Incremental Spoken Dialogue Processing: Architecture and Lower-level Components

Timo Baumann

© Copyright 2013 Timo Baumann, Diplom-Informatiker aus Hamburg.

Dissertation zur Erlangung des akademischen Grades *Doctor philosophiae* (Dr. phil.)
vorgelegt an der Fakultät für Linguistik und Literaturwissenschaft
der Universität Bielefeld am 2. April 2013.

Prüfungskommission:
Prof. Dr. David Schlangen (Betreuer und Gutachter)
Prof. Dr. Petra Wagner (Gutachterin)
Prof. Dr.-Ing. Stefan Kopp
Prof. Dr. Jan de Ruiter

Datum der mündlichen Prüfung: 16. Mai 2013.

♾ Printed on acid-free, aging-resistant paper (according ISO 9706).
Gedruckt auf säure-freiem, alterungsbeständigen Papier (nach ISO 9706).

Overview

	Kurzfassung	4
1	Introduction	14
2	Spoken Dialogue and Spoken Dialogue Systems	22
3	Incremental Processing and its Evaluation	45
4	A Software Architecture for Incremental Spoken Dialogue Processing	73
5	Incremental Speech Recognition	88
6	Short-Term Estimation of Dialogue Flow	138
7	Incremental Speech Synthesis	172
8	Conclusion and Outlook	218

Schritthaltende Sprachdialogverarbeitung: Architektur und signalnahe Komponenten

Diese Arbeit befasst sich mit Sprachdialogsystemen, also Mensch-Maschine-Schnittstellen die (primär) über gesprochene Sprache genutzt werden. Sprachdialogsysteme sind vor allem deshalb attraktiv, weil Sprache eine besonders natürliche und intuitive Interaktionsform darstellt.

Der Nutzen von Sprachdialogsystemen ergibt sich jedoch bisher daraus, dass sie helfen, relativ einfache Aufgaben sinnvoll zu erfüllen – denn tatsächlich ist die Natürlichkeit bisheriger Sprachdialogsysteme stark durch das vereinfachende Verarbeitungsschema der *Ping-Pong-Interaktion* eingeschränkt: Übliche Dialogsysteme erwarten einen vollständigen und abgeschlossenen Redebeitrag, auf den das System (nach einer gewissen Verarbeitungszeit) mit einem gleichfalls vollständigen, oft ununterbrechbaren Redebeitrag antwortet. Die Redebeiträge von System und Nutzer sind also überlappungsfrei, was nicht den tatsächlichen Gegebenheiten natürlichsprachlicher Interaktion entspricht, die von einem beiderseitigen Geben und Nehmen lebt, und bei der auch der jeweilige Zuhörer hilft, den Redebeitrag des jeweiligen Sprechers durch Mimik, kurze Einwürfe, und dergleichen mitzugestalten.

Schritthaltende Verarbeitung ist ein Konzept, bei der die Verarbeitung bereits während der Eingabephase abläuft und Zwischenergebnisse bereits erzeugt werden bevor die Eingabe abgeschlossen ist. Einem Dialogsystem erlaubt schritthaltende Verarbeitung Reaktionen zu erzeugen die zeitgleich zur noch laufenden Eingabe erfolgen, oder entgegengebrachte Nutzerrückmeldungen in laufende Systemausgaben zu integrieren. Dadurch erlaubt schritthaltende Verarbeitung eine schnellere Rückkopplung zwischen Nutzer und System, was zu höherer Interaktivität und besserem gegenseitigen Verständnis führen kann. Das Ziel dieser Arbeit ist, den Nutzen schritthaltender Verarbeitung auf die Interaktionsqualität von Sprachdialogsystemen zu untersuchen. Dabei beschränkt sich die detaillierte Analyse auf signalnahe Komponenten (Spracherkennung und -synthese); Module übergeordneter Abstraktionsgrade sind in den Beispielsystemen teilweise weniger ausgefeilt, oder nur simuliert.

Die Leitfrage der Arbeit ist, inwieweit feingliedrig schritthaltende Verarbeitung technisch realisierbar ist und auf natürlichere Weise interagierende Sprachdialogsysteme ermöglicht. Darüber hinaus wird argumentiert, dass Dialog *proaktives* Handeln verlangt, also nicht rein auf Basis bereits vorliegender Erkenntnis, sondern außerdem auf Grundlage von Schätzungen über die (nähere) Zukunft.

Bei schritthaltender Verarbeitung werden sehr viele Zwischenergebnisse erzeugt und da diese jeweils nur aus einem begrenzten Kontext heraus erzeugt werden, ergibt sich die Erfordernis, sie auch verwerfen zu können; dies erfordert Anpassungen der Systemarchitektur gegenüber bisherigen Systemen.

Kapitel 2 gibt einen Überblick über Fragen der gesprochensprachlichen Interaktion, des Dialogs und Dialogsystemen.

Kapitel 3 vertieft dann die Thematik der schritthaltenden (inkrementellen) Verarbeitung und führt einen Formalismus für die Darstellung von Hypothesen ein, anhand dessen Qualitätsmaße schritthaltender Verarbeitung definiert werden, die ausführlich diskutiert werden.

Kapitel 4 stellt die Architektur des im Rahmen der Arbeit entwickelten Softwaretoolkits für schritthaltende Verarbeitung (engl. *incremental processing*), INPROTK vor und diskutiert Daten- und Verarbeitungsschemata.

Kapitel 5 betrachtet inkrementelle Spracherkennung. Die ‚inkrementelle Qualität' der Spracherkennung wird intensiv auf mehreren Korpora und für unterschiedliche Varianten in all ihren Aspekten untersucht. Schließlich werden Optimierungsmethoden vorgestellt, welche Qualitätsaspekte gegeneinander abwägen. Der Nutzen inkrementeller Spracherkennung wird beispielhaft in einer Spielanwendung gezeigt.

Kapitel 6 geht den Schritt von möglichst reaktiver zu *proaktiver* Verarbeitung, welche erlaubt, den Dialogverlauf aktiv zu steuern. Eine Beispielanwendung zeigt, wie durch schritthaltende Verarbeitung die Rückkopplung zwischen Nutzer und System beschleunigt und dadurch Nutzeräußerungen gemeinschaftlich gestaltet werden können. Schließlich wird ein System gezeigt, welches Nutzeräußerungen synchron mitspricht. Dieses System zeigt, dass inkrementelle und proaktive Verarbeitung synchrone Interaktionsfähigkeiten in Echtzeit ermöglichen, indem alle Systemverzögerungen an anderer Stelle durch Prädiktion ausgeglichen werden.

Kapitel 7 betrachtet inkrementelle Sprachsynthese, bei der die Spezifikation der Äußerung noch während der Synthese erweitert oder abgeändert werden kann. Der Nutzen dieser Fähigkeit wird in einer hochdynamischen Umgebung demonstriert, in der Inkrementalität Reaktionen ermöglicht die als deutlich natürlicher im Vergleich zu einem nicht-inkrementellen System eingeschätzt werden. Schließlich wird die Integration inkrementeller Sprachsynthese mit einem Sprachgenerierungsmodul demonstriert, und der Einfluss auf die resultierende Prosodiequalität des Systems bewertet.

Kapitel 8 fasst die Ergebnisse der Arbeit zusammen: feingliedrig schritthaltende Verarbeitung ist technisch möglich und so erfolgreich, dass dadurch für Sprachdialogsysteme vormals unerreichbare Interaktionsmodi ermöglicht werden (u. a. gemeinschaftliche Äußerungsgestaltung, synchrones Sprechen, Berücksichtigung von Änderungen während Systemäußerungen). Schritthaltende Verarbeitung sollte deshalb die Basis für zukünftige Sprachdialogsysteme bilden.

Detailed Table of Contents

Kurzfassung **4**

1 Introduction **14**
1.1 Thesis Outline 18
1.2 Contributions 19
1.3 Previously Published and Co-authored Material 20

2 Spoken Dialogue and Spoken Dialogue Systems **22**
2.1 Modelling Dialogue 22
2.1.1 The Shannon-Weaver Model of Communication 23
2.1.2 Layers of Communication 25
2.1.3 Emergence of Behaviour in Complex Systems 28
2.1.4 Establishing Common Ground 30
2.1.5 Taking Turns 31
2.1.6 Feedback and the Backward Channel 32
2.2 Components and Architecture for Spoken Dialogue Systems 33
2.2.1 Components of Spoken Dialogue Systems 33
2.2.2 Interconnection of Components 36
2.2.3 Discussion 39
2.3 State of the Art in Spoken Dialogue Systems 40
2.3.1 Commercial, Standards-based Systems 40
2.3.2 Related Research Systems 41
2.3.3 Advanced Commercial Systems 42
2.4 Summary and Discussion 43

3 Incremental Processing and its Evaluation **45**
3.1 Timeliness and Incrementality 45
3.1.1 Aspects of Incrementality 47
3.1.2 Related Work on Evaluating Incremental Processing 48
3.1.3 Relation to Anytime Processing 49
3.2 Our Notion of Incremental Processing 50
3.2.1 Incremental Processors 50
3.2.2 Representing Incremental Data 53

3.3 Evaluation of Incremental Processors 59
3.3.1 Gold Standards for Evaluation 59
3.3.1.1 Evaluation with Incremental Gold Standards . . . 60
3.3.1.2 Evaluation with Non-Incremental Gold Standards 63
3.3.2 Metrics for Evaluation of Incremental Processors 64
3.3.2.1 Similarity Metrics 64
3.3.2.2 Timing Metrics 67
3.3.2.3 Diachronic Metrics 69
3.3.2.4 Interrelations between Metrics 71
3.4 Summary 72

4 A Software Architecture for Incremental Spoken Dialogue Processing 73
4.1 The Data Model: Incremental Units 74
4.1.1 The IU Network 76
4.1.2 Triangular Data Models 78
4.2 The Processing Model 80
4.2.1 Incremental Modules and Inter-Module Communication . . 80
4.2.2 Alternative Processing Schemes 83
4.3 Infrastructure 85
4.4 Discussion 85

5 Incremental Speech Recognition 88
5.1 Automatic Speech Recognition in a Nutshell 89
5.1.1 Modelling Speech Data 90
5.1.1.1 The Language Model 91
5.1.1.2 The Pronunciation Model 93
5.1.1.3 Acoustic Modelling in the ASR Frontend 93
5.1.2 The Speech Recognizer 96
5.1.2.1 Hidden Markov Models 97
5.1.2.2 The Decoding Algorithm 99
5.1.3 Evaluating Speech Recognition 101
5.1.4 Refinements 102
5.1.5 The Sphinx-4 Speech Recognizer 103
5.2 Incrementalizing Speech Recognition 104
5.2.1 The INPROTK Module for Incremental Rich Speech Recognition 105
5.3 INTELIDA: A Workbench for Evaluating iSR 107
5.3.1 The Library 107
5.3.2 Interactive Tools 108
5.4 Evaluation of Basic Incremental Speech Recognition 110
5.4.1 Variations of the Setup and Stability of Results 116

5.4.2 N-Best Processing . . . 117
5.5 Optimization of Incremental Speech Recognition . . . 119
5.5.1 Right Context . . . 122
5.5.2 Hypothesis Smoothing . . . 124
5.5.3 Advanced Smoothing Methods . . . 126
5.6 Example Application: Incremental Command-and-Control . . . 128
5.6.1 The *Greifarm* Domain . . . 128
5.6.2 Cost-based Smoothing . . . 129
5.6.3 Wizard-of-Oz Experiment . . . 130
5.6.4 System Implementation . . . 131
5.6.5 Evaluation . . . 133
5.7 Summary and Discussion . . . 134

6 Short-Term Estimation of Dialogue Flow 138
6.1 Floor Tracking in INPROTK . . . 140
6.1.1 Architecture and Implementation . . . 141
6.1.2 Example Application: Collaborating on Utterances . . . 142
6.1.2.1 Domain and Setup . . . 143
6.1.2.2 Experiment and Results . . . 145
6.1.3 Discussion . . . 147
6.2 Micro-Timing Prediction . . . 147
6.2.1 Motivation for the Task . . . 148
6.2.2 Related Work on Simultaneous Speech . . . 150
6.2.3 System Architecture . . . 153
6.2.4 Two Models for Micro-Timing . . . 154
6.2.5 Evaluation . . . 157
6.2.5.1 Corpus and Experiment Setup . . . 157
6.2.5.2 End-of-Word Prediction: When to Start Speaking 159
6.2.5.3 Predicting the Micro-Timing of the Upcoming Word 162
6.2.5.4 Estimating the Reliability of the Predictions . . . 164
6.2.5.5 Summary . . . 165
6.2.6 Example Application: Speaking in Synchrony With the User 166
6.2.7 Discussion . . . 167
6.3 Summary and Discussion . . . 169

7 Incremental Speech Synthesis 172
7.1 Rationale for Incremental Speech Synthesis . . . 173
7.1.1 Related Work . . . 175
7.1.2 Requirements . . . 176

7.2 Speech Synthesis in a Nutshell . . . 178
7.2.1 Text-based Linguistic Processing . . . 179
7.2.2 HMM-based Waveform Synthesis . . . 180
7.2.2.1 Parameter Estimation with HMMs . . . 181
7.2.2.2 Vocoding . . . 181
7.2.3 Discussion of Alternative Synthesis Techniques . . . 182
7.2.4 Evaluation of Speech Synthesis . . . 183
7.2.5 MaryTTS . . . 184
7.3 Incrementalizing Speech Synthesis . . . 185
7.3.1 Incremental Speech Synthesis in INPROTK . . . 188
7.3.1.1 Utterance Tree-based iSS . . . 188
7.3.1.2 An Incremental Module for Speech Synthesis . . . 189
7.3.1.3 Very Low-Latency Prosody Adaptation . . . 190
7.3.1.4 Automatic Hesitation . . . 191
7.3.2 Conformance to the Requirements . . . 192
7.4 The Merit of iSS . . . 193
7.4.1 Domain and System . . . 194
7.4.2 Evaluation . . . 196
7.4.3 Results . . . 197
7.4.4 Discussion . . . 198
7.5 Example Application: Integration with Incremental NLG . . . 199
7.5.1 Use-Case: Adaptive Information Presentation . . . 200
7.5.2 Implemented System . . . 201
7.5.3 Evaluation . . . 203
7.5.3.1 System Response Time . . . 203
7.5.3.2 Subjective Evaluation . . . 205
7.5.4 Discussion . . . 206
7.6 Evaluating the Prosodic Quality of iSS . . . 207
7.6.1 The Design Space for Incremental Prosody Production . . . 208
7.6.2 Experiment . . . 210
7.6.3 Evaluation . . . 210
7.6.3.1 Qualitative Analysis . . . 211
7.6.3.2 Quantitative Evaluation . . . 212
7.6.4 Conclusion . . . 214
7.7 Summary and Discussion . . . 215

8 Conclusion and Outlook 218
8.1 Summary . . . 218
8.2 Conclusion . . . 219
8.3 Open Questions . . . 221

List of Figures

2.1 The Shannon-Weaver Model 23
2.2 The Chain Model of Communication 27
2.3 Architecture of and Information Flow in Spoken Dialogue Systems . 36
2.4 Blackboard Architecture for Spoken Dialogue Systems 37
2.5 Control and Information Flow in a Spoken Dialogue System 39

3.1 Degrees of Proactiveness 46
3.2 Relation Between Input and Output in Incremental Processing . . . 50
3.3 ASR Hypotheses During Incremental Recognition 55
3.4 One-best, N-best and Lattice Output of an ASR 56
3.5 Task-dependent Gold Standards for a Disfluent Utterance 60
3.6 Gold Standard for Incremental Speech Recognition 61
3.7 Subsequent Outputs for an Incremental Semantics Component . . . 62
3.8 A Semantic Frame Represented in the IU Framework 63
3.9 An ASR's Actual Incremental Output 65

4.1 IU Hierarchy in INPROTK 76
4.2 IU Network 77
4.3 Triangular Data-Driven IU Network 79
4.4 Two Incremental Modules with Contained IUs 81
4.5 Incremental Modules With Update Listening 84

5.1 A Discrete HMM With Three Emitting States 97
5.2 A Branching HMM 99
5.3 Pronunciation Lextree 101
5.4 A Screenshot of the INTELIDA Graphical Interface 108
5.5 Distribution of F0 and FD 113
5.6 Timing Metrics and ASR Errors. 114
5.7 Correction Time and IU Survival Time 115
5.8 Stability of iSR Results Under Varying Conditions 116
5.9 FO Distribution Varying with N-best List Size 118
5.10 Lattice During Incremental Speech Recognition 120
5.11 Right Context for Improving iSR 123
5.12 Hypothesis Smoothing for Improving iSR 125

5.13 Timing Metric Distribution Comparison for Optimized iSR 126
5.14 Per-Word `correction times` 127
5.15 The *Greifarm* Domain for Incremental Command-and-Control . . . 128
5.16 Wizard Interface for the Command-and-Control Task 130
5.17 UML Diagram of the *Greifarm* Prototype 132

6.1 The Place of a Floor Tracking Component in the iSDS Architecture . 141
6.2 The *Pentomino Select System* Domain 144
6.3 Task Durations in the *Pentomino Select System* Versions 146
6.4 The Micro-Timing Task in Co-Completion 149
6.5 Ranked Factors Influencing Human Synchronous Speech Performance 151
6.6 Schematic View of the Co-Completion System 153
6.7 Micro-Timing Models 155
6.8 F0 Distribution/Time Available to Determine Shadowing 159
6.9 EoW Predictions for Micro-Timing Models 161
6.10 Scatter Plot Showing Micro-Timing Reliability 165
6.11 Example of Shadowing a File in the Corpus 166
6.12 Example of Shadowing Live Input 167

7.1 Delivery Adjustments in Incremental Speech Synthesis 173
7.2 Example of Utterance Production in a Highly Interactive Environment 174
7.3 Processing Paths in Speech Synthesis Systems 178
7.4 Processing Shares of MaryTTS Modules 185
7.5 Schematic View of iSS 186
7.6 iSS Based on Utterance Trees 189
7.7 Mediating Push- and Pull-based Processing 190
7.8 Interface to Low-Latency Prosody Adaptation 191
7.9 The CarChase Domain 195
7.10 Evaluation Results for the CarChase System 197
7.11 Preferred System Behaviour in Highly Dynamic Environments . . . 198
7.12 Events in the Calendar Domain 200
7.13 Interplay of Components in the Calendar System 201
7.14 Progress Updates in the Calendar System 202
7.15 Naturalness Ratings for the *Calendar* System 205
7.16 Design Space for Incremental Prosody Production 209
7.17 Incrementally Generated Pitch Tracks 212
7.18 Interrelation of Lookahead and Prosodic Quality 214

List of Tables

2.1 Linguistic Subdisciplines 26

4.1 General Properties of IUs and Exemplary Operations 75

5.1 Overview of iSR Corpora 111
5.2 Overview of Raw iSR Performance 112
5.3 Concepts in the *Greifarm* Domain 131

6.1 Performance of Estimating the End of the Ongoing Word 160
6.2 Performance of Estimating the Micro-Timing of the Next Word . . . 163

7.1 System Response Time in the Calendar Domain 204
7.2 Results for Incremental Prosody Production 213

1 Introduction

This thesis concerns itself with one of the problems of spoken dialogue systems. A spoken dialogue system (SDS) is an interface to a computer that mainly interacts with its human users through spoken language, exchanging multiple spoken turns (questions and answers, commands and replies, …) resulting in a dialogue between the user (or possibly multiple users) and the system.

Spoken dialogue systems have been in use, for example as remote telephone (interactive voice response) applications, since the 1990's: first using touch-tone, then extended by allowing spoken keywords, and later simple commands (Pieraccini and Lubensky 2005). SDSs have also been widely hated by their users (Sharabi and Davidow 2010, p. 196) who tend to prefer to talk to human call center agents instead. Recently, spoken dialogue systems have been introduced to smartphones in the form of *intelligent personal assistants* such as Siri, Google Now, or S Voice. It is especially remarkable that SDSs are now used on smartphones that should provide ideal conditions for visual/tactile interaction given their large touch screens. The fact that SDSs are attractive on these devices means that they will likely become even more important for human-computer interaction in the future.

Spoken dialogue for interacting with a computing device may be attractive for two reasons: (a) SDSs hold the potential for very natural communication, very similar to how humans interact and communicate with each other (though this potential is lacking in current systems). In addition, (b) when compared to most other modes of human-computer interaction, SDSs open up an additional, under-used modality, namely spoken language (currently, the intelligent assistants mentioned above are multi-modal and make heavy use of the screen[1]). While interacting through dialogue is natural and intuitive for humans, dialogue systems so far resolve the issues of intuitivity and naturalness only to some degree – but sufficiently so to already be of use in certain situations.

The success of today's SDSs largely stems from the fact that they help to solve relatively simple tasks in useful ways, such as delivery tracking, buying a train ticket at the station, or when ordering at a restaurant. However, talking to an SDS is not as uncomplicated and straightforward as interacting with a human and is hardly ever a pleasurable experience (apart from the humorous aspects). More human-like

[1]However, this author believes that eventually many personal computing devices will become too small for detailed visual/tactile interaction, leaving spoken dialogue as the predominant form of interaction with the device.

SDSs can be expected to be applicable to more domains with less clearly-defined tasks and goals that require a larger degree of *conversational competence*. Roughly ordered by complexity, these may include: teaching, playing, entertaining, counseling, psychotherapy, or building and sustaining a personal relationship. Such domains are beyond the current capabilities of dialogue systems because they require the interaction to be smooth and natural, not only to be successful on a functional level.[2]

In a usability study, Ward et al. (2005) identified seven issues detrimental to efficient and pleasant dialogue with SDSs (in decreasing order of importance): speech recognition and understanding, time-outs, responsiveness, synthesis, feedback, adaptation, and other factors (prosody, non-lexical sounds, prompt generation). While not being the number one priority, time-outs, responsiveness and feedback share the property of concerning the *way* that the dialogue evolves, whereas speech recognition and understanding are primarily important on the functional level. Together these three factors have a significant share in the usability problems identified by Ward et al. (2005) and are all related to the crude processing mode of conventional dialogue systems: *ping-pong interaction*.

The ping-pong game that SDSs engage in goes as follows: a full user turn is received, recognized, analyzed for its meaning, and integrated into the dialogue manager's model of the state of affairs, before a system response is generated, synthesized, and output to the user. The main problem with this mode of operation is that there are no provisions for accepting as input or producing as output anything smaller than a full turn. Thus, relatively long time-outs are required at the end of user turns to avoid interrupting the user's turn when she makes a short pause within it. The overly long time-outs, together with the time necessary for processing the input and generating output then result in sluggish responsiveness. Finally, the system is unable to give feedback during user turns, or to integrate feedback during its own turns. Instead, when faced with feedback, the system either acts as if it were interrupted and aborts its utterance, or ignores the feedback altogether.

The goal of this thesis is to investigate a processing scheme that allows for *better interaction quality* (and as a result, new applications for SDSs) by overcoming the ping-pong style of interaction. Incremental processing is considered in this thesis to be a crucial requirement for better interaction quality.

Incremental processing means to interpret partial input, to process partial hypotheses, and to generate partial results.[3] Natural dialogue is both produced (Levelt 1989) and perceived (Tanenhaus et al. 1995) in an incremental manner by human dialogue participants, that is, they receive and understand continuously and are able to adapt

[2] One may argue that computers should never be deployed in any of these roles. The author agrees to the degree that computers will remain bad surrogates to humans in all these tasks – however, even a computer can be better than no help at all in many situations.

[3] A more detailed view on incremental processing is given in Chapter 3.

ongoing speech with little delay. In a way, SDSs are inherently incremental in that recognition, analysis and output production are repeated for every user/system turn exchange. In the terminology of incremental processing, conventional SDSs use a *granularity* of turns, that is, turns form the basic units of processing. For better interaction quality, dialogue systems must be incremental at a much more fine-grained level.[4] In contrast to conventional, 'turn-incremental' systems, the granularity of events in the approach to be described can be as low as a few milliseconds – short enough to support quick system reactions and hence better responsiveness and closer feedback loops. Such a fine granularity may not always be necessary – however, it is easy to reduce granularity in a system by combining units, whereas it is very hard to increase it *ex post*.

Among others, Aist et al. (2007b) showed that incremental understanding may be advantageous over non-incremental understanding in human-computer dialogue and Aist et al. (2007a) showed efficiency gains for incremental processing in a dialogue system. Here, we aim to account for fully, fine-granular incremental dialogue processing, where the whole system (instead of only select modules) works incrementally.

The topics of this thesis are the necessary changes in the system architecture to support (fine-granular) *incremental processing*, the evaluation methodology for incremental processing, and the development and performance analysis of low-level incremental processing modules for SDSs.

Thesis Statement

Given the goal of better interaction quality and the topic of incremental processing, it is argued that:

> **fine-granular incremental and proactive processing are viable techniques for more naturally interacting spoken dialogue systems.**

For this statement to be true, it needs to be shown that incremental processing in an SDS is both feasible and successful, and that it helps to develop more naturally interacting SDSs. This thesis presents an architecture for incremental spoken dialogue processing, outlines a notion of incrementality and presents evaluation metrics which are deemed useful to assess the success of incremental processing.

The evaluation metrics are put to use in the analysis of incremental speech recognition (iSR) which is shown to perform sufficiently well to be of use in a spoken dialogue system and some systems based on iSR are shown to handle new behavioural patterns in human-computer interaction which are not possible for non-incremental SDSs:

[4] The more fine-grained processing is what is referred to in the dialogue community when the term "incremental dialogue processing" is used; see for example (Schlangen and Rieser 2011).

immediate visual response in Chapter 5.6, and sub-turn incremental behaviour as a form of collaboration on utterances in Chapter 6.1.2.

Additionally, it is argued that some decisions in dialogue cannot be taken purely reactively (i. e. after all relevant input has become available) but must be taken *proactively* (i. e. where acting early with the possibility of doing wrong is better than acting late with a higher certainty of success). SDSs taking proactive decisions cannot base these only on hypotheses of what has happened so far but must also take into account what is expected to happen in the near future and must be able to act even though only partial information about the future is available. This is again exemplified with SDS modules close to the audio signal: Chapter 6.2.6 shows how a system can estimate micro-temporal aspects of ongoing speech. These estimates can be used to speak in synchrony to the user, to an extent that compares well to human performance in this task, showing that real-time end-to-end incrementality (that is, including all processing lags between speech input and output) is feasible. Chapter 7 treats the topic of incremental speech output, which is necessary to take action if only parts of the output have been decided on, investigates the quality trade-off involved when limiting the synthesizer's lookahead into the future, and finds that incremental synthesis allows for otherwise unrealizable conversational behaviour, such as starting early, or changing one's mind while speaking.

This thesis focuses on "lower-level" components for spoken dialogue processing: speech recognition, speech synthesis, and dialogue flow estimation. These *building blocks* are a prerequisite for further, "higher-level" components (such as understanding or reasoning) to be able to take advantage of incremental processing, to actually take complex decisions at all times in the dialogue and not just at the end of a user's turn (or the lack of a user's turn). Regarding the higher-level components for incremental dialogue processing, some progress can already be seen (Buß and Schlangen 2011; DeVault, Sagae, and Traum 2009; Sagae et al. 2009; Schlangen, Baumann, and Atterer 2009), so the components developed here do not stand solitarily but are part of a growing eco-system of components for incremental SDSs. Within the field of incremental spoken dialogue processing, they lay foundations for those parts of the processing chain that are further away from the audio signal.

Incremental processing is useful beyond SDSs in the more general field of *spoken dialogue processing* where the focus is different than human-computer interaction through dialogue. Some such areas are simultaneous interpreting (Amtrup 1999; Bangalore et al. 2012), overlistening for call-center agent support (Farrell 2004), or simultaneous transcription (e. g. of interviews). These areas are not actively investigated in this thesis, but the requirements towards their "lower-level" components are likely to be very similar to those developed here.

1.1 Thesis Outline

Chapter 2 places the work into context. The chapter reviews findings on dialogic communication *per se*, and architectures for dialogue processing, giving a general overview of the components required for processing dialogue and the requirements that an ideal architecture for conversational dialogue processing should meet. The chapter closes with some state-of-the-art systems and discusses how those relate to the work presented here.

Chapter 3 discusses previous approaches to incremental processing and explains the notion of incremental processing used, in which bits of information are added in a piece-meal fashion. *When* bits of information are hypothesized, *when* the hypothesis becomes reliable, and *how often* bits of information are replaced with other information are the major criteria identified regarding the performance of incremental processing. A detailed view of evaluation metrics that cover the above-mentioned aspects and their interrelations is developed, as well as how trade-offs between the metrics can be made.

Based on the account of incrementality, Chapter 4 presents a software architecture for building incremental SDSs and gives a high-level overview of INPROTK, a toolkit implementation of this architecture.

Chapter 5 gives a detailed assessment of incremental automatic speech recognition (iSR) after first presenting the algorithmic basics for standard, HMM-based speech recognition, explaining how it can easily be incrementalized, and describing INTELIDA, the workbench for evaluating iSR. The evaluation then analyzes iSR on several corpora detailing the different metrics outlined above, and investigates stability of results as well as behaviour for n-best processing. We then present two optimization techniques that rate favourably when trading timeliness against other qualities relevant for incremental processing. The usefulness of a slightly more advanced and problem-specific technique is shown in a first example application that significantly improves in terms of responsiveness when using optimized iSR and that would not be possible with non-incremental speech recognition.

Having dealt with low latency processing, Chapter 6 turns to predictive processing. Predictive processing is necessary in order to actively influence the dialogue flow. Firstly, a floor tracking component is introduced and put to use in a second example application that collaborates on utterances with the user. Secondly, and further increasing the temporal granularity at which decision making in a system may work, the micro-timing of the user's speech flow is modelled, which allows to predict the remaining duration of words currently being spoken. This is shown to work with reasonable performance and an example application uses the predictions to shadow a speaker's utterance, showing that end-to-end incrementality is indeed possible in real-time.

