraint ranking in OT, here the weights are language-specific. Flemming proposes similar solutions to a range of problems in phonology and phonetics in a *Dispersion theory* (Flemming, 1995).

Flemming's constraints have much in common with the soft constraints used to plan and realize multimodal output in the much more practical UI on the Fly system. Both approaches use an efficiency constraint, as well as weights rather than a ranking.

4.3 Weighting the constraints

The most basic way to see the interaction of hard and soft constraints is the generate&test methodology. Here, multiple potential solutions (i.e. output variants) are generated using the grammar that encodes the hard constraints. Each of these potential solutions fully satisfies the hard constraints. Then, the solutions are scored using a *fitness function*, which looks at a range of properties of each of the solutions, assigning a score to each. The one solution that is assigned the highest score is the solution which satisfies most or the most important soft constraints.

In a practical system, generate&test is inefficient thus impractical. Therefore hard and soft constraints are applied simultaneously using an appropriate *control strategy*, which will be discussed in Chapter 6. The remainder of this chapter is concerned with the fitness function that implements the soft constraints.

There are different approaches to formulating the fitness function, and usually there are several weighted considerations to make. SUPPLE (Gajos & Weld, 2004) implements a similar generation strategy, with an over-generating grammar and a fitness (i.e. cost) function. The system optimizes the predicted effort a user has to make in order to reach each element of an interface. Such a user-model driven fitness function still leaves the designers with many choices – for example, whether the cost for each user interface element should also depend on the corpus-based (e.g. maximum-likelihood) probability for it's actual use.

In typical UI on the Fly applications, one generates simpler, but multimodal interfaces for small-screen (bottleneck) devices. The number of elements shown on a screen is small, and the user interface widgets defined by the MUG do not differ greatly in the time it takes to operate them. We see cost differences, however, in the degree to which the voice modality is used (it takes time to listen to system prompts). We therefore model the efficacy of a particular multimodal output as a combination of reading / listening time versus the benefit of presenting important information.

The fitness function in UI on the Fly is a *heuristic* function. It combines empirically known models (such as one for the screen reading speed, based on the number of words) and reasonably well motivated (yet unproven) assumptions about how useful particular output might be. By default, we try to be as helpful as possible, with information that is deeply embedded in the semantic structure receiving lower priority than higher elements. Redundant information, that is, information that is presented in both modes, receives less than a double benefit. Information that needs to be presented according to the assumed dialogue management component leads to a heavy penalty if it is left out during generation stage.

The trade-off defined in the function lies in the cost of the output, which is estimated in terms of the cognitive load imposed on the user, who needs to read new text on the screen or listen to the voice output.

4.4 Situation profiles

The fitness function specifies soft constraints. These are parametrized with models for the current device hardware (screen size?) and the usage situation. The scoring approach could accommodate device/channel-specific constraints such as screen size or slow loading times for high-resolution graphics as well. We assume that each available mode is well-specified in terms of its output characteristics, both in terms of the MUG that is able to generate mode-specific markup (such as HTML or VXML) and of the user and situation models that contain (default) data for each mode.

Situation	screen-dynamic	screen-static	voice
At Work	10,1	5,1	10,1
Public Transport	1,5	2,1	10,1.5
Restaurant	10,10	10,10	5,5

Table 4.1: Example situation-specific coefficients for the α and β vectors.

The models encode the situation and device characteristics in terms of a number of mode-specific coefficients, as shown in Figure 4.1. These coefficients are α, β for the situation model, and ϕ for the device model. α modifies the benefit (efficacy) derived from output in a particular mode for a certain situation. For instance, one can read changing elements on the screen more easily at work ($\alpha = 10$) than while on the bus ($\alpha = 1$). β modifies the cost of output in a particular mode for a certain situation. For instance, voice output is considered more disturbing in the restaurant ($\beta = 5$) than in it is while on public transport ($\beta = 1.5$).[1] ϕ represents differences in output benefit for the modes.

The models used in VPA are pre-defined. Why? After all, we could fit the parameters given by the models to experimentally obtained data. These data would be a corpus of ideal system output in all situations covered. Since the term *situation* only covers up a range of varying conditions, the corpus would need to encompass a high number of examples. It is questionable whether such a costly approach would result in a measurable, significant improvement of adaptation quality. [2]

[1] We differentiate between changing (dynamic) and constant (static) screen output as separate modes, as implemented in VPA. The previous chapter has simplified this to only the screen and voice modes for illustrational purposes.

[2] A more realistic method not implemented here pertains to dynamic adaptation. Situation models adapt, to be retrieved at a later time by the user. An intelligent system would probably tie the situation models to position information such as coordinates retrieved via GPS. Such adaptation requires a feedback loop: the system needs to know if its behavior is appropriate. However, asking the user of an actually used system for feedback countless times is not a realistic proposition. An further alternative to the adaptation of a model would be to translate sensory input into situation models. Such sensors are commonly available on modern PDAs and include a microphone and some light-sensing device, such as a built-in camera. The connection of the device to a cradle, such as used for cellphones in cars, would give additional information. These approaches require the integration of sensor technology and generation, certainly leaving the scope of this thesis.

4.5 The fitness function

These constraints are formalized in a score that is assigned to each variant represented by the attribute value matrix ω, given a set of available Modes M. Also needed are a situation model $< \alpha, \beta >$, a device model ϕ. The models are vectors, containing a coefficient (α_m, β_m and ϕ_m) for each mode, which represents situation and device-specific properties. The fitness function is a weighted sum of the efficacy benefit (first part) and the efficiency cost (second part):

$$s(\omega) = \sum_{<e,d>\in E(\omega)} u(e,d) - max_{m\in M}(\beta_m T_m(\omega))$$

$$u(e,d) = P(d, \sum_{m\in M} (\phi_m \alpha_m e_{m|realized}), e_{realize})$$

To produce the efficacy benefit, the function E returns a set of *semantic entities* in e and their embedding depths in d. A semantic entity of ω is defined as any substructure of ω whose attribute path was already contained in the original semantic specification that was input to the algorithm. The semantic entities arising from the dialogue act shown in Figure 3.14 (p. 52) would be a top-level dialogue act (ASKCONFIRMATION), an action (of type TASK), a task (SEND-EMAIL), an e-mail, a contact (i.e. a person), an email address, a string with the subject line, and a string with the text of the message.

For each of these entities, the function P looks at whether it was requested by the dialogue manager (attribute `realize`) and whether it was actually realized in the particular output variant (the mode-specific attribute `realized`), which means that it would actually be shown on the screen or included in the speech output. The value of the `realize` attribute is expressed as $e_{realize}$ in the formula, the one of the mode-specific attribute `realized` as $e_{m|realized}$.

P penalizes the non-realization of requested entities, while rewarding the (possibly redundant) realization of an entity.[3] The reward decreases with the embedding depth d of the semantic entity. As d, we can simply assume the length of the attribute path leading to the entity. The e-mail in Figure 3.14 would have $d = 3$, while the ASKCONFIRMATION dialogue act would have $d = 0$. This mechanism assumes that deeper entities give less relevant details, which is intuitively (and according to our experience) a good heuristic. Nevertheless, the system's knowledge base may specify a `salient` attribute, which contains an ordered list of attributes that are most salient for the given type. For example, the `contact` type defined in the VPA domain specifies the list $< firstname, lastname >$ as salient – as opposed to, for example, that person's telephone number. (For the type hierarchy used in VPA, see Figure 3.7, p. 38)

The efficacy given by P is higher for redundant realization in several modes. However, it does not increase linearly, which reflects the intuition that redundant information is only slightly more useful than non-redundant information. As shown in the formula, the argument to P takes into account a situation-specific mode coefficient α_m.

[3] Section 3.3 discusses the dialogue manager interface with the dialogue acts referred to here.

The cognitive load (second part of the sum) is represented by a function that predicts the time $T_m(\omega)$ it would take to interpret the output. This equals the utterance output time for a text spoken by the text-to-speech system, or an estimated reading time for text on the screen.

4.6 A first evaluation of the fitness function

Natural Language Generation systems are notoriously difficult and costly to evaluate; adaptive systems even more so. In NLG, we don't deal with linguistic models that make particular predictions about language. The space of "good solutions" is rather large, and human subjects need to be asked to judge output. Obviously, the subjects are a heterogeneous group. Within-subject tests can make up for that; however, results are almost never comprable to those of experiments with other systems.

The fact that tests cannot be automatically reproduced, as it is the case with analysis algorithms (with a corpus and a gold standard annotation), or with dialogue management (Walker et al., 1997), makes NLG testing expensive, and iterative development even more so.

Dialogue systems research has produced methods of systematic evaluation, most prominently the PARADISE framework (Walker et al., 1997) for speech systems or the derived PROMISE method for multimodal systems (Beringer et al., 2002). One advantage of these is that evaluation can be carried out automatically, using subject data collected only once. This helps in iterative development. To do so, PARADISE encodes dialogues and their sub-dialogues in slot-filler structures for the information being passed between the computer agent and human user. A κ statistic, calculated on the confusion matrix of the empirical data and the target dialogue descriptions (slot-filler arrays) yields a form of dialogue success. Efficiency measures are introduced as cost functions. User satisfaction and other subjective measures are correlated to the automatically derived data. Thereby, a weighting is calculated: how important are the different cost measures for user satisfaction? By means of this correlation, new iterations of the system can be analysed and user satisfaction measurements can be estimated.

In some ways, the PARADISE methodology relates to the soft constraint based optimization approach employed in UI on the Fly. We already try to optimize our communication strategy based on weighted cost and efficacy measures. Measuring the success of such a system using a similar, or even the same fitness function makes little sense.

Adaptive systems pose further challenges to evaluators. Adaptation to users, in particular special-needs users, should be tested with a representative group of subjects. These are hard to come by, if, as in FASiL's VPA, a multimodal system needs to be evaluated. Such systems cater for users with partial loss of hearing or vision.[4]

[4]Whether a system is developed with the needs of representative or average users in mind is hardly important in the view of accessibility-advocates. Addressing a broad range of (sensory) deficiencies is the goal. The fact that some sensory impairments are rare may mean that the performance of the system in these cases is statistically almost irrelevant. Yet the system would be criticized as inaccessible.

If a mobile system adapts to its usage situation, the test procedure will need to control for the environment. Simulated distractions are almost always only a weak substitute. If we employ real-life distractions, we face ethical or liability constraints: who would want to run subject trials about the use of a multimodal PDA in cognitively and physically distracting situations, such as driving on a busy freeway or walking on the sidewalk?

An alternative might be a field trial that would let users decide when to use the device. Their interactions would be evaluated in relation to the communicative situations. Unfortunately, a field trial is not very practical, for three reasons:

1. We would need a system, not just a natural language generation module. UI on the Fly balances constraints, and some of its choices will only prove worthwhile in a complete dialogue. Other multimodal NLG systems will also optimize interactivity, which again needs a full dialogue cycle.

2. The system has to be stable and free of major problems. For mobile, voice-based systems, this is currently not yet the case. Whether the NLG system runs in a demonstration system built only for evaluation, or whether it runs in a complex environment constructed in a multi-party effort, they still show major usability issues. In a user-evaluation, these will outweigh the more subtle differences in NLG choices. In other words, if the machine does not understand the verbal input even after several tries, the user won't be impressed with the beautiful multimodal output.

3. If the field evaluation is successful and shows a distribution of quantitative data, it will be difficult or even impossible to point our fingers at a single system component. We will not find out if the NLG module did it's job well or not so well in certain situations, unless the subjects are actually able to criticize particular behavior. Indirect measurements, such as task completion time, or cognitive load imposed, measure the system as a whole.

Therefore, the soft-constraint balancing method employed in UI on the Fly was evaluated with a small study that simulates situations and a dialogue system, gaining direct and indirect quality measures.

4.6.1 Experimental configuration

We showed subjects not just single out-of-context utterances, but whole dialogues. System output on the screen and by voice was generated with the UI on the Fly system.[5] The user input was pre-recorded and voice only. The dialogues played automatically without the subject's intervention.

In the dialogue, the (virtual) user sends an e-mail to a number of people. Recipient names were randomly modified; so were the subject line and the body of the e-mail. Recipients were spread over the "to", the "carbon-copy", and the "blind-carbon-copy" fields of the e-mail. This varied across dialogues, too.

[5]The voice output was rendered by Apple's Mac OS X TTS with a high-quality female unit-selection voice. The screen output was shown with Internet Explorer (IE). We used a server-client architecture with the HTTP protocol and IE as frontend.

The principal dependent variable of this experiment is the subjects' recall of the contents of the dialogue, measuring whether subjects were able to notice differences in the e-mail that was being sent during the multimodal dialogue and a complete e-mail presented at the end.

Therefore, subjects were first asked to pay attention to the dialogue. After the dialogue had been presented, subjects were presented with a questionnaire that showed a visual representation of a complete e-mail. They were asked whether this e-mail equaled the one that was sent in the previous dialogue. In half of the trials, the e-mail shown did not fully represent the e-mail from the dialogue (independent variable 1), and the subject's task was to notice the difference or equality. The "mistakes" hidden in the e-mail presented at the end are: a) One or more recipients had in a different recipient type, for example, a person who was on the "cc" (carbon-copy) list in the dialogue, ended up being on the "bcc" (blind-carbon-copy) list in the questionnaire e-mail. b) Recipients were missing or added and c) the subject line or the body of the e-mail were significantly altered.

The two other dependent variables examined were subjective measures of *efficiency* and *reliability*. The questionnaire contained, for each judgement, a 7-point scale, which ranged from *inefficient* to *efficient* and from *reliable* to *unreliable*.

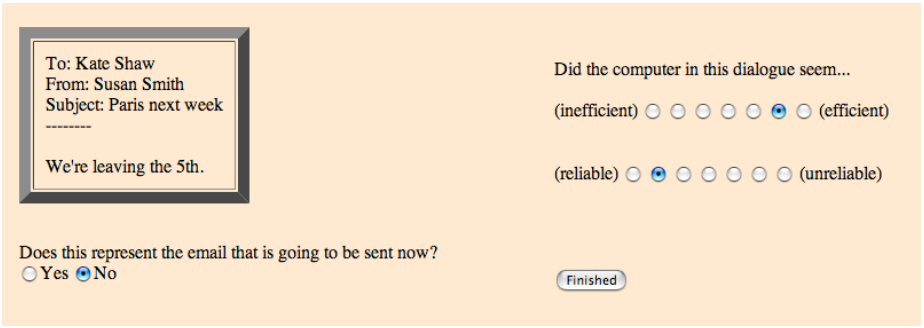


Figure 4.2: After the recorded dialogue, subjects were presented with a questionnaire. It shows an e-mail with or without differences from the e-mail sent during the dialogue, and scales for two subjective judgements.