Chapter 7 turns from input to output processing, primarily incremental speech synthesis (iSS). After giving some background on speech synthesis and explaining how it is incrementalized in the InproTK iSS component, a series of experiments investigates (a) the merit of using iSS in a highly dynamic environment (which dialogue certainly is), (b) the integration of iSS with incremental NLG to form a fully incremental speech output pipeline, and (c) the trade-offs between incremental timing aspects and resulting quality when incrementally producing prosody.

Chapter 8 briefly summarizes the major findings of the thesis work and concludes that incremental and predictive processing is indeed feasible to an extent that it can help improve interaction quality and should form one of the architectural foundations of next generation spoken dialogue systems. The chapter also briefly sketches some ideas for future work.

1.2 Contributions

This thesis contributes:

- an evaluation methodology for assessing incremental processing performance (Section 3.3.2), an evaluation workbench implementing the metrics for incremental speech recognition (Section 5.3), a detailed analysis of these aspects in iSR (Section 5.4), as well as the idea of optimization methods that lead to a better trade-off of incremental performance metrics, specifically: reducing the degree of non-monotonicity (Section 5.5);
- a software architecture for developing incremental spoken dialogue systems based on the IU model by Skantze and Schlangen (2009) (Chapter 4) with fully integrated iSR (Section 5.2.1) and iSS (Section 7.3.1) components, as well as floor-tracking (Section 6.1) and micro-timing prediction (Section 6.2) capabilities, which is released as free and open-source software;
- incremental HMM-based speech synthesis embedded into the incremental architecture providing incremental access to manipulations of content and delivery parameters on various levels (Section 7.3.1) which provides for highly interactive system behaviour (Sections 7.4, 7.5);
- a system that co-completes a user's utterances (given that the content is known), showing that real-time end-to-end incrementality in connection with proactive behaviour works in practice (Section 6.2.6); and
- an incremental system that interactively collaborates with the user on utterances with the resulting interactions being more natural and more reactive than a baseline system without incremental processing (Section 6.1.2).

1.3 Previously Published and Co-authored Material

This thesis has not been conceived in an isolated environment and much of its content has previously been published, at conferences and elsewhere. This section clarifies which portions have previously been published (and where), and details the present author's share in previously co-authored material.

Evaluation of Incremental Processing The approach to incremental processing and its evaluation has previously been published in **(Baumann, Atterer, and Schlangen 2009)** and more comprehensively in **(Baumann, Buß, and Schlangen 2011)** on which much of **Chapter 3** is based (including the structure of the chapter and some of the actual, revised text). Chapter 3 additionally introduces a formalism for the dynamic evolution of incremental hypothesis sequences which is also used to formulate the evaluation metrics in that chapter and the iSR optimizations in **Section 5.5**.

INPROTK The Incremental Processing Toolkit INPROTK, presented in **Chapter 4**, has previously been described in **(Baumann, Buß, and Schlangen 2010)**, **(Schlangen et al. 2010)**, **(Baumann and Schlangen 2012d)**, and **(Baumann and Schlangen 2012e)**. The descriptions have been expanded and more details have been added, as well as the concept of *triangular data processing*.

Incremental Speech Recognition The INPROTK iSR module, its methods for iSR optimization, and evaluation metrics have first been reported on in **(Baumann, Atterer, and Schlangen 2009)** and **(Baumann, Buß, and Schlangen 2011)**, from which some text is borrowed. All experiments in **Chapter 5** have been re-run to reflect bug-fixes and the analyses and discussions have been considerably extended. The evaluation of N-best processing (which was first considered in **Baumann et al. 2009**) has been completely reworked to focus specifically on iSR metrics (instead of understanding as in Baumann et al. 2009).

INTELIDA The workbench for iSR evaluation presented in **Section 5.3** has been published, together with a viewer for incremental data, TEDview, in **(Malsburg, Baumann, and Schlangen 2009)**. While TEDview was developed primarily by Titus von der Malsburg, INTELIDA is the present author's work. Descriptions have been greatly expanded and updated to reflect the many changes since 2009, especially the new integrated GUI tool for evaluation.

Floor Tracking The floor tracking component as well as the example application to the *PentoSelect* system presented in **Section 6.1** have been first presented in **(Buß,**

Baumann, and Schlangen 2010). Experiments and results are re-reported and the overall structure of the paper has been re-used in the section. The architectural description of floor tracking has been extended, and discussions extended.

Overall, while Okko Buß is the first author of the aforementioned paper, care has been taken to report only the present author's findings, or to point out the taking over of material.

Micro-Timing Prediction **Section 6.2** is based on **(Baumann and Schlangen 2011)**, reflecting the structure and re-using much of the text. However, analyses have been performed again and in more depth and many details have been added. The section on estimating the reliability of predictions is entirely new.

Incremental Speech Synthesis The requirements for iSS and the InproTK iSS component have previously been described in **(Baumann and Schlangen 2012c)** but these descriptions have been considerably extended in **Chapter 7**. The experiments on incremental prosody production have been reported in **(Baumann and Schlangen 2012a)** and much of **Section 7.6** is based on that material but has been extended and deepened.

Integration with iNLG The integration with iNLG originally appeared in **(Buschmeier et al. 2012)** and the system description in **Section 7.5** has been re-focused to only those parts that the present author is primarily responsible for.

Hendrik Buschmeier has defined the domain and performed the evaluation as well as the statistical analysis, which is also pointed out in the relevant subsections.

2 Spoken Dialogue and Spoken Dialogue Systems

The naturalness of interaction (and with it the potential conversational competence) of a system depends to a high degree on the system's architecture which must provide for the sorts of behaviours that natural interaction entails. Of course, decisions on the architecture can only be taken after carefully modelling the problem at hand, in the present case spoken dialogue.

This background chapter looks at models for dialogic communication, especially regarding the "low level" parts of dialogue and communication and only lightly touching the "high level" issues such as understanding or reasoning. This stands in contrast to other descriptions of dialogue modelling which often only ever start on the level of full turns (e. g. Jurafsky and Martin 2009, Ch. 22; McTear 2002, pp. 55-75), and completely disregard *in-turn* processing.

We then discuss the general approaches to the architecture of spoken dialogue systems and describe the processing components conventionally used before pointing out some state-of-the-art example systems, and the extent to which they reflect the requirements for naturally interacting spoken dialogue systems will be examined. This analysis forms the basis for the approach to incremental spoken dialogue processing taken in the subsequent chapters.

2.1 Modelling Dialogue

This section deals with dialogue modelling *from the bottom up*. We start our description with a basic *model of communication* in Subsection 2.1.1, then discuss how complex communication systems can be managed through the use of *layering* in Subsection 2.1.2 and introduce how behaviour *emerges* in a complex system (which, as will be developed, dialogue certainly is) in Subsection 2.1.3.

After this discussion of communication *per se*, Subsection 2.1.4 approaches spoken dialogue more closely by discussing the issue of *grounding* between communicative partners. Acoustic communication through a common medium (usually air) is not trivial, in part because interference effects prohibit simultaneously sending and receiving messages. In plain words: it is hard to listen carefully to someone speaking while simultaneously speaking oneself. Human dialogue deals with this limitation by two means: (a) the use of a *backward channel* which is discussed in Subsection 2.1.6 and helps with grounding and channel management, and (b) channel management proper, discussed in Subsection 2.1.5. *Channel management* determines who is allowed

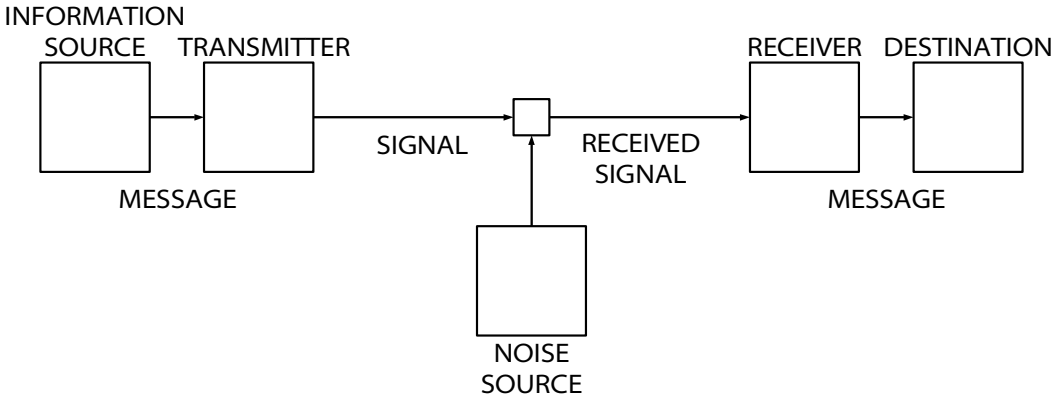


Figure 2.1: The Shannon-Weaver model as shown in (Shannon and Weaver 1949, p. 7).

to speak when in a dialogue, that is only towards the end of our discussion of dialogue modelling we finally reach the fact that in dialogue predominantly only one person speaks at a time – that people *take turns* speaking. This fact may seem self-evident at first thought and this is also the level at which the modelling of *dialogue content* often only begins. However, we try to develop that even though turn-taking is the major organizational principle, content is exchanged on a far more fine-grained level between dialogue participants and it is the goal of this thesis to show how this exchange can be enabled in a system (by using incrementality) and to show that it may improve a dialogue system's performance.

As we are primarily interested in the *means* of dialogic communication, not in the content of the dialogue, the discussion of dialogue modelling closes at the level of the dialogue turn. For higher-level dialogue modelling the reader is referred to Jurafsky and Martin (2009, Ch. 22), McTear (2002, pp. 55-75), Jokinen and McTear (2010, Ch. 2), and especially to Jokinen (2009).

2.1.1 The Shannon-Weaver Model of Communication

The now classic model of communication formalized by Shannon (1949) and popularized by Weaver (1949) is shown in Figure 2.1. Information from a *source* is transmitted in the form of *messages* which are *transmitted* as *signals* and reverse-transformed by a *receiver* to reach their *destination*.

Regarding the application of the model to the problem domain of spoken dialogue, we can conveniently rely on Weaver's own words:

> When I talk to you, my brain is the information source, yours the destination; my vocal system is the transmitter, and your ear and the associated eighth nerve[1] is the receiver. (Weaver 1949, p. 7)

Weaver does not explicitly mention the *signal* that is communicated via the *communication channel* (all that lies between the transmitter and receiver, shown as a small central square in the figure). In spoken communication, an acoustic waveform forms the *primary signal* to be communicated, that is, the signal that forms the basis for communication. In face-to-face spoken communication, the channel which transports the sound wave is plain air.

Signals may be perturbed by noise on their passage through the channel, which, in the model, is introduced by a specific *noise source*. In fact, it is convenient to assign all perturbations, whether they arise from the channel, or the transmitter, or the receiver to this noise source as well, so as to only have one such source in the model (which also gave rise to the term *noisy channel model*). An important aspect of successful communication lies in modelling the noise source itself and in determining the amount of redundancy in the signal that is necessary to allow the receiver to correctly decipher the original message given the received signal.

The model of communication that we're discussing was developed as part of a mathematical theory of communication (Shannon and Weaver 1949), and in the context of information theory. Thus, a great amount of detail is spent on specifying properties of the signal, the channel, and the noise source so as to allow for an unambiguous reverse transformation of the perturbed signal into the original message. There seems to be an assumption that either the correct message or no message at all will be extracted by the receiver (or that there are methods in place that allow for the unambiguous validation of received messages). Another assumption is that receivers and senders match each other perfectly.

For the context of spoken dialogue, it must be pointed out that the received signal may be so much perturbed by noise, or the receiver may be mismatching the transmitter to such a degree that the original message may not be recovered but that instead the *received message* differs from the original. In fact, if my brain is the information source and yours is the destination, and given that our brains differ, the messages must differ. More importantly, there is currently no way of finding out whether the original message in my brain equals the message that reaches yours. Section 2.1.4 below will come back to this problem.

[1] The outer and middle ear, the *nervus vestibulocochlearis* and the *nucleus cochlearis* comprise the human auditory system (Greenberg 1996), which Weaver seems to imply as the receiver.

2.1.2 Layers of Communication

Only rarely are communication systems as direct as in Figure 2.1 above, where the message is only encoded and decoded once to become a primary signal. When conversing over the telephone, for example, sound waves are not directly transported by air (only to the microphone and from the loudspeaker onward) but also by some sort of telephone machinery. This telephone machinery can itself be modelled using the above terminology with the microphone acting as transmitter, the telephone cord as the channel through which voltage differentials are transported, and the loudspeaker as the receiver. Notice that the sound wave, which had been the signal in the primary level, becomes the message on the lower level, with the sound wave message being signalled by voltage differentials (or zeros and ones in modern telephony).

This encapsulation forms the basis for communication network standards (like the OSI model; Tanenbaum 1981; Zimmermann 1980) which are using the above layering technique as a method of abstraction: details of the communication protocol on lower levels are decoupled and partially hidden from the higher levels of the *communication stack*. In the following, we examine how well this engineering approach works from a scientific/linguistic viewpoint. While *layering* (or, more formally, stratification) will be the result of our analysis, we have to start by looking at how to *structure* the language communication system.

Communication is a complicated business, and hence communication systems are complex systems which are potentially difficult to study. Following the *analytical method*, a large problem can be studied by dividing it into smaller sub-problems while keeping track of the division process (Descartes 1824, pp. 141-142). For this, we group similar phenomena to form the sub-problems which can then be looked at in isolation. To keep things simple, we will abstract away from details of phenomena which can be deemed irrelevant in the analysis of the sub-problems. (That is, we assume that small inaccuracies in the analysis of sub-problems will have only small effects in the overall analysis; see Subsection 2.1.3.) Transferring this methodology to the analysis of systems and their structure, we try to differentiate the parts that comprise the system (we will call these parts sub-systems or *components*) and find the relations between those parts.

Spoken language is probably the most versatile, sophisticated and hence most complex communication system the human mind has developed. Language comprises many different phenomena (sound waves, words, phrases, ideas, to name a few) which interact in many ways and it may be obvious that some sort of separation of the different sub-tasks is necessary (or at least helpful) when analyzing and modelling dialogue. Thus, we have to take close looks at the language system and its structure.[2]

[2] An excellent discussion and differentiation of the terms *system* and *structure* can be found in (Serébrennikov et al. 1975, pp. 6-15).

Table 2.1: The conventional division of linguistics into subdisciplines.

pragmatics	the study of meaning in context
semantics	the study of meaning
syntax	the study of sentence structure
lexicology	the study of words
morphology	the study of forming words
phonology	the study of a language's sound system
phonetics	the study of speech sounds

The structure of language as a system has been analyzed by many and for a long time, not least intuitively by those who construct or enjoy poetry and other plays on words. The history of scientific analysis of language is probably as old as science itself.

Linguistics has been conventionally structured – using the scientific method – into a number of subdisciplines that mostly follow a layering approach. Textbooks often handle the fields separately (e. g. Grewendorf, Hamm, and Sternefeld 1989). Fields build on each other, as can be seen in Table 2.1, but the question of organization into fields is often seen primarily by tradition (Grewendorf, Hamm, and Sternefeld 1989, p. 38).

In contrast, Serébrennikov et al. (1975, pp. 33-70) discuss the history of structuring the language system with regards to how well a given structure fits the problem. They, too, find *stratification*, i. e. the structuring in layers the most fitting analysis of the language system. According to Serébrennikov et al. (1975, pp. 71-76), a *layer* (or *stratum*) can be characterized by a homogeneity in structure and content, and by an inventory of elements which are atomic from the point of view of this layer (i. e. they cannot be uniformly sub-divided with the methods of this layer). Layers are autonomous in the sense that their internal structuring, the rules that they follow, is independent of other layers. However, layers are interdependent with other layers in the sense that there are connections between layers and relations between the layers' elements. These relations are representational, i. e. some element on one layer is represented by or itself represents one (or more) elements on another layer. Serébrennikov et al. (1975, pp. 78-79) argue that relations are not constitutional, i. e. one element does not comprise (or is not comprised of) elements of other layers. I would argue that the difference of whether a word is represented by some phonemes or

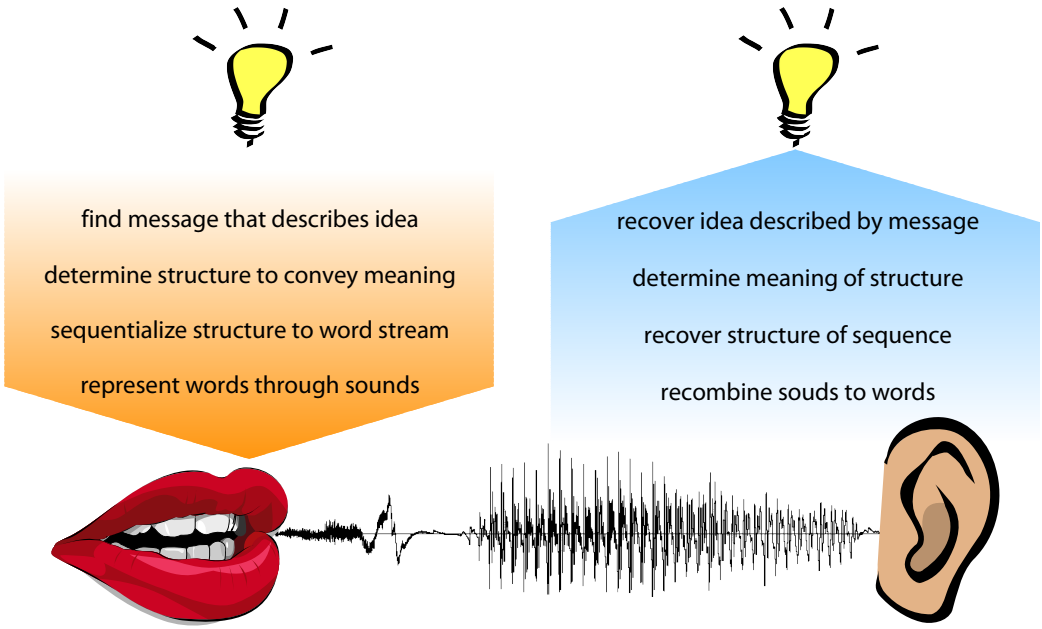


Figure 2.2: The chain model of communication.

whether it is constituted by these phonemes is of little relevance in practical systems.[3] In short, Serébrennikov et al. (1975) give a detailed and convincing argumentation for the well-known linguistic partitioning into layered subdisciplines such as phonetics, phonology, morphology, syntax, semantics, and so on.

Where Serébrennikov et al. (1975) embrace layering from a very theoretical, systems-based perspective, Levelt (1989) gives proof for layering from a psycholinguistic viewpoint, with a focus on speech production. Levelt's partitioning into levels (conceptualization, formulation, and articulation) is less fine-grained than the differentiation of linguistic fields, as he argues that only these levels do not interact beyond simple transmission of information. In contrast, the other, more fine-grained levels (such as grammatical and phonological encoding) can be shown to interact more closely (Levelt 1989, p. 15). As a conclusion, the autonomy/interdependence of modules and granularity of sub-division (be it into layers or otherwise) cannot be answered absolutely. Rather, it is a decision to be weighed for the goals at hand.

[3] However, the notion of representation instead of constitution more easily explains why it is possible that phonemes may be *ambisyllabic*, that is, belong to two adjacent syllables (Swadesh 1937).

Layering communication along the Shannon-Weaver model results in the *chain model of communication* (Pétursson and Neppert 1996, p. 35) as shown in Figure 2.2: an idea (e. g. "I want the soup to be saltier") is to be transmitted from one to another communicator and transformed multiple times when it passes from one processing layer to the next. The processing layers use different protocols and message forms for their tasks. All messages are ultimately transformed to, and all protocols are ultimately based on some physical transport layer. For speech, this is the audio waveform.

Both language and computer networks are communication systems. Thus, re-applying the models for computer networks to language may be a worth-while endeavour. Taylor (1988) and Taylor and Waugh (2000) carry the ideas of the OSI model for communication systems into the domain of human-computer interaction and specifically to dialogue analysis with their *layered protocol theory* (LPT). In LPT, a *layer* is constituted by the fact that there may be feedback between the sender and receiver of messages on this layer. This distinguishes their layers from the simple example for layers in the telephone example at the beginning of this subsection, which are seen as "merely stages in the coding and decoding of a message, and not as communicators" (Taylor 1988, p. 186). Feedback is necessary when high-level messages cannot unambiguously be determined based on the incoming message. The feedback messages on a lower layer that are passed back and forth in the process of establishing the message on a higher level is called the *protocol loop* (Taylor and Waugh 2000, p. 195) and extends the pure chain model of communication by allowing sub-systems to communicate back and forth independently to establish a higher-level message.

In LPT, layers are seen as independent and autonomous (as they often are in networked computer systems). Sub-problems of communication are handled independently from the other system components but in direct coupling with their respective counterpart in the other communicating system. The following subsection further investigates the concept of coupling.

2.1.3 Emergence of Behaviour in Complex Systems

The method of *divide-and-conquer* that has been the guidance in the previous subsection and which helps to manage the complexity of language by dividing it into manageable sub-components, has been criticized as *reductionist*: phenomena are grouped and looked at in isolation. A main question is: can all phenomena be handled in this way? That is, is it possible to assign each phenomenon to one (or only a few) of the system's modules, but not to the interaction of components in the system as a whole? The answer is: no.

The question is not restricted to dialogue (or to linguistics) but exists in many scientific domains. Coming from the field of biology, Bertalanffy approaches similar problems in his General Systems Theory (Bertalanffy 1972). General Systems Theory

uses the more holistic concept of *emergence* where phenomena may emerge from the *interaction* of separate modules. In other words, phenomena are not necessarily assigned to a sub-component but to the combination of sub-components and their interrelation.

The ordering of speakers in a dialogue is a perfect example of emergence: while it has been recognized that prosody plays an important role in determining speaker change (Gravano and Hirschberg 2011), it is only together with word information that turn-taking events are registered by human listeners with full performance (Wesseling, Son, and Pols 2006). Thus, it would remain unclear whether a component for word recognition, or one for prosody analysis should be in charge of turn-taking analysis. More importantly, while it is possible to have one module in the system take care of an issue that different linguistic layers are involved in, behaviour also emerges from the combination of modules across independent systems: one of the largest influencing factors for speaker change arises from the reaction of the interlocutor, who has a large share in determining speaker change.

As a result, modelling of complex behaviour such as turn-taking must take into account the other speakers involved and conclusions can only be drawn from investigations that take into account the *interaction* between speakers and all relevant sub-systems. Studying complex behaviour purely on the basis of corpus experiments, for example, is likely to fail. The question of relevance of sub-systems for final system behaviour is also stressed by Larsen-Freeman and Cameron (2008) who explain why some results that seem to work in isolation fail to work in full systems.

Another criticism of the divide-and-conquer approach from above is the problem of stability: it is assumed that small inaccuracies or errors in one component will only have a small effect on the overall outcome in the full system. This assumption is called *linearity* but real systems turn out to be highly non-linear and chaotic: small changes often have large effects. Dialogue is highly non-linear: initially small misunderstandings are often only resolved some utterances later in which case all that has been said in the meantime may have to be invalidated and reconsidered. Timing is also highly non-linear: if I fail to jump in with my witty remark now, I will likely be unable to do so one or two words later. In many cases, I will have to completely rephrase or give up on commenting altogether.

Complex systems are stabilized by *coupling* and *attractors* as outlined by Larsen-Freeman and Cameron (2008). Coupling is the close interaction of components across different independent systems (and can also be achieved by 'communicators' in LPT; Taylor and Waugh 2000). The coupled system is steered towards attractors, which are states that avoid non-linearities breaking havoc and instead provide relative stability.

To conclude, stratification (layering) is an important principle in language itself and hence should be considered in language processing systems. Further, *strict* layering has many implementational advantages (see next section) but it is somewhat insufficient to fully deal with all language phenomena. Some phenomena only emerge when combining components to form a system and the relations for combination should ideally be more powerful than simple chaining of processing components as in the chain model of communication. In the approach described below (see Chapters 3 and 4) the implementation is limited to a principled architecture for layering of components that at least avoids some of the disadvantages of conventional layering implementations by allowing faster reaction and shorter feedback loops through incremental processing. While the *module* architecture to be developed is relatively traditional, the *data* that is produced in the different layers follows the connectionist approach and enables the principled interaction with minimal elements from other layers that represent partial information, permitting modules deep access to other modules' data.

In any case, when developing systems that interact through dialogue with humans, the idea of *dialogue as a coupled system* should be kept in mind: all SDS actions will be responded to by the human interlocutor, who may either balance or multiply small mistakes, depending on whether a stabilizing attractor is available and sufficiently attractive. An example application in which small errors are balanced by the human user is given in Chapter 5.6.

The discussion above has stressed the view on the overall system but in this thesis only small, partial systems will be built and some evaluations will even operate on individual modules in isolation. The author hopes that he has paid enough attention to the full system perspective while working on these individual modules, so that the results obtained will remain valid also in complete SDSs.

2.1.4 Establishing Common Ground

A fundamental goal of communication is the establishment of mutual understanding. When conversing, we want to share understanding. For this, the conversants need acknowledgement that their messages are being understood, that is, that the conversation is actually functioning. One of the strengths of natural language is its ambiguity (which allows for compact representations and can most often be disambiguated through context). However, misinterpretations are also possible: Shannon's model (cmp. Subsection 2.1.1) expects that 'faulty', uninterpretable messages are identifiable but this is often not the case for natural language messages where 'faulty' messages are instead just mis-interpreted as some other message. As faulty messages cannot simply be identified and re-queried from the sender, a different, more powerful and more flexible method is required to negotiate whether mutual understanding is be-

ing achieved. In dialogue, *grounding* is the method to establish that the dialogue participants are in fact talking about the same, shared thing.

Grounding happens on many conversational levels, and often subconsciously as part of the joint activity between speaker and listener that is dialogue interaction (Clark 1996). Grounding happens through the acknowledgement of 'contributions' which are grouped in 'adjacency pairs' (Schegloff 1968), for example question/answer pairs or greetings. The size of these contributions can be argued about but often full utterances are seen as contributions in dialogue. Going a bit further, I believe that grounding can be more fine-grained, with subtle signals from the listener to the speaker and back. Incremental processing can hence significantly improve a system's grounding abilities and some results highlighting this ability will be shown in an example application in Chapter 6.1.2.

2.1.5 Taking Turns

The previous subsection mentioned 'contributions' without detailing their progression in spoken dialogue. The present subsections aims to clarify this matter.

Spoken dialogue is naturally transmitted through air and thus over a shared communication channel. The shared channel has to be *divided* (to allow for the back-and-forth interaction that allows the establishment of common ground) and there are (at least) two constraints on this: (a) shadowing and interference effects of overlapping audio signals in the absence of channel division, and (b) limited brain resources that forbid simultaneous production and reception of speech. There are several ways that a channel could be divided by. For example in digital communication, modems divide the common channel (the wire) by transmission frequency. Humans do not simultaneously speak at different frequencies – instead they largely *time-divide* the channel. The job of channel management is thus to coordinate which participating communicator speaks *when* during the dialogue (also termed: 'who holds the floor'). Time-dividing the shared communication channel appears to be a language universal (Miller 1963, as cited by Duncan and Niederehe 1974) and the sub-system of natural dialogue dealing with it is called the *turn-taking* system.

A "simplest systematics for the organization of turn-taking for communication" has been developed by Sacks, Schegloff, and Jefferson (1974) which states that a dialogue participant's speech is constructed from *turn-constructional units* (TCUs) which are segments of speech of various sizes for which the duration can be projected by the listener and which allows a speaker change to follow. A turn-constructional unit

ends in a *transition relevance place* (TRP), at which the floor *may* transition from the current speaker to another.[4]

There are some relatively simple rules for the turn allocation to a next speaker and according to Sacks, Schegloff, and Jefferson (1974), turn-taking is managed *locally*, that is, floor changes are determined from the dialogue participants' behaviour only in the vicinity of the decision to be taken.[5]

For the most part, turn-taking is not managed on the content layer of the interaction: while the floor can be verbally assigned to someone, most often this is done by other, more subtle means (gaze, gesture, posture shifts, prosody). Additionally, turn-taking seems to be dealt with partly on the sub-conscious layer: even if interlocutors disagree or have an argument, they follow the turn-taking rules, at least to some degree.

Turn-taking signals that inform about upcoming TRPs have been investigated in great depth (Duncan and Niederehe 1974; Gravano and Hirschberg 2011) but often based on corpus studies which leave out the dynamics of dialogue as a coupled system that was outlined above. Section 6.1 covers turn-taking and turn-taking signals. However, the detection of turn-taking signals is only rudimentary in the implemented system.

2.1.6 Feedback and the Backward Channel

Dialogue requires bi-directional communication for which various variations can be conceived. A common channel can be divided so that messages can be passed along in both directions simultaneously (often called 'duplex'), or the channel can be time-divided, which is called simplex or 'half-duplex' communication. Often, communication that involves longer messages is sometimes divided into one content-bearing channel and an additional *control* or *backward channel* where status about delivery on the main channel is reported (for example, the *file transfer protocol*, FTP, uses two distinct connections for control and data; Postel and Reynolds 1985).

At first sight, turn-taking appears to be sufficient to organize dialogue in light of the single (acoustic) channel using time-sharing. However, this is an over-simplification: dialogue also involves what Yngve (1970) has first called "back-channel" utterances that overlap with ongoing turns (without being interpreted as out-of-place or as interrupting an ongoing turn; Duncan and Niederehe 1974). Such utterances can be short feedback like "`yeah`", "`good`", "`right`", conversational grunts (Ward 2006), audible inhalation, lengthening or softening of delivery in the primary channel, and

[4] Sacks, Schegloff, and Jefferson (1974) emphasize the "place"-aspect in TRPs, possibly to avoid the discussion of whether a TRP is a point in time (without temporal extent), or a period in time (with temporal extent).

[5] This is actually what distinguishes dialogue from e. g. group discussions where lists of next speakers are being kept and speaker turns are assigned by a discussion leader.

so on, that are used to give feedback on a more fine-granular level than that of the full turn.

Thus, human dialogue is neither simplex nor duplex but rather somewhat *one-and-a-half directional*: the primary, content-bearing channel works simplex, but there is also a backward channel, transmitted over the same signal path, on which feedback regarding the current state of grounding is given, and which supports the primary mechanisms for channel assignment and grounding.

Speakers integrate feedback on the back-channel (or the absence of feedback) into their ongoing turn on the forward channel. Thus, failing to present a back-channel where it would be appropriate may disturb the dialogue flow.

2.2 Components and Architecture for Spoken Dialogue Systems

SDSs are modular not only to mirror the structure of speech as a communication system but also for engineering reasons: developing large, monolithical systems is inefficient and at some size typically becomes impossible, especially when the system should be adaptable and extensible (Parnas 1979). Thus, if the system is not to be monolithical, the questions of the system's components and their interconnection, i. e. the question of the system architecture arises.

By and large, the partitioning of the SDSs into components – especially for those on the lower level – does not vary much across systems and follows the layers of communication outlined in Subsection 2.1.2 above. A short description of these standard components and their tasks is given in Subsection 2.2.1. Components can be combined to form a system in several ways and Subsection 2.2.2 discusses approaches to this aspect of the architecture. Finally, Subsection 2.2.3 describes the architectural decisions taken for the toolkit developed as part of the thesis.[6]

2.2.1 Components of Spoken Dialogue Systems

In practice, the partitioning of the system into components varies little between a broad range of spoken dialogue systems and the following paragraphs briefly describe the major components in the order that they are triggered when a user starts to talk to the system.

Voice-Activity/Turn-Taking Estimation One basal (but very important) task for a dialogue system is to determine whether the user is speaking or not. In speech recognition, this is called *voice activity detection* (VAD) and is used to reduce the amount of processing on silences preceding and following the user utterance. In a

[6]The toolkit itself, InproTK, is described in detail in Chapter 4.

dialogue system, voice activity detection is additionally required to determine whether the user starts speaking (i. e. requests the floor) or finishes to speak (i. e. releases the floor), presumably waiting for a system response; a dialogue system should deliver it's own turns (i. e. take over and give up the floor) accordingly. Standard dialogue systems use the VAD's *endpointing* to determine when a user turn is over and the *onset* to determine user *barge-in* (i. e. the user interrupting the system's turn). A time-out can be started when the system releases the floor allowing the user only a certain lag before she starts her own turn. For more complex, conversationally competent spoken dialogue systems, we prefer the term *turn-taking estimation*: the task of determining the user turns' beginnings and ends. Turn-taking estimation will be covered in Chapter 6.

The decision of whether the interlocutor is speaking or not may sound trivial, but it is in fact quite hard as SDSs are often confronted with speech over the telephone, with background noise, and a range of different speakers; thus, the field of VAD has remained an active area of research.[7] In dialogue, the end of a user's fluent stretch of speech does not necessarily also indicate the end of the user's turn but could also indicate a user hesitation, complicating the issue of turn-taking estimation.

Speech Recognition Speech recognition (also called *automatic speech recognition*, ASR) is the process of determining the words that were spoken in a speech waveform. We speak of *rich speech recognition* for speech recognizers that additionally output durations, prosodic aspects, and other properties of recognized words (which may be useful for disambiguation in further processing). Speech recognition is dealt with in great detail in Chapter 5.

Natural Language Understanding Natural Language Understanding (NLU) interprets the meanings of the words that have been recognized in the previous step. This may include syntactic and semantic analyses, integration with dialogue history, and so on. Ultimately, some *logical form* of the user utterance is produced which describes the user turn's meaning in context. Individual units of relevant user behaviours are conventionally called *dialogue acts* (DA).

Dialogue Management The dialogue manager (DM) contains the system's *dialogue state*, updates the state by integrating user contributions (based on the incoming DAs from NLU) and determines from the updated state what the system should say next (in terms of one or more output DAs to pass on to natural language generation, NLG).

[7] All *Interspeech* conferences between 2007 and 2012 have featured full sessions specifically on the topic of VAD.

Domain Reasoning A domain reasoning component connects the dialogue system to other relevant systems that are necessary in the domain, for example information and booking systems for SDSs that handle the booking of airline tickets, hotel rooms, or pizza delivery. In other domains, domain reasoning is responsible for interacting with the operating system of a robot (querying sensors and commanding actuators). Domain reasoning is often not a completely separate component from the dialogue manager but is at least partially integrated into the DM which limits the latter to the particular domain. While domain independence would be nice in principle, it is much easier to devise domain-dependent DMs.

Natural Language Generation Natural Language Generation (NLG) turns dialogue acts into word sequences. A broad range of techniques, from simple, pattern-based techniques that are very limited in domain to grammar-based broad approaches (e. g. SPUD, Stone et al. 2003) can be used.

Speech Synthesis Speech synthesis usually turns written text into spoken audio (which is why speech synthesis is often called *text-to-speech*, TTS). In fact speech synthesis may also use more general general linguistic representations, possibly blurring the line with NLG.

Due to the limited quality of synthesized speech, many current-day applied SDSs rely on *canned speech*, audio snippets that are specifically recorded from a speaker to closely match the intentions assigned to a particular DA. While perfect in this regard, pre-recorded messages leave no room to flexibly reacting to the specific situation which is necessary for more user-centered interactions. Speech synthesis is dealt with in great detail in Chapter 7.

Many of the components used in SDSs are not specifically developed for spoken dialogue but for other usage scenarios. For example, speech recognition is vastly used in dictation, and speech synthesis is used in reading out texts – thus, most off-the-shelf speech recognition and synthesis software is tailored towards written language even when used in a spoken dialogue system. Likewise, many NLU and NLG tasks are related to (text-based) web applications, such as generating weather reports or understanding/categorizing the content of forum messages or tweets. This may be one of the reasons why incremental processing in input and output components has not been a part of dialogue systems development from the start, even though it is natural, obvious, and enables more human-like behaviour. Also, especially the lower-level components (ASR and TTS) internally work in a left-to-right (i. e. easily incrementalizable) fashion anyways and have not been used incrementally

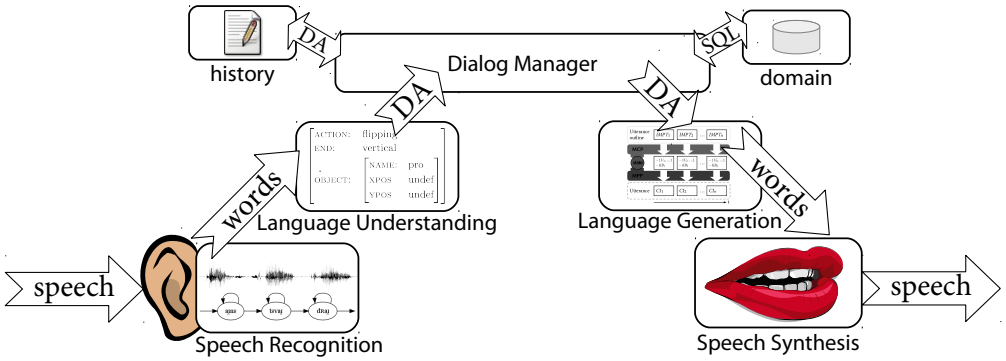


Figure 2.3: Schematic view of the architecture of and information flow in (most) spoken dialogue systems.

only because this feature was of no concern in the applications the off-the-shelf components were mainly developed for.

2.2.2 Interconnection of Components

This subsection is concerned with the interconnection of the components described in the previous subsection to form a modular system. There are two possible views on the interconnections in the architecture: one is to follow the *information flow* in the system, the other is to follow the *control flow* in the system. Incidentally, these two flows partially overlap.

The simplest form of information flow is realized in a *pipeline*, following the chain model of communication (cmp. Subsection 2.1.2) as depicted in Figure 2.3. In a pipeline (also: cascade, Guhe 2007, p. 67), the modules of the system are ordered sequentially and the output of one component is passed on as input to the next. Pipelines are the predominant architecture for at least the lower-level parts of most spoken dialogue systems.(McTear 2002, p. 105; Jokinen and McTear 2010, Chap. 1.2)

The main advantage of a pipeline is its simplicity: in a plain pipeline, a module receives all its input from the preceding module and the generated output is passed exclusively to the next module. Thus, only two modules need to agree on the interchange format for the data that flows between them and formats can be differentiated for different types of data. Management of concurrency is straightforward in a pipeline, as coordination between neighbouring modules can be achieved by waiting for the next module to be ready to accept data (or waiting for the previous module to provide data). However, concurrent processing requires that input become available continu-

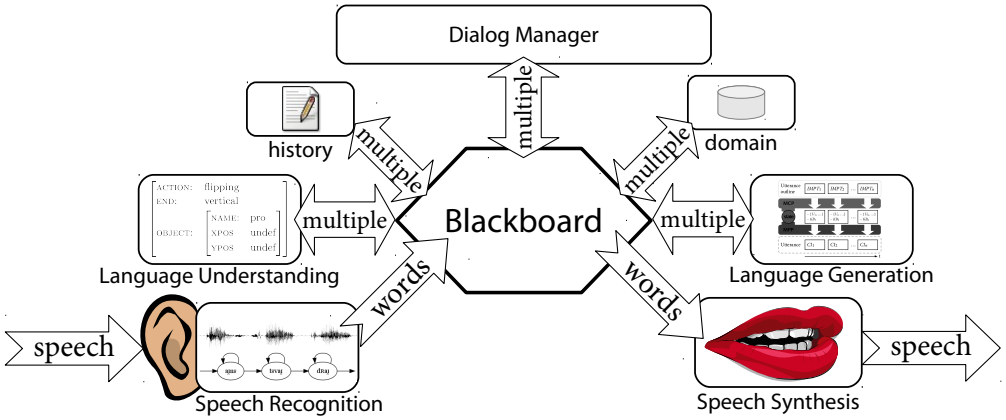


Figure 2.4: Schematic view of the architecture of a blackboard-based spoken dialogue systems; arrows labelled 'multiple' use multiple input and/or output data formats.

ously and in small chunks, in order to spread processing load over a larger period of time. If input is delivered infrequently and in larger units, consumers will often be blocked waiting for input and be overwhelmed when input is provided all at once.

Pipelines can be operated *pull*-based, or *push*-based, that is, either a module waits until it is queried for output, or a module is triggered into action by some input being provided. Either way, enabling bi-directional communication between modules, especially in the concurrent processing case, neutralizes much of the conceptual simplicity of the pipeline approach.

While conceptually and structurally simple, some of the interaction that is necessary for natural dialogue capabilities is difficult to achieve in simple pipelines such as the integration of high-level knowledge on lower levels, or passing on low-level information to the highest levels (i. e. surviving intermediate levels). As an example, conceptual pacts (Brennan and Clark 1996) or mimicking the user are impossible to realize without passing on lower-level information because the precise words spoken by the user (and interpreted by NLU) remain unknown to the NLG component. To conclude, pipelines are simple, but maybe too simple.

Another architectural model allows the interchange via a common *blackboard* as depicted in Figure 2.4. In this model, components write data in their own formats into a common data store (the blackboard), and other components read these data to derive their own output.

In such a *connectionist* model, every module has access to all other modules' data (and, depending on the implementation, might even be able to change other modules' data). Thus, decisions can be based on the full information that is available in the system, regardless of which module has generated it (potentially resulting in better decisions). However, data formats need to be standardized for an effective use of this advantage. Furthermore, access to 'distant' data is only possible if the provider of that data is part of the system, limiting the modularity and re-usability of parts of the system in different contexts.

While in principle all modules can work independently and concurrently, a major disadvantage of blackboard architectures is that of synchronization of data access in light of concurrent processes. Multiple independent accesses to the blackboard can lead to inconsistent reads if some of the accessed data is written to in-between. Inconsistent reads can be avoided if the blackboard supports transactions and blocks manipulation of data while a transaction is in progress. This, however, incurs considerable complexity on the implementation which then needs to implement full-blown database capabilities.

Finally, modules might circularly react on each other (e. g. module A provides some data X that module B reacts on by providing some data Y to which module A reacts with retracting data X, and so on). Systematically detecting and handling such loops, either at runtime or beforehand, may be problematic.

In short, where pipelines may take too seriously the chain model of communication, blackboards completely disregard the layered structure of spoken dialogue. Blackboards do not come with the limitations inherent in a pipeline but this power comes with an increase in complexity and issues of stability that implementing modules must be aware of.

The other view on the architecture is that of *control flow*, the paths that *decision making* takes in the architecture. Figure 2.5 gives a schematic view of the exemplary control flow in an SDS in which control is indicated by dark (red) arrows. As can be seen, all control is exerted by a dialogue controller that doubles as dialogue manager in that system. Other models are possible as well: a system can be purely reactive, with a pipeline of modules that is triggered into action by their respective input (starting from incoming user speech). Alternatively, control can be shared between multiple modules in the system (e. g. time-shared with different modules being responsible in different moments, or sub-divided responsibilities for different aspects of control like turn-taking and contribution-determination).

Where pipelines have a natural control flow, plain blackboards do not define any control flow and must resort to update-notification mechanisms or frequent polling.

Commands may be needed during a dialogue, to influence processing modules (e. g. ASR grammars may need to be changed per dialogue state). However, such

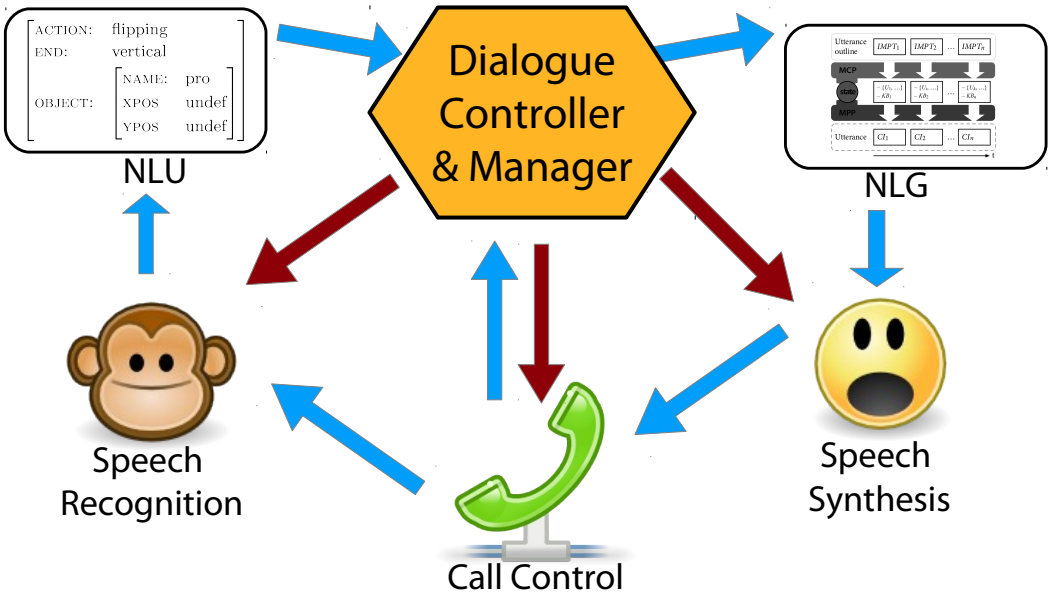


Figure 2.5: Schematic view of control and information flow in a spoken dialogue system; dark (red) arrows indicate control, light (blue) arrows indicate information flow.

control messages may interfere with ongoing processing in the pipeline and technical limitations may exist when control messages can be executed.

2.2.3 Discussion

Our architecture aims to combine the advantage of pipeline architectures (structured information, decoupled components, defined control flow) with the flexibility of blackboard architectures (concurrent processing, access to and manipulation of all system data). The architecture fully embraces incremental processing and supports efficient increment updates, enabling concurrency with minimal locking in the pipeline. Furthermore, a uniform data container for all data and an interlinking of units to form a network that represents the system's state enables modules to also access data across multiple steps in the architecture, partially unifying the pipeline and blackboard approaches.

The issues of control flow are potentially more important in full, deployed SDSs that need to robustly handle thousands of calls without human intervention even when sub-components are sometimes faulty. In contrast, the systems developed in this

thesis were never meant to be "mission-critical" and are often only partial, enough to demonstrate and assess the value of a certain component. Thus, control is often reactive, driven by user input once the system has been started. However, one system will demonstrate partially distributed control for turn-taking and content issues, respectively (see Chapter 6.1).

The architectural decisions taken will be further justified in Chapter 4 where the toolkit for incremental processing InproTK is described. However, we first describe some specific *systems* and their capabilities in the next section to analyze what exactly are some of the missing capacities of current-day spoken dialogue systems.

2.3 State of the Art in Spoken Dialogue Systems

This section very briefly reviews the state of the art in spoken dialogue systems by first presenting the industry standard for applied SDSs, then looks at some related research systems, and finally follow up on some recent advances in commercial systems.

2.3.1 Commercial, Standards-based Systems

An industry consortium led by the *World Wide Web Consortium* has developed since 1999 (Lucas 2000) a suite of (XML-based) standards for simple, state-based dialogue systems (often called *voice user interfaces*, VUIs in the industry):

VoiceXML is the core standard (and eponymously used for the whole family). It defines dialogue management based on a simple slot-filling approach. VoiceXML documents are linked together and user replies determine what link is followed, which is also why processors for VoiceXML are often called *voice browsers*.

SRGS, the *speech recognition grammar specification* is a standard way of specifying the expected user utterances (and which ASR is limited to understand). Grammars may change between dialogue states (that is, the voice browser controls ASR per-dialogue state) and may contain embedded ECMAScript to shape the speech recognizer output into a logical form.

SISR, *semantic interpretation for speech recognition* specifies the format of the logical form that ASR returns to the voice browser.

SSML, *speech synthesis markup language* is used to influence prosodic and other factors of the system's speech output. Finally,

CCXML, *call-control XML* is a standard to manage the setup and tear-down of a dialogue with telephony or other voice-streaming hardware.

The family of protocols around VoiceXML has spurred development and application of voice user interfaces since 2000 (Lewis 2011, Chap. 5.1), as it has established a framework for modular, interoperable components (developed by different, specialized companies). In this respect, the loose coupling of components via standard

network protocols based on XML has helped tremendously. Furthermore, commercial toolkits for developing VoiceXML-driven GUIs exist, often providing graphical interfaces and other productivity enhancements.

While dialogue management as defined by VoiceXML itself is relatively limited (or rather, becomes cumbersome for complex dialogue strategies), a big advantage of the standard is the fact that individual VoiceXML snippets can be generated dynamically on a web-server, which may itself transparently implement more powerful dialogue models and use VoiceXML only as its interface language towards the voice browser.

However, VoiceXML is too narrowly defined for exploring larger dialogue issues of research interest. Specifically, VoiceXML uses an inherently turn-based processing scheme, making meaningful incremental and sub-turn-based processing impossible.

2.3.2 Related Research Systems

The *Let's Go!* Systems The CMU Communicator Toolkit (Rudnicky et al. 1999) and its successor Olympus/RavenClaw (Bohus and Rudnicky 2009) are architectures that allow for a modular design of SDSs, with modules connected in a strict pipeline (but using a central control hub) via the Galaxy message passing system (Seneff et al. 1998).

Let's Go! is the name of a public bus information system run by CMU and implemented in the RavenClaw architecture (Raux et al. 2003). The toolkit and system have been shown to work in a highly reactive system with fast turn-taking capacities (Raux and Eskenazi 2009), which forms a fast-path signal from the audio input to the DM that a response should be output. In that system, there is no deep understanding during the utterances so that pro-active/predictive interventions are impossible. Furthermore, processing on the fast-path and in the processing component could result in different intermediary and utterance-final behaviours (such a system might nod repeatedly during the utterance to signal understanding and finally reply "`I did not understand`").

Data for the bus domain has also been made public and a challenge for bus information systems has been organized (Black and Eskenazi 2009; Black et al. 2011). In this context, the domain is also used for incremental SDS research, foremost by Selfridge in his ongoing dissertation work: Selfridge et al. (2011) investigate iSR and find similar results to those presented in Chapter 5. Selfridge et al. (2012b) presents a (partially) incremental SDS for the *Let's Go!* domain and features a combination of iSR and POMDP-based dialogue management (Selfridge et al. 2012c).

USC-ICT Systems The virtual humans group at the *Institute for Creative Technologies* develops multi-modal virtual embodied conversational agents that include some incremental technology. Incremental understanding of partial utterances has been

shown by Sagae et al. (2009) and DeVault, Sagae, and Traum (2009) aimed to find the point of maximum understanding during the utterance, which together allow to finish an utterance from that point on (DeVault and Traum 2012b) and has been extended to take system confidence into account (DeVault and Traum 2012a). A model for incremental grounding during the utterance is also being developed in this context (Visser et al. 2012).

SEMAINE The SEMAINE project aims to build receptive, sensitive, virtual artificial listening agents that aim to engage the user in a conversation (for as long as possible) (Maat 2011). The agent focuses on swift turn-taking and emotion recognition (based on prosodic analysis), while it in fact does not truly understand what it is being told.

Joustra (2011) finds all agent reactions to be sluggish when compared to human-human interactions. This once more indicates that prosody alone is insufficient for turn-taking estimation and highlights the need for turn-taking based on a fully incremental systems.

The YMIR Architecture Work on the YMIR architecture started with an interactive multi-modal *humanoid* agent that performed gestures (Thórisson 1997) and was later extended to learn and perform full turn-taking (Jonsdottir, Thórisson, and Nivel 2008). The architecture is very fine-granular and tries to model the complexity of real-time dialogue (Thórisson and Jonsdottir 2008). However, processing in the architecture is shallow and does not include understanding. As a consequence, it cannot be used to build full dialogue systems.

The NUMBERS System The NUMBERS system (Skantze and Schlangen 2009) is a direct predecessor of the work presented here and also based on the IU model (Schlangen and Skantze 2009). It incorporates some early findings on iSR (to be presented in Chapter 5) and is otherwise based on the Higgins architecture (Skantze 2007) and was later extended into JINDIGO (Skantze 2010), an alternative toolkit for building iSDSs.

2.3.3 Advanced Commercial Systems

Commercial systems have been greatly advanced since Apple introduced *Siri* to their smartphones in late 2011,[8] where it can be used as an additional modality for interaction with the system. Siri's technological breakthrough is primarily in the area of semantic understanding and system integration, not so much in interaction schemes that would advance over other commercial systems.

[8] http://www.sri.com/work/timeline/siri

Google had previously developed an incremental web search method[9] and correspondingly, Google Voice Search[10] also features incremental speech technology.

According to McGraw (2012), Google's incremental speech recognition now uses incremental hypothesis smoothing as detailed in (McGraw and Gruenstein 2012), which itself is based on the methods to be described in Chapter 5, however in combination with a much more advanced speech recognizer and using machine-learned stability estimates.

Side-by-side comparisons clearly indicate the advantage of incremental processing, for example when answers to spoken queries are given with less latency and visual feedback on the recognition result is returned incrementally.[11]

The fact that incremental dialogue processing has successfully entered the market does not only prove that this thesis should have been written much more rapidly, but also shows that the topic at hand is economically viable.

2.4 Summary and Discussion

This chapter argued that dialogue as a communication system is complex, as it encompasses many different types of information on different layers, and that both the production and integration of feedback occurs with little latency. Dialogue uses turn-taking as organizational principle. However, time-divisioning is not strict but rather a gradual process in which the speaker/hearer relation is determined collaboratively, and the floor is shared among several dialogue participants with proportions that change over time. Dialogue aims to exchange information (in the form of messages) that need to be grounded, that is, being given feedback on the success of delivery of the message. The division between main and backward channels are not clear cut and grounding feedback can be transported on both. Feedback is given concurrently to ongoing 'main' messages, resulting in tight feedback loops. To model these tight feedback loops, flexible incremental processing is needed.

Dialogue requires a multitude of different types of processing, thus a practical system will need to be composed of multiple components that work together. Regarding the modularization of dialogue processing, layered protocol theory describes communicators that independently provide reflexive behaviour (this will be explored with hesitating speech synthesis in Chapter 7.4). In LPT, modules also use informed decision strategies to determine whether to pass on a (partial) hypothesis yet or to wait for more input (iSR optimization performs this task and is dealt with in Chapter 5.5). Still, LPT specifically dis-allows interaction across processing layers.

[9] http://www.google.com/instant/

[10] http://www.google.com/mobile/voice-search/

[11] http://vimeo.com/52497584

A different approach is taken by Thórisson (2008) who argues that dialogue is a *heterogeneous, large and densely coupled system* (HeLD) that is best described as a large number of very small modules that interact tightly with many of the other modules in the system. This view is successfully implemented in the YMIR architecture for turn-taking (Thórisson and Jonsdottir 2008). However, while it has been successfully extended to multi-modal and multi-party turn-taking (Thórisson et al. 2010), its modules are far more fine-granular (and at the same time less sophisticated) than those used by practical systems for spoken dialogue (and the degree of sophistication of the modules is lacking). In fact, YMIR focuses solely on turn-taking and is missing components for speech recognition or synthesis, or higher-level processing, thus not showing that these are possible in the HeLD architecture. It is certainly more than a thesis worth of work to build all of these components (and decide on their interconnections) before full SDSs following that approach could become a reality.

The proposed solution to the question of modularity vs. dense coupling is principled, standardized data exchange using IUs (Schlangen and Skantze 2009, 2011) within a largely pipeline-based architecture. The proposed system architecture is detailed in Chapter 3 and its implementation in a software toolkit in Chapter 4. The architecture broadens the information exchange between modules in a stratified system by keeping links to data on previous levels that can be queried by later processors and allows acyclic graphs of modules instead of plain pipelines, as well as two-way exchange between layers (in a somewhat limited way).

Of course, the architecture is incremental, thus, despite its simplicity in some respects, it is a step forwards towards more naturally interacting systems as it provides for close feedback loops which will be the topic of Chapter 6.

3 Incremental Processing and its Evaluation

This chapter, based on (Baumann, Atterer, and Schlangen 2009; Baumann, Buß, and Schlangen 2011) gives a short review on previous endeavours in incrementality to motivate the notion of incremental processing that is used throughout the thesis. Measuring the quality of incremental processing requires new kinds of metrics, and we lay out this need and develop such metrics in Section 3.3. The metrics will then be used in the following chapters to measure the performance of incremental speech recognition, timing estimation, and incremental speech synthesis.

This chapter approaches evaluation as a purely quantitative analysis and develops appropriate metrics for individual system components. A different approach, *subjective evaluation*, is also frequently used in dialogue research: test subjects interact with, or inspect the output of full systems and system quality is measured in terms of interaction quality or questionnaire ratings. We will perform and discuss such evaluations on a small scale for the example applications in later chapters (6.1.2, 5.6, 6.2.6, 7.5) to supplement quantitative analyses.

3.1 Timeliness and Incrementality

This thesis' topic is incrementality and incremental processing. However, incrementality is not an end in itself. Who should care if a dialog systems works incrementally or not? In this section I will try to motivate why incrementality is important for building naturally interacting spoken dialogue systems and one key to this is *timeliness*.

Dialogue is a collaborative process (or "joint action", Clark 1996) and collaborating means to work together. Hence, a collaborative process is one where the processors (in this context, interlocutors of the dialogue) work together *during* the process of creating a dialogue. Conventional dialogue systems collaborate on a turn-by-turn basis: a user turn is received, valuated, and only then responded to with a system turn. However, we have seen in the last chapter that this is not how humans interact: speakers expect feedback to their ongoing contribution, and will give their own feedback (and expect it to be received) during a system turn. Finally turn-changes should be swift, which may require the system to have a reply ready by the time the user turn is over. In short, system reactions are not sufficiently timely. In fact, Ward et al. (2005) found sluggish turn-taking to be one of the main obstacles in

predictive < anticipatory < **proactive** < reflexive < reactive

Figure 3.1: Progression of ever stronger knowledge when taking decisions

SDS usability. The above-mentioned behaviours require *timely* system reactions.[1] Timing can be improved using incremental processing: where in standard systems, processing load is squeezed into the short moment between the end of the user's turn and the beginning of the system's turn, incremental systems can *fold* processing time by effectuating some processing during speech delivery, where system load is low and processing time is abundant. Chapter 5 focuses on the user's speech delivery (and aims at folding analyses into this delivery time); Chapter 7 is concerned with output generation, where processing can be folded into the system's speech delivery. To conclude, timeliness is the goal and incremental processing is a means of reaching this goal.

Timely behaviour will often mean that a system has to go beyond *reacting* as quickly as possible: replies, feedback, or turn-changes often have to be initiated before they should commence (e. g. to overcome processing lags), so the system needs to be able to *predict* upcoming events and prepare reactions beforehand. The processor is called *proactive* if it is able to take a decision even before all evidence that a decision is to be based on has become available. A clear line between *reactive* and *proactive* behaviour cannot be drawn: the main difference in classifying an action is on how much evidence a decision is being based on and on how likely this evidence is to change in the future. As a matter of fact, language knows a lot of words to distinguish how decisions come about: a *reaction* can be expected when all evidence needed has been compiled. A *reflex* happens sooner, without complex decision making. We can *predict* or *anticipate* a correct action even without context, or with only little context available. See Figure 3.1 for an illustration. Thus, proactiveness can be seen as a continuum with a post-hoc reaction at one of the extremes, and purely speculative behaviour at the other. As evidence can be established over time, a more proactive system will usually be able to act more quickly, while at the same time being less informed about whether this action will in fact turn out to be the correct one.

A proactive system will occasionally perform false actions and it thus must be able to 'change its mind', in order to correct its behaviour as quickly as possible. Overall, a proactive system should optimize the trade-off between spending time to gather evidence and spending time to repair false actions.

[1] We prefer the term "timely system behaviour" over the computational term "real-time processing" as the latter implies stronger timing requirements (on the order of micro- not milliseconds).