The dialogues were presented under two types of varied conditions as outlined below. Each of the conditions (dependent variables) had two possible values.

1. In one half of the trials, the UI on the Fly system was configured to output a top-ranked variant, which is a variant that was deemed appropriate for the situation by the soft constraints. In the other half, the UI on the Fly system picked a low-ranked variant. (Independent variable 2).

2. In one half of the trials, the user could pay full attention to the dialogue. In the other half (Independent variable 3), we simulated a situation that posed a visual and manual demand: the subject was asked to play a computer game (see Figure 4.3). Subjects had to steer a boat down a river, avoiding obstacles by moving to either side

Figure 4.3: Physical setup with MUG screen and second PC providing a distraction.

using the left and right arrow keys. While the game distracted users heavily, they could still hear the system's voice output and occasionally glance at the UI on the Fly screen.[6]

Each subject (n=20) completed 8 dialogue-questionnaire pairs.

The efficiency and reliability judgements were normalized for each subject using the z-score

$$z_x = \frac{x - \mu_x}{\sigma_x}$$

4.6.2 Results

The number of discrepancies between the e-mails presented during the multimodal dialogue and in the questionnaire were evaluated, with respect to whether a *good* or a *bad* variant was presented by the system. The results found differ for the distracted and non-distracted situations (Figure 4.4).

In the distracted condition, subjects noticed a higher number of faulty e-mails, when presented with a *good*, highly-scored variant (mean proportion $M = .28$) as opposed to when they were shown a *bad* variant ($M = .20$). This should only be seen as a trend. In the non-distracted condition, the hit-rate remained the same: there was

[6] This distraction was inspired by the idea of users keeping busy in the household while sending an e-mail. While the studio setup certainly falls short of providing a realistic simulation, we believe that the visual, manual and cognitive demands are comprable to the real-life situation.

no effect of *appropriateness* according to the soft constraint scoring mechanism used ($M = .25$).

Subjective efficiency and reliability judgements: An effect could only be seen for the non-distracted situation. Here, subjects saw the high-ranking interactions as more efficient ($M = .59$) than the low-ranking ones ($M = -.17$). In the distracted situation, subjects assigned very close ratings for the high-ranking ($M = -.20$) and low-ranking ($M = -.14$) variants (see Figure 4.6). A two-way ANOVA showed that the effect for the non-distracted situation is statistically significant ($F(1, 19) = 5.29, p < .05$).[7]

Perceived reliability did not differ significantly for high- and low-ranking variants. In the non-distracted situation, subjects judged the interactions as similarly reliable for high-ranking ($M = .01$) and low-ranking variants ($M = -.06$) and also did so for the distracted situation for high-ranking ($M = .06$) and low-ranking ($M = 0.00$) variants (see Figure 4.6).

4.6.3 Analysis

In distracted situations, subjects were slightly better at spotting mistakes in the dialogue, when they were shown highly-scored utterances. A weak conclusion could be that the ordering imposed by soft constraints on the two variants (top-ranked vs. lowest-ranked) may show a trend to improve the reliability of the communication.

However, there was no effect for the hit-rate in the non-distracted situation. There is a simple explanation of this phenomenon: if both modes are available, subjects made use of all the information given over the course of the dialogue and equally for the voice and screen modes. They had enough information to spot the errors.

Overall, subjects only spotted few mistakes, which may have to do with the fact that subjects were not sending their own messages in an interactive system, but merely watching the dialogue.

The users perceived efficiency improved slightly through the use of the situationalized trade-off with the soft constraints used by UI on the Fly. More work has to be done to look into further constraints and improve adaptivity in order increase the effect seen. In conclusion, the finding can be considered relevant for the development of both generation components and such systems which use similar soft constraints to automatically evaluate user interface outputs.

4.6.4 Conclusions

The data gathered during the limited evaluation experiment indicates trends. Users do take note of the (believed) efficiency of a user interface, and the choices of the soft constraints make a difference.

[7] I thank Michael Cody, Fred Cummins and Erin Panttaja for their help in implementing and conducting the experiment. An earlier version of the evaluation methodology was developed in Panttaja et al. (2004).

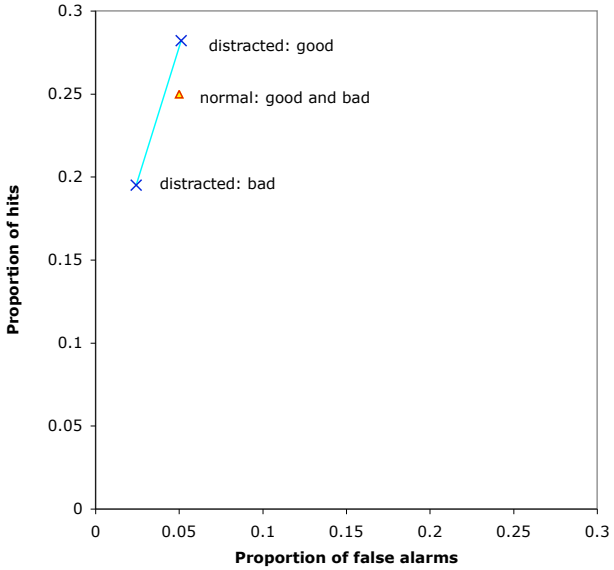


Figure 4.4: False Alarms versus Hits: in the distracted condition, there are more hits (higher recall) and only slightly more false alarms (lower precision).

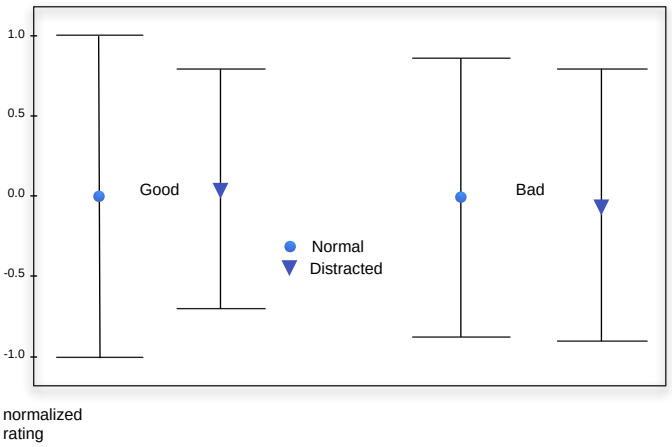


Figure 4.5: Reliability ratings: no effect could be shown for good and bad variants under different conditions.

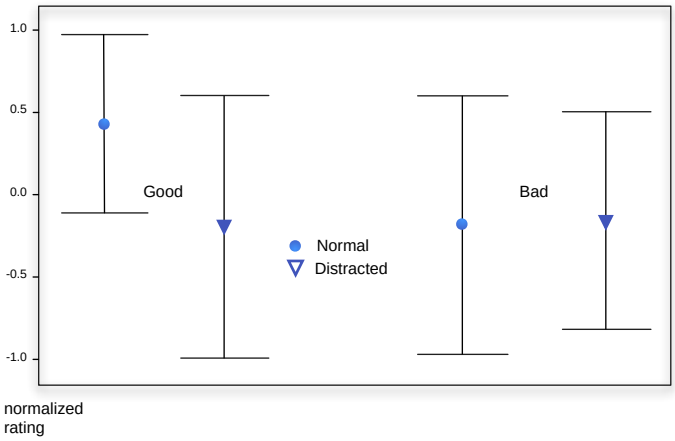


Figure 4.6: Efficiency ratings: Only in the non-distracted situation, users judged the efficiency to be higher for variants deemed good by the fitness function.

The construction of the experiment emphasized the non-subjective measure of reliability rather than its subjective measure (*perceived reliability*). I assume that the experimental design impeded us in finding an effect for good and bad variants with respect to perceived reliability.

A shortcoming of the experiment was that the task of spotting mistakes was too difficult, even though the experiment had been improved after pilot trials. The task is obviously different from spotting real-life mistakes, where the user's own content is to be validated – for example when a system misinterprets spoken user input.

The effects shown should motivate further efforts in developing methods that optimize a system's actual communicative efficiency. This feature is noticed by users. We conjecture, it is not just noticed, but also *appreciated*.

5 Coherence

Coherence is a property of text or multimodal communication that contributes to what is commonly called *good style*. It is most evident when several alternative variants of text. Even though such variants may bear the same meaning, human communicators show a strong preference for only a few of them. These preferred variants are usually easy to read and understand, in other words, they are fluent. Texts with a high degree of coherence are those in which *everything fits nicely together*.

From a linguistic or cognitive point of view, there is much to say, yet no final conclusion can be drawn about why some texts or communications appear more coherent than others. I distinguish between two types of coherence: *Cross-modal coherence*, which is a property of simultaneous, coordinated multimodal communication, where the speech, text or graphics presented simultaneously align with each other, and *discourse coherence*, where consecutive sentences in a text or dialogue fit together.

Coherence is a key element in designing a dialogue-based interface. The multimodal interface outputs created with UI on the Fly can be coherent. This chapter presents the two forms of coherence and defines models that can account for them in the context of multimodal output generation with UI on the Fly.

In the Virtual Personal Assistant domain as well as in similar dialogue contexts (flight bookings or other database query interfaces), the need for coherence is limited. Dialogues tend to be short and to the point. Focus shifts steadily between different pieces of information entered by the user, except in error recovery situations. For other dialogue systems, for example in tutoring, dialogues are longer. Referring expressions, where coherence plays an important role, are common. The UI on the Fly system has provisions to accomodate such phenomena. But for the above reasons, they were tested with toy domains rather than with the VPA.

5.1 Cross-modal coherence

5.1.1 Motivating cross-modal coherence

In a dialogue, interlocutors are usually consistent in their own idiosyncrasies, and they also adopt their conversation partner's style to some extent. Similarly, we would expect humans and computers to be consistent in speaking and writing.

Many of the linguistic choices we make have no bearing on the semantics of communication or its stylistic qualities, if viewed in isolation from their context. However, being consistent in one's choices contributes to good style. For example, lexical choice should not vary: it is either *cell phone* or *mobile phone*. Cross-modal coherence implies

consistency of this choice across all verbal modes.[1] Humans display different forms of coordination between communicative modes (McNeill, 1992; Oviatt et al., 1997). The empirical evidence for a range of similar and related phenomena will be detailed in this section.

The UI on the Fly generation system aims to be coherent and consistent across all modes. It presents redundant content, for example, by choosing the same lexical realizations. It ensures cross-modal references to be accomodated. For example, a deictic expression such as *these two e-mails* (by voice) forces the specific e-mails to be put in focus on the screen.

Cross-modal coherence motivates changes in MUG compared to grammars using the more conventional Functional Unification Formalism. These changes consist of a very simple principle encoded in the generation algorithm: all components realizing one semantic entity must unify. Components may still contain mode-specific information in an attribute named after the mode. AVMs within this attribute will not interfere with the realization instructions of a component that realizes the same semantic entity in another mode. All other branches of the AVM, however, must unify across modes. In short, the AVMs allow us to distinguish information a) that needs to be shared across all output modes, b) that is specific to a particular output mode, or c) that requires collaboration between two modes, such as deictic pronouns. The unification principle replaces explicit integration rules for each coordination scheme, such as the ones used by Johnston (1998).

An example of cross-modal coordination: The lexical realization mentioned (*cell phone* vs. *mobile phone*) is an easy one, which can be taken care of by simply hard-coding one choice. An example more relevant for the e-mail domain used in the VPA would be the realization of a name. People can be referred to by their first or last name, or by their full name. I expect any combination on screen and by voice to be allowed, except for the case where a person would be referred to by first name in one mode, and by last name in the other one.

Related priming experiments show that such an incoherent *sequential* combination of referring expressions is more difficult to process. An effect of cross-modal alignment or coherence in *simultaneous* stimuli on the performance of processing communicative output is hypothesized.

Classical cross-modal priming experiments. Generally, a first visual stimulus allows users to access an associated semantic or lexical entity more easily, when they process a second stimulus, or produce language. When presented visually with a ambiguous word, subjects can identify (decide about lexicality of a synonym) one of its meanings more quickly (Swinney, 1979). This works for picture naming tasks (with a verbal, visual prime), (Federmeier & Bates, 1997), and for words in two languages (Liu, 1996). Cross-modal priming suggests that perception and/or production are easier if a related prior stimulus was given. If we assume that the visual depiction is processed

[1] A *verbal* modes is one that employs natural language, whether in spoken or written form.

before the spoken system output, it follows that coordinated output should be easier to process.

Alignment. (Pickering & Garrod, in press) argue that dialogue participants align their phonological, lexical, and syntactic choices. Carefully controlled experiments support their dialogue model. Alignment is particularly evident in question/response scenarios, or when one interlocutor finishes the other one's sentence. In the Virtual Personal Assistant domain, there are examples where the graphical output determines the user's choice of responses: Let's assume the system asks a (verbal) question and offers a set of choices (radio buttons) for the user to pick. Obviously, we would want the screen output to align with the voice output, for example in morphosyntactic case marking in German. In the following examples, ACC stands for *accusative* case, DAT stands for *dative* case, and NOM for *nominative* case.

Voice:

Finde	*persönliche*	*Nachrichten*	*oder*	*solche*
Find	personal	Messages–ACC	or	the-ones–ACC
vom	*Institut*	*oder*	*vom*	*Direktor?*
from-the–DAT	department–DAT	or	from-the–DAT	director–DAT

Screen: Three radio buttons allow the user to choose exactly one out three options, labeled with "persönliche" (*personal ones*-ACC), "vom Institut" (*from the–DAT department*) and "vom Direktor" (*from the–DAT director*).

* **Screen:** Three radio buttons allow the user to choose exactly one out three options, labeled with "persönlich" (*personal*-Unmarked), "das Institut" (*the–NOM/ACC department*) and "der Direktor" (*the–NOM/ACC director*).

In this case, a user's reply would use the same prepositions and thus, the same morphosyntactic case (required by the German prepositions). Offering *personal*, *the department*, *the director* in nominative case and without prepositions as shown in the alternative marked with (*) would have impeded the user's alignment with the system.[2]

Managing expectations consistently. Experience from dialogue systems shows that users generally tend to expect the system to understand phrases that it utters.[3] This could be seen as a natural alignment-related assumption about dialogue. Cross-modal coordination allows a system to send clear signals about the system's capabilities, rather than sending different ones.