3.1.1 Aspects of Incrementality

Incremental processing, in its intuitive form, means to process bit-by-bit, in a piece-meal fashion. Incremental processing has a long tradition in spoken language processing (e. g. Reddy et al. 1976; Young, Russell, and Thornton 1989). However, incrementality was primarily used as a means of reducing memory requirements, and as a way to integrate knowledge components to efficiently prune the search space during recognition.

In linguistics, Levelt (1989) coined the term *Wundt's principle* (after Wilhelm Wundt) to describe bit-by-bit, incremental processing: "Each processing component will be triggered into activity by a minimal amount of its characteristic input." (Levelt 1989, p. 26)

One drawback of Levelt's definition is that input processing ('being triggered into action') is not observable to the outside world. However, processing should only be called incremental if output can also be *observed* incrementally. Two alternative views have been given for incremental output in the literature:

Levelt's definition of Wundt's principle was extended by Guhe (2007) to append "and produces characteristic output as soon as a minimal amount of output is available." (Guhe 2007, p. 70) Kilger and Finkler (1995) instead stress not the possibility of generating output but the *necessity* of that output for further operation. Both these definitions are lacking. In a modular system, a processing module is unable to find out whether its output is necessary for further operation, rendering Kilger and Finkler's notion problematic. At the same time, Guhe's extension never forces the processor to generate output during processing (i. e. there is no constraint on the 'availability' of incremental output). We thus propose that a processor should be called incremental if it consumes input in a piece-meal fashion (Levelt 1989), and generates characteristic output in a piece-meal fashion (Guhe 2007) *at least in some situations* – without this condition, any incrementality a processor might possess internally would be unobservable to the outside world. The aspect of whether output is generated at all times or only sometimes has been called *massive* vs. *moderate* incrementality by Hildebrandt et al. (1999).

The *granularity*, i. e. what counts as 'minimal amounts' of input is crucial for incremental processing: the more fine-granular a processor, the earlier it can be triggered into action. Of course, granularity varies between processors depending on the task at hand.

Most of the work cited above does not systematically deal with the fact that incremental processing requires that a component be able to "change its mind", i. e. it requires the possibility for an incremental processor to revert previously output hypotheses in the light of additional input. Guhe (2007) calls the property of simply

extending previously output results *monotonicity*. Correspondingly, a processor that is able to change previous hypotheses may be called *non-monotonous*.

A model suitable for non-monotonous incremental processing is described by Schlangen and Skantze (2009). The model supports changes and revocation of previously output parts of hypotheses of an individual processor. This allows a processor to output an hypothesis as soon as possible, while keeping the option of changing or revoking this hypothesis in favour of a different, better one, later on. Significant changes and extensions in evaluation methodology become necessary for incremental processing following this approach which is described in some depth in Section 3.2.

3.1.2 Related Work on Evaluating Incremental Processing

In early systems, incrementality was used to improve non-incremental system performance but not used as a feature in its own right and no evaluation of incremental aspects took place. "Consciously incremental" processors for many different tasks have also been described in the literature (e. g. Sagae et al. 2009; Wachsmuth, Fink, and Sagerer 1998), and these descriptions usually also include evaluations. An often-used method here is to use standard metrics, such as word error rate (WER; Hunt 1990) for speech recognizers, for comparing a non-incremental and an incremental processor which never change their own previously output hypotheses. Such non-monotonous incremental processing is limited and hence results will be worse than non-incremental processing. Thus, the trade-off between the degree of incrementality and degraded results can be assessed (e. g. Wachsmuth, Fink, and Sagerer 1998).

To our knowledge, incremental evaluation metrics of ASR for incremental systems have not been covered in the literature before. Most closely related, Wachsmuth, Fink, and Sagerer (1998) show results for an ASR which fixes its results after a given time Δ and report the corresponding word error rate (WER). This unfortunately confounds the incremental and non-incremental properties of their ASR's performance.

A notable exception from the tradition of using standard metrics for the evaluation of incremental processors is (Kato, Matsubara, and Inagaki 2004) in the field of incremental parsing, which deals with the evaluation of incrementally generated partial parse trees. Kato et al. define a partial parse's validity given partial input and implement a method for withholding output hypotheses for a given length of time (similarly to our methods to be presented in Section 5.5), measuring the imposed average *delay* of this method as well as loss in precision compared to non-incremental processing. While this evaluation method goes some way towards capturing peculiarities of incremental processing, it still cannot account for the full flexibility of the incremental model by Schlangen and Skantze (2009), which allows for incremental processors to revise previously output hypotheses so that they may eventually produce final results that are as good as those generated by their non-incremental

counterparts. Such an incremental processor may trade timeliness of incremental results for quality of final results. As we will argue here, for this kind of incremental processing evaluation metrics are needed that capture specifically the incremental aspects of the processors and the *evolution over time* of incremental results. What needs to be measured is *what happens when*, instead of simply comparing final results of incremental and non-incremental settings.

In own previous work on incremental processors, the author has helped to develop metrics – out of a need to do justice to the complexity of the problem of incremental processing, which is not captured by standard metrics. For evaluating incremental ASR and, more generally, for balancing incremental quality and responsiveness (Baumann, Atterer, and Schlangen 2009); for evaluating incremental reference resolution (Schlangen, Baumann, and Atterer 2009), where the focus was on measuring the "stability" of hypotheses; for evaluating incremental natural language understanding more generally (Atterer, Baumann, and Schlangen 2009), looking at the distribution of certain important events (correct hypothesis first found, and final decision reached); and finally for n-best processing (Baumann et al. 2009). A first consolidation and generalization of these metrics has been presented in (Baumann, Buß, and Schlangen 2011) which forms the basis for the present chapter.

3.1.3 Relation to Anytime Processing

There are similarities between the problems posed by incremental processing and those posed by what is often called "anytime processing" (Dean and Boddy 1988). Particularly relevant for the discussion here is (Zilberstein 1996), who discusses the evaluation of anytime processing. *Anytime processing* is concerned with algorithms that have some result ready at every instant, even when all processing possibilities have not yet been exhausted. As noted by (Schlangen and Skantze 2009), incremental processing can be seen as a generalization of anytime processing, where responses are required also for partial input. In anytime processing, there exists a trade-off between the processor's *deliberation time* (when to stop a heuristic search) and the quality of results. In incremental processing, the deliberation time can be seen as the amount of input that the processor awaits before returning a partial result.

Zilberstein (1996) notes for anytime processing that "[t]he binary notion of correctness is replaced with a multi-valued quality measure associated with each answer." (Zilberstein 1996, p. 73), and we will argue below that incremental evaluation cannot be condensed to one single metric but that there are different dimensions of incremental performance. Zilberstein also introduces *quality maps* and *performance profiles* as ways to characterize the "expected output quality with execution time t" for anytime processors, a notion which we will adjust to relate output quality to amount of context.

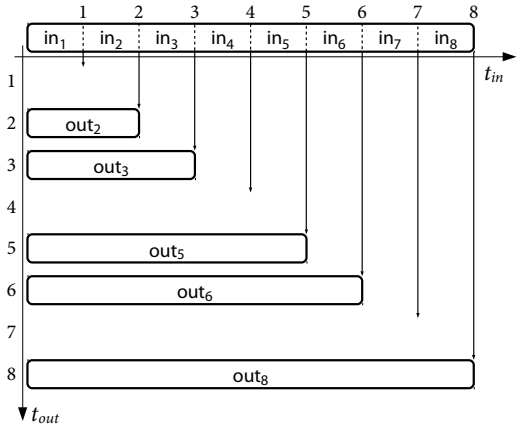


Figure 3.2: Schematic view of relation between input increments and growing output of an incremental processor.

3.2 Our Notion of Incremental Processing

As found in the previous section, incremental processing is the processing of minimal amounts of input (increments) (Guhe 2007, p. 70), and the (at least occasional) generation of characteristic output in a piece-meal fashion.

Processors meeting these two requirements are called *incremental processors*. The input to an incremental processor should be structured to support this processing mode. In order to also support *modular* processing, the output must be structured similarly because that output must be usable as the input to a next incremental processor. Input and output structure will be further discussed in Subsection 3.2.2.

The input and output of an incremental processor and data structures suitable for this purpose are discussed in the following subsections.

3.2.1 Incremental Processors

The input and output behaviour of an incremental processor, which is also schematically represented in Figure 3.2, is defined as follows:

Definition 3.1. An *incremental processor* accepts input in a piece-meal fashion and produces (partial) output before having consumed all input *at least in some situations*.

In Figure 3.2, the input, consisting of a sequence of eight input increments, is shown in the top row and the output after the processing of each input increment is shown

in the subsequent rows. The time spanned by the input increments is shown on the horizontal axis, while the times at which the outputs have been produced are indicated on the vertical axis. The figure assumes the output at some instant t to be based on all input up to t and represents this correspondence through the width (and alignment) of the bars representing output. As an example, output out_5 is the output generated at time t_5, after all input increments up to and including in_5 have been consumed. It is incidental in the figure that input increment times are equidistant, and in many cases, there will be irregular intervals between increments. However, time is discrete in our model:

Definition 3.2. A *frame* forms the basic temporal unit and determines the maximum *granularity* of input, processing, and its results with regards to timing. All times are discretized to a multiple of the frame duration.

Frames *discretize* time forcing the change of incremental output to be discrete as well. Thus, gradual output change is not allowed (at least below the level of temporal granularity). Frames also simplify time handling and the interconnection of separate, concurrently running processors. In our implementation (see Chapter 4) time progresses at a granularity of 10 ms (for speech recognition) or 5 ms (for speech synthesis). Additionally, we require for evaluation that measured time is *finite*, i. e. that there is a t_{max} after which no more processing occurs and beyond which no hypotheses extend.

In the evaluation methodology presented here, delays imposed by the actual consumption of input, processing proper, and generation of output by the processor are ignored. (In reality, t_5^{out}, the time at which output out_5 has been created would be slightly after t_5^{in}, the time at which input in_5 was consumed.) In the terminology of Ramalingam and Reps (1993), as cited by Guhe (2007, p. 75), processing time is assumed to be *bounded*, i. e. constant per input increment, and negligible compared to the time spanned by each increment. (However, no steps are taken to enforce boundedness, especially, incremental processors are not restricted to limited contexts as proposed by Guhe (2007).)

Boundedness can also be phrased as *sustainability*: if processing time regularly exceeds the frame duration at which new input is fed into the processor, then *real-time* processing becomes infeasible. Processing lags accumulate in unsustainable incremental processing, reducing to absurdity our goal of using incremental processing to allow timely system behaviour. Luckily, with modern day computers, many tasks, such as speech recognition and speech vocoding, allow for sustainable incremental processing; many other tasks, such as NLG and parsing, are within reach of sustainable real-time incrementality.

Notice that not always is new output being generated after consuming an input increment (there is no new output after consuming in_1, in_4, and in_7) (*moderate* in-

crementality in the sense of Hildebrandt et al. 1999, p. 22). For anytime processing, Zilberstein (1996, p. 74) calls the property of having a result ready at all times *interruptibility* and notes that any non-interruptible processor can be made interruptible by caching the most recent result. This holds analogously for incremental processing. In the following, we assume processors to always have incremental output ready, transparently caching most recent results if processing itself is not massively incremental (in the sense of Hildebrandt et al. 1999).

Figure 3.2 illustrated that there are two temporal dimensions that are relevant when talking about incremental processing: one is the time when the output was made (shown along the vertical axis), the other is which segment of the input that output is based on (shown along the horizontal axis). As a general rule, when evaluating, accuracy of the output should only be determined relative to the input that was available, and timeliness of the output should be determined relative to the relevant timing of the input. Finally, the way that the output evolves will have to be considered.

Monotonous incremental processing cannot change partial output that later turns out to be wrong. Being unable to change previous output in the light of additional input will produce inferior results if there are long-range dependencies. Such inflexible processing may be called *stubborn*, in a sense, with the opposite being yielding processing:

Definition 3.3. An incremental processor is called *yielding* if its final result (i. e. the output produced after processing all input) is equivalent to the result of a similar non-incremental processor for the same task (one that equals the incremental processor in internal data modelling, etc.).

An incremental processor can meet this criterion by deferring the production of output until the situation is unambiguous to guarantee monotonic output. However, this conflicts with our central goal of timeliness. Such a processing strategy will often lead to very few intermediate results, or no intermediate results at all, possibly rendering the processor fully non-incremental. In fact, both monotonous output together with yieldingness can only be simultaneously guaranteed if an input increment's influence is limited to a local context (e. g. a few words into the future may unambiguously determine a word's part-of-speech tag). Natural language often contains long-range dependencies (relevant e. g. in parsing), so just as humans are sometimes 'garden-pathed', a system should be able to output an hypothesis if it has a strong belief about a situation but also be able to change the hypothesis if future input indicates this. While the frequency of long-range dependencies is certainly lower for low levels of processing, the distance spanned by dependencies does not seem to be restricted.

Monotonous hypotheses have advantages, foremost that a consuming processor can perfectly rely on intermediate results. However, the interpretation of natural language *is* non-monotonous, due to long-range dependencies, as outlined above,

making it desirable to allow non-monotonicity in an incremental processor's output and we will describe how this can be handled systematically in the next subsection. For non-monotonous output, the need to measure the *degree* of non-monotonicity, that is, the amount of changes to incremental hypotheses arises. Recently, the term hypothesis *stability* has been used (Selfridge et al. 2011) to describe the degree of non-monotonicity of a processor's output. We will describe metrics that describe diachronic output evaluation to measure stability in Subsection 3.3.2.3.

Early works on incrementality (e. g. Wachsmuth, Fink, and Sagerer 1998) used 'stubborn' processing schemes. These were often evaluated in terms of quality degradation (e. g. WER increase) compared to their non-incremental counterparts. As our yielding processors ultimately generate the same output as non-incremental processors, no WER increase can be measured. Thus, our evaluation scheme brings us closer to actually measuring the incremental properties rather than performance decrease under a limited incremental processing scheme.

To conclude, in our notion of incrementality, a processor's future hypotheses must not be restricted by its intermediately hypothesized, partial results that it has output before. Instead, it should be able to take back hypotheses that later turned out to be wrong (rendering its output non-monotonous). This requirement is reflected in our representation of incremental data which we turn to next.

3.2.2 Representing Incremental Data

Figure 3.2 deliberately left out the question of how an incremental processor's output should be structured (the output was drawn simply as long unstructured bars). Finkler (1997) introduced the distinction between *quantitative* and *qualitative* incrementality. In the former, all output is repeated after every input increment and hence the question of output structure is irrelevant for evaluation methodology. However, quantitative incrementality does not facilitate building modular incremental systems where one processor feeds the next, as output from one component cannot easily be fed to a subsequent component in a piece-meal fashion. The link between individual bits of information is lost. Contrastingly, in qualitative incrementality, output increments build on the output of the preceding processing step. Thus, to support modular incremental systems, we have the same requirements for a processor's output that we have for the input: it must be structured to support piece-meal feeding to a succeeding processor, that is, it must come in pieces:

Definition 3.4. We call $iu = (payload; start; end) \in \mathbb{I} = \mathbb{P} \times \mathbb{N} \times \mathbb{N}$, an evaluation *increment* of input or output with a payload from some set of payloads $\mathbb{P}$ and under the constraint that $start \leq end$.

The payload of an increment may be a word, a phoneme, a concept, or whatever other minimal amounts of data the processor generates and that is to be evaluated for appropriateness. Start and end times are measured in frames.[2]

Additionally, to ease processing in a modularized, yet highly coupled and interconnected system, we may be interested in tracing which parts of the input have led to a certain output increment in order to make more informed decisions in later modules. We use the notion of *incremental units* (IUs), introduced by Schlangen and Skantze (2009) and extended in (Schlangen and Skantze 2011) to arrive at a parsimonious representation for both input and output. IUs meet our definition of evaluation increments. As defined by Schlangen and Skantze (2009), IUs are the smallest 'chunks' of information that can trigger a processor into action. IUs typically are part of larger units, in a manner as individual words are parts of an utterance. This relation of being part of the same larger unit is recorded through *same level links*; the IUs that were used in creating a given IU is linked to it via *grounded in links*.[3] As IUs are connected with these links, an *IU network* emerges that represents the current output of the processor at the given instant. During incremental processing, the processor incrementally builds up the network of IUs that will eventually form its final output, once all input has been consumed.

IUs that describe the state of the world are grounded either directly in (some parts of) the *primary signal*, or in other IUs and following their grounded in links will eventually *ground out* to the primary signal. Thus, finally, all IUs can be attributed to some portion of the signal, thus, have a start and an end time and meet our timing requirement.[4]

IUs as increments are 'minimal units' (Schlangen and Skantze 2009) and an hypothesis of a processor consists of several increments in a sequence:

Definition 3.5. We call $hyp = (iu_1, iu_2, \ldots, iu_k) \in \mathbb{H} = \mathbb{I}^*$ a processor's hypothesis about the state of affairs.[5]

[2] For simplicity (e. g. to allow easy comparison of different methods), times can be measures relative to the beginning of processing, not on an absolute scale.

[3] The sequential relationship expressed by same level links is represented implicitly in the input sequence in Figure 3.2 simply by placing the increments in sequence; Figures 3.3 (right side), 3.4 and later figures make them explicit with diamond-headed arrows between IUs. Informational dependency between input and output is represented in Figures 3.2 and 3.3 (left side) through horizontal alignment of inputs and outputs while the input has been left out in Figure 3.4 and no dependencies are shown. In later figures, grounded in links for informational dependency will be expressed by arrows with regular heads.

[4] This is tailored towards events that span a *period* of time. Events that are based on a *point* in time could be handled by assigning identical start and end times.

[5] Sometimes $hyp = (iu_1, iu_2, \ldots, iu_k)$ will be abbreviated as $iu_{1..k}$ below.

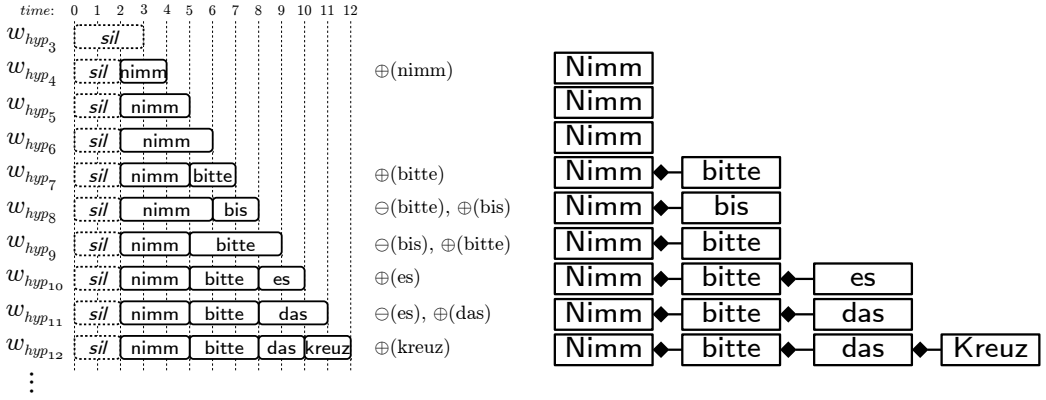


Figure 3.3: ASR hypotheses during incremental recognition. Raw ASR hypotheses are shown on the left, the corresponding IU network on the right, and step-wise edits in the center.

Incremental processors have a *current hypothesis*, composed of a sequence of IUs, at every time t:

Definition 3.6. We call $hyp_t = (iu_1, iu_2, \ldots, iu_k)$, with $t \in \mathbb{N}$ the hypothesis of an incremental processor after processing all input up to (and including) time t (measured in frames); $t \leq t_{max}$.

Many processors on the lower levels may provide for the additional constraint that IUs are non-overlapping and temporally ordered: $\forall i \in (1 \ldots k-1) : end(iu_i) \leq start(iu_{i+1})$.[6] Notice that we do not restrict the timings of the hypothesized IUs: while most often, $end(iu_k) \leq t$, i. e. the processor generates output only for the input that it has received so far, this need not necessarily be the case: a processor will be called *predictive* if it generates some output increments describing future events without having received any direct supporting evidence so far (i. e. given $hyp_t = iu_{1..k}$, $end(iu_k) > t$).

The *current hypothesis*, i. e. the current 'total output' of an incremental processor during incremental processing can be read off the output IU network by following the same level links from the newest IU backwards. Figure 3.3 shows (stylized) output of an incremental speech recognition component. In the figure, ASR is recognizing the utterance "`Nimm bitte das Kreuz [Take please the cross]`" with intermittent

[6] The function *start* (and similar functions used below) are meant to select the corresponding field from their argument.

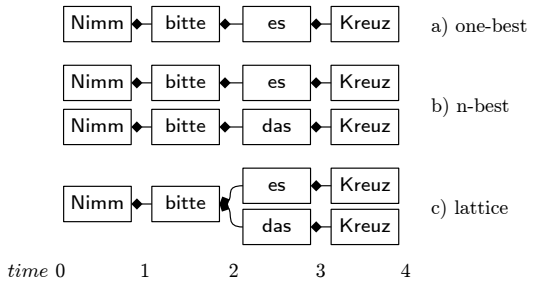


Figure 3.4: One-best (a), n-best (b), or lattice (c) IU output of an ASR in the IU framework.

misrecognitions substituting "bis" ("until") for "bitte" ("please") and "es" ("it") for "das" ("the"). The representation on the left of the figure is similar to that of Figure 3.2, but here the actual content and structure of the output hypotheses is shown. The rightmost column shows the IU network as it emerges after each processing step. Each IU contains a word hypothesis and IUs are connected via same level links, eventually forming an utterance.

Figure 3.4 shows more IU networks for ASRs in different configurations; for one-best output, there is one final node, for n-best lists there are n final nodes with unconnected networks and for lattice-/tree-like structures paths are connected in a common root, ending possibly in several final nodes. In this case, not only one, but several current hypotheses (or: variants of the current hypothesis) can be derived from an IU network at the same time.[7]

In our scheme, non-incremental processors produce only one hypothesis *hyp* for $t = t_{max}$, i. e. after processing the whole input. In contrast, an incremental processor should output *multiple* (consecutive) hypotheses, finally reaching $hyp_{t_{max}}$. Using the terminology for hypotheses, we can say that an incremental processor is *yielding* iff its $hyp_{t_{max}}$ always equals its non-incremental counterpart's hypothesis.

We will now define an abstract notion of similarity in order to be able to describe the possibilities for succession of consecutive hypotheses below.

Definition 3.7. We call two increments *similar* ($iu_1 \approx iu_2$) iff they have the same payload: $iu_1 \approx iu_2 \Leftrightarrow payload(iu_1) = payload(iu_2)$.

[7]The evaluation metrics defined in this chapter have to be extended to handle multiple current hypotheses. This will be further explored in Chapter 5.4.2. For the time being it suffices to say that one or more hypotheses can be derived for all IU networks at any time.

We do not deem it necessary to restrict the amount of variation in start and end times for two increments to be similar. In practice we will use increment similarity almost exclusively in conjunction with the position of the increment in the list of hypothesized increments, which restricts timing variation to a sensible level.[8]

Similarity can easily be extended to hypotheses: as we are dealing with full hypotheses, we extend the definition of similarity to increment sequences and additionally define the concept of prefixes:

Definition 3.8. Two hypotheses $hyp = (iu_1, iu_2, \ldots, iu_k)$ and $hyp' = (iu'_1, iu'_2, \ldots, iu'_l)$ (or for short $iu_{1..k}$ and $iu'_{1..l}$) are called similar ($iu_{1..k} \approx iu'_{1..l}$) iff $k = l$ and $\forall\, i \in (1 \ldots k)$: $iu_i \approx iu'_i$.

One feature of incremental language processing (in our experience) is that much of the previous output remains valid between processing steps (that is, even when allowing non-monotonicity, the degree of monotonicity is high). In incremental speech recognition, for example, two consecutive hypotheses (hyp_t and hyp_{t+1}) are often similar in that they mostly contain the same words. (Very often for ASR, all words remain the same apart from the last one consuming one more frame: $(\forall\, i \in (1 \ldots k) : word_i(hyp_{t+1}) = word_i(hyp_t)) \wedge end(word_k(hyp_{t+1})) = end(word_k(hyp_t)) + 1$.) When adjacent hypotheses are different, differences often regard the "right end", i. e. the most recent parts of the hypothesis: iu_k and/or a few increments preceding iu_k. For speech recognition, this happens if an additional, new word is recognized, or a previously hypothesized word is found to be wrong and retracted, or a word hypothesis is replaced with a different, better matching word hypothesis. As most of the change occurs towards the 'newest' part of the hypotheses, it is sensible to have a notion of the common prefix of two hypotheses:

Definition 3.9. Given an hypothesis $hyp_t = (iu_1, iu_2, \ldots, iu_k)$, we call any increment sequence $iu_{1..i}$ with $i \in (1 \ldots k)$ a *prefix* of hyp_t: $iu_{1..i} \lessapprox hyp_t$.

Before making use of this notion of a prefix, we will now describe three simple *edit* operations that can be applied to the right end of an hypothesis *hyp* in order to turn it into hyp' and that can be used to describe the change between consecutive hypotheses:

Definition 3.10. We call $\oplus : \mathbb{H} \times \mathbb{I} \longrightarrow \mathbb{H}$ the *add* edit operation. Given an hypothesis $hyp = iu_{1..k}$ and an increment iu, $apply(hyp, \oplus(iu))$, or for short $hyp \oplus iu$ results in the hypothesis $hyp' = (iu_{1..k}, iu)$.

[8] This valuation is from experience and purely qualitative; testing whether all similar IUs (using above definition) meet this notion would be an enormous (and tedious) undertaking.

Definition 3.11. Likewise, we call $\ominus : \mathbb{H} \times \mathbb{I} \longrightarrow \mathbb{H}$ the *revoke* (remove) edit operation. Given an hypothesis $hyp = iu_{1..k}$ and an increment iu, $apply(hyp, \ominus(iu))$, or for short $hyp \ominus iu$ results in the hypothesis $hyp' = iu_{1..k-1}$, given that $iu_k \approx iu$.

Definition 3.12. We call $\mathbb{E} = \oplus \cup \ominus$ the set of edit operations.

Edit operations can be applied one after the other to turn any hypothesis into any other.[9] To make this simple, we allow to *apply* lists of edits instead of just one; the edits in the list are applied one after the other.

In this thesis we do not regularly refer to *substitution* as an edit operation (which substitutes the last increment of an hypothesis for another). However, if needed, substitution can be seen as the concatenation of a revoke and an add operation:

Definition 3.13. We call $\oslash : \mathbb{H} \times \mathbb{I} \times \mathbb{I} \longrightarrow \mathbb{H}$ the *substitute* edit operation. $\oslash(iu_1, iu_2) = (\ominus(iu_1) \oplus (iu_2))$.

Similarly to constructing new hypotheses from (lists of) edits, we would also like to generate the lists of edits that are needed to do so. We make use of the notion of hypothesis prefixes defined above:

Definition 3.14. We call $diff(hyp_1, hyp_2) \in \mathbb{E}^*$ the operation that generates the minimal list of edits necessary to turn hyp_1 into hyp_2:
Given $hyp_1 = (iu_{1...k})$ and $hyp_2 = (iu'_{1...l})$, and let $j = \arg\max_j iu_{1..j} \approx iu'_{1..j}$ denote the last element of the maximum common prefix of hyp_1 and hyp_2; then $diff(hyp_1, hyp_2) = (\ominus(iu_k) \ominus (iu_{k-1}) \ldots \ominus (iu_{j+1}) \oplus (iu'_{j+1}) \oplus (iu'_{j+2}) \ldots \oplus (iu'_l))$.

Edit lists between consecutive hypotheses ($diff(hyp_{t-1}, hyp_t)$) are shown in the middle column of Figure 3.3. Of course, the edit operations just described correspond to IUs being added to or unlinked from the IU network.

Edit operations are deliberately limited to operate on the most recent parts of an hypothesis, which simplifies the model. While redundant edits are required to exchange a unit that was constructed a while back, such exchanges become sparser and sparser the further the distance. Thus, for sequential data the edit operations presented here are an ideal tradeoff between model complexity and efficiency. (However, the trade-off may differ e. g. for parsing, where proximity cannot be measured linearly in numbers of words.)

To conclude, using the notation developed in this section, both the current full hypotheses as well as the changes between consecutive hypotheses can be traced

[9] Of course, more complex edit operations could be introduced to support changing hypotheses 'in the middle'. However, apart from the fact that the vast majority of edits occurs at the right edge of hypotheses (see Chapter 5.5), and that infix edit operations would add a lot of complexity, our edit operations are already sufficient to support any transformation of hypotheses.

between processors. The representations allow us to define metrics that describe (a) the quality of the results encoded in the IU network, (b) the times at which the individual contributions that form this result were created by the processor, and (c) the diachronic evolution of the network during processing. We will describe how to evaluate these three aspects of incremental processing in the following section, after discussing the types of gold standard information that are required.

3.3 Evaluation of Incremental Processors

This section describes quantitative evaluation schemes for incremental processors which are based on comparing a processor's output against some given ideal output. There are various sub-properties that are not independent of each other, and trade-offs are involved if either of those is to be optimized.

The requirement for specific incremental metrics arises foremost from the flexibility of the processing scheme, with most processors being yielding (i. e. ultimately producing the same output as non-incremental processors). In contrast to using conventional evaluation metrics for incremental processors, the metrics presented here follow the goal of separating incremental performance aspects from overall, non-incrementally observable performance.

We are solely concerned with the evaluation of individual incremental processors or their combination into pipelines but we do not consider the combination of processors to arbitrary systems. As a consequence, we do not have to consider the inter-play of loops when processors feed back information to predecessors.[10] In the sense of Wirén (1992), the evaluation scheme is limited to processing that is ordered *left-to-right*.

As explained in the previous section, an incremental processor produces a sequence of partial outputs and there are several aspects to the sequence that will be covered by different metrics. Before defining metrics for these aspects, we first turn to the targets for comparison: the gold standard.

3.3.1 Gold Standards for Evaluation

The ideal output for comparison (often called *gold standard*) can be manually constructed or automatically generated using additional knowledge that is not available to the processor being evaluated.

The ideal output of a process looks differently, depending on the task, and the specific goals set for a processor. The disfluent utterance shown in Figure 3.5 is an example for this: even though "`knife`" is shown to be uttered by mistake with the

[10] However, this evaluation methodology could be used if loops are only internal to the collection of processors that are to be evaluated.

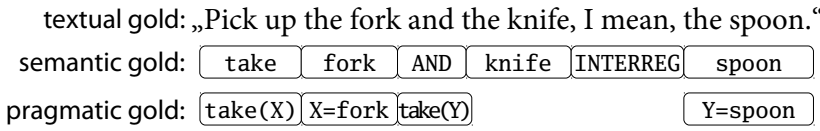


Figure 3.5: Three gold standard annotations of a disfluent utterance: for speech recognition, semantics and pragmatics.

interregnum "I mean" and later corrected by the reparans "spoon", it should form part of the output of an incremental speech recognizer as these words have undeniably been spoken. Similarly, a simple semantic analyzer should probably generate output for both "knife" and "I mean", though the latter should be marked as being an interregnum. However, a pragmatic interpreter should *not* output "take(knife)", not even intermittently, as this meaning is not what the speaker intends. Instead it should only output "take(spoon)" and only this should be marked in the gold standard for evaluating a pragmatic interpreter. Such an interpreter's prudence will come at the cost of timeliness, because output can only be generated when it is reasonably certain. Such considerations should be taken into account when designing the gold standard for a specific application.

We have said above that quantitative evaluation is the comparison of *actual* output of a given processor to some sort of *ideal* output. The previous section has shown how output can be represented in a way that makes all dimensions of incremental information (content, timing, and evolution of incremental results) accessible in the evaluation of an incremental hypothesis. The format and availability of ideal output will be discussed now. Sometimes, obtaining the ideal output is easy; in other cases, some required information cannot be recovered from typical evaluation resources, and an approximation has to be found. The former case will be discussed first, and then the approximation case.

3.3.1.1 Evaluation with Incremental Gold Standards

Figure 3.3 from the previous section shows the output of an incremental processor after each input increment consumed. This is the format in which we would like a gold standard for evaluation to be, in order to be able to evaluate every incremental hypothesis. Luckily, the information required for this is often available in existing ASR evaluation resources: for the content, we need the output increments, which in this case is a sequence of words which is provided by the transliteration of the input. We also need the link between input increments and output increments which in this

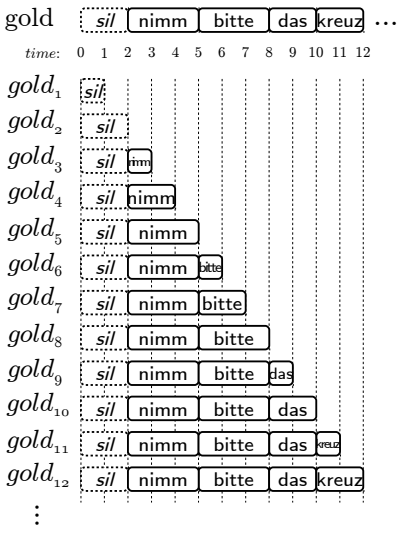


Figure 3.6: The gold standard for incremental speech recognition evaluation.

case means that we need an alignment between words and audio signal. Again, this is often provided by ASR evaluation resources, and if not, can be produced automatically via forced alignment.

In Figure 3.6, an aligned, non-incremental gold standard sequence is shown in the top row (labelled "gold"). This is the final state the incremental processor should ideally reach. The intermediate stages necessary for incremental evaluation can be created from the final state by going backwards through the input increments and removing the current rightmost output word whenever we go past the input increment that marks its beginning as can be seen in the figure (e. g. at time 10 we would remove "Kreuz", at time 8 "das", and so on). Following this method, the resulting gold standard demands that an output increment be created as soon as the first corresponding input increment has been consumed; e. g. a word-IU should be produced by an ASR as soon as the first audio frame that is part of the word in the gold standard is received. While this will often be impossible to achieve, it provides us with a well-defined upper boundary of the performance that can be expected from an incremental processor. In analogy with 'current hypothesis', we call the resulting intermediate stages the *current gold standard* relative to a given input increment:

Definition 3.15. Given a non-incremental, aligned gold standard $gold = iu_{1..k}$, the *current gold standard* at time t is the prefix for which all increments start before t: $gold_t = iu_{1..l} : iu_{1..l} \lessapprox iu_{1..k} \wedge start(iu_{l+1}) \geq t$.

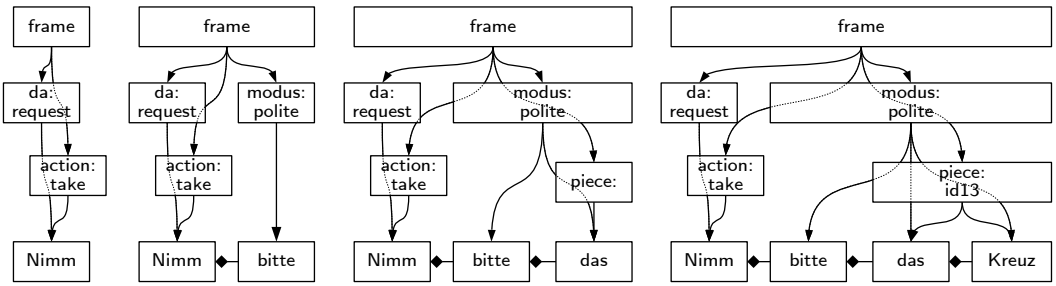


Figure 3.7: Four subsequent outputs for an incremental semantics component as input words are being processed.

This method is directly transferable to other kinds of input and output. Figure 3.7[11] shows the incremental growth of a network representing a frame semantics; this time the input increments (in this case words, not bits of audio) are shown in the bottom row, and the grounded in links which relate output to input are represented by arrows. (As the networks in this example are more complex, steps are drawn next to each other and not in rows as in the previous figures.) If a corpus is available containing utterances annotated with their final semantics together with information about which words are responsible for which bits of that final semantics, we can use the same method to go backwards through the input IUs and create the full corresponding set of IU states for partial inputs.[12] For semantics, this type of resource (containing the relevant 'timing' information) is rare, as making the link between what should be known based on partial input may not even be easy for human annotators (but see (Gallo et al. 2007) for an effort to create such a resource). Typically, only the final correct semantics is available, with no indication of how it was conceived (see e. g. the ATIS corpus as used in He and Young 2005). In such a case, the intermediate outputs must be approximated from the final state; we will explain how in the next subsection.

[11] In the example given, the slot filling for "`modus`" depends on "`bitte`" and all following words. This is because the modus could easily turn out to be e. g. 'sarcastic' if some other word had been added later on.

[12] For conceptual simplicity, hypotheses have been defined as sequences of increments which is sufficient for simpler output such as from speech recognition. More complex output demands for definitions based on sets of increments but otherwise works identically. The gold standard definition for sets would be $gold_t = \{iu \in gold : start(iu) < t\}$.

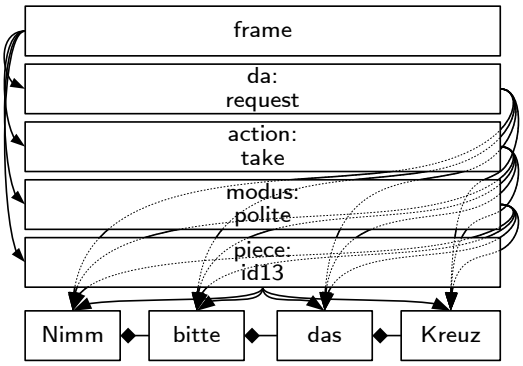


Figure 3.8: A semantic frame represented in the IU framework without specific dependencies between slots and words.

3.3.1.2 Evaluation with Non-Incremental Gold Standards

Figure 3.8 shows a situation in which a fine-grained link between input and output increments cannot be recovered from the available evaluation resource. We then simply assume that all output IUs are grounded in all input IUs, which is the equivalent of saying that every input increment contributed to every output increment. The figure only shows the final state and we again derive the incremental steps from this by going backwards through the input IUs, as above. However, no desired output will disappear from the gold standard because every output is already grounded in the very first input increment (as we don't know what input increment some output increment *logically* depends on). Viewed in the direction of time this means that the gold standard is demanding that all output increments be known from the beginning; this is clearly an unreasonable assumption, but as it is kept constant, it allows to measure the gradual approach towards this ideal.

Such a representation then of course gives us less information, and hence an evaluation based on it can only give a coarse-grained insight into the processor's performance. If we assume that in reality not all output information is available immediately, the best a processor can do against such a gold standard is that it fares better and better as more input comes in, and as more and more of what will be the final representation is discovered. Likewise, we loose the ability to make fine-grained statements about the timeliness of each output increment.

There is a third common case that can be subsumed under this one. Sometimes one may want to build a processor that is only incremental on the input side, producing for each input increment an output of the same type as it would for a complete,

non-incremental input. An example for this would be a processor that predicts a 'complete' utterance meaning based on utterance prefixes. (This has been explored by Sagae et al. 2009; Schlangen, Baumann, and Atterer 2009 and Heintze, Baumann, and Schlangen 2010.) In IU-terms, the output IU is grounded in all input IUs and hence such a processor can be evaluated against a non-incremental gold standard without loss of information.

3.3.2 Metrics for Evaluation of Incremental Processors

We now discuss metrics that quantify differences between actual and ideal output (i. e. the gold standard). We identify three categories of metrics: Overall *similarity metrics* (measures of equality with or similarity to a gold standard), *timing metrics* (measures of the timing of relevant phenomena w. r. t. the gold standard) and *diachronic metrics* (measuring change of the incremental hypotheses over time), which we will look at in turn. These metrics illuminate the different aspects of incremental performance of a processor, but they are not independent of each other (e. g. timing can only be measured if something is correct, absolute correctness entails perfect timing and evolution, etc.). Interrelations of metrics will be further discussed in Subsection 3.3.2.4.

3.3.2.1 Similarity Metrics

Similarity metrics compare what should ideally be known at some point in time to what *is* known at that point. The only difference that incremental evaluation brings with it is that the comparison is not done only once, for the final output given complete input, but also for all stages that lead to this final output. An incremental similarity evaluation hence will result in a sequence of results per full input token (e. g. per utterance), where non-incremental similarity evaluation yields only one. To be able to evaluate after every input increment, we need a gold standard that covers the ideal outputs after every input increment, as described above. Figure 3.9 shows such an incremental gold standard and the IU network (for the same utterance as in Figure 3.3) produced by an incremental ASR.

The most basic measure of similarity is `correctness`: we simply count how often the output hypotheses' payloads are identical to the current gold standard (i. e. how often $hyp_t \approx gold_t$)[13] and divide this by the total number of incremental results. In Figure 3.9, the output is correct four times, resulting in a `correctness` of 40 % (ignoring the empty, trivially correct hypotheses 1 and 2 in the calculation). Incremental processors often lag behind in producing output for recent input. If this delay (Δ) is known

[13] Again, the ignorance against increment timings is based on the experience that timing deviations are small and tolerable.

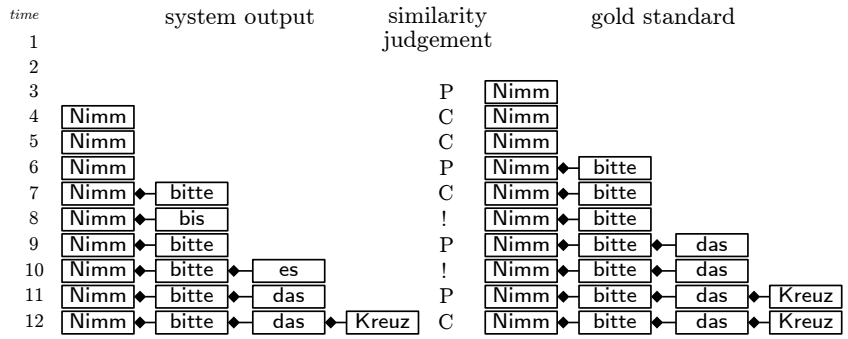


Figure 3.9: An ASR's actual incremental output (left), the incremental gold standard (right) and a basic similarity judgement (center): correct (C), prefix-correct (P), or incorrect (!).

in advance, it can be taken into account, defining a `delay-discounted correctness` which, if measured at time t only expects a correctness relative to the gold standard at time $t - \Delta$; see Section 5.5.1 for a method that employs this type of evaluation. However, the processor's lag will often vary with the input, which a fixed Δ cannot account for. In this case, we propose counting the number of times that the processor's output is a *prefix* of the ideal output at that instant ($hyp_t \lesssim gold_t$) and call the corresponding metric `p-correctness`. In Figure 3.9, `p-correctness` is 80 %.

`Correctness` has a shortcoming, however, namely that many processors generate output that is often not correct: even the final output (i. e. the output when all input increments have been considered) may contain errors compared to the gold standard. (Imagine that the ASR in the example from Figure 3.9 had generated "`nehme`" instead of "`nimm`"; this would have rendered all output increments incorrect in this strict understanding.) In such a case, a processor's non-incremental deficiencies (i. e. deficiencies that also show in non-incremental processing) block the incremental evaluation. There are two possible remedies for the problem: Relaxing the equality condition to some task-dependent similarity measure (that is, changing what counts as correct), or changing the gold standard in such a way that it is guaranteed that final results are regarded as correct. We discuss the latter approach first as it allows for a clean separation between incremental and non-incremental performance.

If we want to ensure that the final output of a processor counts as correct, we can simply use this final output as the gold standard and derive from it the set of current gold standards. To distinguish this from `correctness` as compared to an independent gold standard, we call the measure `r-correctness` (for *relatively* correct, relative to the

processor's final output). This separates the evaluation of incremental and non-incremental quality aspects: to learn about the overall (non-incremental) quality of the processor, we use standard metrics to compare its final output with a "true" (externally generated) gold standard; to focus on the incremental quality, we use `r-correctness`.

The alternative approach of relaxing the equality condition leads us to using task dependent evaluation techniques that make it possible to measure the similarity (and not just identity) of IU networks. Which non-incremental measure should be used as the basis for such incremental performance measure completely depends on the task at hand: for example, incremental speech recognition can be evaluated with WER or CER (Boros et al. 1996), syntactic parsing with measures for tree comparison (Carroll, Briscoe, and Sanfilippo 1998), semantics by calculating f-measure between frame pairs, or, as a more complex example, specific metrics for natural language generation (Reiter and Belz 2009) or machine translation (Papineni et al. 2002). We can 'incrementalize' such metrics simply by making comparisons at each input time step, comparing actual output and current gold standard, as explained above. Typically, we will be interested in the average of this metric (how similar in general is the output, no matter how partial the input?), but we may also want to explore results at different grades of completeness (does the processor perform worse on smaller prefixes than on longer ones?), or the performance development over time.

Also, we may want to allow for certain output differences over time, something that is unique to incremental processing. For example, an ASR may produce in sequence the hypotheses "`grün`", "`grüne`", and finally "`grünes`" as it successively consumes more input. In certain settings, already the first hypothesis may be close enough so that the consuming processor can start to work, and the subsequent hypotheses would not count as contradictions. In such a case, we can set up the similarity metric so that it would allow all variants in the comparison with the gold standard, creating a kind of *incremental concept error* metric which weighs "sensible" mistakes as less serious than non-sensible mistakes.

We close with a discussion of two more similarity metrics. `F-score`, the harmonic mean of precision and recall, is a useful metric for similarity when the output representation is a 'bag' of concepts and no timing information is available in the gold standard. In such a setting, we can expect the score to be very low at the beginning (hardly any slot, compared to the correct representation for the complete input, will have been filled in the beginning) and to rise over time, as more slots are being filled (hopefully correctly). In this case, the shape of `f-score curves` plotted over different grades of input completeness can become an informative derived measure. Using this method, we found (in Atterer, Baumann, and Schlangen 2009) that considerable knowledge about what the speaker says can be inferred within the first 40 % of the utterance (Atterer, Baumann, and Schlangen 2009, p. 1034).

Finally, mistakes can be valued differently at different stages through the input. This is especially appropriate for processors that predict complete outputs (see discussion above at the end of Section 3.3.1.2) and captures that not making a decision can be an adequate decision early on but becomes more and more just like making a wrong decision the more input has been seen. `Time-adjusted error` captures this notion in valuing certain events (e. g. the processor deciding on the special class "undecided") differently, depending on how much of the input has been seen so far. For example, "undecided" could start at 0 % error and rise to 100 % error towards the end of the utterance (e. g. using time or word counts, depending on the task).

When no timing information is available in the gold standard or the processor's output, `time-adjusted error` is the best we can do in measuring timeliness of behaviour. However, more fine-grained evaluation metrics can be conceived when timing information is available and these will be discussed next.

3.3.2.2 Timing Metrics

Timing metrics measure *when* some notable event happens in incremental output relative to some reference time from the gold standard. All other things being equal, incremental processors are better that give good and reliable results as early as possible;[14] our timing metrics make this precise.

As mentioned above, a characteristic of our notion of incremental processing is that hypotheses may be revised in the light of subsequent input. This revision means that there are two events specifically that are informative about the performance of the processor: when output becomes available, and when it is not revised any more. We capture this with the following two metrics:

Definition 3.16. The *first occurrence* (FO) of an output increment $iu_j \in hyp_{t_{max}}$ is the time t at which it (and all its predecessors) are correctly hypothesized for the first time: $\text{FO}_{absolute}(iu_j) = \arg\min_t iu_{1..j} \lessapprox hyp_t$.

FO is the moment at which no further edit to the increment in the j'th position of the hypothesis is *required*. However, even though no edits to the increment are required anymore, it may still be changed erroneously. Thus, it is also important to know when a processor's hypothesis about an increment is not changed anymore (has become *stable* in the sense of Selfridge et al. 2011):

Definition 3.17. The *final decision* (FD) for an increment $iu_j \in hyp_{t_{max}}$ is the time t at which it (and all its predecessors) are correctly hypothesized and not changed anymore: $\text{FD}_{absolute}(iu_j) = \arg\min_t \forall t_i \in (t, \ldots, t_{max}) : iu_{1..j} \lessapprox hyp_{t_i}$

[14] However, if output for the future is generated, we might want to be notified by the processor that this output relates to future input, e. g. via correspondingly set increment start and end times.

As a first illustration for FO and FD we return to Figure 3.9: for "bitte", the first occurrence is at step 7. "bitte" is temporarily replaced by "bis" in step 8, and the final decision is at step 9.

To determine timing measures, we use the time difference between the occurrence of the IU in the output and the gold standard IU's timing. There are two issues that need to be resolved: (a) FO and FD are only defined for IUs that are eventually recognized correctly (and only if all predecessors are recognized correctly as otherwise the prefix relation does not hold); and (b) for correctly recognized increments, a decision on how to anchor the timing comparison needs to be taken.

The easy solution to incorrect hypotheses is to use the final hypothesis of the processor to derive the incremental gold standard (as was done for `relative correctness` above), even though it may be less accurate or partially incorrect when compared to a true gold standard. Again, this separates the question of how well the processor performs compared to an external standard from the question of how well it performs incrementally; here, the measures will tell us how fast and how stable the processor is in making the decisions it will ultimately make anyway, regardless of whether they turn out to be correct. However, one might want to only valuate truly correct increments sometimes (e. g. as it does not really matter how fast the processor is in taking wrong decisions, especially if they remain wrong). In this case, some sort of automatic alignment (such as minimum edit distance, Levenshtein 1966) between final hypothesis and gold standard needs to be found which can be used to match references to incorrect increments. Of course, the automatic alignment also has to be taken into account when computing hypothesis similarity.

Regarding the anchoring of the metrics, FO and FD should be differentiated because they capture different aspects of processing. Just like for similarity measures, where we assume an increment should be generated as soon as it begins to appear in the gold standard, we propose to *anchor* FO relative to the beginning of the increment in the gold standard:

Definition 3.18. $\mathrm{FO}_{anchored}(iu_j) = \mathrm{FO}_{absolute}(iu_j) - start(gold_j)$.

The above definition encourages *speculation* to happen as soon as some of the cues relevant for the IU become available. The 'faster' a processor is, the lower its average FO will be – some processors may even achieve a negative average FO, if they often predict output before any (direct) evidence for it was available.

Conversely, but again resembling the approach taken with similarity metrics, we propose to anchor FD relative to the end of the increment in the gold standard, as it is only reasonable to take a final decision once all the corresponding input (as specified by the gold standard) has been observed:

Definition 3.19. $\mathrm{FD}_{anchored}(iu_j) = \mathrm{FD}_{absolute}(iu_j) - end(gold_j)$.

The more reliable a processor is, the lower its average FD will be. The remarks on partially incorrect hypotheses given above similarly apply to anchoring. FD can only be measured after processing has finished; during processing it is unknown whether an IU will be withdrawn later on.

Returning again to Figure 3.9, "bitte" started at 6 but only appeared at 7, hence $\text{FO}_{anchored}$(bitte) is 1. Looking back at the time-alignment in Figure 3.3 we see that the alignment of "bitte" changes between steps 8 and 9 (and is only correct at step 9). As timing is ignored in the prefix operation used above (which is based on increment similarity not identity), $\text{FD}_{anchored}$(bitte) is also 1.

Whether timing metrics should be measured on an absolute or relative scale depends on the task at hand and the granularity of the available gold standard. For example, in Chapter 5, timing of speech recognition output will be evaluated in milliseconds but timing has previously been measured as utterance-percentages in an evaluation of a reference resolution task in (Schlangen, Baumann, and Atterer 2009).

The statistics of these events over a test corpus convey useful information about the processor: the distributions for FO and FD indicate how much the processor lags behind in producing output given the relevant input, and how long it takes for the processor's decisions (given all relevant input) to become stable, respectively. The average FO can be used to determine the (average) *lag* that is incurred by the processor before output becomes available (that, however, may be revoked afterwards). FD can be used to determine stability, at least under the assumption that hypothesis timing and gold timing do not deviate much. Processors with low variance of timing metrics are more predictable in their processing outcomes.

In some cases, these metrics may be more important for specific tokens than others, and this will be explored in Section 5.6. Even though the full distribution contains useful information, analyses in this thesis will often be restricted to reporting means, medians and standard deviations to describe the distributions.

3.3.2.3 Diachronic Metrics

In some sense, the types of metrics presented so far cast a static look on the processing results, with the similarity metrics describing *whether* a result (at a given time step) is correct or not, and the timing measures describing *when* correct results become available, given the whole set of outputs. The metrics discussed now round out the set of metrics by describing what happens over the course of processing the input – the *diachronic evolution* towards the final results.

One metric is based in the number of edits between successive hypotheses that did occur as compared to the number of edits that should have been sufficient to generate the final result:

Definition 3.20. Let $N_{optimal} = |diff(hyp_{t_0}, hyp_{t_{max}})|$ the number of edits that is *necessary* (to generate the final hypothesis without any intermediate errors) and $N_{actual} = \sum_{t=1}^{t_{max}} |diff(hyp_{t-1}, hyp_t)|$ be the number of all the edits that actually occurred over the course of incremental hypothesis generation. Then we define the *edit overhead* (EO) as the proportion of all edits that result in unnecessary overhead when interpreting all the edits that are output by the processor: $\text{EO} = \frac{N_{actual} - N_{optimal}}{N_{actual}}$.

EO thus measures the proportion of unnecessary or even harmful edits among the overall edits to the processor's output. If an incremental processor ever 'changes its mind' about previous output, it will need more edit operations to revoke or substitute a previously output increment and EO increases. EO, being a proportion, can be measured as a percentage between 0 % (for an ideal processor) and 100 % (for a processor that changes its mind infinitely).[15] In the running example from Figure 3.3 above, there are 10 edit operations for a total of 4 words, resulting in an EO of 60 %.[16]

Why does edit overhead matter, and why do we need another metric to cover this? Remember that in incremental systems, subsequent components may start to work immediately with incremental outputs of their predecessors. Hence, unnecessary edits mean unnecessary work for all consumers (and, possibly, for their consumers). A processor that frequently changes previously output hypotheses (we call such changes *jitter*) may still go unpunished in similarity metrics (as it may produce the same amount of correct intermediate results, but in a "harmful" order) and while there are influences on timing metrics (see next subsection), edit overhead allows for a direct quantification.

EO describes the overall stability of the output of a processor. However, EO does not describe the stability of individual units. Such a measure can be derived from timing metrics: the time that it takes for an increment to settle after it has first been hypothesized (i. e. the time that it takes to become stable) can be called `correction time`. Average `correction time` can simplistically be computed from the difference between FO and FD. An external confidence measure can be based on this statistic, combined with the 'age' that an increment has reached at runtime (the time that it has survived without being revoked): for example, if a processor is known to have an average `correction time` of less than 500 ms for, say, 90 % of its output increments, then one can be certain to a degree of 90 % that an increment that has been around for 500 ms will not be revoked anymore. While this does not tell whether an increment is final or not, it helps to make probabilistic judgements. Furthermore,

[15] Depending on the operational costs for the *consumers* of the incremental output, different costs might be assigned to addition, revocation and substitution which would result in an extended version of EO.

[16] This counts substitution as two operations: first revoke the old, then add the new word. Considering substitution as one operation would result in an $\text{EO}_{\oslash}$ of 43 %.

the proportion of increments that are immediately correct (i. e. have a correction time of 0, respectively FO = FD) is informative as it describes the degree of non-monotonicity of the relevant, final output.

One drawback with above stability judgements is that timing metrics like FO and FD can only be computed for increments that 'survive' until the very end. However, a processor (with EO > 0) generates many increments that are not part of the final hypothesis. This makes the probabilistic assumptions from above overly pessimistic, because the survival time of increments that are later abandoned is systematically shorter than that of increments that persist. Stability can also be estimated based on survival times of all the increments generated 'along the way' and the two estimates will be compared in Chapter 5.4. However, this stability estimate is much more computationally expensive, as *all* increments have to be considered, not only those that make it into the final hypothesis.

3.3.2.4 Interrelations between Metrics

In general, a processor should perform as good as possible in all the metrics presented above. In practice, the importance of some metrics may be higher than for others. There are interrelations between metrics and these can be exploited by trading one aspect of incremental performance against another. This subsection aims to describe the interrelations between metrics and on this basis, favourable trade-offs will be explored in Chapter 5.5.

To begin with, there are some simple interrelations: perfect correctness is equivalent to perfect timing (FO and FD). Both entail zero EO, but the reverse does not hold as edits may have happened too late (or too early). While late decisions are accounted for by p–correctness, edits that come too early hurt correctness. As a consequence, a clairvoyant processor that generates the correct final result right from the beginning would be incorrect against an incremental gold standard up until the very end. Thus, high correctness is not a suitable target *per se*. However, similarity is a good indicator of whether processing is running as expected and the development of a measure over time can give insights in both processing and properties of the corpus.

A processor that is not always correct is faced with trade-offs: improving timeliness (FO) by making it hazard guesses earlier means that it is more likely to get something wrong (hurting both correctness and EO). In reverse, using more time for deliberation of a hypothesis may reduce EO (which is a major objective for incremental speech recognition; cmp. Chapter 5.5), but also delays decisions, hurting FO. The impact on FD may be different than on FO, as faulty intermediary decisions can be avoided with longer deliberation.

Care must be taken because a processor can also 'cheat' (maybe unintentionally) by exploiting the fact that final hypotheses are proposed to be used as the gold standard

for incremental evaluation. A processor that outputs some hypothesis at t_0 and never changes it reaches perfect scores in all metrics, but the non-incremental performance is nil. Thus, incremental metrics should never be the only target for optimization. In our experience, improving a processor's non-incremental performance (that is, the result for the complete input sequence) will often also improve the incremental properties, as non-incremental performance is an upper bound for all incremental processing.

Finally, good performance in some metric may be more important than in another for a given application. This has to be taken into account when comparing different processors for a given task. Especially, once *decisions* are to be based on incremental hypotheses (e. g. decision to nod to signal understanding), a certain degree of confidence in that hypothesis may have to be reached which can be derived from `correction time` curves.

3.4 Summary

This chapter has laid out the notion of incremental processing to be used throughout the thesis, based on the insight that incremental language processing *needs* to be non-monotonous, that is, allow to change previously generated intermediate hypotheses.

The requirement of structured incremental data (for both input and output) using interlinked units of incremental processing (IUs) was described and we have formalized a scheme for evaluating incremental processors along three dimensions: similarity of results to a given gold standard, timing of individual output increments, and diachronic evolution of the incremental hypotheses.

The various aspects of incremental processing cannot be reduced to one single performance measure, as there are different *dimensions* of processing. However, the performance of processors along these dimensions interrelates. Knowing these interrelations, application-specific optimizations can be performed that result in good trade-offs.

The next chapter presents the software architecture of our toolkit for incremental spoken dialogue processing that operationalizes our notion of incremental processing. Processors implemented in the toolkit are then described in the later chapters and are evaluated along the evaluation scheme presented here.

4 A Software Architecture for Incremental Spoken Dialogue Processing: INPROTK

This chapter describes INPROTK, the software toolkit for incremental spoken dialogue processing that has been developed in the context of this thesis and that is used in the example applications.[1]

Chapter 2 showed that dialogue depends to a large degree on timely interaction (in order to quickly establish common ground and to quickly react to feedback) and inferred that incremental processing is a way of meeting this requirement. In addition, Chapter 2 argues that architectures for dialogue processing need to be modular in order to be flexible, to support concurrent processing, and for cognitive plausibility. At the same time, information exchange between modules should be broad and a module should be allowed access to all lower-level information as this can potentially be relevant for the module's decision making (e. g. a semantic processor might need access to prosody to differentiate between different meanings). Finally, we saw that current systems lack a stringent model of incrementality and, as a consequence can only be partially incremental and only partially meet timing requirements.[2]

Chapter 3 introduced the IU model for incremental processing that forms the basis for our software implementation. Chapter 3 represented information both as full hypotheses as well as using differences between successive incremental hypotheses. INPROTK supports both, absolute and differential representations for hypotheses. Following the IU model, INPROTK builds and uses a network of highly interconnected incremental units at runtime to represent all information derived from the dialogue as well as the system's information state at any point in time. This (dynamic) IU network is produced by a (static) network of processing modules, or other, less formalized processing schemes.