[2] This constructed in-domain example is given to support the hypothesis of cross-modal alignment – it is not implied that alignment as shown is implemented in UI on the Fly.

[3] This is a common deficiency of some, mostly commercial systems, which use different resources for generation of (canned) text and natural language understanding.

Gestures, body posture and rhetorical moves. Gestures are known to be coordinated with speech (usually preceding it). Beyond the simple case of deictics (*this, that*), subjects showed shifts in body posture that coincided with high-level rhetorical moves, in particular changes in topic (Cassell et al., 2001). While discourse-level effects are addressed in the second part of this chapter, we can learn from discourse structure that there is cross-modal temporal coordination of discourse units. UI on the Fly with its generation formalism MUG incorporates that for screen and voice output.

As shown, both in temporally distinct (alignment, priming) as in simultaneous language (or action) production (gestures), we can see coordination effects. The conclusion is a strong hypothesis for cross-modal coordination of verbal and non-verbal content.

The next sections deal with the way MUG generates cross-modally coherent content.

5.1.2 Examples of cross-modal coordination

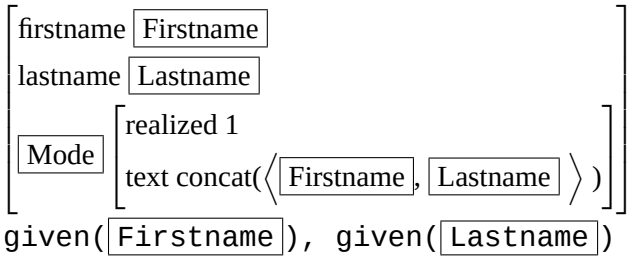
The cross-modal coherence motivated here can be found in MUG in the parallel realization of content in all modes, where components that realize a particular semantic entity must unify across all modes.

In the following, we will take a look at some cases where the unification principles of MUG ensure cross-modal coordination.

Names. One of the MUG components in the FASiL VPA grammar realizes the full referring expression for a *contact*. This component is controlled by a higher set of rules that deal with the generation of referring expressions in general, employing pronouns when appropriate. A name can be realized in various ways. We could refer to the person in question by his/her first name, by last name, or by full name.

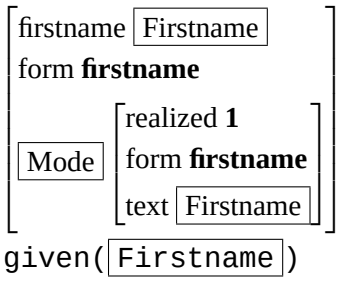
Without any cross-modal coordination, the system could freely choose how to refer to a person. For example, it could refer to the same person by first name on the screen, and by last name via voice. This is clearly inconsistent, and would be interpreted as a flaw. In particular, if the user does not remember the full name of the person very well, she might believe that the screen output and the voice output refer to different people.

The solution lies in an attribute that takes on different values depending on the form of the name. Competing components that realize the name can have a `form` attribute, which is of values **firstname** or **lastname**. The attribute ensures that a first name only output may never be combined with a last name only output. They may, however, be freely combined with a full name version of the same entity:

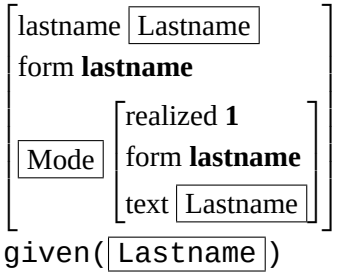
(1)


The additional constraints given(X) here state that Firstname and Lastname must be instantiated. (Recall (Section 3.4) that a component consists of an AVM and optional other constraints. These are, however, kept to a minimum.)

In some cases, we want to give only their first name, as in this component:

(2)


... or just the last name:

(3)


Components (2) and (3) do not unify, which avoids a situation where a first name is given in one mode, and a last name is given in the other.

This realization of the person demonstrates cross-modal coordination. Note that there are other considerations when realizing a reference to a person. Cultural and linguistic variation prescribe certain choices for different levels of familiarity (or social

difference) with the person.[4] A full name reference is usually only chosen if the generation system needs to point out a particular person from others with the same first/last name (a distractor set). For algorithms addressing this, see Dale & Reiter (1995), or Horacek's work, for instance Horacek (1997).

Deictics The deictic demonstrative pronoun "this" can refer to an entity that is emphasized visually (e.g. by pointing) in conjunction with the utterance. Thus, a lexical entry for a spoken "this" will stipulate certain properties for the visual realization.

To generalize this behavior, an intermediate component layer of category REFEXP takes on the task of planning referring expressions. It realizes anaphoric pronouns (e.g. "it"), deictic pronouns (e.g. "this") and definites (e.g. "the file") with generic components. Full forms are realized by calling components specific to the type of entity to be realized, which means that there is a group of components that realize names, and other components that realize references to an e-mail, for example by giving the recipient and its subject line.

Syntactically higher-level components, e.g. realizing sentences or verbal phrases always call the REFEXP layer to generate referring expressions. Via structure sharing, they unify the semantic representation of the entity in question from the dialogue act representation with the value of an attribute called `sem` within the REFEXP structure.

The component from the REFEXP layer shown in Figure 5.1 represents a deictic realization for an arbitrary entity. The entity is supplied from another component (the one that "calls up" this component) in the `sem` attribute. Its object type is unified with variable ObjectType. In the branch `typetext`, another component is called to provide a realization for the type. The text is bound to variable TypeText. It could contain the string "document" for the *document* type. The result of this component constructed in `voice|text` is a string such as "this! document". (The exclamation mark serves as markup for the text-to-speech system to put a non-nuclear accent on the demonstrative *this*).

Finally, the component states in the `screen-dynamic` branch that the accompanying screen output must realize the entity fully (i.e. not as pronoun or definite), and that the entity must be highlighted. Note that this component shows only the realization for the *voice* mode. The *screen-dynamic* realization differs – however, the component for the other modes will have to unify with this one to ensure cross-modal coherence.

The approach shown here does not necessarily make a definitive choice of a single referring expression. Instead, it leaves most of the decision-making tasks to the soft constraints (described in Chapter 4). The interaction of cross-modal coherence and discourse coherence may play a role in refining the soft constraints, ultimately leading to a smaller search space (see also Section 5.2 on local discourse coherence).

MUG avoids the use of explicit coordination rules, such as used for signal fusion in the MATCH system. MATCH employs a unification-based formalism, not unlike

[4] We would also refer to a person by last name as Dr X, Ms X and the like – if title and gender information was available in the FASiL VPA domain and comprable systems.

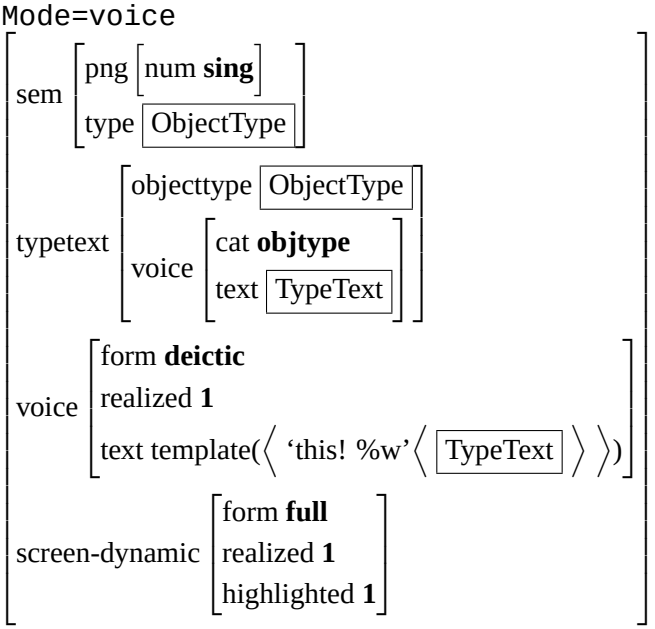


Figure 5.1: REFEXP component realizing a deictic definite description, such as “this person” or “this document”.

the one MUG uses, for semantic signal fusion (Johnston, 1998). Types of cross-modal coordination (e.g. location by pointing, commands by voice) are explicitly encoded in rules. This makes sense given the fact that only a certain set of non-verbal cues is to be integrated, mainly when users combine complementary types of information. MUG, in contrast, emphasizes cross-modal coherence and avoids the use of specific rules. Specific coordination schemata (such as deictic pronouns plus pointing) are broken down into separate, quasi-lexical entries in the grammar. They combine to form the coordinated structures.

5.2 Discourse coherence

Just like cross-modal coherence, discourse coherence is a property that models *stylistic preferences*. This kind of coherence, however, is concerned with the relationship between sentences that occur in sequence. While there may be many texts that express the same meaning, the *coherent* ones will be more pleasant to read and easier to understand.

Coherence criteria assume that we follow a set of basic principles when determining what information to present when in a text. As for the ordering of utterances, there are other strong criteria to affect it – not just coherence. But within a single sentence, coherence plays a role when we decide when and how to refer to contextual information.

For example, we could say *Joelle has sent you three messages.* or, alternatively, *There are three messages from Joelle.* Which of these two output variants to choose depends on how the sentences before and after this sentence are constructed.

In the following, I will introduce different aspects of discourse coherence, some of which fall within the scope of *Centering Theory* (Grosz et al., 1995). Centering has been developed since the mid-1990's as a cognitively motivated theory of discourse coherence. I demonstrate that Multimodal Functional Unification Grammar can accomodate aspects of the theory, in order to allow for discourse coherence to be a soft constraint in the realization process.

5.2.1 Different aspects of discourse coherence

In the following, different natural language discourses expressing the same four statements can be seen. The statements (or: *propositions*), which express one factoid, are:

1. Jackie is an 11 year-old girl from Johannesburg.
2. Jackie owns a German Shepherd.
3. Jackie's brother owns a rabbit.
4. One day, Jackie's German Shepherd mauled her brother's rabbit.

I will start with a seemingly random realization. Each of the following sentences is grammatical and expresses one of the above propositions:

> (A) *One day, the dog mauled the rabbit. Jackie's brother owns a rabbit. Jackie is an 11-year-old girl from Johannesburg. Jackie owns a German Shepherd.*

This short text is difficult to understand and would not be formulated by a considerate, normally intelligent speaker. It violates several communicative principles relating to coherence.

One of them is that determiners such as *a* and *the* in referring expressions such as *a dog* or *the rabbit* need to signal whether they introduce new entities in the discourse or refer to old ones. Correcting this mistake results in a slight stylistic improvement:

> (B) *One day, a dog mauled a rabbit. Jackie's brother owns the rabbit. Jackie is an 11-year-old girl from Johannesburg.She owns the German Shepherd.*

A further communicative principle is that pronouns are employed when the object that they refer to has already been introduced, and that we usually do not introduce a new person in a possessive attribute, as *Jackie* is introduced in *Jackie's dog*. Speakers also tend to use the more specific description first, and a more general description later on:

> (C) *One day, a German Shepherd mauled a rabbit. Jackie is an 11-year-old girl from Johannesburg who owns the dog. Her brother owns the rabbit that was bitten to death by the animal.*

Even with the referring expressions corrected, the order in which all the people and animals are introduces makes processing the language difficult. The text does not appear to be *fluent*. This is where Centering theory provides a valuable explanation, as to why following discourse tends to be preferred over each of the previous ones:

> (D) *One day, a German Shepherd mauled a rabbit. The dog's owner is Jackie, an 11-year-old girl from Johannesburg. The rabbit was kept by her brother.*

Another aspect of discourse coherence deals with the *rhetorical structure* of text. Theories of rhetorical structure explain, how propositions are ordered to maximize the credibility of the argument that is made. It provides some (insufficient) rules for the use of discourse connectives, such as *while, but* or *even though*. Rhetorical structure takes place in longer texts or dialogues, where rhetorical relationships can be found to hold between larger spans of texts. Rhetorical theories such as *Rhetorical Structure Theory* (Mann & Thompson, 1988) often provide an important means to aggregate (structure) discourse in Natural Language Generation. The above dialogue could emphasize the parallelism between the second and third sentence:

> (D) *One day, a German Shepherd mauled a rabbit. The dog's owner is Jackie, an 11-year-old girl from Johannesburg, while the rabbit was kept by her brother.*

In multimodal user interfaces, however, only a limited set of rhetorical moves can be expected to be found. Therefore, I opt not to deal with rhetorical structure in this thesis.

Most aspects of discourse coherence are difficult to capture. The ordering of utterances interacts with various contextual contraints, such as the standards set by the text genre. For instance, a newspaper article would provide a compact summary in a lead sentence: *A dog of an 11-year-old Johannesburg girl has mauled a rabbit. The dog, a German shepherd, mauled the pet, when (...).* An alternative ordering would be chosen by a police report, or during an oral account given by the rabbit's owner.

Given comprable circumstances and communicative intents to compare different sequences of utterances, however, a theory of local discourse coherence can predict human preferences for some of the discourses. For the purposes of this discussion, I assume an *utterance* to be whatever is generated by the application of a grammar (MUG), but no more than one syntactic sentence.

5.2.2 Centering Theory

Centering is a model of local coherence, developed by Grosz et al. (1995): it looks at fluency as a property emerging from two adjacent utterances at a time. Centering uses two violable constraints. When these constraints are fulfilled, the transition between the two utterances is coherent: *Cohesion* is a constraint stating that utterances should maintain

the same center of attention, which I will call *topic* in the further discussion. *Salience* postulates that an utterance should realize its topic in the most prominent position.[5] This is (depending on the language) usually the subject. I will define formally, how centering defines the topic and what the relative preference for Cohesion and Salience is.

Centering theory provides a model that can explain the reasoning leading to steps (C) and (D) above. Apart from this model, an algorithm to choose valid determiners (*the*, *a*) as shown in step (B) can be formulated in MUG.

Attentional state

Centering describes the relationship between adjacent utterances in terms of the *discourse entities* they *realize*. A discourse entity is basically anything that can be referred to: living beings, material things, ideas. A discourse entity is realized by an utterance, if it is mentioned or referred to. Example: the sentence *She lives in Johannesburg and owns a dog* realizes the following discourse entities: *Jackie, Johannesburg, Jackie's dog* provided *she* actually refers to Jackie. Each utterance influences the attentional state through the entities it realizes.

The attentional state is defined in terms of three data structures (*centers*), thus:

1. The *forward-looking centers* are organized in a list (CF) of discourse entities, ordered according to their syntactic role in the sentence. While a discussion of syntax would extend beyond the scope of this introduction, it is important to note that the subject is usually the first element in the list, and optional elements (which could be left out without impeding the grammaticality of the sentence) appear at its end. The ordering is deemed to be specific to the language.