The IU model has predominantly been used to describe input processing, that is, the analysis of incoming information and aggregation of information to larger, more abstract units of information. INPROTK also supports incremental output generation which requires the fission of larger IUs into smaller sub-units. The implementation of both input and output in one system explains many of the differences between INPROTK and the original IU model that we point out in this chapter.

[1] The INPROTK open source software project is hosted at SourceForge: `http://inprotk.sf.net`.

[2] An exception being the NUMBERS system (Skantze and Schlangen 2009), which is also based on the IU model and which can in parts be seen as a predecessor to INPROTK.

InproTK has been developed since 2007 in the incremental dialogue processing project at the *University of Potsdam*.[3] The goal of that research project was to explore methods of incremental dialogue processing (based on the IU model) with a focus on prototype implementations for limited sub-problems of dialogue. InproTK mirrors this focus in that it is fully incremental from the ground. However, some practical ingredients necessary for building fully operational SDSs may still be missing.

InproTK is being developed as an open research platform, implemented completely in the (widely used, industry standard) Java programming language, and distributed as open source software.[4] As a research platform, most software modules allow for low-level access and perform relatively lax checks on their inputs. Of course, this flexibility incurs responsibility on the side of the programmer who uses such modules, calling for a cooperation-based programming style that requires to check all important assumptions. Often, interfaces were later adapted to meet new requirements. It turns out that the author is surprised how well InproTK has coped with ever-changing requirements and how stable much of the foundational code and concepts have been.

While the core of InproTK is very general, the implemented processors follow an evidence-driven (bottom-up) approach where incremental units are gathered by lower-level processors, passed on to higher-level processors which analyze them and ground new IUs on this evidence. The opposite, expectation-driven (top-down) processing is certainly possible as well; however, allowing bi-directional data exchange would complicate inter-module communication immensely. As a final remark, security and stability are of limited interest in research systems (that only need to last until the experiment is over) – it is strongly advised *not* to rely on InproTK in open and potentially hostile environments.

This chapter covers the properties of incremental units (IUs), the IU hierarchy and how the IU network is constructed in Section 4.1, the processing schemes and inter-module communication in Section 4.2, additional infrastructures provided to facilitate building SDSs in Section 4.3, and finally discusses the current state of InproTK in Section 4.4.

4.1 The Data Model: Incremental Units

Incremental units (IUs) in the model by Schlangen and Skantze (2009, 2011) have the two related properties of holding minimal amounts of information and of being minimal units for processing. The 'size' (and timespan covered) of the units varies widely and depends on the level of abstraction of the contained information (and the implemented granularity of the producer). As units are usually combined in

[3] More information on the incremental dialogue project *InPro* is available at `http://www.inpro.tk`.
[4] InproTK and its version history are available via SVN from `http://sf.net/p/inprotk/code/`.

Table 4.1: Five classes of general properties of the IU base class and exemplary operations illustrating the available capabilities.

comparison:	identity	`getID()`
	equality	`payloadEquals()`, `compareTo(IU)`
	timing	`startTime()`, `duration()`
network:	groundedness	`ground(IU)`, `groundedIn():List<IU>`
	same level	`getSLL()`, `getNextSLLs():List<IU>`
status:	information status	`commit()`, `isRevoked()`
	delivery progress	`getProgress()`, `getOngoingGroundedIU()`
updates:	update behaviour	`addUpdateListener()`, `updateOnGrinUpdates()`
	notification	`notifyListeners()`
debugging:	textual representation	`toTEDviewXML()`, `toLabelLine()`, *`toPayLoad()`*

the abstraction process, the higher the level of abstraction, the larger the units (e. g. individual phonemes are combined to form words, one or more words are combined to a semantic unit, …) and the less 'concrete' the contained information.

IUs are *typed* objects using the Java type system, all sub-classing the same *abstract* base class IU. IUs of different types represent different types of information (phonemes, words, phrases, …). As a short overview, general properties and some exemplary operations of the IU base class are shown in Table 4.1 and largely mirror the definition of IU properties by Schlangen and Skantze (2011). Extending that classic IU model, InproTK defines a *progress* status for every IU to explicitly account for IUs that represent data about the past, the future, or events that are currently ongoing.[5]

Not shown in the table and not included in the abstract IU base class are all payload field and payload operation definitions (apart from the abstract `toPayLoad():String` used for debugging purposes). These fields and operations have to be defined by the concrete IU sub-types. For example, `SegmentIU` defines phonemic identity of a segment and start- and end-time in the recognized user speech. As can be seen in Table 4.1, InproTK's IUs are full-blown objects which feature relatively complex operations (such as finding the IU among the subordinated IUs that is currently ongoing). This is especially true for some payload operations, where, for example, a word could be queried for its accentuation status, which it will determine by querying

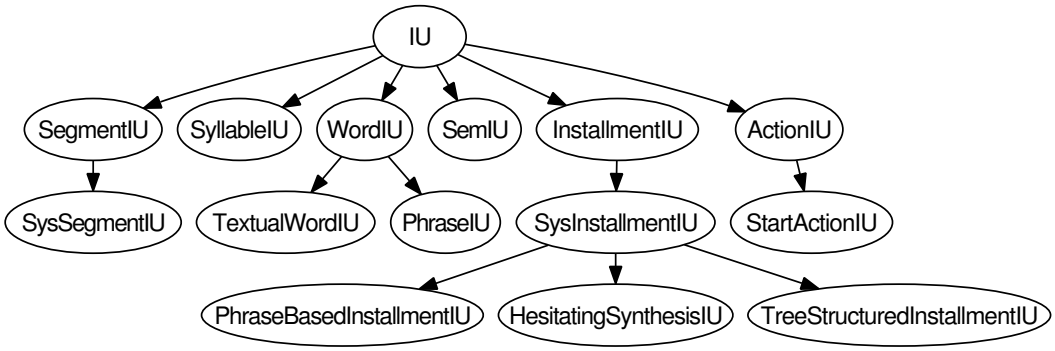


Figure 4.1: An excerpt of the InproTK built-in IU type hierarchy.

for accentuation the syllable IU that holds lexical stress. In contrast, IUs as defined by Schlangen and Skantze (2011) are primarily passive containers of information.

A part of the IU hierarchy in InproTK (for phonemes, words, installments, ...) is shown in Figure 4.1. As can be seen, the hierarchy often contains multiple levels with IU types for specific uses building on more general types, which allows to share concepts (and code). For example, some IU type definitions for both input and output processing are partially shared (`SegmentIU` for input is sub-classed by `SysSegmentIU` for output) or even identical (`WordIU`). This is a characteristic feature of InproTK.

In this section, only the main features of IUs are explained and the IU hierarchy is shown to exemplify how sub-classing is used to simplify IU handling. Interfaces or implementation details of sub-types are discussed only as far as they are necessary for explanation. More details will be explained in later chapters, when necessary.

4.1.1 The IU Network

IUs, being minimal units, need to be linked together to form a whole, complex information representation. The IU model uses two types of links for this: grounded-in links to mark hierarchical dependencies, and same-level links to organize units on the same level.

In InproTK, *grounded-in links* (*GRINs*) are used to point to related IUs on a lower level of abstraction, that is, GRIN links always point down, towards lower levels of abstraction and, ultimately, the primary speech signal. This differs from the original IU model, where grounding marks informational dependency. For bottom-up

[5]The progress information could, in principle, also be derived from the IU's timing and the current wall-clock time; however, IUs that describe the future may still be missing precise timing.

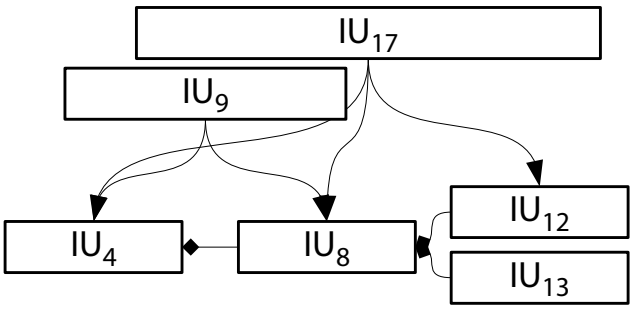


Figure 4.2: IUs interconnected to form a network (including interpretation alternatives).

input processing, this makes no difference but for output processing would result in links pointing upwards, in the opposite direction than on the input side. This reversal would render the use of IU types on both input and output sides difficult or impossible. In InproTK, some IU types (e. g. WordIUs) can be used for both input and output processing as GRINs always point in one direction (towards segment IUs) even though informational dependency differs between input and output side. In the implementation, links are bi-directional, that is, the IU network is a doubly-linked structure that can be efficiently traversed in both directions, so that the logical link direction does not impose any limitations on processing capabilities. As GRINs always point down, they can be used to avoid replication of information: for example, a design principle is to use recursive definitions of start- and end-time of any IU by following GRINs until IUs that represent speech audio, the system's primary signal (cmp. Section 2.1.2), are reached (which contain the actual timing information).

Units on the same level of abstraction are linked together with *same-level links* (*SLLs*); a link between two IUs indicates the (temporal) adjacency of two units. This is a simplification of the more general IU model that allows many types of relations via different types of SLLs; so far, there was no need for more than the adjacency relation in InproTK.

The SLL relation has a cardinality of 1:n, that is, each IU has at most one same-level link to its predecessor unit but multiple IUs may have the same predecessor. This can be seen in Figure 4.2, where both IU_{12} and IU_{13} point back to IU_8. This allows for alternative hypotheses using tree structures. The more general case of allowing to share partial alternatives via any directed acyclic graph (i. e. a cardinality of m:n) was deemed too complex for implementation purposes. The implemented incremental modules (cmp. Section 4.2) only output one-best hypotheses, which

allows for the efficient computation of hypothesis differentials. GRIN is a many-to-many relation, that is, each IU may be grounded in several units and may itself ground several other, as can also be seen in Figure 4.2. This is necessary to allow the association (aggregation) of multiple lower-level IUs to one higher-level, and to allow for alternative interpretations across IU processing levels (in Figure 4.2, IU_9 and IU_{17} would probably be interpreted as representing alternatives.)

During processing, IUs are added to the network as new insight is gathered, or removed from the network if hypotheses (or parts thereof) are abandoned. At any point in time, the network represents all the information that is known to the system. The network is highly dynamic, with changes to the network reflecting the system's changing internal state over time. The next subsection describes the way that processing in InproTK may extend and evolve the IU network structure.

4.1.2 Triangular Data Models

Systems built with InproTK have mostly used data-driven (bottom-up) processing schemes. The IU architecture (and InproTK's data models) would likewise support expectation-driven processing schemes (where expectations guide upcoming data analysis). However, expectations would need to be communicated towards the lower-level processors. InproTK's processing has also mostly followed a one-best approach (which allows easy computation of differential hypotheses as outlined in Section 3.2.2). One-best and expectation-based processing do not match well as only one single expectation could be transported. This subsection instead explores the data-centric bottom-up, one-best approach to incremental processing as implemented and used in InproTK.

As explained above, IUs on higher levels are less granular than on lower levels, and higher-level IUs are often associated to several IUs on the lower level. This also means that during input processing some IUs on the lower level cannot yet be associated to an IU higher up because some 'ingredients' for that higher-level IU are still missing. As a consequence, structure building on higher levels lags behind in input processing. This can be seen in Figure 4.3 (left side), where no word has been recognized yet for the stretch of audio that follows "`in`", and "`in`" itself is still missing an interpretation. Input processing forms a bottom-up left-to-right triangular data structure.

Similarly, for output processing, units are larger on higher levels, and more context (i. e. lookahead into the future) is typically required to make sensible decisions. On lower levels, in contrast, units need not be constructed immediately, but only *just-in-time*, as they are required by lower-level processors (and ultimately, speech output). Using the top-down left-to-right triangular processing scheme shown in Figure 4.3 (right side), the network remains flexible and efficient when accommodating re-interpretations and only minimal amounts of re-processing are required.

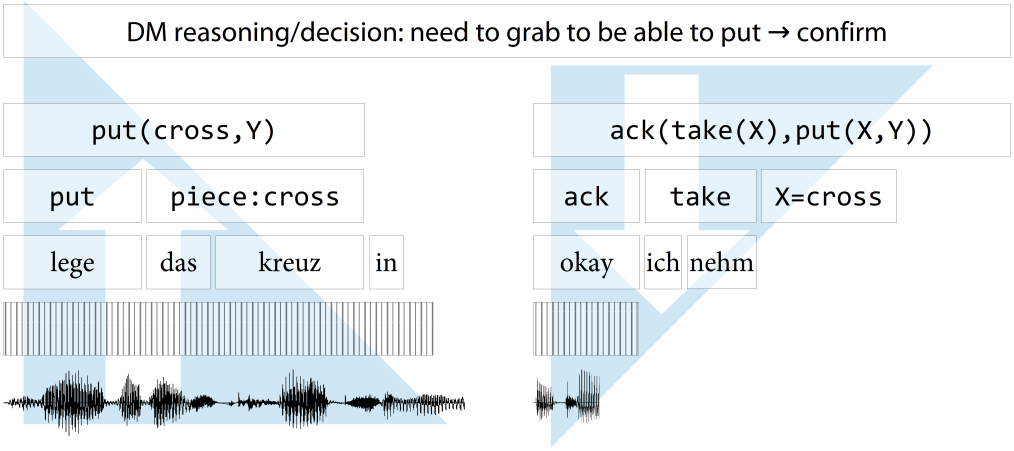


Figure 4.3: Data-driven processing results in triangular IU structures in InproTK: a bottom-up 'input triangle' on the left, a top-down 'output triangle' on the right; both are governed by dialogue management and decision making at the top.

Imagine that in the example in Figure 4.3, the next word recognized in the input could be "`gelb`", resulting in a reference ("`das Kreuz in gelb`") that is unresolvable in the domain. Dialogue management would need to abandon the current output plan of generating an acknowledgement and replace it with some other response (e. g. signal non-understanding). However, only minimal re-processing overhead would have occurred as, for example, NLG had not yet decided on a referent and speech synthesis only pre-generated a small amount of output (as shown in the figure).

Dialogue management, shown at the top of Figure 4.3, has the task of governing the two triangles, taking the decision when to actuate output processing given some input interpretation. This decision-making may 'break' the re-interpretation processes: when a decision becomes observable to the interlocutor, she is faced with a system action to which she will inadvertently react herself, even if the system abandons the underlying hypothesis. For example, if dialogue management decides to start speaking and revokes this decision quickly after, the interlocutor will notice a false start and may react to it. Buß and Schlangen (2011) have implemented dialogue management strategies to cope with this inevitable problem as best as possible. DeVault, Sagae, and Traum (2009) try to find points of maximum understanding to identify where breaking the reinterpretation does not do any harm.

The IU network as a whole is interlinked so that links can also be traversed between output and input triangles: for example, mirroring the user's referring expression helps to build rapport (Brennan 1996); the generation component can follow the GRINs to find out whether the user said "`Kreuz`", "`X`" or "`das rote Teil`" to refer to a piece (represented by a single logical form) and easily mirror that referring expression.

The strength of the IU model is its capability to handle incrementally evolving and changing *hypotheses* in a principled way. An implemented system is additionally required to keep *factual* data which does not change, and on which these changing hypotheses are built. Thus, the final aspect of the InproTK data model is the BaseData store. The *BaseData store* contains the user-provided input in several streams that can be accessed based on their timing information such as speech from the microphone, derived ASR feature vectors, or potentially camera feeds from a webcam, or derived gaze information. While not currently implemented, system output could also be stored for future reference. Base data is not a matter of hypothesis but a matter of fact, which is the principled difference to IUs. Hence, base data need not be revisable. IUs do not need to actually store copies of the input data but merely refer to the base data store which results in a space-efficient implementation.

The next section discusses the processing modes that can be used to produce IU networks as described in this section.

4.2 The Processing Model

The IU network can be created and manipulated in several ways. The conceptually simplest way is by using incremental modules as proposed by Schlangen and Skantze (2009) which is detailed in the following subsection. Two other processing schemes, one based on active IUs, the other based on IU update listeners, will be outlined in Subsection 4.2.2.

4.2.1 Incremental Modules and Inter-Module Communication

Incremental modules are the 'classic' processing scheme in the IU model. They match very well the division of SDSs into components as presented in Section 2.2, especially Figure 2.3, which showed the information flow between components in a dialogue system. Each component in the dialogue system would be realized as an incremental module in InproTK. Section 3.2.1 defined abstract incremental processors which the incremental modules presented here implement.

Incremental modules consist of a *left buffer*, a *processor*, and a *right buffer*, where the processing mode is to consider input in the left buffer, process it (possibly considering

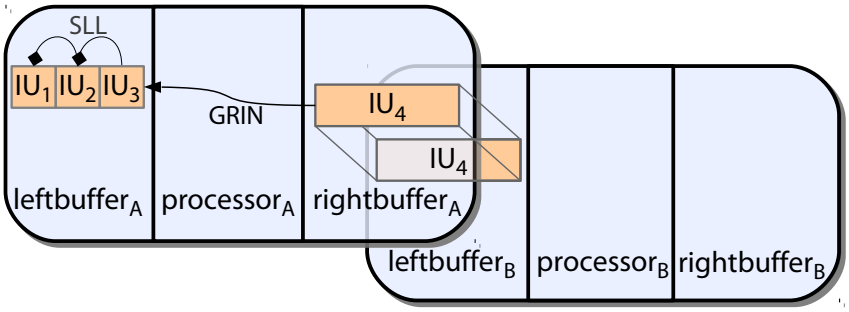


Figure 4.4: Two incremental modules A and B consisting of a left buffer, a processor and a right buffer. Module A's right buffer shares its content with module B's left buffer. Input and output IUs are connected with grounded-in links (GRIN); successive IUs on one level are linked with same-level links (SLL).

some state internal to the processor), and to provide (or alter) output in the right buffer.[6] As can be seen in Figure 4.4, incremental modules are interconnected by conceptually superimposing one module's right buffer with another module's left buffer. Thus, output that is changed by one module automagically results in an input change in connected left buffers. 'Superimposed buffers' are implemented by directly passing on data from from a right buffer to all connected processors' left buffer update methods. (That is, left buffers only exist as a concept but are not explicitly represented in the implementation.)

The module network is (statically) configured in a system configuration file and modules need not necessarily form a pipeline, as one module may output to and receive input from several other modules.[7] Most modules, however, expect input IUs of a specific type only, and deliver output IUs of another type (or a few related types). Thus, care must be taken to connect modules to each other that actually can work together in a meaningful way. Currently, there are no checks implemented that enforce the configured module connections to be compatible (or meaningful). Care must be taken by module implementers to check the incoming IUs for their type and to handle misconfigurations (e. g. by throwing errors). Some debugging modules,

[6]The IU model by Schlangen and Skantze (2011) additionally allows for the opposite direction in information flow. For implementation simplicity, this is not implemented in InproTK, as difficult-to-solve concurrency problems could potentially arise with such a circular processing scheme.

[7]In principle, a module may also connect its right and left buffer; this may be useful for parsing, where constituents need to be combined to form other constituents in a later step. However, as noted in Footnote 6, concurrency issues must be addressed.

such as a generic hypothesis dump utility as well as a module visualizing the current state of the IU network accept any type of IUs.

Section 3.2.2 developed that incremental data can be dealt with either as full hypotheses or as differences between successive hypotheses and defined *edit operations* to describe these differences. Three types of edits are implemented in InproTK to inform about buffer changes:

- an *add* edit informs the processor about an IU newly added to the left buffer,
- a *revoke* edit informs the processor that the IU that previously ended the current hypothesis has been revoked from the left buffer, and
- a *commit* edit informs the processor that an IU in its left buffer has become stable and will not be revoked in the future.

When receiving one or multiple edits, this may trigger processing and lead the processor to change its own right buffer (adding, revoking, or committing to its own IUs) and these edits are again passed on to connected modules.

Exchange of edits can be more space-efficient than passing around full hypotheses, and operations can often be defined more easily based on edits, if only the changed part of the hypothesis is relevant. However, depending on the task of the module, using full hypotheses may be more convenient than using edits (i. e. differential hypotheses). For this reason, hypothesis changes in InproTK are (redundantly) characterized by passing both the complete current buffer (a list of IUs) as well as the difference in edits between the previous and the current state (a list of edits of IUs), leaving modules a broader choice of implementation. InproTK's abstract implementation for incremental modules that all implemented modules are based on, `IUModule`, allows access to both full and differential representations as input and gives the implementer the choice to provide as output either a full hypothesis (in which case the edits are computed), or the edits that should occur (in which case the full hypothesis is computed).

Incremental modules in InproTK are not fully separate components in the sense of execution threads in a concurrent, multi-threading environment (as defined by the Java virtual machine) in that module boundaries are not automatically also thread boundaries. In contrast, execution threads may enter an incremental module by calling the left buffer update method, incur some processing and then exit the module via right buffer update calls, continuing on to further modules. The permeability of component boundaries may result in synchronization issues when multiple threads are used but it avoids the cost of multiple thread-boundaries when using many but relatively simple modules. No systems built with InproTK have required a thread separation so far. If necessary, it would be simple to introduce thread-boundaries in the abstract `IUModule` class. However, as IUs are interconnected, full separation of components and access only via calls to buffer interfaces, as may be seen in the pure IU model, is not a goal of InproTK.

Most modules can afford to be fully event-driven, only being called into action when their left buffer changes. However, some modules may need to endogenously create, retract or manipulate IUs, for example based on time-outs. They may either implement above-mentioned thread-boundaries and use their own time-out mechanism, or use the system-wide time-out facility, which uses a separate signalling mechanism and currently supports to set time-outs for several types of turn-taking events.

Finally, as currently implemented, all modules live in the same process and shared memory. However, inter-module communication may transparently use a messaging protocol by plugging pairs of appropriate communication modules into the module network.[8] In such a use-case it may be profitable to only send edits and to leave the reconstruction of the full state to the receiving side. Notice that IU links would not be readily available across processes, unless special care is taken to replicate these as well, or to obtain them on demand. Contrasting INPROTK's implementation, IPAACA (another incremental dialogue architecture that was first described together with INPROTK and JINDIGO, yet another incremental dialogue architecture, by Schlangen et al. 2010) is tailored for dialogue systems spanning multiple processes (potentially on different computers) and with modules implemented in different programming languages. IPAACA uses message passing between modules and remote procedure call mechanisms when querying or modifying IUs that are owned by a different module (the concept of IU ownership, a feature of the abstract IU model, is not used in INPROTK).

4.2.2 Alternative Processing Schemes

Incremental modules are a conceptually simple way of structuring the full system into smaller (and re-usable) sub-components which perform simpler tasks. However, as currently implemented, some processing steps, such as signalling back to an earlier module are impossible, and implementing these would erase much of the current simplicity (both conceptually and implementation-wise). We hence propose two other schemes for more advanced processing that can be combined with incremental modules: one based on active IUs and the other based on update listeners. Of course, combinations of these processing modes with IU module-based processing is possible, even though it may increase the complexity of the resulting control flows.

We call IUs *active IUs* if they do arbitrarily complex processing themselves, including the generation of more IUs (that may themselves effect more processing). IUs are informed about their commit and revocation status (cmp. Table 4.1) by right buffer objects, and hence can act upon these events, allowing for a mixture of this processing

[8] Early versions of INPROTK used the open agent architecture (OAA, Martin, Cheyer, and Moran 1999) for inter-module communication but other communication means could also be used.

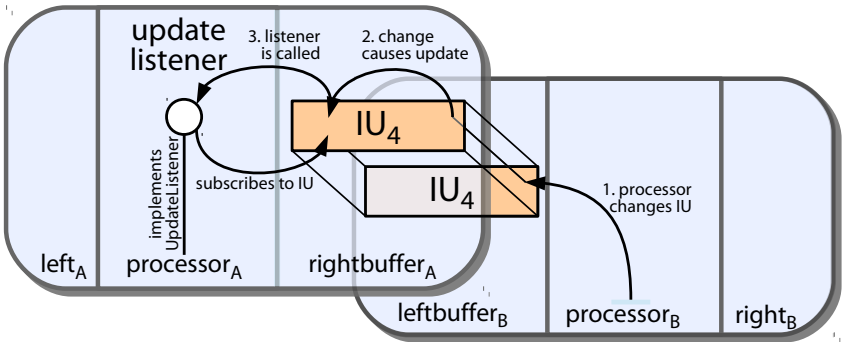


Figure 4.5: Updates to IUs enable right-to-left communication across the left-to-right interface between incremental modules: processor A implements the `IUUpdateListener` interface and subscribes to updates for each IU that it puts into the buffer. If processor B later changes an IU, the change will cause an update, the listener is called, and the processor can react.

scheme with module-based processing. Systems that employ active IUs are presented in the example application in Section 5.6, as well as in (Baumann et al. 2013).

Update listeners can be added to IUs and are called upon specific events such as edits, progress change, or other events. Thus, they can perform all the processing that modules can, including the generation of new IUs (to which they would need to subscribe corresponding update listeners similarly to processors that are interconnected). Update listening can also be employed by modules to enable right-to-left feedback across the left-to-right buffer interface as shown in Figure 4.5. The power of update listeners comes with an increased complexity of the control flow and requires careful implementation in multi-threaded applications. Implemented somewhat differently from InproTK, JINDIGO (Schlangen et al. 2010; Skantze 2010) makes extensive use of update listening (from where this addition to InproTK was inspired), even to implement standard module communication.

To conclude, incremental modules provide a robust and stable interface that can most easily be re-used across systems. In contrast, while active IUs and update listening may sometimes require careful tuning to the application, these schemes allow for advanced processing options that may be required for some intended behaviour.

4.3 Infrastructure

InproTK also includes infrastructure not yet discussed that supports rapid prototyping of (partial) incremental dialogue systems in research environments.

Command-line applications that start up InproTK help to get a first grasp at using InproTK's basic modules, simplify common configuration choices via command-line options and can transparently be interchanged to switch between speech and text input for testing purposes. InproTK builds on CMU Sphinx' configuration management for simple yet powerful configuration of system modules and their interconnection.

Audio input and output can be configured to use the sound card, audio files, or RTP (Schulzrinne et al. 2003) for audio transport via a network. Multiple channels can be used simultaneously, for example to log an interactive session to disk.

InproTK includes a range of GUI components for building partially multi-modal SDSs in the Pentomino domain that can also be used for other puzzle-playing or command-and-control tasks. Some GUI components are included to aid in Wizard of Oz experiments where the dialogue manager is replaced by a human experimenter.

InproTK comes with incremental modules for speech recognition based on CMU Sphinx-4 (Walker et al. 2004, see also Chapter 5) and speech synthesis based on MaryTTS (Schröder and Trouvain 2003, see also Chapter 7) which can also be accessed in simpler, non-incremental modes with support for several different Mary versions, synthesis modes and voices. In addition, input is prosodically analyzed for pitch, using a (reduced) version of the YIN algorithm (Cheveigné and Kawahara 2002), and signal loudness is measured using the revised B-curve filter (Skovenborg and Nielsen 2004) as proposed by Zemack (2007).

4.4 Discussion

This chapter presented InproTK, the toolkit for incremental spoken dialogue processing developed in the context of this thesis. In the toolkit, data is represented in minimal units of information that are interconnected to a network which represents the system's understanding of the world. The network is constructed incrementally, by processing modules or through other processing schemes, as input becomes available to and output is generated by the system.

As can be expected, InproTK is the result of a learning process and this learning process is far from over; in other words: InproTK does not represent the ground truth for implementing the abstract IU model (let alone for incremental processing in general). Neither do its two main competitors, JINDIGO and IPAACA.

Some choices for an ideal embodiment of the IU architecture and the associated processing patterns remain open questions. One fundamental trade-off to be con-

sidered is that of modularity and data encapsulation on one side vs. integration and availability of data for processing on the other side. InproTK tries to mediate between these extremes through the use of clear and simple module interfaces on one side that nonetheless leave free access to all information via the IU network. While the present author has become very familiar with how much one module may fiddle with parts of the IU network that 'belong' to a different module, it remains to be seen how well this approach scales to other modules, to larger systems, and to programmers less experienced in using InproTK.

Many patterns for incremental processing within the IU model are possible and three were described in Section 4.2 and are implemented in InproTK. The original IU model opens up many possibilities at the cost of some structural redundancy. InproTK implements only a limited set of IU model capabilities (extending it in other places), and both JINDIGO and IPAACA do the same but take different choices. Future work, and with it more experience with incremental processing, will help to bring about a required set of generic capabilities for IU-based incremental processing. Hopefully, tested design patterns for modularized incremental processing will emerge that may provide guidance when tackling new problems. While processing based on incremental modules as implemented in InproTK provides for a simple and robust pattern, its insufficiencies in complex systems (pure bottom-up processing with no feedback to or expectations for earlier modules, limitation to one-best processing) are obvious. They are justified by InproTK being part of the first generation of incremental SDS architectures and will undoubtedly be overcome in the future.

InproTK is by no means finished, as important components such as incremental NLU and NLG for general use and incremental dialogue management are still being developed, with decision making remaining a major challenge. Modules that do exist and were implemented as part of this thesis (iSR and iSS, see Chapters 5 and 7, respectively) are singular and the universality of InproTK would be well tested if alternative implementations existed. To this end, it is currently planned to integrate the *Android* iSR component into InproTK as an alternative to the open-source version to be described in the next chapter.