2. The *preferred center (CP)* is the highest-ranking element of the CF.

3. The *backward-looking center (CB)* is a single entity that has been realized in both the current utterance and the preceding one. If there are several such entities, it is the one entity that was ranked highest (frontmost) in the CF of the preceding utterance. It can be seen as the *topic* of the previous utterance.

In the following, simple examples are given to illustrate what is predicted by centering. Table 5.1 lists the centers for these example sentences. The two principles of Cohesion and Salience will be formally defined and ranked. Their fulfillment determines the quality of the *transition* between two utterances.

Some examples

The following example discourses use the same semantic propositions as above. The discourses given, however, are formulated to demonstrate the different Centering principles of Cohesion and Salience. Consider the following short discourses, which differ in the *cohesion* of utterances. Which one is the stylistically preferred one?

[5] I follow Kibble (2001) in his breakdown of the original Centering model into Cohesion and Salience.

Ex.	Utt.	CF	CB	CP	Transition
E/F	1	Jackie	0	Jackie	
E/F	2	Johannesburg, dog	Jackie	Jackie	CONTINUE
E	3	rabbit, dog	dog	rabbit	ROUGH SHIFT
F	3	rabbit, dog, Jackie	Jackie	rabbit	RETAIN
G/H	1	dog, rabbit	0	dog	
G	2	Jackie, dog, Johannesburg	dog	Jackie	RETAIN
H	2	dog, Jackie, Johannesburg	dog	dog	CONTINUE
I/J	1	Jackie	0	Jackie	
I/J	2	Jackie, dog, Johannesburg	Jackie	Jackie	CONTINUE
I	3	dog, rabbit	dog	dog	SMOOTH SHIFT
J	3	dog, rabbit, Jackie	Jackie	dog	RETAIN

Table 5.1: Attentional structure for the examples in the text

(E)

1. Jackie is an 11 year-old girl.
2. She lives in Johannesburg and owns a dog.
3. One day, a rabbit was mauled by **the** dog.

(F)

1. Jackie is an 11 year-old girl.
2. She lives in Johannesburg and owns a dog.
3. One day, a rabbit was mauled by **her** dog.

The Centering model predicts that (F) is preferred over (E). In (F), the phrase *her dog* refers back to *Jackie*. Therefore in (F), and not in (E), the topic (CB) does not change between utterances 2 and 3, it remains *Jackie*. (E) violates the Cohesion principle. In the following two texts, another principle is violated:

(G)

1. One day, a German Shepherd mauled a rabbit.
2. Jackie is an 11-year-old girl from Johannesburg who owns the dog.

(H)

1. One day, a German Shepherd mauled a rabbit.
2. The dog is kept by Jackie, an 11-year-old girl from Johannesburg.

Centering predicts that (H) is preferred over (G), because it realizes the topic (the dog) in the most prominent position of the second utterance. (G) does not fulfill the Salience principle.

Which principle is more important? Examples such as the following can give a first indication:

(I)

1. Jackie is an 11-year-old girl.
2. She lives in Johannesburg and owns a dog.
3. One day, the dog mauled a rabbit.

(J)

1. Jackie is an 11-year-old girl.
2. She lives in Johannesburg and owns a dog.
3. One day, her dog mauled a rabbit.

Utterance (I3) shifts the topic from *Jackie* to *the dog*. However, the sentence realizes this topic in a prominent position. Therefore, the sentence violates only the *cohesion* principle. Utterance (J3) does the opposite: it maintains the topic (by realizing *Jackie* in the possessive pronoun *her*), but violates the *salience* principle, as the most prominent position (subject) in (J3) realizes *the dog*.[6]

Centering, as seen by Walker et al. (1998), predicts (J) to be more acceptable than (I): Coherence is more important than Salience. Intuition may or may not share this view.

Cohesion and Salience: the ranked utterance transitions

Now that the centers are defined, the constraints mentioned above and utterance transitions that depend on these constraints can be formalized. Centering formulates the Cohesion principle as the following constraint:

Do not shift the topic (if there is any)!

$$\mathrm{CB}(U_i) = \mathrm{CB}(U_{i-1})$$

The Salience principle demands:

Formulate a center in the most prominent position!

$$\mathrm{CB}(U_i) = \mathrm{CP}(U_i)$$

[6] It is assumed that possessives such as *her* are ranked lower than the noun (*dog*) that they modify in the CF, or at least not as high as the subject of a sentence.

Table 5.2: Transitions between utterances U_{i-1} U_i and their conditions in Centering

	CB(U_i) = CB(U_{i-1}), or CB(U_{i-1}) = [?]	CB(U_i) $\neq$ CB(U_{i-1})
CB(U_i) = CP(U_i)	CONTINUE	SMOOTH SHIFT
CB(U_i) $\neq$ CP(U_i)	RETAIN	ROUGH SHIFT

Grosz et al. (1995) formulate three transitions between adjacent utterances, to which a further distinction (of topic shifts) has been added by general consensus since. These transitions result from the violation of some of the above constraints. Table 5.2 lists the transitions and their dependence on the constraints. The following ranking is a direct result from prioritizing Coherence over Salience:

CONTINUE $>$ RETAIN $>$ SMOOTH SHIFT $>$ ROUGH SHIFT

Given this ranking, it becomes clear why (E) is preferred over (F), and (G) over (H), and (J) over (I): the transitions occurring of the preferred variants are ranked higher. These transitions are listed in the rightmost column of Table 5.1.

5.2.3 Pronominalization rule

Pronominalization occurs when a pronoun (*he, it, this*) is used to refer to a discourse entity, which may otherwise be referred to by name or with a definite referent (*the man*). Centering claims that

If any element of CF(U_i) is realized as a pronoun in U_i, so is CB(U_i).

The pronominalization rule is illustrated in Figure 5.2, which shows a different example in an alternative form of noting the centers, as they change during the discourse. In the figure, we see a sequence of five utterances, whereas alternatives are given for utterances 4 and 5. Whether 4a or 4b are chosen, the transitions remain the same. However, 4a violates the pronominalization rule. 5b is preferred over 5a, because it has the higher-ranking transition (RETAIN instead of ROUGH SHIFT), independent of the choice taken in 4.

5.2.4 A parametric, evolving theory

The ranking of transitions established since Grosz et al. (1995) seems to be universal and hard: if the best transition available is not used, text is predicted to be incoherent. Why, then, do corpora contain more SHIFT than CONTINUES (in spoken discourse, Passonneau (1998)), or generally a higher percentage of SMOOTH SHIFTS than RETAINS?

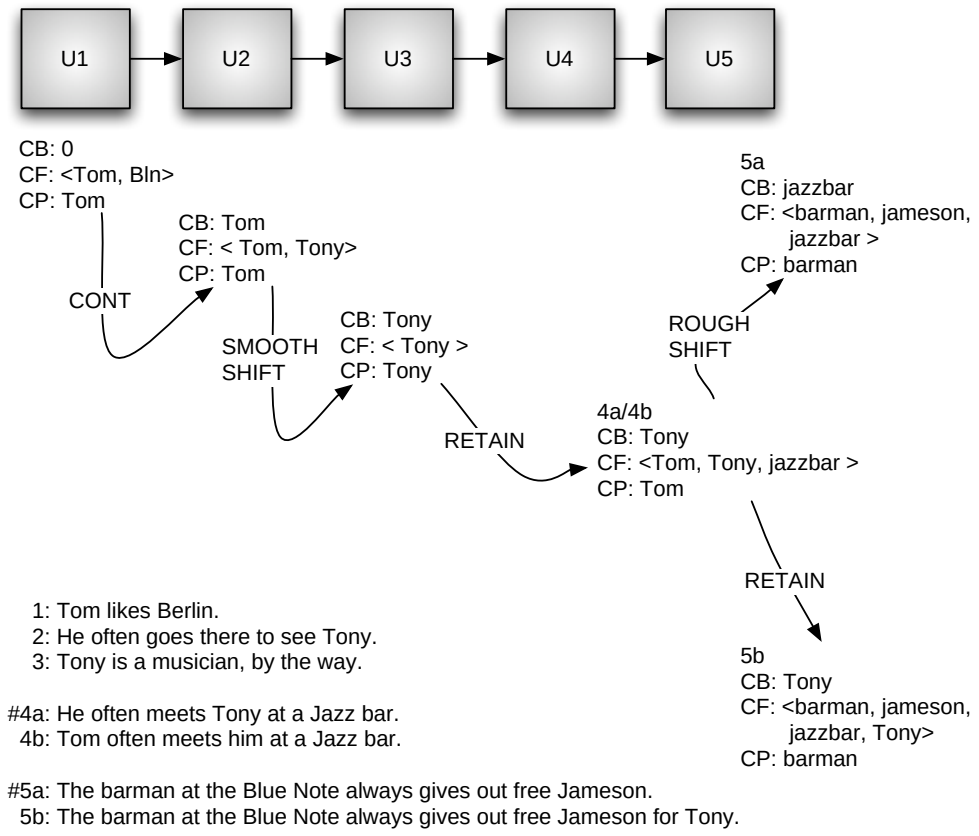


Figure 5.2: An example of how centering explains local coherence. Utterances marked with # are assumed to be less acceptable in their context, in the case of 4a) because it violates the pronominalization rule, and in case 5a) because the RETAIN transition is preferred to a ROUGH SHIFT.

(Kibble, 2001)? This would mean that Salience is stronger principle than Cohesion, as Kibble suggests.

An alternative explanation is that coherence is only one goal among many. A natural language realizer, in particular a hybrid one that relies on a small set of syntactic constructions, has limited expressivity. Dialogue goals need to be pursued, among them the confirmation of understood voice commands, or the correction of factual mistakes an interaction partner has made in tutoring applications. These goals may be more important than generating perfectly coherent discourse.[7]

Therefore, coherence measures are a good candidate for a *soft constraint*, to be included in the MUG fitness function. Obviously, weights are domain-specific and will have to be trained on a corpus or adapted iteratively. Soft constraints may be broken down in subgoals, for example Cohesion and Salience.

Not only the ranking of transitions has been subject of intense discussion. Originally, the ranking of grammatical functions in CF was a language-specific element of the model. Furthermore, it is unclear what constitutes an utterance, or whether centers are aligned among discourse participants. Such model parameters may be specific to language or genre.

5.2.5 Centering in MUG

Local coherence seems to govern many syntactic decisions, and it plays an important part in the generation of referring expressions. A sensible way of generating more coherent text is by means of *planning*: before a sentence is realized, choices affecting coherence are made for a sequence of sentences, not just for a single one. Planning and realization can then be seen as separate system modules.

This architecture makes hard and early decisions, rendering it efficient. However, the planning stage knows very little about the realizer's options. A perfectly coherent text plan may not work out, because the linguistic means to realize it are missing from the grammar. For example, it might be coherent to realize a certain discourse entity again in an utterance, but not as a subject. If there is no combination of grammar rules for the specific dialogue act, which would realize the entity in non-subject position, the process would fail. This is a realistic scenario, given that most application-tailored grammars contain templates rather than a full range of linguistic rules.

From a linguistic point of view, an architecture that separates planning from realization ignores the underlying consequences that planning decisions have. For example, coherent text might be longer. It is reasonable to assume that human speakers would sometimes forego perfect coherence in favor of a more concise discourse. This might be another explanation as to why there are more SMOOTH SHIFT than RETAIN relations in many corpora.

An implementation of centering directly in the generation grammar (MUG) allows for coherence judgments to be made during the realization process, before the final

[7] Otherwise, humans or voice interfaces might as well stay quiet, since silence arguably requires the least effort in parsing!

output has been generated. Obviously, only the transition from the previous to the currently generated utterance can be deduced. The planning implemented here does not include aggregation (the process of accumulating information to be realized in each utterance) or the choice of rhetorical structure, both of which would increase the search space (in the search for the optimal solution) exponentially.

One possible implementation of Centering in a unification-based framework shifts the perspective from an utterance-centric one to an entity-oriented one. While the original formulation defines several centers for each utterance, these data structures can be implemented as mostly binary values for each discourse entity (Figure 5.3). For each entity, such value stores whether the entity is the backward-looking center (CB), the preferred center (CP) and how it is ranked in the forward-looking center (CF). (The CF information is not binary.) A further mechanism has to ensure that there can be only one backward-looking center per utterance. This information is kept directly in the semantic descriptors of each entity, that is: on the MUG blackboard. Additionally, a further structure in the `previous` branch notes, for each entity, the state of that entity in the previous utterance.

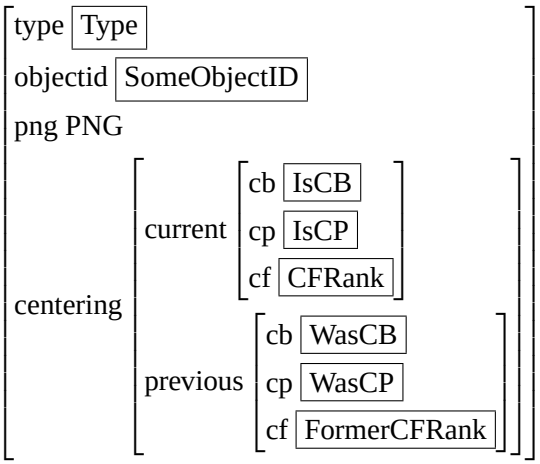


Figure 5.3: The centering structure stores the attentional state of a particular discourse entity.

IsCP and IsCB are set procedurally, as soon as enough information about the discourse entities in the current utterance becomes available. For example, if the grammar realizes the subject of a sentence, IsCP can be set for all discourse entities. IsCB can be set as soon as it is clear that no higher-ranked entity in the previous utterance is realized in the current one. CFRank is set by the grammar components that determine the syntactic realization of an entity.[8]

One way to determine if a certain transition holds for the current utterance is to attempt unification of each entity with one or more specific AVMs defined for the

[8] In this implementation, low values indicate a high CF ranking by convention.

transition. This unification carries out the checks of Cohesion and Salience. Essentially, it compares the centering information stored for the particular entity in the previous and the current utterance. Figure 5.4 contains the specific AVMs pertaining to the RETAIN transition. These AVMs state that the CB needs to be retained (Cohesion): `cb` is either 1 or 0 for both the current and the previous utterance. However, the CB is not realized as CP in the current utterance (no Salience): `cp` is 0, in case the entity is the CB.

$$\left[\begin{array}{ll} \text{objectid} & \boxed{1} \\ \text{centering} & \left[\begin{array}{ll} \text{current} & \left[\begin{array}{ll} \text{cb} & \mathbf{1} \\ \text{cp} & \mathbf{0} \end{array}\right] \\ \text{previous} & \left[\text{cb } \mathbf{1}\right] \\ \text{utterance} & \left[\text{cb-obj } \boxed{1}\right] \end{array}\right] \end{array}\right] \vee \left[\text{centering} \left[\begin{array}{ll} \text{current} & \left[\text{cb } \mathbf{0}\right] \\ \text{previous} & \left[\text{cb } \mathbf{0}\right] \end{array}\right]\right]$$

Figure 5.4: Disjunction of AVMs: each discourse entity of an utterance needs to unify with one of these two AVMs for the RETAIN transition to hold.

Obviously, an alternative, procedural check could be carried out instead. Such a check allows for a simple constraint implementing the pronominalization rule. Whether represented by AVMs and unification, or other forms of constraints, coherence can be measured at some point during the realization process. It can be used to guide the search for an optimal realization variant in the form of a *soft constraint*, which the generation process optimizes. Therefore, the transition detected should influence the score of the generated output. Augmenting the fitness function defined in Section 4.3, a schematic version would be:

$$s = \text{efficacy} - \text{cognitive load} + \text{coherence}$$

Efficacy, cognitive load and coherence need to be normalized to a common scale. The normalization factor for coherence is then a system design parameter: how important is *coherence* compared to the ease of communication of a single utterance (cognitive load) and the efficacy of the system?

5.3 Conclusion

This Chapter gave an introduction to cross-modal and discourse coherence. Cross-modal coherence was motivated by contextual evidence of comprable phenomena, and implemented directly in MUG. Discourse coherence was demonstrated using linguistic data, and a model that explains some of the phenomena was given with Centering theory. Centering can be implemented in MUG. It integrates elegantly with the hybrid hard/soft constraint approach followed in this thesis. The implementation allows decisions to be made about the coherence of the currently generated utterance as early as possible, without reducing the flexibility of the grammar.

Further work needs to unify cross-modal and discourse coherence. If we suppose an element on the screen is highlighted (or, an interlocutor points at something). Does this constitute a separate utterance in the Centering sense? If accompanied by verbal discourse, probably not. However, how is a visually salient element ranked in the CF? Possibly higher than the verbally realized entities. Only corpus data could hint at the possibly complex interaction between discourse entities presented in different modes.

6 Generation as a Constraint Optimization Problem

Multimodal Functional Unification Grammar uses soft, violable constraints to optimize appropriateness in a given situation. The technique comes at a cost: efficiency. Because we cannot exactly determine appropriateness, before the final output is generated, we need to *search* for a good variant.

Efficiency is an important problem for natural language generation systems, if they are applied in a dialogue system context, where near-realtime performance is needed to give prompt replies to the user. Natural language generation systems have long operated with maximally specific dialogue acts and constraints that were hard rather than violable. Other systems, producing discourse instead of single utterances in a dialogue systems, simply take their time to come up with a result. With constraint optimization as followed in this thesis in the context of a dialogue system, *efficiency becomes an issue*.

A hybrid system using hard and soft constraints like the one presented in the previous chapters can only run efficiently if the right methods are used to find the best output variant. Then, however, the results are promising. While they do not allows us to generate complex dynamical output on today's embedded systems like PDAs, they can be used on server-client-architectures like the one employed in FASiL's Virtual Personal Assistant, where output is generated on a powerful, dedicated server.

To address the issue, we can recast the search for a good output variant as a *constraint optimization problem*. The *solution* to this problem is a set of grammar components that are applied to form our final output. The grammar defines *constraints*, mainly through attribute-value matrices and the implicit unification algorithm. These constraints need to be fully *satisfied* by any solution. Finding the best solution that violates the fewest number or the least important soft constraints means *optimizing* it. Solving constraint satisfaction problems, in the general case, is an NP-complete task, let alone solving constraint optimization problems, which are NP-hard (Meseguer et al., 2003). However, heuristic functions which estimate the final cost, given a partial solution, can help achieve good performance.

Various strategies to solve constraint optimization problems have been developed in Artificial Intelligence research, and which ones are promising depends on the nature of the specific task. In this chapter, I will review well-known solutions to search algorithms and apply them to generation with soft-constraint-based functional unification grammars. Performance figures for these algorithms in the context of natural language generation will be presented.

Apart from efficiency, a further desirable property of some of the algorithms presented in this chapter is that they allow *anytime* search. That means that the realization

process can be stopped at any time, and a complete output variant is returned: the one that is judged to be the best i.e. most appropriate at the time.

6.1 An efficient implementation

The steps involved in generation with a MUG can be summarized in three types of actions.

1. Identifying grammar components (rules) which unify with part of the input-structure (i.e. satisfy the hard constraints) and, as a set, yield the optimal or a good solution according to the fitness function (i.e. optimize towards the soft constraints). This involves multiple executions of the next step.
2. Unification of each selected grammar rule
3. Evaluation of functional expressions

Techniques to efficiently implement the unification (step 2) of feature structures commonly employ destructive unification, which avoids the copying of structures. In the MUG implementation, unification is additive, so that it is reversible when needed by backtracking. Profiling experiments with the implemented algorithms were conducted. They showed that unification accounts for less than a fourth of all processing time, if destructive unification and a stack-based backtracking framework is used.

The evaluation of functional expressions (step 3) is optimized in that shared functional expressions are only evaluated once. Evaluation is cached across solutions (generation variants). Lazy evaluation techniques, which only evaluate partial expressions when needed, would not help us in MUG, since the final results, e.g. of string concatenation operations, are needed by the fitness function.

To identify the right grammar components (step 1) and apply them in an efficient order is a more complicated matter altogether. A range of techniques can be applied – some of them result in significant improvements in efficiency. The search techniques discussed here apply to other natural language generation systems with optimization elements. Therefore, they will be present in more detail in the remainder of this chapter.

6.2 Formalizing the problem - the search tree

The generation algorithm recursively unifies a grammar component with each constituent substructure of the current blackboard structure. Therefore, it faces a choice point whenever there is more than one unifiable component available. The goal of the algorithm is to find the single combination of choices that minimizes the fitness function[1]. Following common practice, I view the search as a process whose state can be represented in tree

[1] I follow AI practice is using a *cost* or *penalty* in this discussion, while the fitness function as defined in Chapter 4 uses a score. The best solution is the one with the lowest cost, or the highest score.

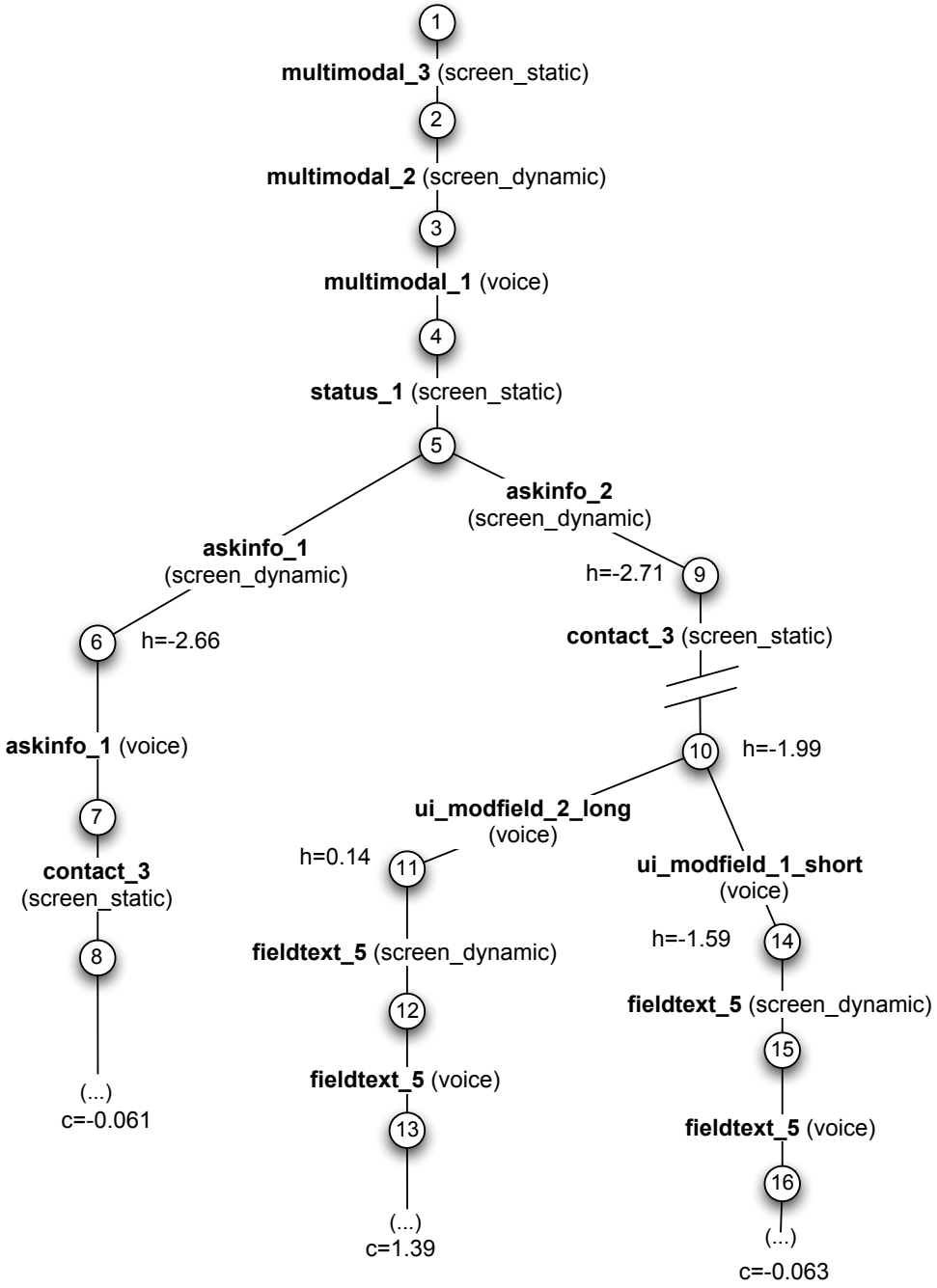


Figure 6.1: A partial search tree. It shows the first steps to generate a multimodal output for an ASKINFORMATION dialogue act. Modes are given in parenthesis.

form (Figure 6.1). Each choice point is depicted with a node in the tree, with one leaving arc (to daughter nodes) to each possible choice. The leaves (terminal nodes) of the tree represent partial solutions during the process. The choices leading to the solution are encoded in the path from the root node to a leaf node (see Figure 6.1).

A search algorithm can be optimized in different ways:

1. The overall order in which the terminal leaves of the tree (partial solutions) are explored further (see Sections 6.3, 6.4.1, 6.4.4)
2. The order in which the arcs of a single node are explored (see Section 6.4.3)
3. The order in which the free variables are instantiated, i.e. which remaining unexpanded constituents on the blackboard are expanded next (see Section 6.5.1)
4. which terminal leaves are selected for pruning (not explored further) (see Section 6.4.2)

In the remainder of this chapter, common implementation techniques will be discussed and applied to the generation problem. Many of the underlying search algorithms are classical (see, e.g., Russell & Norvig (1995)), others have been investigated more recently. For a formal overview, see Meseguer et al. (2003). Results of the different algorithms for the Figure 6.2 compares runtimes of each of several combinations of search strategies for eight test cases taken from the *Virtual Personal Assistant* application.

6.3 Depth-first backtracking search

Depth-first backtracking search can be seen as a default search technique. It simply explores each first available choice for each node until a solution is found, receiving no direction in which paths to explore when. The leftmost arcs of the search are always build first: nodes are created in ascending order as enumerated in Figure 6.1.

If at some point in the search, no choice is available that would satisfy all prior constraints, the search algorithm makes a set of choices undone: it *backtracks*, until the next earlier choice can be revised. Choices are undone in a last-in-first-out fashion: first, the current node is explored. If no further choice is available at this node, the next choice farther up in the tree is undone in a last-in-first-out fashion, and so forth.

In backtracking search, the effect of choices is usually encoded in elements on a stack, so choices may be undone by removing elements from the stack. In MUG, this means that only the changes introduced by a unification are stored on the stack. Rewinding to an earlier state is cheap. Still, depth-first in general is costly (time), as it explores all solutions completely before looking at any soft constraints. Space requirements only depend on the size of the currently calculated solution, as competing solutions are not stored concurrently.

As a default, we assume that structure sharing does not imply structure copying. The implementation avoids copying unless necessary. Thus, constraints imposed by the

instantiation of variables – i.e. constraints imposed by a unification process – are propagated and checked immediately. If unification succeeds, the new constraints imposed by a grammar component are compatible with the solution found so far.

6.4 Methods based on a heuristic function

Most prioritization involves a heuristic that predicts the final cost, given only a partial solution.[2] If a heuristic is *admissible*, it never overestimates the cost, ideally giving a lower bound of the cost. Commonly, such heuristics are difficult to come by. The quality of the heuristic influences the efficiency of the algorithm. The faster the heuristic converges to the final cost, the better.

The admissible heuristic function used for the experiments discussed here (and for the UI on the Fly system) calculates the lower bound of the predicted final score. It is equivalent to the fitness function, except that for unknown values (e.g. `realized` attributes), we assume such values which minimize the cost. That means that we assume that all semantic entities are realized (`realized` attribute is 1), and that the produced textual output is empty, unless it is already instantiated. (Concatenations and text templates are factored in as much as possible). A non-admissible heuristic would not make these conservative assumptions. It would rather assume realistic defaults, in particular for the length of the produced text.

Heuristic costs are given for some nodes as h in Figure 6.1, final costs are c. The figure shows part of an actual search tree from a generation process with MUG.

Calculating the heuristic costs time, however, as Figure 6.2 shows (bars 2 and 4 from the left, whose basis only differs in that the heuristic is calculated and a bound is applied): if it is calculated (as in bar 4), but does not lead to a sufficient reduction in actually explored partial solutions, it will increase overall runtimes. The admissible heuristic, for example, does not rule out many variants. A non-admissible heuristic (not shown) brings the desired effect, but optimality cannot be guaranteed, which would make comparison of efficiency more difficult. The same effect can be seen for the best-first algorithm as described in the next section: when the bound is calculated (bar 3), overall runtimes are higher than otherwise (bar 1).