A final issue is that of error recovery: generic support for high-level processing exceptions could help in error cases and a principled handling of partial failure would increase system reliability immensely. An error recovery strategy, if implemented in a specific module that governs other, sub-ordinated modules could also be used to support running multiple, concurrent variations of modules or sub-pipelines that the governor could select from at runtime.

Another, move would be to abandon the current home-grown object structure and to base the IU network on a full object-based distributed database. The advantage would be full database capabilities, such as transactions, locking, and consistency.

Reproduced with kind permission by Heise Verlag and Ritsch & Renn from c't 16/2012.

5 Incremental Speech Recognition

A spoken dialogue system needs to be able to listen to a user's speech and hence speech recognition is the main input channel for any SDS. *Incremental* speech recognition (iSR) is hence a requirement for any *incremental* spoken dialogue system. If incremental speech recognition were infeasible, iSDSs could not become a reality.

This chapter reports the results of the 'incrementalization' of the open-source speech recognition engine that is integrated into InproTK. That speech recognition can in fact be used incrementally is anything but granted and this chapter reports a detailed evaluation of the relevant aspects for incremental processing.

To provide some background, the next Section 5.1 presents a short introduction into automatic speech recognition culminating in the central idea of the token pass algorithm in Subsection 5.1.2.2 which forms the basis for incrementalization as detailed in Section 5.2. Readers familiar with HMM-based speech recognition may want to skip the background section.

Chapter 3 detailed the notion of incremental processing underlying our approach, where hypotheses can be changed after they are first generated, and defined metrics suitable for evaluation. In addition, the trade-offs between aspects of incremental performance, such as timeliness and edit overhead, were discussed.

Section 5.3 presents Intelida, an evaluation workbench for incremental speech recognition that implements the incremental performance metrics and evaluation results are presented in Section 5.4. Section 5.5 explores optimization techniques that aim at optimizing performance trade-offs with a focus on the reduction of edit overhead while keeping the reduction in timeliness low.

Finally, Section 5.6 presents an application of optimized iSR to a speech-control task. The section details the implementation and integration with InproTK, presented in Chapter 4, an application-specific iSR optimization strategy, and a small user study that shows that iSR allows for well-timed interactions and in which the application-specific optimization strategy somewhat outperforms the generic techniques developed before.

Section 5.7 summarizes and discusses the findings in this chapter and outlines some possible improvements.

5.1 Automatic Speech Recognition in a Nutshell[1]

Automatic speech recognition (ASR) faces the problem of identifying the sequence of words (if any) that are spoken in a given stretch of speech audio.[2] Mainstream ASR technology follows the metaphor of the noisy channel (cmp. Chapter 2) and uses *Hidden Markov Models* (HMMs) to handle the separation between noise and signal.

In the application of the noisy channel model to speech recognition, the observed speech waveform ($\mathbf{O}$) is the signal and the message ($\mathbf{W}$) is a sequence of words. The task for the ASR (the *receiver* in noisy channel terminology) is to find the one sequence of words among all possible sequences that is encoded by the signal. All sources of uncertainty about the message – arising from unclear speech, actual interfering noise, homophony (different words sounding alike), and others – are modelled by the channel's noise source.

An important simplification that makes the ASR task feasible is to assume a *closed language*, that is, to assume that the user's utterance is comprised of words that are part of a known *lexicon* ($\mathbb{L}$) and that the spoken sequence of words ($\mathbf{W} = w_1, w_2, \ldots, w_n$) is limited to being one of all possible sequences of these words (the ASR's *language*: $\mathbf{W} \in \mathcal{L} = \mathbb{L}^*$). A slight extension is what we call *rich speech recognition* which also provides the time alignment of words. In that case, the ASR's output includes timing information for each word: $w = (\mathit{lexeme}; \mathit{start}; \mathit{end}) \in \mathbb{W} = \mathbb{L} \times \mathbb{N} \times \mathbb{N}$ (with the additional constraint that $\mathit{start} < \mathit{end}$ and these times measured in full frames).

In spoken language, two utterances of the same words will always sound slightly differently. HMMs are a model to deal with this. *HMMs* are productive stochastic models, which may generate many alternative observations for a given state sequence. The *hidden* states of the HMM model the word (or phoneme) sequence, which is said to *emit* observations that correspond to the speech signal. HMMs can be used *in reverse* to find the probability for a state sequence given an observation sequence ($\mathrm{P}(\mathbf{W}|\mathbf{O})$). When using *maximum likelihood estimation*, the ASR's task is to choose

[1] Speech recognition has been described by many, and in excellent ways. Notably, Jurafsky and Martin (2009, Chapters 6 and 9) give an excellent and broad overview, Taylor (2009) details acoustic modelling, Rabiner and Juang (1993) give detailed proofs of the learning processes, and Gales and Young (2007) also cover many advanced topics. The structure of most descriptions is similar, and it is adopted for this introduction which aims to provide the background necessary and sufficient for the present purposes.

Readers familiar with HMM-based speech recognition may want to skip this section.

[2] This defines *continuous* speech recognition – we leave out the simpler problem of *single word* recognition because stretches of speech in natural dialogue most often consist of multiple words.

the one sequence of words ($\hat{\mathbf{W}}$) from all possible word sequences that it finds *most likely* to be the message given the observation:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W} \in \mathcal{L}} \mathrm{P}(\mathbf{W}|\mathbf{O}) \tag{5.1}$$

Speech recognition uses *machine learning* and the development of speech recognition systems is largely *data-driven*. This means that the types of models for speech recognition are chosen by engineers, but that the models' *parameters* are estimated from large amounts of training data. Even though the training processes are important, they are not covered in detail in the descriptions below, as they are not necessary to understand the remainder of this thesis. However, the requirement of being trainable is important to understand modelling choices.

In the remainder of this section, we will look at how Equation 5.1 can be broken into feasible parts using the Bayesian method, how to estimate parameters for the data models that describe the parts, and how to efficiently apply them in ASR *decoding* using HMMs.

5.1.1 Modelling Speech Data

The speech recognizer's probability estimation task formulated in Equation 5.1 can be broken into two parts by applying Bayes' rule[3], resulting in:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W} \in \mathcal{L}} \frac{\mathrm{P}(\mathbf{O}|\mathbf{W})\,\mathrm{P}(\mathbf{W})}{\mathrm{P}(\mathbf{O})} \tag{5.2}$$

Equation 5.2 can be simplified by dropping the denominator $\mathrm{P}(\mathbf{O})$. (The denominator is constant for all $\mathbf{W}$ and hence doesn't change the outcome of the arg max function.) Thus, we get:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W} \in \mathcal{L}} \mathrm{P}(\mathbf{O}|\mathbf{W})\,\mathrm{P}(\mathbf{W}) \tag{5.3}$$

Practical large-vocabulary continuous speech recognition (LVCSR) systems use an additional pronunciation model. This describes how the words are expected to be spoken, i. e. determines the phoneme sequence that corresponds to the word sequence ($\mathbf{Ph}(\mathbf{W})$). As a result we get:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W} \in \mathcal{L}} \mathrm{P}\left(\mathbf{O}|\,\mathbf{Ph}\left(\mathbf{W}\right)\right)\mathrm{P}\left(\mathbf{W}\right) \tag{5.4}$$

[3] Bayes' theorem proves that $\mathrm{P}\left(x|y\right) = \frac{\mathrm{P}(y|x)\,\mathrm{P}(x)}{\mathrm{P}(y)}$.

The acoustic observations in the observation sequence are not independent of each other (stemming from the fact that speech parameters change slowly over time). However, this is assumed by the way the acoustic observations are calculated (see Subsection 5.1.1.3). This results in different weights between acoustic model and language model in the overall probability estimate, which must be countered by a *language model (LM) weight* resulting in the slightly adapted equation (Jurafsky and Martin 2009, p. 349):

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}\in\mathcal{L}} \mathrm{P}\left(\mathbf{O}|\,\mathbf{Ph}\left(\mathbf{W}\right)\right)\mathrm{P}\left(\mathbf{W}\right)^{LM\ weight} \tag{5.5}$$

Values for the LM weight are set empirically to optimize recognition on held-out test data. A typical range for the factor is between 5 and 20 (Jurafsky and Martin 2009, p. 349).[4]

The two factors in Equation 5.4 can be described as the prior probability of the word sequence, $\mathrm{P}(\mathbf{W})$, and the likelihood of observing a signal given the word sequence, $\mathrm{P}(\mathbf{O}|\,\mathbf{Ph}(\mathbf{W}))$, where $\mathbf{Ph}(\mathbf{W})$ determines the expected pronunciation for the word sequence $\mathbf{W}$. Factoring out the phonemization, the speech recognizer closely resembles the standard linguistic structure of grammar, lexicon, phonology, and acoustic phonetics, and the division of linguistics into sub-disciplines as shown in Table 2.1.

The three ingredients use different formal models (to capture well the respective properties of the underlying problems) and model parameters are estimated independently for the *language model*, *pronunciation model*, and *acoustic model*, respectively, and are easier to estimate from available training data than the direct probability of how well a word sequence matches a given observation. The formal models are recombined in the HMM-based decoding algorithm that we describe in Subsection 5.1.2.

5.1.1.1 The Language Model

The language model's task is to estimate the probability $\mathrm{P}(\mathbf{W})$ of any word sequence $\mathbf{W}$ allowed by the language $\mathcal{L}$. Additionally, the language model needs to enumerate all $\mathbf{W}$ in $\mathcal{L}$ during the decoding process, see Subsection 5.1.2. There are two basic kinds of language models: structured grammars and unstructured, *flat* N-gram models.

Grammars are sets of rules that define the structure of the language. *Phrase structure grammars* define re-write rules such as "S → NP VP" or "N → elephant" with capitalized non-terminal and lower-case terminal symbols. Starting with an initial string "S", symbols in the string are re-written according to the rules until no non-terminal

[4]Finally, the changes introduced by the LM weight prefer sequences with fewer, longer words than more, shorter words. This, again, can be balanced by an additional *word insertion probability*. Both LM weight and word insertion probability are set from held-out validation data to minimize error rates.

symbols are left. If rules are *weighted* the resulting sequence is assigned a probability estimate by multiplying the weights of the applied rules.[5] An advantage of grammars is that they provide a structure for the word sequence that can be useful in further processing (e. g. natural language understanding). However, they lack robustness against unexpected or malformed input: if there is no rule to cover a certain structure, then all word sequences exhibiting this structure will be excluded. Accurate rule-sets for probabilistic language models are hard to estimate from data and tedious to write by hand. A hybrid approach could be to write the rules by hand and to then estimate rule weights from data.

An *N-gram model*, in contrast, is a probabilistic model that operates only on the language surface, assigning probabilities to the word sequence based on the probabilities of sub-sequences. The correct probability of the sequence **W** can be written as the product of the conditional probability of each word following its predecessors (Jurafsky and Martin 2009):

$$\mathrm{P}(\mathbf{W}) = \prod_{i=1}^{n} \mathrm{P}(w_i | w_1, w_2, \ldots, w_{i-1}) \tag{5.6}$$

The key idea of N-grams is to approximate the true probability of a word following *all* of its predecessors by looking at the probability of the word following just the previous N − 1 words (together forming sub-sequences of length N). In the simplest case – *unigrams* with N = 1 – the probability of a sequence **W** is approximated by the product of individual word probabilities: $\mathrm{P}(\mathbf{W}) \approx \Pi_{i=1}^{n} \mathrm{P}(w_i)$. For larger N, this generalizes to:

$$\mathrm{P}(\mathbf{W}) \approx \prod_{i=1}^{n} \mathrm{P}(w_i | w_{i-\mathrm{N}}, w_{i-\mathrm{N}+1}, \ldots, w_{i-1}) \tag{5.7}$$

Limiting the left context is necessary to be able to reliably estimate the probabilities by counting the relative occurrence in training data, where the complete word sequence might not occur (resulting in zero probability for the sequence) even though shorter sub-sequences do occur.

Now, the larger N, the closer the approximation approaches the true probability. However, our estimation of an N-gram's probability will become less reliable for larger N due to *data sparsity*: e. g., some N-grams may never be seen in training data and still occur during recognition. In the simple model, such N-grams will be assigned zero probability, and as a consequence the whole sequence probability will be zero (as it is modelled as the product of its trigrams).

The trade-off between larger N (better approximation of true probability) and lower N (better estimates from training material) can be improved upon by combining

[5] If rules are unweighted, all sequences allowed by the rules are assigned the same probability.

higher-order and lower-order N-gram models. The inaccurate estimation of higher-order models can be *smoothed* by distributing some probability mass to N-grams that have never occurred in the training data (but might still occur in the application), by *interpolating* higher N-grams with lower N-grams, or *backing off* to lower N-grams if the higher N-gram never occurred. Thus N-gram models typically assign some (very low) probability to any word sequence, avoiding the zero probability problems of grammars. The generic term *Statistical Language Model* (SLM) is often used as a synonym for N-gram models.

5.1.1.2 The Pronunciation Model

The pronunciation model's task is to determine the sequence of phonemes that are used to pronounce a word sequence. In the simplest case, the pronunciation model just performs a dictionary lookup for each word's pronunciation. In more complex models, post-lexical phonological processes can be used to model elision effects that may change phonemization at word boundaries.

Going beyond dictionary lookup, statistical models (e. g. joint-sequence models, Bisani and Ney 2008) trained from data can be used to determine the most likely phoneme sequence. Such models that build an abstraction over the training data can provide phoneme sequences for words that were never seen in the training process. In practice, this reduces the model's interdependence with the language model and as a result reduces development effort.

Statistical models as well as dictionaries may provide several likely pronunciations for the same word sequence. In this case, the ASR can include this information into the search for the most likely word sequence by opening up alternative paths in the search space. While alternatives allow for better pronunciation matching, this may not actually translate into better recognition results as probability mass gets distributed among more hypotheses.

5.1.1.3 Acoustic Modelling in the ASR Frontend

The task of the acoustic model proper is to estimate $P(\mathbf{O}|\mathbf{Ph})$, that is, to capture how well an acoustic observation matches a given phoneme sequence. This task can be split into two parts: the first is to bring the acoustic waveform into an appropriate form that focuses on relevant and disregards irrelevant signal variation and will be described in this subsection. The second is to actually make use of the observation sequence for speech recognition. That part is tightly integrated with the HMM methodology and will be explained in the next subsection.

The acoustic speech waveform is continuous[6] but for speech recognition with HMMs it is modelled as a sequence of observation *frames* ($\mathbf{O} = o_1, o_2, o_3, \ldots, o_n$), equally distributed in time, often every 10 milliseconds. During digitization, speech audio is commonly sampled at 16 kHz and quantized using 16 bit samples. Thus, each frame may be one out of a total of $160 \times 2^{16} \approx 10$ million distinct observations that are possible for every 10 ms of audio. Given the fact that languages distinguish no more than 50 phonemes (Ternes 1999), the huge amount of redundancy in the speech signal should be clear.[7] The goal of acoustic modelling in the speech recognizer's *frontend* is to reduce this redundancy. The methods used are based on (simplified) phonetic knowledge about human speech perception and production.

Human auditory perception is characterized by a focus on spectral characteristics of the signal, that is, on analyzing the subfrequencies contained in the signal. By and large, the auditory organ measures the peaks of the signal envelope as well as the fundamental frequency of the speech signal (Taylor 2009). At the same time it is robust against noise and different loudness conditions, and exhibits some non-linearities in the frequency domain and the signal power.

Human speech production can be described by the *source-filter model* in which the vocal folds in the glottis are the source of a primary sound (somewhat resembling an impulse train for voiced, and white noise for unvoiced speech) which then passes the *vocal tract*, which acts as a resonator and filters out frequency ranges and amplifies others (the *formants*) depending on its setting (Pétursson and Neppert 1996). The soft tissue in the vocal tract causes resonance to show mostly the attenuation of frequencies, and very little amplification. This is especially true for higher frequencies which are attenuated more strongly, resulting in a phenomenon called *spectral tilt* of the signal. When articulating, the vocal tract's setting is adjusted (and with it the resonance frequencies change) so as to produce the different speech sounds which are marked by their characteristic signal envelopes. Orthogonally to phonation in the vocal tract, the fundamental frequency of the primary signal determines intonation (or more specifically, *pitch*) and does not affect the identity of the articulated speech sound. The result is a signal that is *quasi-stationary*, i. e. it is stationary in the short-term (there is little spectral change between adjacent frames) and slowly changes over time (as adapting the vocal tract is relatively slow).[8]

The methods to model the speech signal in practical systems that we will describe below together define *mel frequency-adjusted cepstral coefficients* (MFCCs, Mermelstein 1976), which are the features used in the speech recognition system described

[6] At least before digital sampling in the sound card.

[7] Although we might need to model the transitions between phonemes (and there are on the order of 50^2 transitions) – we will come to that further below.

[8] The burst phases of plosive sounds are not quasi-stationary and as a result are somewhat imprecisely modelled by the ASR frontend.

below (and used in many state-of-the-art systems). MFCCs use a technique called cepstral analysis which can be used to deconvolve (i. e. separate) the fundamental frequency of the signal (caused by the source of the source-filter model) from the spectral envelope of the signal (caused by the filter). In cepstral analysis, the signal is Fourier transformed into the frequency domain, logarithmized and Fourier transformed one more time (hence the name of the technique, the reverse of *spec*tral). The parameters describing the source (mostly its fundamental frequency) are now clearly separated from those describing the filter parameters (determining the signal envelope). This seemingly arbitrary procedure works because taking the logarithm in the spectral domain (where the convolution of signals is reduced to a multiplication of spectra) further reduces convolution to addition and allows deconvolution with the second (re-)transformation (Oppenheim and Schafer 2004). (Also, this only works for signals which can be sufficiently well described by the source-filter model but not for arbitrary signals.)

To model spectral tilt and the higher auditory sensitivity to higher frequencies, the signal is first *pre-emphasized*, that is, filtered to enhance higher frequencies. As a next step, the speech signal is reduced into a sequence of observation frames by *windowing* with the window being advanced by 10 ms between observations (Taylor 2009). Most commonly, the window size is slightly larger (e. g. 25.6 ms), resulting in an overlap, and uses a Hamming window to reduce the introduction of windowing artifacts into the frequency analysis.

Each window is transformed into the frequency domain using the Fourier transformation. The phase information is discarded by ignoring the imaginary components of all coefficients. Auditory determination and discrimination of frequencies is linear only up to a frequency of about 500 Hz and pitch is perceived log-linearly with lower spectral resolution above. This relation is captured by the *Mel scale* and the ASR uses band-pass filtering to aggregate the energy in different frequencies into consecutively wider and broader spaced frequency bins, commonly 40 bins for 16 kHz (Mermelstein 1976).

These binned frequencies are then logarithmized and reverse-transformed into the cepstral domain. Overall signal power and the first twelve cepstral coefficients that together describe the signal envelope are then combined into the MFCC vector. The remaining cepstral coefficients that cover irrelevant details of the signal (including the fundamental frequency) are discarded.

Finally, to filter out additive noise from the MFCCs and to focus our features on deviations rather than means, each coefficient is normalized by subtracting its global mean in a process called *cepstral mean normalization*.[9]

[9]Of course, in incremental processing we do not know the global mean because the signal's future is still unknown. However, the global mean can be approximated from a sliding window that captures

Articulation changes relatively slowly (compared to the windowing frequency) and the transitions between speech sounds are just as important as the *stationary phases*.[10] This somewhat breaks the short-term stationarity assumption from above. To explicitly model gradual change, the delta between adjacent frames' parameters is used, as well as the the delta of deltas, for change and acceleration, respectively, resulting in a total of 39 parameters per feature frame.

While MFCCs reduce the original data and help to 'sharpen' the signal in a way suitable for recognizing speech, they still allow many degrees of freedom. To reduce computational needs, *quantization* can be used to reduce the MFCCs to a limited number of, say, 256 or 1024 different observation classes. Recognition would then proceed in determining the phonemes corresponding to the sequence of classes.

More precise but much more computing intense (and requiring more training data for reliable estimation) is to calculate the multidimensional *Gaussian distributions* that best describe all occurrences of each phoneme (or parts thereof), which can be represented by means and covariance matrices. To reduce model complexity, the covariance matrix is often assumed to be diagonal (i. e. features are assumed to be independent) which approximately holds for MFCCs (Taylor 2009), resulting in two vectors μ and σ to describe the distribution. The Gaussian distributions can then be used in *continuous Hidden Markov Models*. Not all distributions are well described by a Gaussian. *Mixtures* of Gaussians are uesd to describe such distributions that overlay several Gaussian distributions (often 4 or 8, also called *Gaussian Mixture Model*, GMM).

The context of a phoneme determines its acoustic realization due to *coarticulation* between speech sounds. Coarticulation can be modelled by developing different models for each phone's preceding and following context resulting in about $50^3 = 125000$ *triphones*.[11] To counter this increase in search space, *state-tying* can be used where different triphones refer back to the same Gaussian distributions, to unify, e. g. the states of [a] following [b], and [m] (which share certain phonological characteristics). Whether to tie states and how is determined during training, usually with decision trees that build on manually defined phonological features.

5.1.2 The Speech Recognizer

We have described above how to model word sequences with grammars or N-gram models, the mapping of words to phones, and how to model speech audio as a sequence

the past resulting in a long-term mean subtraction. This has the additional advantage of adapting to slowly changing additive noise.

[10] In fact, diphthongs, like [aʊ̯] are characterized by not being stationary at all but through a transition that instead features relatively constant change.

[11] Of course, not all of the possible combinations will be realized.

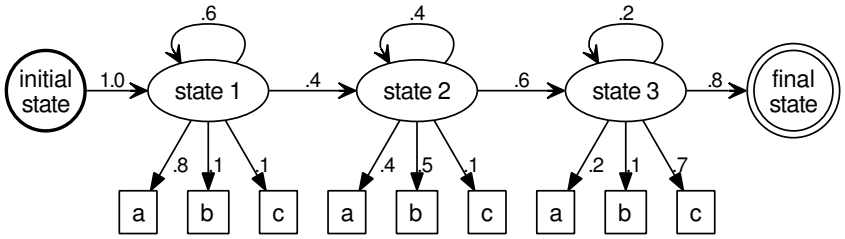


Figure 5.1: A discrete HMM with three emitting states (in addition an initial state, and a final state), and three discrete, emittable symbols (boxes a, b, and c) with varying emission probabilities for each state.

of observation parameters. This subsection now describes the speech recognition process proper, by first describing the stochastic modelling technique used in speech recognition, Hidden Markov Models, and then describing how the recognition process is operationalized in the decoding algorithm.

5.1.2.1 Hidden Markov Models

Formally, *Hidden Markov Models* are state-based stochastic, productive models (Rabiner and Juang 1993, Ch. 6.3.3) and are comprised of:

- a set $\mathcal{S}$ of discrete states that the model is in at any given time step including specific start and end states. Under the *Markov assumption*, the current state fully encapsulates the model's history. In practical LVCSR systems, each phoneme is modelled by three states: one for the transition from the preceding phoneme, one for the stable phase of the phoneme, and a third one for the transition to the next phoneme.
- a matrix $\mathcal{A}$ to describe state-transition probabilities between timesteps. Under the Markov assumption, $\mathcal{A}$ is independent of time and previously visited states. Most often, the topology of the HMM is limited to a left-to-right model where only self-transition (i. e. staying in the same state) or going to the next state is allowed, as in Figure 5.1. This limitation simplifies learning the transition probabilities from data and can be made because speech unfolds over time and does not normally reverse or jump around.
- a set $\mathcal{M}$ of observation symbols (for discrete HMMs), and
- an observation probability function $\mathcal{B} : \mathcal{S} \times \mathcal{M} \mapsto [0..1]$. For discrete HMMs, this can be described by a single matrix. For continuous-density HMMs, the probability distribution is modelled by a Gaussian distribution (or GMM) which

can be computed from (multidimensional) μ- and σ-matrices, as described in the previous subsection.

The denotation of HMMs as *hidden* stems from it being a two-level stochastic process with the higher level (the states) not being directly observable (i. e. hidden). Often, one observation may be emitted (or, seen in reverse, be accepted) by multiple states[12] and the most probable state for an observation can only be determined from the whole *sequence* of observations (Rabiner and Juang 1993, p. 326). This nicely models the context-dependent nature of the speech signal.

The HMM in Figure 5.1 does not leave any options: all states have to be passed in order to reach the final state. Such an HMM can be useful to find the best *alignment* of an observation to the given state sequence, and to calculate the overall probability of the observation being emitted by the HMM. Also, such HMMs are used to re-estimate emission and transition probabilities given an alignment: the iterative training process for HMMs re-aligns and re-estimates repeatedly until matching of the data does not improve anymore. Finally, a non-branching HMM can be used to generate an observation sequence based on a given state sequence (a capability that will be explained and put to work in Chapter 7.2.2).

The HMM in Figure 5.2, in contrast, contains a branch that allows two paths from the initial to final states and the most likely state sequence in this model depends on the observation. For example, the likelihood of the observation sequence aabba is more likely to be emitted by a path through the HMM that passes along state 2, while aaccc is much more likely to be emitted if state 3 is passed instead.[13]

For speech recognition, we build a much larger branching HMM that accepts all the word sequences possible in $\mathcal{L}$. An incoming observation sequence can then be matched against all possible paths in that HMM and the most likely path is the best estimate $\hat{\mathbf{W}}$ for the word sequence spoken. The details of an efficient algorithm for the matching task will be described in the next subsection.

HMMs for *large vocabulary continuous speech recognition* as described in Subsection 5.1.1 have a somewhat more complex structure than in Figure 5.2. Specifically, we distinguish between normal emitting and special *non-emitting* states. Groups of emitting states (usually three) relate to phonemes (or triphone contexts as described above) and are chained together (with transition probability 1.0) to form a word as specified by the pronunciation model. The language model is then used to combine these chains into the branching model and non-emitting states are used to model

[12] For continuous-density models employing Gaussian distributions, any observation can be emitted by any state, albeit with radically different probabilities.

[13] The best alignment for aabba via states 1 and 2 is 11222 with a probability of $1.0 \times .8 \times .5 \times .8 \times .3 \times .5 \times .4 \times .5 \times .4 \times .4 \times .6 = .0009216$, via states 1 and 3 the best alignment is 11113 with a probability of $1.0 \times .8 \times .5 \times .8 \times .5 \times .1 \times .5 \times .1 \times .2 \times .2 \times .6 = .0000192$. For aaccc the probabilities are .0000144 and .002107392 for the best alignments.

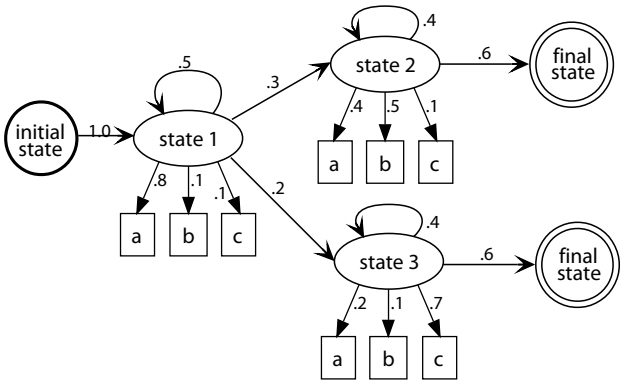


Figure 5.2: A branching HMM that emits different 'types' of observation sequences, such as aabba, or aaccc.

the language model probabilities via transition weights. Depending on the type of language model used and the search strategy employed (see below), these *word states* may either precede or succeed their phoneme states.

5.1.2.2 The Decoding Algorithm

Recalling the task formulated in Equation 5.5, we want to find the sequence of words from $\mathcal{L}$ with the highest probability, using the arg max function. A naïve implementation could build an HMM that completely describes $\mathcal{L}$, generate all possible alignments of the observation sequence to emitting HMM states, compute the respective probabilities (by multiplying transition and emission probabilities), pick the best alignment, and infer the words from the (non-emitting) states.

There are several problems with the naïve approach: the first is excessive recomputation as many of the alignments share common parts. Viterbi decoding avoids these recomputations through the use of dynamic programming (and also solves the problem of finding the best alignment along the way). In *dynamic programming*, all hypotheses (i. e. word sequences) are constructed in parallel, and step-by-step, using all the results from the last step for the next. In plain Viterbi decoding, given an observation of length T and an HMM with S states, we construct a matrix $\boldsymbol{v}$ of size $S \times T$. Additionally, a zeroth column is filled with all zeros, and a single 1 for the initial state. We then fill the matrix column-by-column, following the rule:

$$v[s,t] = \max_{s'=1}^{S} v[s',t-1] * a_{s',s} * b_s(o_t) \tag{5.8}$$

with a and b the transition and observation costs given by the HMM. Using the maximum, we discard all but the most likely path to a state under the rationale that the globally optimal alignment must also be locally optimal. Beyond the probabilities themselves, we store the state s' which the maximum function selected in each step in a separate *backpointer* matrix. Once the matrices are filled, we can trace back the optimal state sequence via the *backpointer* matrix, starting at the final state (Jurafsky and Martin 2009) with the highest probability of being reached.

We should note that the Viterbi algorithm does not in fact find the best word sequence but the word sequence that corresponds to the best alignment. These may in fact differ as the probability of a word sequence should be summed over *all possible* alignments which, however, are prohibitively expensive to compute. In most cases the best-matching alignment's probability dominates all other alignments' probabilities, making the Viterbi assumption useful. The use of the Viterbi algorithm is also the reason why adding multiple pronunciations to a word may in fact deteriorate recognition performance, as mentioned above.

The complete Viterbi algorithm is guaranteed to find the optimal state sequence but its runtime complexity is $O(S^2T)$. For large state spaces this quickly becomes infeasible. Additionally, much of the computation is concerned with sequences that are unlikely to turn out to be optimal as they are far less likely than the current best hypothesis. The token-pass algorithm can be used to re-conceptualize the search problem and allows easy implementation of a *beam-search* strategy, which at every time-step focuses on only the X best instead of all possible hypotheses.