6.4.1 Breadth first, best first and beam search

Breadth first search explores all paths (and partial solutions) in parallel, thus requires keeping several choice stacks to note the effects of choices (i.e. the additional constraints). Nodes in Figure 6.1 are created in this order: 1, 2, 3, 4, 5, 6, 9, 7, 10, 8, 11, 14, 12, 15 (...).

[2] In the search tree, partial solutions are such tree leaves from which branches can originate (downwards). The branches haven't been explored yet and are not known beyond the fact that they exist. Branches exist in the MUG expansion algorithm if there are unexpanded constituents on the blackboard.

Best first search (BFS) increases the chances of finding a good solution at an early point of time. It explores the most promising leaf first, using a fitness heuristic. In this case, the search may be stopped at any time, yielding some (presumably good) result.

The caveat here is that these search methods have untractable space requirements, but can be improved by only looking at the best k searches at a time (beam search) and stopping when the first solution is generated. This search is not optimal, which means it is not guaranteed to find the best solution. However, it may represent a good trade-off between memory usage, runtime and quality of the solution.

Unfortunately, it has proven insufficient in the MUG algorithm to simply discard all the partial solutions that fall outside the search beam: whenever hard constraints cause a search path to be cancelled at a later point, the search beam becomes narrower, because previous solutions cannot be retrieved any more. Cancellations are frequent in MUG: either because there is not unifying grammar component available for a certain partial solution, or, more commonly, the lower bound (admissible) cost heuristic raises above the best previously found solution (see next section). Therefore, our beam search is only practical without bounding, and with a "safety net" that resorts to another search method, if no solutions are found.

An attempt was made to dynamically increase the beam size and re-run the search with the new beam size, until a solution is found. This method could be termed *iterative broadening*. Unfortunately, it showed only limited success. This applies to an increase of beam size (in terms of an increased number of solutions), as well as to an increase in threshold from the highest beam edge, so that variants with a higher predicted cost can be kept within the dynamic threshold. One reason for the failure of such an algorithm may be that the search often fails (cancels all the solutions in the beam) at late stages, causing the search to duplicate many steps. Also, with increasing beam size, more and more structures need to be stored, which is costly.

6.4.2 Branch & bound

Branch & bound (B&B) uses a standard depth-first search until the first solution is found. Then, in the standard backtracking fashion, further solutions are explored. However, partial solutions are continuously evaluated by the heuristic function. If a heuristic cost rises beyond the actual cost of the best previously found solution, the partial solution is pruned. For example (Figure 6.1), the branch originating from state 11 can be pruned, if the solution with cost -0.061 is already known.

If used with an admissible heuristic, this algorithm is optimal. For the MUG algorithm with an admissible heuristic, however, the improvement through bounding is eaten up by the additional cost of calculating the heuristic cost. In fact, very few search paths are pruned, as the lower bound converges only slowly towards the final cost.

Only if we use a non-admissible heuristic, which sometimes may overestimate the cost, we see a significant improvement. The heuristic used is good enough to not prune the best paths, so while we are not guaranteed an optimal solution, we probably receive a good one.

6.4.3 Leaf ordering (local best first search)

It is important for the branch & bound algorithm to quickly reach a first solution. This solution should be a good one, i.e. have a low cost, in order to allow the algorithm to prune a high number of search tree branches without exploring them further. Combining the B&B with *local best first search* (LBFS) represents a step in just this direction. This search has the advantage (over BFS) that its memory requirements will not get out of hand. LBFS optimizes *locally*, choosing the best next arc from a single node before its siblings. The advantage of LBFS over LBS is that there are fewer open alternatives with costly memory needs at a time. Only the branches originating at this node need to be stored at one time, and standard backtracking can be employed to explore all of them.

For example (Figure 6.1), LBFS would prioritize state 9 over state 6, because the heuristic cost of 9 (h=-2.71) is lower than the one of 6 (h=-2.66), and similarly for state 14, which is preferred over 11. However, other than in BFS, once 9 is chosen, it will be fully explored: upon reaching 14, the algorithm does not jump back to 6, just because the heuristic shows a higher cost (h=-1.59) than what node 6 (h=-2.66) had to offer. The first solution found is the one with cost -0.063: a good solution, yet not the best one. In the following backtracking, we can immediately prune state 11.[3]

Local decisions do not generally lead to global optimization. Consider a case where the grammar may go for a "simple" component, which realizes the utterance directly with canned text and incurs a penalty for not realizing some important detail in its output. Alternatively, a more compositional component may be chosen, which leads to the use of many others, building the utterance phrase by phrase, word by word. The second choice may seem better in the eyes of a non-admissible heuristic, as it seemingly does not incur penalty points (yet). Therefore, it is explored first. In the long run, however, the final result may still be more costly than the "simple" one.

So, combining LBFS with a global cut-off (generating only the first k solutions) often misses good solutions, in particular if they are introduced by early choices. However, in general it results in better and earlier first results.[4]

6.4.4 Iterative deepening

Iterative deepening (Korf, 1985) brings a significant improvement for natural language generation with soft constraints. The general idea of iterative deepening is to start a branch & bound search with an initially too low cost bound, so that all (or most) partial solutions are pruned using the heuristic. Then, search is repeated with an iteratively increased bound. As soon as at least one solution is found, iteration stops, and the best solution found during this iteration is returned.

[3] In larger searches, it is obvious that a lower initial bound allows us to prune more branches than a higher one would do. In this case, we would prune at node 11 even if leaf 8 would be reached first.

[4] An interesting extension would be to explore other ways of prioritizing decisions, such as using regression to estimate the score rather than a heuristic, or looking at the average number of subsequent choices. This would effectively prioritize *early* instead of *good* solutions.

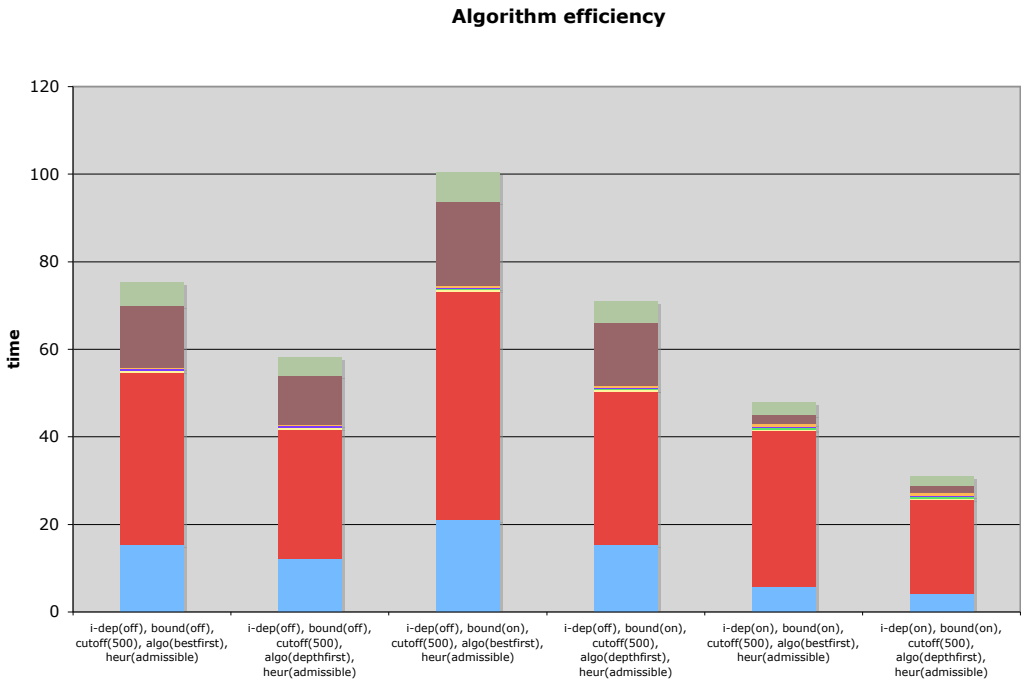


Figure 6.2: Runtimes for various combinations of optimization techniques. (Lower numbers are better.) Colors encode different test cases. Only optimal algorithms are shown. The performance of non-optimal algorithms depends largely on their non-admissible heuristic function, and the quality of their output would have to be quantified.

Just like in a standard B&B search, the algorithm prunes solutions, whose most optimistic score estimate (the lower cost bound) is below the initial bound, and below all other solutions that might be found during the same iteration. Therefore, if an admissible heuristic is used, the algorithm is optimal.

One might think that the repetition of search steps will impede efficiency. Luckily, it does not. Searches with a low initial cost bound in MUG tend to fail early, which is why the first iterations (which do not return a result) are not time-consuming and need no optimizations as proposed by Reinefeld & Marsland (1994).

Iterative deepening does not depend on an early good solution, since all solutions are explored. It would be reasonable to expect an additional benefit from early good results, since additional bounding as in the standard B&B algorithm could still be enforced. However, when we employ LBFS as proposed earlier, we see its small benefit eaten up by the cost of copying local partial results during LBFS.

Experimentally, the diagram in Figure 6.2 shows consistently lower runtimes for iterative-deepening search (abbreviated *i-dep* in the legend).

6.5 Further methods

Iterative deepening depth-first branch & bound search has proven to be the most efficient search method in MUG. There are other methods which we will describe briefly including their drawbacks that made their exploration for MUG unlikely to produce favorable results.

6.5.1 Variable reranking (or: minimum remaining values)

The previously discussed optimization methods make changes to the order in which partial solutions are explored further, or they cut them short. The following optimization, in turn, takes care of another choice the algorithm has to make. To discuss this, we will focus on the task of finding components to unify with each constituent structure, as mandated by the MUG principles.

After the MUG application algorithm has identified all constituent substructures on the blackboard, they may be expanded in any order, and also the modes that are chosen may be expanded in any order. The expansion often means to make a choice between several possible grammar components. *Minimum remaining values optimization* (MRV) expands those constituents first, which have the fewest alternative choices. MRV focuses on the most constrained variable, and, in the context of constraint optimization problems, choosing a component can be seen as the equivalent of instantiating a variable.

SUPPLE achieves better efficiency thanks to MRV in two out of three problems (target devices) (Gajos & Weld, 2004). Unfortunately, MRV is not very effective in MUG. One reason might be the that the particular grammar used to test allows for a fairly free combination of components: the only effective constraint imposed by ASKINFORMATION on the REFEXP component that realizes some referring expression is the morphosyntactic case. Also, such constraints may be introduced by either component: if the REFEXP component instantiates the value for *case* first, it effectively constrains the choice of ASKINFORMATION.[5]

In SUPPLE, the list of variables is much larger. Different UI elements need to be realized either way. In MUG, the lower-level components are not known before the higher-level components have been selected. Therefore, the variable ordering is skewed towards a *top-down* expansion.

[5] Actually, no askinformation component would be available for, say, a dative referring expression for certain verbal complements. But since unification is fairly cheap in MUG, this wrong choice has little consequence.

6.5.2 A* search.

This search method traditionally involves calculating a lower bound for the cost of future choices, and add it to the cost of previous choices. Many classical constraint optimization problems such as finding the shortest path in a graph with weighted arcs[6] can be solved elegantly and very efficiently. However, A* search suffers greatly from memory requirements: with an increasingly bad heuristic function, it closes in on breadth-first search and becomes unmanageable. Unfortunately, the MUG heuristic function is not very good, as is the case in many real-life problems. Applying the Iterative Deepening technique to A* search alleviates this problem, but still requires the current partial solution (i.e. the blackboard) to be copied time and again. Similar to global-best-first search, it would require substantial changes to the backtracking nature of the algorithm.

6.5.3 Grammar compilation.

The MUG algorithm starts out with the input specification (dialogue act) on the blackboard. An expansion would work equally well without this specification – it would be minimally constrained, generating all possible output variants for all possible inputs. These variants could be saved at compile-time. At run-time, those structures which unify with the input specification may be scored and ranked. (We simply delay the enforcement of some constraints imposed by the input specification, while pre-compiling all other constraints.) The selection of matching structures may be organized efficiently with a decision tree algorithm (e.g. binary search), which typically gives logarithmic time complexity. For a small domain, this would be very manageable – the FASiL VPA grammar's search space is estimated to be less than 10000 solutions. However, for linguistically more sophisticated grammars, as implemented in functional unification manner and otherwise, or in broader domains, combinatorial explosion might make this approach unlikely to succeed. One should keep in mind that the functional evaluation step still needs to be carried out for all solutions with the dialogue act data instantiated, and this step is expensive.

6.5.4 Obtaining an initial bound by choice classification.

Cabon et al. (1996) have proposed to apply *Mean Field Optimization*. This means to run several iterations of the optimization process as a pre-processing step. This gives a probability for the variable-value assignments. As component choices can be seen as the variables in the MUG context, one could choose the most likely components first, produce a solution and use the cost of this solution as an initial bound. This addresses the need of Branch & bound algorithms for an early and good bound. In an adaptive system, a simple maximum-likelihood estimation might not be ideal. A number of properties (e.g. about the current situation) would have to be taken into account. Training a

[6] In more down-to-earth terms, such a graph could represent a road network with way points, where road sections differ in length and speed limit.

classifier with a large feature set seems to be a promising approach to obtain an initial lower bound.

6.5.5 Improving the fitness heuristic by regression.

Classifiers such as Support-Vector-Machines (SVM) are able to handle a high number of features, while dealing with sparse data well. They have been applied with some success to linguistic tasks, such as text classification (Joachims, 1998) or rhetorical analysis (Reitter, 2003b). While a fitness heuristic is usually a function motivated by intuition and trial&error, an SVM used for regression would give a theoretically sound estimate based on a corpus of grammar applications rather. It seems reasonable to expect that this would be not only better motivated, but also more precise than a bound derived directly and solely from the fitness function. Further work is needed to explore machine learning of heuristic functions.

6.6 Conclusion

I have applied a number of constraint optimization algorithms to the special problem of generating natural language output with MUG. As the results (Figure 6.2) demonstrate, consistent and substantial improvements in efficiency can be achieved using a bestfirst/depthfirst iterative-deepening Branch & bound search. If an admissible heuristic is used, the algorithm is optimal.

An efficient generation algorithm is not enough. The grammars used must allow the algorithm to make an early estimate of a solution's appropriateness. Hard constraints must reduce the search space enough. The test cases used here demonstrate this. Variance in generation time is high. Adaptive systems are likely to succeed in real-life if they foresee future dialogue steps and pre-generate output as templates, rather than doing so in real-time.

7 The MUG Workbench: A Development Environment for Generation Grammars

7.1 Introduction

When grammar-based techniques for natural language generation (and analysis alike) find their way into collaborative projects or actual application, big grammars tend to become hard to extend and debug. The MUG system provides a new tool set with a graphical debugging environment for functional unification grammars, which is designed to help grammar developers inspect the results of their work.

The particular formalism supported is Multimodal Functional Unification Grammar (MUG), as discussed previously in this thesis. When compared to other natural language generation grammar formalisms, one of the distinguishing features is that MUG supports several coordinated modes, such as voice prompts or structural and/or language-based screen displays. Furthermore, MUG is designed to over-generate: For each input description, the grammar produces a range of coherent realization variants, which are ranked by a scoring function in order to optimize the output towards situational and device-related factors.[1]

7.2 System overview

The MUG System is a development tool that consists of several components. The *MUG Formalism* is a grammar specification syntax. The *MUG Engine* handles the generation and adaptation process and offers interfaces to connect external components. *MUG Workbench* is a graphical development environment. The workbench works as an inspection tool, which runs a test case, generating all possible *variants* of the output, and then gives access to the various steps taken during content realization. This was found to be faster than the step-by-step execution in a graphical debugger.

7.3 Debugging grammars

Unification-based formalisms are generally difficult to maintain and debug. There are various reasons for this.

[1] This chapter is an extended version of Reitter (2004).

- Grammars are considered independent of a parsing or generation algorithm. They declare licensed structural operations. How a complete structural description is derived from an input word (in parsing) or how some structure is built up to achieve output (during generation), is up to the algorithm. Grammar development should be independent of the algorithm, therefore, it should play no role in debugging. Consequently, there is usually no clear temporal (step-by-step) view of the grammar application process.

- Unification is destructive. Once a unification operation is completed, the resulting structure does not contain information about the origins of the data. *Which grammar rule (component) instantiated feature* xy *in the structure? Why did these structures not unify?*

- Unification-based formalisms make heavy use of structure-sharing. With this method, a higher-level grammar rule (component) specifies what information is available to its daughters. This, however, makes the structures difficult to visualize and inspect.

Unification-based grammar development tools address these issues in various ways. *Linguistic Knowledge Builder* (LKB, Copestake (2001)) for instance allows users to attempt unifications and gives verbal messages if they fail. It shows tree structures to visualize parses. Users can inspect lexical entries and type hierarchies.

MUG differs from the typed feature structure grammars used with LKB in several ways. MUG is designed for generation rather than parsing. MUG uses only weak types. Important ideas in MUG include over-specification and multimodality. The differences warrant a slightly different approach to grammar development.

The two most important questions a grammar developer asks when debugging are: *Why was a specific undesired output generated?* And: *Why was a certain desired output not generated?* In the MUG Workbench, developers use the *variants view* to answer the first, and the *log view* to answer the second question.

7.3.1 Inspecting variants of output: variants view

In the *variants view* (Fig. 7.1), these variants may be inspected and compared: the workbench lists, for each variant, the components used.

One common problem with the grammar used in the FASiL project were structural ambiguities that did not lead to an ambiguity in the output. In most cases, this was caused by semantic information being used for the generation of text that was, in the end, not used for the final product. This clearly seems undesirable. The workbench shows such structual ambiguities and lists any differences in the subsets of the components that were used to generate the ambiguous variants. The value of each component – as it occurs in

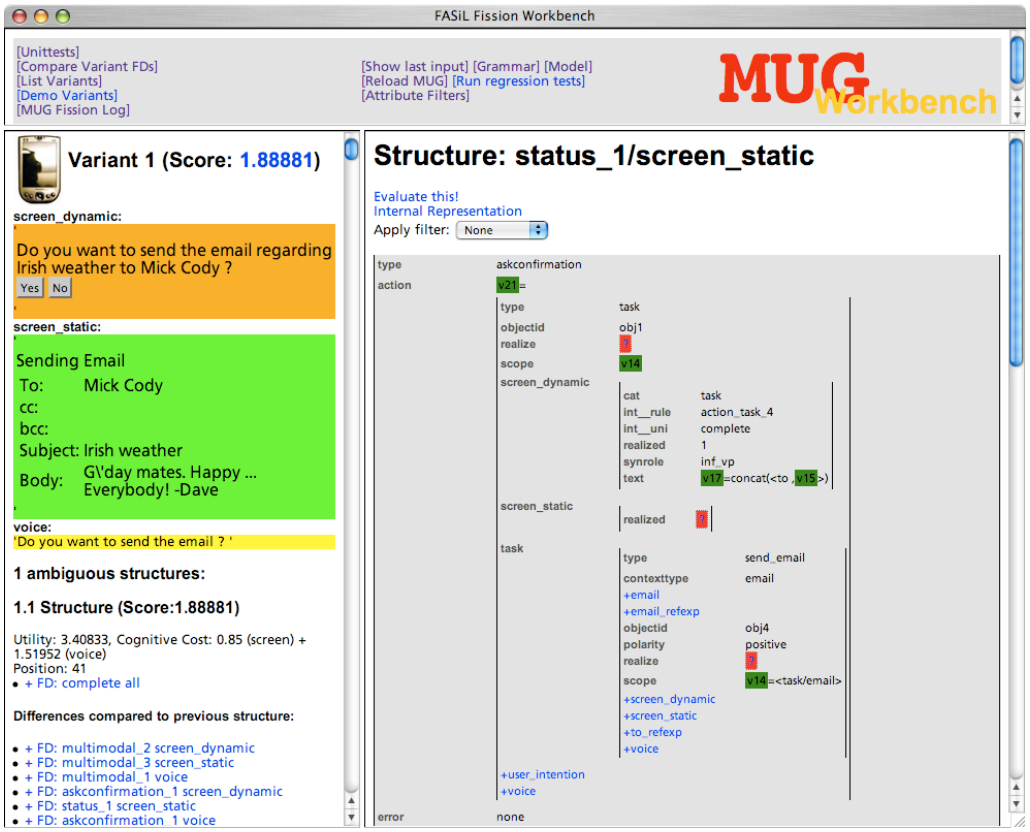


Figure 7.1: Variants and a large AVM. Attributes can be collapsed.

the final, overall AVM[2] – may be inspected. Generally speaking, the variants view is a good way to deal with faulty or extraneous variants.

Misspelled variable names, but also variables in the wrong positions in AVMs are another common source of errors. Such variables remain unbound. The workbench marks them clearly in the display. The developer can also inspect variables easily and collapse or filter the rather large feature structures. Syntax errors are shown when the grammar is loaded via the workbench user interface.

Soft constraints as formulated in the fitness function may be inspected, too. Since they are noted in the overall AVM, the particular fitness function may be viewed by viewing the AVM. Device and situation models can be selected (see Figure 7.2).

[2] In the screenshots, AVMs are called FDs, which stands for *Functional Description*. This is a reference to the underlying linguistic Functional Unification Formalism and the associated terminology. AVM and FD are equivalent for our purposes.

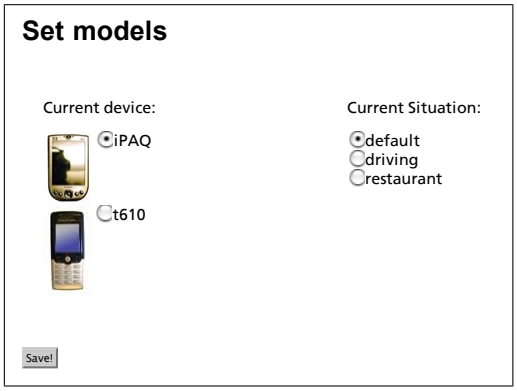


Figure 7.2: A choice of device and situation models is available. These are extensible.

7.3.2 Tracing the steps of the generation algorithm: log view

This view of the process is purely logical: there is no conceptual time-line in unification-based grammars as in procedural programs. We found that a more procedural view may help to spot problems with variants that failed to come up, furthermore it is a way to spot efficiency bottlenecks or to simply learn about how the formalism works. We offer a *log view* (Fig. 7.4) that enumerates all the steps that the MUG interpreter takes to apply a grammar to the input. These steps are shown for each variant of the output. This view allows inspection of the state of the sub-substructures as they were before and after a component (for a given mode) was applied.

A useful feature in this view is the marking of steps that were undone by means of backtracking, because – at a later stage in the generation process – an application of a rule failed. In many cases, the cause of the failure is a bug in the grammar. In other cases, it is desired behavior, but computationally inefficient. Such effects are visualized in the log view.

When unification fails, the grammar developer may want to know, why. Typically, this is the case when a desired variant is not generated by the grammar. Every AVM can be manually unified with any grammar component. The workbench uses a relaxed notion of unification: the process always succeeds. However, conflicting features are visually emphasized (see Figure 7.3).

7.4 Applications

The MUG Workbench aided two experienced and one novice grammar writers to create a multimodal UI for personal information management (handling e-mail), which contains 126 components (190 with disjunction compiled), and a second, smaller MUG (39 com-

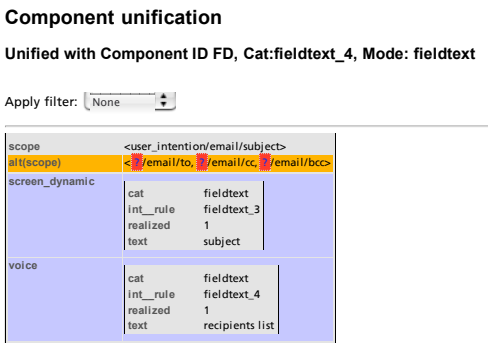


Figure 7.3: Alternative unifications (rule choices) can be attempted from the log view. A typical question that can be investigated with this method would be: Why did a particular component not unify with a structure?

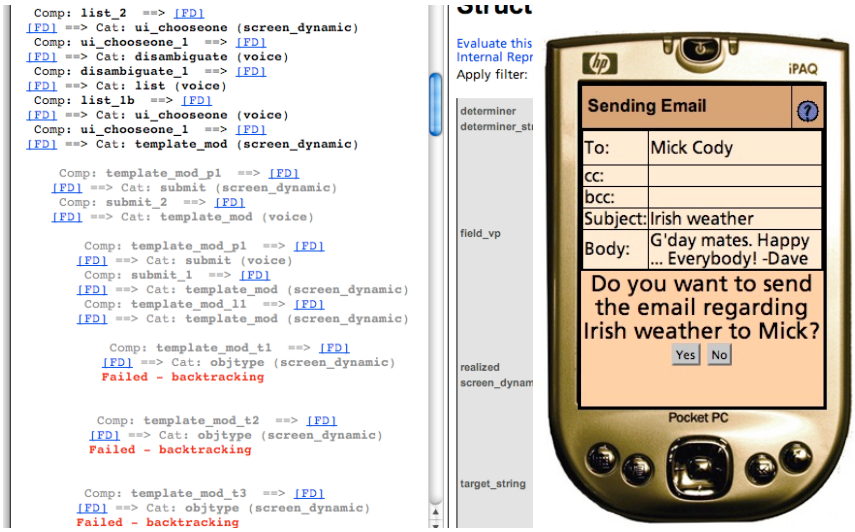
ponents), which generates a short coherent discourse with pronouns. To test (see Panttaja et al. (2004)) and also demonstrate the grammar, multimodal output can be made on any networked device (e.g. PDA) with an HTML client, with text-to-speech voice rendered on the server, as well as on simulated devices on the local workstation. A prototypical dialogue system, not described here, has been implemented in Java to demonstrate the use of MUG through defined application interfaces in a heterogeneous environment.

7.5 Availability

The MUG Workbench has been made available in source form at

http://www.media.mit.edu/d̃reitter/mug/

It runs on Unix and Windows architectures. The Workbench and all dependencies are available for free under open-source licenses.



(a) (b)

Figure 7.4: a) In the log view, steps taken back during backtracking (because they didn't lead to valid solutions) are greyed out. b) life-size results can be demonstrated from the workbench (design: E. Panttaja)

8 Conclusion and Outlook

I have discussed an approach of over-generation that is hybrid in two ways. It can combine canned text and fine-grained, linguistically motivated generation. Secondly, it combines the hard constraints that a grammar encodes with soft constraints encoding utility and cognitive load. It implements the idea that communicative choices take a variety of unequally weighted factors into account. These factors are formulated in grammar and fitness function, separately from the control strategy that tries to find the optimal solution.

I have presented a formalism to encode the grammars and algorithms to efficiently implement the system, based on the well-researched unification grammars. Soft constraints have been motivated and formalized as a fitness function. An evaluation with naive test subjects showed that the predictions of the soft constraints increase perceived efficiency of the system. A practical, real-life application around personal information management provided the basis to develop, demonstrate and evaluate the system.

Beyond the immediate needs of this application, I extended the notion of coherence from cross-modal to discourse level, and described a practical application to design grammars. On a practical level, the thesis presents an implemented, reusable and user-friendly system that allows fissioning and realization multimodal output and debugging and extending the grammars.

There are lessons to be learned from the work on and with the current system.

8.1 Content selection depends on further factors

The algorithm shown incorporates two functions into a generation grammar: content selection, and the distribution of content to modes. Doing so, it bases its decision on the following critera.

- Input specification of the dialogue management component. The dialogue manager may specify that certain semantic entities should be realized.
- Utility of the output.
- Cost for the user to read or listen to the content presented.

Using these criteria, the algorithm can render output so that it is useful and does not overwhelm the user. Content is presented in a mode that makes sense given the user's situation, and it adheres to the requirements of the dialogue, for example when content needs to be confirmed and therefore rendered to the user.

There are further criteria, however. Is the user familiar with the task? Does he need additional explanations? For example, can we use the less common term 'cc' (as a verb) in "CC to whom?" or should be more explicit, as in "To whom would you like to send a copy of it?".

The important question of *alignment* is not (yet) addressed: interlocutors align their lexical choices (unless they want to explicitly correct the user). They strive for parallelism of syntactic constructions. A modern generation component for a dialogue-driven user interface should make an effort to adapt to the user in certain cases, following human communication paradigms.

Another deficiency becomes clear when the heuristic function doing the utility/cost trade-off is examined. Cognitive cost is predicted on the basis of the length of the surface form of the output. Surely, that is not all of the cost. However, expressed in time, it is the voice output that takes longest, so that further processing cost only plays a marginal role. However, this cost still fails to predict how quickly the user is going to be able to react. For verbal interaction, the effect might be small. For the visual/tactile mode (graphical user interfaces with a pointing device such as a mouse), there are significant differences. Gajos & Weld (2004) provide a detailed model of the user's interaction with potential output and optimize their generated user interfaces according to that which would reduce the interaction effort. In the FASiL domain, again, only use few graphical user interface elements are used – such as radio or OK/Cancel buttons, and where they are used, there is no variation, so as not to confuse the user. Even those buttons could be adaptive: they would be bigger, if the situation suggests there is no stylus available.

8.2 Playground

The domain chosen, *Virtual Personal Assistant* with conversations about whether to send a short e-mail to one's colleagues requesting a meeting, is not nearly rich enough to support the sophistication discussed in the previous section.[1] The first target application has turned out to pose a number of problems for efficiency and some for adaptivity. However, it is not rich enough to exemplify fine-grained adaptivity in mobile devices. Nor is it rich enough to allow us to study multimodal discourse coherence phenomena.

8.3 Models based on empirical knowledge

The additional criteria for an NLG system to follow, which were proposed in the previous sections, are unlikely to be balanced by manually estimated, situation-specific weights, as has been done for MUG. The variety of constraints warrants empirically justified weights, with possibly a large corpus of comprable multimodal interaction in different situations. A trainable model of multimodal coherence, alignment and situation-adaptivity is a long-term goal. And a worthwhile one.

[1] Apart from that, for alignment questions, relevant information from the analysis stage is simply not available to the generation component in FASiL.

Bibliography

André, E., Finkler, W., Graf, W., Rist, T., Schauder, A. & Wahlster, W. (1993). WIP: The automatic synthesis of multimodal presentations, *in* M. T. Maybury (ed.), *Intelligent Multimedia Interfaces*, AAAI Press, Menlo Park, CA, pp. 75–93.

Bateman, J. A., Kamps, T., Kleinz, J. & Reichenberger, K. (2001). Towards constructive text, diagram, and layout generation for information presentation., *Computational Linguistics* **27**(3): 409–449.

Beringer, N., Kartal, U., Louka, K., Schiel, F. & Türk, U. (2002). PROMISE - a procedure for multimodal interactive system evaluation, *Proceedings of the Workshop 'Multimodal Resources and Multimodal Systems Evaluation'*, Las Palmas, Gran Canaria, Spain.

Bolt, R. A. (1980). Put-that-there: Voice and gesture at the graphics interface, *Computer Graphics* **45**: 337–348.

Bouayad-Agha, N., Scott, D. & Power, P. (2000). Integrating content and style in documents: a case study of patient information leaflets, *Information Design Journal* **9**(2): 161–176.

Bourbeau, L., Carcagno, D., Goldberg, E., Kittredge, R. & Polguère, A. (1990). Bilingual generation of weather forecasts in an operations environment, *in* H. Kargren (ed.), *Proceedings of the 13th. International Conference on Computational Linguistics (COLING'90)*, Helsinki, Finland, pp. 318–320.

Boves, L. & den Os, E. (2002). Multimodal services – a MUST for UMTS, Multimodal multilingual information services for Small mobile Terminals (MUST),, *Technical report*, EURESCOM Project.

Cabon, B., Verfaillie, G., Martinez, D. & Bourret, P. (1996). Using Mean Field Methods for Boosting Backtrack Search in Constraint Satisfaction Problems, *Proceedings of ECAI96, Budapest, Hungary*, pp. 165–169.

Carpenter, B. (1992). *The Logic of Typed Feature Structures*, Cambridge University Press, Cambridge, England.

Cassell, J., Nakano, Y. I., Bickmore, T. W., Sidner, C. L. & Rich, C. (2001). Non-verbal cues for discourse structure, *Proceedings of the 41st Annual Meeting of the Association of Computational Linguistics*, Toulouse, France, pp. 106–115.

Cohen, P., Johnston, M., McGee, D., Smith, I., Oviatt, S., Pittman, J., Chen, L. & Clow, J. (1997). Quickset: Multimodal interaction for simulation set-up and control, *Proceedings of the Applied Natural Language Conference*, San Francisco, CA.

Copestake, A. (2001). *Implementing Typed Feature Structure Grammars*, CSLI Lecture Notes, Center for the Study of Language and Information, Stanford.

Dale, R. & Reiter, E. (1995). Computational interpretations of the gricean maxims in the generation of referring expressions, *Cognitive Science* **19**: 233–263.

Dale, R., Oberlander, J., Milosavljevic, M. & Knott, A. (1998). Integrating natural language generation and hypertext to produce dynamic documents, *Interacting with Computers* **11**(2): 109–135.

Denecke, M. (2000). Informational characterization of dialogue states, *COLING 2000*.

Denecke, M. & Waibel, A. (1997). Dialogue strategies guiding users to their communicative goals, *Eurospeech '97*, Rhodes, Greece, pp. 1339–1342.

Elhadad, M. & Robin, J. (1992). Controlling content realization with functional unification grammars, *Proceedings of the 6th International Workshop on Natural Language Generation*, Springer-Verlag, pp. 89–104.

Elhadad, M. & Robin, J. (1998). An overview of SURGE: A reusable comprehensive syntactic realization component, *Technical Report 96-03*, Dept. of Mathematics and Computer Science, Ben Gurion University, Beer Sheva, Israel.

Federmeier, K. & Bates, E. (1997). Contexts that pack a punch: lexical class priming of picture naming, *Newsletter of the Center for Research in Language*.

Feiner, S. K. & McKeown, K. R. (1998). Automating the generation of coordinated multimedia explanations, *in* M. T. Maybury & W. Wahlster (eds), *Intelligent User Interfaces*, Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Flemming, E. (1995). Phonetic detail in phonology: Towards a unified account of assimilation and coarticulation, *in* K. Suzuki & D. Elzinga (eds), *Proceedings of the Arizona Phonology Conference 5, Features in Optimality Theory*.

Flemming, E. (2001). Scalar and categorical phenomena in a unified model of phonetics and phonology, *Phonology* **18**(1): 7–44.

Frege, G. (1892). Über Sinn und Bedeutung, *Zeitschrift für Philosophie und philosophische Kritik* **100**: 25–50. Translation by Max Black in Geach and Black (1970): 56–78.

Gajos, K. & Weld, D. S. (2004). Supple: Automatically generating user interfaces, *Proceedings of the 9th international conference on Intelligent user interfaces*, Funchal, Portugal.

Grice, H. (1975). Logic and conversation, *in* P. Cole & J. Morgan (eds), *Syntax and Semantics*, Vol. 3, Academic Press, pp. 41–58.

Grosz, B. J., Joshi, A. K. & Weinstein, S. (1995). Centering: A framework for modeling the local coherence of discourse, *Computational Linguistics* **21**(2): 203–225.

Horacek, H. (1997). An algorithm for generating referential descriptions with flexible interfaces, *in* P. R. Cohen & W. Wahlster (eds), *Proceedings of 35th ACL / 8th EACL (ACL/EACL 1997)*, Association for Computational Linguistics, Somerset, New Jersey.

Joachims, T. (1998). Text categorization with support vector machines: learning with many relevant features, *in* C. Nédellec & C. Rouveirol (eds), *Proceedings of ECML-98, 10th European Conference on Machine Learning*, Springer, Heidelberg, Germany, pp. 137–142.

Johnston, M. (1998). Unification-based multimodal parsing, *Proceedings of COLING-ACL 1998*, pp. 624–630.

Johnston, M., Bangalore, S., Vasireddy, G., Stent, A., Ehlen, P., Walker, M., Whittaker, S. & Maloor., P. (2002). Match: An architecture for multimodal dialogue systems, *Proceedings of ACL-2002*.

Kibble, R. (2001). A reformulation of rule 2 of Centering Theory, *Computational Linguistics* **27**(4): 579–587.

Knight, K. & Hatzivassiloglou, V. (1995). Two-level many-paths generation, *Proc. of the 33rd Conference of the Association of Computational Linguistics (ACL-95)*, Boston, MA, pp. 252–260.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search, *Artificial Intelligence* **27**(1): 97–109.

Kvale, K., Warakagoda, N. D. & Knudsen, J. E. (2001). Speech-centric multimodal interaction with small mobile terminals, *Norsk Symposium I Signalbehandling*, Trondheim.

Langkilde, I. & Knight, K. (1998). Generation that exploits corpus-based statistical knowledge, *COLING-ACL*, pp. 704–710.

Levelt, W. (1989). *Speaking: From Intention to Articulation*, MIT Press, Cambridge MA.

Lindblom, B. (1998). Systemic constraints and adaptive change in the formation of sound structure, *Approaches to the Evolution of Language: Social and Cognitive Bases*, Cambridge University Press, Cambridge.

Liu, H. (1996). *Lexical Access and Differential Processing in Nouns and Verbs in a Second Language*, PhD thesis, University of California at San Diego.

Mann, W. C. & Thompson, S. A. (1988). Rhetorical Structure Theory: Towards a functional theory of text organization, *Text* **8**(3): 243–281.

Martinet, A. (1952). Function, structure, and sound change, *Word* **8**(1): –.

Matthiessen, C. M. I. M. & Bateman, J. A. (1991). *Text generation and systemic-functional linguistics: experiences from English and Japanese*, Frances Pinter Publishers and St. Martin's Press, London and New York.

McNeill, D. (1992). *Hand and mind: What gestures reveal about thought*, University of Chicago Press.

McTear, M. (1998). Modelling spoken dialogues with state transition diagrams: experiences of the cslu toolkit.

Meseguer, P., Bouhmala, N., Bouzoubaa, T., Irgens, M. & Sánchez, M. (2003). Current approaches for solving over-constrained problems, *Constraints* **8**(1): 9–39.

Moore, J., Foster, M. E., Lemon, O. & White, M. (2004). Generating tailored, comparative descriptions in spoken dialogue, *Proceedings of the 17th International FLAIRS Conference*.

Oberlander, J. & Brew, C. (2000). Stochastic text generation, *Philosophical Transactions of the Royal Society of London, Series A* **358**: 1373–1385.

O'Donnell, M., Mellish, C., Oberlander, J. & Knott, A. (2001). ILEX: An architecture for a dynamic hypertext generation system, *Journal of Natural Language Engineering* **7**: 225–250.

Oviatt, S. (1999). Ten myths of multimodal interaction, *Communications of the ACM* **42**(11): 74–81.

Oviatt, S., DeAngeli, A. & Kuhn, K. (1997). Integration and synchronization of input modes during multimodal human-computer interaction, *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, pp. 415–422.

Panttaja, E., Reitter, D. & Cummins, F. (2004). The evaluation of adaptable multimodal system outputs, *Proceedings of the Workshop on Robust and Adaptive Information Processing for Mobile Speech Interfaces at COLING-04*.

Papineni, K., Roukos, S. & Ward, T. (1990). Free-flow dialog management using forms, *EUROSPEECH'99*.

Passonneau, R. (1998). Interaction of discourse structure with explicitness of discourse anaphoric noun phrases, *in* M. A. Walker, A. K. Joshi & E. F. Prince (eds), *Centering Theory in Discourse*, Clarendon Press, Oxford, pp. 327–358.

Pickering, M. & Garrod, S. (in press). Toward a mechanistic psychology of dialogue, *Behavioral and Brain Sciences*.

Pollard, C. & Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*, Chicago: University of Chicago Press.

Prince, A. & Smolensky, P. (1993). Optimality theory: Constraint interaction in generative grammar, *Technical Report RuCCS No 2*, Rutgers University, Center for Cognitive Science.

Reinefeld, A. & Marsland, T. A. (1994). Enhanced iterative-deepening search, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**(7): 701–710.

Reiter, E. (1994). Has a consensus NL generation architecture appeared, and is it psychologically plausible?, *in* D. McDonald & M. Meteer (eds), *Proceedings of the 7th. International Workshop on Natural Language generation (INLGW '94)*, Kennebunkport, Maine, pp. 163–170.

Reiter, E. (2000). Pipelines and size constraints, *Computational Linguistics* **26**(2): 251–259.

Reiter, E. & Dale, R. (2000). *Building Natural Language Generation Systems*, Cambridge University Press.

Reitter, D. (2003a). *Rhetorical analysis with rich-feature support vector models*, Master's thesis, University of Potsdam.

Reitter, D. (2003b). Simple signals for complex rhetorics: On rhetorical analysis with rich-feature support vector models, *LDV-Forum, GLDV-Journal for Computational Linguistics and Language Technology* **18**(1/2): 38–52.

Reitter, D. (2004). A development environment for multimodal functional unification generation grammars, *Proc. Third International Conference on Natural Language Generation 2nd Volume. ITRI Technical Report.*

Reitter, D., Panttaja, E. & Cummins, F. (2004). UI on the fly: Generating a multimodal user interface, *Proceedings of Human Language Technology conference 2004 / North American chapter of the Association for Computational Linguistics (HLT/NAACL-04)*.

Russell, S. & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ.

Seneff, S., Hurley, E., Lau, R., Pao, C., Schmid, P. & Zue, V. (1998). Galaxy-II: A reference architecture for conversational system development, *ICSLP 98*, Sydney, Australia.

Swinney, D. (1979). Lexical access during sentence comprehension: (re)consideration of context effects, *Journal of Verbal Learning & Verbal Behavior* **18**(6): 645–659.

W3C (2000). Multimodal requirements for voice markup languages (draft), *Technical report*, World Wide Web Consortium, http://www.w3.org/TR/multimodal-reqs.

Wahlster, W. (2000). *Verbmobil: Foundations of Speech-to-Speech Translation*, Springer, Berlin - Heidelberg - New York.

Wahlster, W. (2002). Smartkom: Fusion and fission of speech, gestures, and facial expressions, *Proceedings of the 1st International Workshop on Man-Machine Symbiotic Systems*, Kyoto, Japan.

Wahlster, W. (2003). Towards symmetric multimodality: Fusion and fission of speech, gesture, and facial expression, *KI 2003: Advances in Artificial Intelligence, 26th Annual German Conference on AI, KI 2003, Hamburg, Germany, September 15-18, 2003, Proceedings*, Vol. 2821 of *Lecture Notes in Computer Science*, Springer.

Walker, M., Joshi, A. & Prince, E. (1998). Centering in naturally occurring discourse: An overview, *in* M. A. Walker, A. K. Joshi & E. Prince (eds), *Centering Theory in Discourse*, Oxford University Press, Oxford, pp. 1–28.

Walker, M., Litman, D., Kamm, C. & Abella, A. (1997). PARADISE: A framework for evaluating spoken dialogue agents, *in* P. R. Cohen & W. Wahlster (eds), *Proceedings of the ACL-EACL-1997*, Association for Computational Linguistics, Somerset, New Jersey, pp. 271–280.

Zipf, G. K. (1949). *Human Behavior and the Principle of Least Effort*, Addison Wesley.