In *token passing* (Young, Russell, and Thornton 1989), tokens are used as pointers onto states in the HMM network. Each token contains a back-pointer to it's predecessor and the associated hypothesis' *score*, possibly split into language model and acoustic model scores. Starting with one token in the initial state of the HMM, it generates tokens for all successor states (discarding all but the best token per state as in the Viterbi maximum selection). Tokens are then sorted by score and all but the best X tokens can be discarded, or all tokens with a score too far from the best (or both of these pruning rules can be applied). By discarding unlikely tokens, less computations are spent on unlikely hypotheses and memory requirements are reduced. After pruning, the algorithm returns to deriving next possible states and calculating state probabilities given the observed data frame. This loop continues until all frames have been consumed.

Additionally, token passing supports 'infinite' search spaces, by *dynamically constructing* the HMM network. Dynamic construction of the HMM network is especially important for cross-word triphone acoustic models, and for long-range (more than bigram) N-gram models (Odell et al. 1994) which would both increase static layouts by orders of magnitude. A technique for dynamic HMM network construction is used in the *LexTree* decoding approach (Odell et al. 1994). A lextree (also called 'trie') is a

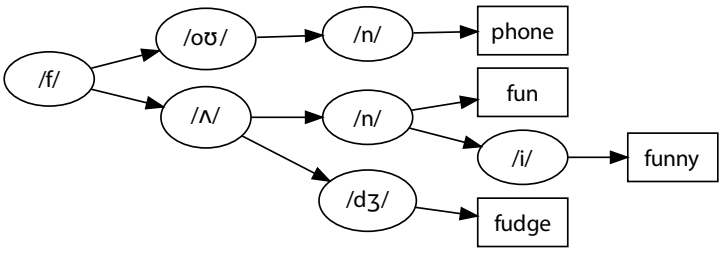


Figure 5.3: A pronunciation lextree for the words: phone, fun, funny, fudge.

method to store the ASR dictionary's pronunciations in a compact form by folding prefixes together and attaching the word labels as leaves. A small example lextree is shown in Figure 5.3. Lextree decoding starts with a copy of the lextree as initial HMM network and whenever a token reaches a word node, a new copy of the lextree is attached to it. Even though many copies are created, this is much more efficient than constructing the full static network ahead of time, especially because most paths will be pruned quickly. In lextree decoding, the word token must be attached to the end of the tree (once all corresponding phonemes have been passed). Thus, the word's language model score remains unavailable and is approximated by the maximum score of all words still accessible on the tree (slightly under-restricting the search space).

5.1.3 Evaluating Speech Recognition

Speech recognition aims to optimize $P(\hat{\mathbf{W}})$ and the easiest is to measure its success at doing so by counting the times that the utterance was correctly recognized. This is called *sentence error rate* (SER).

SER, however, ignores the fact that a speech recognizer's task is to recognize a *sequence* of words, and there may be different degrees of error in a sequence. In view of SER, being almost correct is as wrong as being totally wrong, whereas we would like to value partial correctness of the sequence.

In contrast, *word error rate* (WER) evaluates the proportion of correctly recognized words in the recognized sequence by counting the number of mis-recognized words

in terms of wrongly inserted words (#ins), substituted words (#subst), and omitted words (#del). It is computed as:

$$\text{WER} = \frac{\#\text{ins} + \#\text{subst} + \#\text{del}}{\#\text{words in transcript}} \times 100\,\% \tag{5.9}$$

Of course, as the recognizer may wrongly insert and delete more words than are actually in the transcript, WER is not bound to 100 %.

Two approaches could be taken for an even more detailed evaluation: (a) the confusion of similar sounding words can be punished less hard (as intuitively, the ASR's error is less grave), or (b) only those errors that really matter to the application are counted.

This latter approach is computed by the *concept error rate* (CER, Boros et al. 1996) which, instead of focusing on the inserted/deleted/substituted words only regards the *concepts* relevant to further processing. While more meaningful for a specific application at hand, CER cannot be computed in isolation without knowing what the recognition task is going to be, which counters the idea of a modular system.

5.1.4 Refinements

The token-pass algorithm does not only find the one best recognition result but the token list can also be used as an ordered list of next-best candidates. The decision making by the recognizer does not necessarily reflect the right quality criteria and in practice it does happen that more appropriate hypotheses are lower ranked than the top-ranked hypothesis (e. g. Rayner et al. 1994). Even though the lower candidates are considered less likely by the recognizer, the *n-best* list can be useful if additional information is considered and the list is *re-ranked* accordingly. N-best lists have been shown to be valuable in SDSs (in the non-incremental case) as they allow to revaluate hypotheses given higher-level information (Chotimongkol and Rudnicky 2001; Lee, Jung, and Lee 2008; Purver, Ratiu, and Cavedon 2006; Williams 2008).

Related to n-best list processing, *lattice generation* is the process of folding common parts of hypotheses in the n-best list which results in far less processing overhead when considering all hypotheses.

Confidence measures give an account of how confident an ASR is about its results, either on a per-utterance or even a per-word basis. Confidence measures can be computed from lattices in a second pass (which is somewhat incompatible with incremental processing), or be computed locally and frame-synchronously (Razik et al. 2008, 2011). Word confidences are not considered in the remainder of the thesis, even though they could offer additional value.

We limited our discussion of decoding algorithms to *one-pass* decoding, as the alternative, *multi-pass* decoding is of little use for incremental systems. Multi-pass decoding uses an initial imprecise recognition result to adapt following, more complex models (e. g. to speaker characteristics such as dialect), possibly repeating this process several times. This leads to better recognition results but only ever generates final results when all speech data has been processed several times. (However, see Section 5.2 which reports work by Imai et al. (2000) on incremental progressive multi-pass decoding.)

Standard ASR technology ignores *prosody*; this author has helped to demonstrate that prosody can be incorporated in the language model (following the intuition that a word's likelihood of being spoken correlates with the prosodic context) to improve ASR results (Ward, Vega, and Baumann 2012). However, incorporating prosody increases computational demand, as the language model's output is no more a priori information but has to be re-computed for every prosodic context. We hence ignore these results in the remainder of the thesis.

5.1.5 The Sphinx-4 Speech Recognizer

The *Sphinx* speech recognition engines are a series of open-source speech recognizers developed at *Carnegie Mellon University* since 1988 (Huang et al. 1992; Lee, Hon, and Reddy 1990; Placeway et al. 1997; Singh 2004). Apart from Sphinx-4, which we use, other still actively developed members in the Sphinx family are Sphinx-3 (the slowest but potentially most accurate recognizer), Pocketsphinx (based on Sphinx-2 and meant to be used on small devices where processing power is limited), and SphinxTrain (a program collection to train acoustic models for all members of the Sphinx family).

Sphinx-4 has been developed since 2003 (Lamere et al. 2003) in collaboration with *Sun Microsystems* who were interested in showing that the JAVA programming language could also be used to build computing-intensive research systems. (At the time, JAVA was still avoided in the research community, partly because the hotspot compiler was missing in the first JAVA releases and all code was interpreted.)

Being designed by 'real programmers' (as opposed to ingenious scientists) and with modularity, extensibility, configurability, and flexibility in mind, Sphinx-4 is the best-engineered piece of scientific software the present author has seen in his career. Sphinx-4 has had a great share in making the work presented in this thesis possible, and fun.

5.2 Incrementalizing Speech Recognition

The task of incremental speech recognition is to recognize words in speech as soon as they are spoken (or at least with as little delay as possible).

The token-pass algorithm forms the basis for our incremental speech recognizer. As developed in Subsection 5.1.2.2 above, it is a *time-synchronous* algorithm, that is, it builds all structure necessary for finding the speech recognition result while consuming its data. In other words, the input side of the algorithm is incremental as-is.

At runtime, the token-pass algorithm keeps a list of all partial results that may later be extended to become the overall best hypothesis (as determined by acoustic and language models). However, it is not trivial to determine the one partial result that will later become the prefix of the overall result. Different approaches have been used in the literature to determine these prefixes: Brown et al. (1982) searched the token history for a prefix that is common to *all* active tokens, that is, finds the 'undisputed' portion of the utterance so far. As these partial results are undisputed (also called 'immortal' by Selfridge et al. 2011), no additional errors can be introduced compared to non-incremental recognition, meeting the yieldingness criterion for incremental processing (cmp. Definition 3.3). However, the undisputed prefix may lag behind significantly (or even remain empty if the first word remains disputed until the end), resulting in long delays of this strategy.

Wachsmuth, Fink, and Sagerer (1998) evaluate the best-ranked hypothesis at time t only up to $t - \Delta$ (effectively leaving a fixed right context of Δ, cmp. Section 5.5) and considered the words contained in this prefix as final and unchangeable. Seward (2003) uses the same strategy for incremental phoneme recognition and find the results of the incremental strategy almost identical to non-incremental processing using a Δ of 150 ms. Imai et al. (2000); Imai et al. (2003) use a progressive two-pass setup to counter the restrictions of their simple, bigram-based first-pass decoder. In their setup, some portion of the first pass is unfolded into a lattice which is then passed to the second pass. If two consecutive hypotheses of the second pass match, then all but the M most recent words are considered as decided upon and passed on as output. These latter techniques are similar as their final output may differ from the non-incremental output (they are not *yielding*) and hurt recognition performance.

The approach taken here and first presented in (Baumann, Atterer, and Schlangen 2009) can be seen as an extension of the above approaches which however allows to change previously output hypotheses, by exploiting the flexibility introduced by the IU model. The *right context* technique in Section 5.5 is similar to (Wachsmuth, Fink, and Sagerer 1998), and the *smoothing* technique generalizes the idea of Imai et al. (2000) to consider the similarity of consecutive hypotheses. More recently, Selfridge et al. (2011) and McGraw and Gruenstein (2012) have extended the idea

of changing hypotheses, which holds a binary notion of correctness, to computing stability estimates for partial hypotheses.

5.2.1 The INPROTK Module for Incremental Rich Speech Recognition

The token-pass algorithm makes the current results available as a list of tokens after every frame, sorted by recognition score. Sphinx-4 makes it especially easy to acquire intermediate hypotheses, as there exists an interface (`ResultListener`) for exactly this purpose which can be configured to be called after every frame consumed by the decoder. When extracting the token for the 'best' hypothesis, there are two possibilities: (a) the token is in a final state of the HMM and hence specifies an hypothesis that does not require more speech to follow, or (b) the token is in a non-final state of the HMM and more speech must follow. For incremental recognition, all but the very last recognition result will later turn out to have been among non-final states. However, a major goal of incremental processing is to obtain information about the full utterance as soon as possible and recognition for the full utterance will end in a final state. Additionally, words only become known in the search graph when all the corresponding emitting states (phonemes) have been passed. Often, a final hypothesis with somewhat shortened phonemes for the last word will become available early, allowing the incremental recognizer to come up with an hypothesis for the word before the word is objectively over. Therefore, INPROTK uses the best token that is in a final state, and only backs off to the token in the best non-final state if no token in a final state is available. It should be noted that this choice does not have a tremendous impact as n-gram models (which INPROTK has predominantly used for speech recognition) tend to allow an utterance ending after any word (via LM smoothing). A detailed investigation of whether final or non-final state tokens lead to better results (and what the trade-offs are) has not yet been undertaken but would certainly be valuable, also because this choice is neglected in the related literature on incremental speech recognition cited above.

Once a best token has been selected, the list of words and phonemes along the token path can be (trivially) extracted and a word hypothesis (as a list of `WordIUs`) can be generated and words can be augmented with phoneme information (`SegmentIUs`) in their grounded-in links, as word and corresponding phoneme states are interleaved on the token path. However, the implementation is somewhat complicated by the fact that the ordering (and implemented class) of phoneme and word states in the search graph depends on the decoding strategy and type of language model (SLM or grammar) used for recognition. (For LexTree decoding with SLMs, phonemes precede their assigned words, whereas words precede their phonemes for grammar-based decoding.) INPROTK supports all orderings and hence all recognition modes of Sphinx-4 can be used incrementally.

Thus, at every time step, a best hypothesis (hyp_t) is available in its full form. Edit messages (which are required for inter-module communication in InproTK) are computed from these by the *diff* operation defined in Definition 3.14. The computational cost of deltification increases with hypothesis length and the implementation could be made more efficient by caching previous best tokens and their associated hypotheses and only trace back the token network until a previously matched token is found. However, utterances are relatively short in practice so this is only of theoretical interest.

Incremental processing in the IU model using incremental modules relies on edits between successive hypotheses and edits should come as rarely as possible to avoid re-computation by consecutive modules. Very often, consecutive hypotheses do not differ in words but only in the attribution of signal frames to phoneme segments. In these cases, the corresponding words are not changed (and no updates are sent via inter-module communication) but only the timings in associated `SegmentIUs` are updated.

In addition to features required for speech recognition, InproTK adds a pitch detector to the Sphinx-4 acoustic processing pipeline so that pitch marks are available as one of the tracks in the *BaseData* store (cmp. Chapter 4.1).

The iSR component of InproTK structures words into syllables by extending the Sphinx pronunciation dictionary to allow for syllable boundaries. During recognition, syllable boundaries are recovered from the dictionary and `SyllableIUs` are introduced as a separate layer between words and phonemes. (In other words, `WordIUs` only indirectly ground in `SegmentIUs`, via `SyllableIUs`). This functionality is meant to be useful (together with pitch information from the base data store) for prosodic analysis of words, e. g. to support lexical disambiguation of homophones, or for phrase boundary detection (cmp. Chapter 6.1).

To conclude, the raw iSR component of InproTK outputs hierarchical IU networks up to the word level with syllables and phoneme-level segments below. The speech recognizer's decoding algorithm and its search strategy are handled as a black box regarding their hypothesis generation. All the component requires is that the recognizer provides a best hypothesis (for some definition of best) after every time step, a requirement that is met by the token-pass algorithm. No optimizations for incremental output on the search *per se* have been performed. A more informed selection of best tokens, or a radically different search method (e. g. A* search, Kenny et al. 1991) are candidates for such endeavours.

The performance of the InproTK iSR component will be evaluated in Section 5.4, following the presentation of the evaluation tool for this purpose in the next section. As mentioned above, it is important to keep the amount of editing low, and Section 5.5 presents and evaluates the methods for edit reduction implemented in the iSR component.

5.3 INTELIDA: A Workbench for Evaluating iSR

This section presents INTELIDA, the workbench for Incremental timing evaluation of linguistic data implemented in the context of this thesis. INTELIDA is written in the PERL programming language and consists of a library to handle incremental and non-incremental timed data in several formats, two generic command-line programs for incremental and non-incremental data, respectively, and an interactive graphical user interface. Additionally, the author has used the INTELIDA library in many short scripts for corpus conversion, validation, extraction and measuring tasks.

5.3.1 The Library

The INTELIDA programming library defines classes that provide an object-oriented interface to labels, alignments of labels, and collections of alignments, as they are found in the data models of many annotation tools, and which can be used to non-incrementally describe linguistic corpus data. INTELIDA reads and writes non-incremental data in common formats (Praat's *TextGrid*s, Boersma 2002; as well as Wavesurfer alignments, Sjölander and Beskow 2000) to allow for easy exchange with external tools.

INTELIDA's main feature is support for incremental data, a feature that is not found in standard tools. The focus of INTELIDA is on the post-hoc analysis of incremental processing results rather than on the efficient handling of incrementally evolving hypotheses as in INPROTK. Correspondingly, the data models used differ between the two: temporal change between hypotheses (which is only implicit in the data model of INPROTK through changes to the IU network) is made explicit by representing and storing all hypotheses at every timestep and including their full timing information, resulting in a high degree of redundancy of information. In contrast to INPROTK and to simplify data storage, no links between related units are kept and units are not typed – relations between different kinds of data could only be recovered via comparison of timing information. This is not a shortcoming, however, as INTELIDA is meant mainly for the analysis of incremental data of one type at a time (mostly timing analyses) whereas the task of INPROTK is the production and manipulation of incremental data of multiple types simultaneously.

For incremental analysis, INTELIDA combines individual aligned hypotheses to *alignment sequences* that directly implement operations to calculate incremental evaluation metrics as presented in Chapter 3.3.2. Alignment sequences apply to one-best processing only and the computation of evaluation metrics for the n-best case differs as will be further detailed in Section 5.4.2 below. To support the analysis of n-best incremental data, INTELIDA supports *n-best alignment sequences* that combine sequences of n-best lists of alignments.

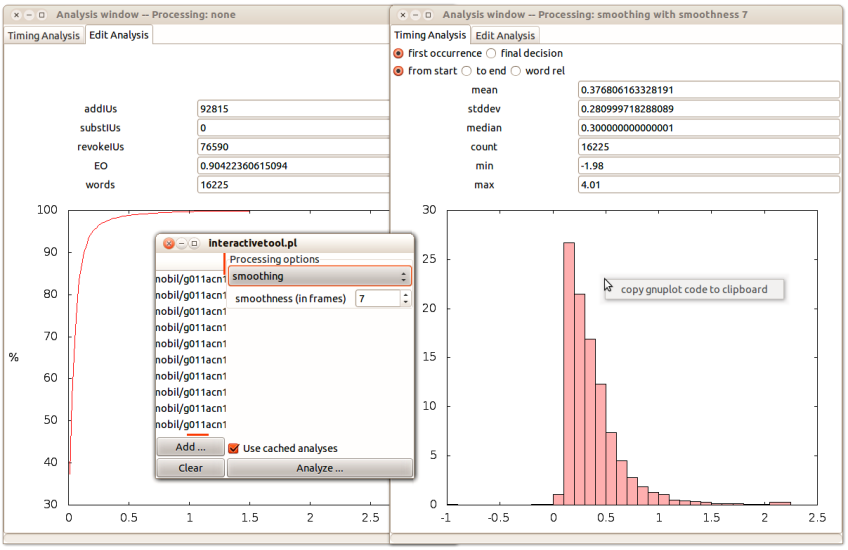


Figure 5.4: A screenshot showing the main INTELIDA evaluation window in the front and two analysis windows in the back.

Incremental data is stored in a simple, text-based format: sequences of alignments are delimited by empty lines; each alignment starts with a timestamp at which it is valid and the content of the alignment is stored in the same (tabulator-delimited) format as used by Wavesurfer. In the n-best case, multiple alignments share the same timestamp.

5.3.2 Interactive Tools

The INTELIDA interactive tools can be readily used to perform incremental performance evaluations for timed linguistic data, especially iSR output.

Graphical Evaluation Tool The graphical evaluation tool allows to evaluate performance metrics on a collection of files that each describe the incremental output of an incremental processor on some evaluation data. In the context of this thesis, the data to be analyzed was incremental speech recognition output from the INPROTK iSR component resulting from recognizing speech in a collection of audio files. For large collections, reading, parsing and pre-processing of input files can be extremely time-consuming; therefore, the tool supports a caching mechanism to speed up loading of previously analysed collections.

The analysis window has two tabs that relate to incremental performance, one presenting timing metrics and another for diachronic analysis. Both tabs are shown in Figure 5.4 (multiple analysis windows can be opened at a time). Graphs visualizing the performance metrics are plotted by calling *Gnuplot* (Janert 2009). The Gnuplot code used to plot the graph can be copied to the system clipboard by right-clicking the plot (as can be seen in Figure 5.4). This highly useful feature enables refinements and extensions to the plots via external editors and also helps to find small differences that cannot be exactly determined in the graphical representation. Most of the analyses of incremental behaviour presented in this thesis have been produced in this way, by using INTELIDA's graphical evaluation tool.

The tool also supports the incremental optimization methods outlined in Section 5.5 (in fact, they were first implemented in INTELIDA before being ported to INPROTK) which allows the easy exploration of their behaviour on a given corpus. For example, in Figure 5.4, the timing analysis is performed for smoothing with factor 7, as selected in the main window.

The GUI code of the tool is based on GTK and hence the graphical tool works out of the box on Linux only and is, unfortunately, hard to set up on other platforms.

Command-Line Interfaces Two command-line interface programs, `TGtool` and `INtool` for non-incremental corpus data and incremental evaluation data, respectively, complement INTELIDA's evaluation tool. Both support a variety of commands to load, inspect, organize, view, edit, and save input files in various formats, and feature a built-in help system.

Both can also be used offline in scripts, by passing (multiple) commands as (multiple) arguments on the command line, or by piping commands into it, which is helpful to rework collections of files. Finally, both programs are able to pass on data to TEDview, a tool for presenting incremental data that is described next.

TEDview TEDview (Malsburg, Baumann, and Schlangen 2009) is a graphical viewer for incremental data that was developed in parallel to INTELIDA but is not an integral or integrated part of the evaluation toolbox.

The additional value of TEDview over standard alignment viewers (such as Wavesurfer or Praat) is the capability of presenting incremental data. For each type of data (track) that is visualized, a full alignment sequence can be visualized by showing the alignment that is valid at a point in time. During playback, the currently visible alignment changes, always showing the state of affairs (e. g. the content encoded in the IU network) at the corresponding point in time. Thus, incrementally changing hypotheses and their interrelations can be easily inspected and understood, much

easier than by looking at alignment sequence files. Of course, non-incremental data can also be visualized.

5.4 Evaluation of Basic Incremental Speech Recognition

This section reports and discusses the incremental performance of the raw InproTK iSR component. We first describe the corpora used in the experiments and give results for a basic setting, discuss these, including possible variations of the metrics employed and compare between corpora. Subsection 5.4.1 explores the stability of results given a systematic variation of speech recognizer performance and Subsection 5.4.2 explores the influence of considering n-best speech recognition results.

Three corpora are used in the experiments: *OpenPento acted*, *OpenPento WOZ*, and *Verbmobil*. The first corpus is a small, relatively ad-hoc collection of utterances from the Pentomino domain,[14] mostly focusing on puzzle piece description and selection and resembling actual utterances from study subjects interacting in a Pentomino setting. The utterances were re-recorded by two speakers in an 'acted' way, trying to avoid the character of read speech and instead resulting in semi-spontaneously uttered speech (Baumann, Atterer, and Schlangen 2009).

The second corpus was collected in a Wizard-of-Oz setting of the Pentomino domain again focusing on puzzle piece description and selection and contains utterances from 9 different speakers (Baumann et al. 2009).

Finally, *Verbmobil* is a publically available dialogue corpus in an appointment scheduling domain (Jekat, Scheer, and Schultz 1997). For this much larger corpus, only a small portion was used for testing, as the speech recognizer's acoustic and language models were trained on the remainder of the corpus. The speech recognizer for the two OpenPento corpora used models trained on human-human dialogue in the Pentomino domain (Fernández et al. 2006), as well as acoustic data from the Kiel Corpora of Read and Spontaneous Speech (IPDS 1994).

For comparison, some corpus statistics are presented in Table 5.1. As can be seen in the table, corpora mainly differ in size and also in complexity, with *OpenPento acted* being the least and *Verbmobil* being the most complex. (Complexity is, among other factors, inherent in the vocabulary sizes.)

Table 5.1 also shows the speech recognizers' (non-incremental) word error rate (WER) which varies radically between the setups. The (low) WER for the *OpenPento*

[14] The Pentomino domain is a puzzle game with the 12 geometrical shapes that can be formed by attaching 5 squares by their edges (irrespective of symmetry or orientation). In the settings, humans discuss how to combine the pieces to form an elephant shape, which requires them to name puzzle pieces (a non-trivial task resulting in highly varying descriptions) and to describe target positions or required movements.

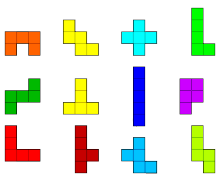


Table 5.1: Overview of corpus statistics of the three corpora used for incremental speech recognition evaluation.

	OpenPento (acted)	OpenPento (WOZ)	Verbmobil
utterances	85	255	1187
transcribed words	706	2364	17322
vocabulary size	179	335	1722
total audio duration	4'49"	28'43"	1:49'49"
avg. word duration (in s)*	0.378, 0.210, 0.335	0.395, 0.310, 0.320	0.299, 0.195, 0.240
word error rate (WER)	18.8 %	62.4 %	34.6 %

*averages are reported as means, standard deviations, and medians.

acted corpus can be explained by the fact that utterances were simple, very similar to utterances contained in the language model, and the fact that the speakers were known to the acoustic models. *OpenPento WOZ* shows that these recognition models are hurt badly by other speakers in actual human-computer interaction, which included different ways of interacting and describing puzzle pieces, as evidenced by the high error rate. Finally, WER for *Verbmobil* is moderate given the much broader vocabulary and the differing speakers between training and test sets. This is because the models used were trained on much more in-domain data than for the other two corpora.

In principle, the author would have preferred to present low word error rates for all corpora. However, the training of acoustic and language models for German dialogue speech from scratch was only one among the many tasks to be performed for this thesis. Finally, while low WER is certainly advantageous when building successful SDSs, the range of error rates spanned by the recognition results for the three corpus setups considered in this chapter are very useful for comparison purposes in the investigation of incremental performance metrics.

Table 5.2 summarizes the incremental performance metrics for the three corpus experiments. As explained in Chapter 3.3.2, `r-correctness` is determined by the proportion of incremental hypotheses that are identical (apart from label timing mismatches) to what should be known at the time the hypothesis is active, relative to the final recognition result. Likewise, `p-correctness` captures the proportions where the incremental hypothesis is a prefix of what would be `r-correct` (to account for the lag introduced by recognition).

The table shows that correctness is very stable across the three conditions, with the recognizer agreeing with its ultimate decision for this stretch of speech for roughly

Table 5.2: Overview of incremental performance metrics for raw incremental speech recognition.

	OpenPento (acted)	OpenPento (WOZ)	Verbmobil
r-correct	30.9 %	25.6 %	23.1 %
p-correct	53.1 %	55.0 %	54.2 %
edit overhead (EO)	90.5 %	90.7 %	90,2 %
$EO_{\oslash}$ (with substitution)	83.8 %	84.8 %	83.1 %
avg. FO*	0.275, 0.187, 0.230	0.332, 0.301, 0.260	0.274, 0.249, 0.210
avg. FD*	0.021, 0.299, –0.05	0.143, 0.492, 0.070	0.160, 0.418, 0.070

*averages are reported as means, standard deviations, and medians, in seconds.

25 % of the time. An additional 25-30 % of the time, the recognizer is lagging behind but it is not following wrong hypotheses. Finally, for the remainder, about 45 % of the time, the recognizer *is* following wrong hypotheses. Although some proportion may be spent in states where the current hypothesis is actually running ahead (i. e. a correct word is recognized even though it hasn't started yet according to the gold standard), the high EO makes this unlikely.

Edit overhead (EO) is a metric to capture the dynamics of the change between good and bad hypotheses: it counts the number of edits (adding a word: ⊕; or removing a word: ⊖) and gives the proportion of superfluous edits (i. e. edits that would not be necessary for an ideal recognizer). It thus complements `correctness`, which does not give any hint about the ordering of wrong hypotheses. (The ordering can be crucial, especially in 1-best processing, as it determines the amount of re-processing for the consumer of the incremental hypotheses.) As can be seen, EO is again stable between corpora at about 9/10, that is, only one out of 10 edits should be necessary.

A further analysis of the edits shows that out of the 9 superfluous edits, 8 are revoke/add pairs that could be combined to substitution edits (⊘). With substitution, the overall number of edits is reduced, resulting in an $EO_{\oslash}$ of about 5/6, with 2/3 being substitutions, 1/6 being bad single add or revoke edits, and the final 1/6 being good edits. Neither EO nor $EO_{\oslash}$, or the proportion of revoke/add pairs that can be represented by substitution change much between corpora; hence only EO will be considered in the remainder of the chapter, without any loss of information.

However, whether an architecture for incremental processing (like InproTK) implements substitution as a special case of revoke and add *does* make a difference if some of its processors can handle substitution with less effort than pairs of revoke and add. (For example, a simplistic parser might not need to act upon the substitution of

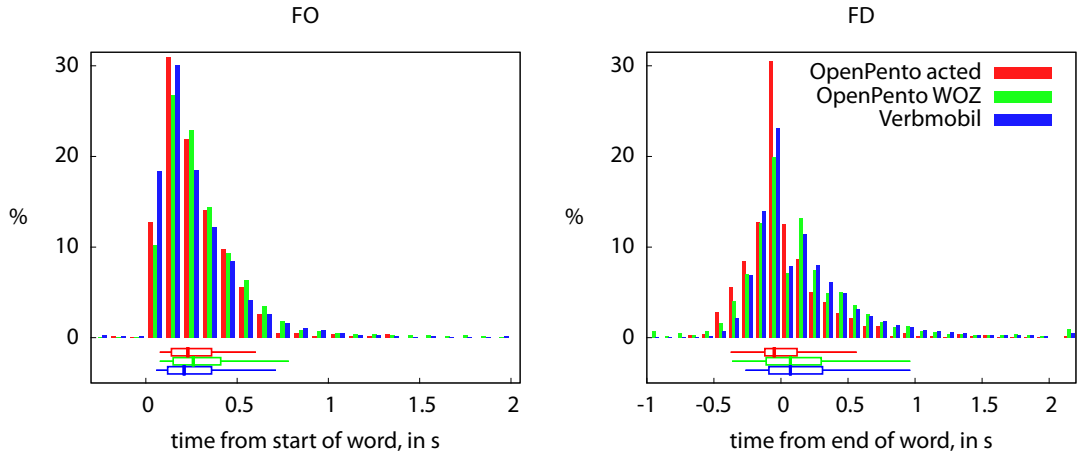


Figure 5.5: Histograms showing the distribution of FO (left) and FD (right) for the three corpora. At the bottom, box plots show the median, the range of the quartiles (box), and the 5 % and 95 % quantiles (whiskers).

a word if the part of speech does not change.) INPROTK currently does not provide substitution edits, favouring code simplicity over potential efficiency gains.

Where edit overhead is important mostly to estimate processing effort induced by hypothesis changes, timeliness metrics, to be discussed now, describe the possibilities opened up by incremental processing: *first occurrence* (FO) describes when the word (and its predecessors) first appears as part of the recognition, *final decision* (FD) describes when the recognition of a word does not change anymore.

On average, a word is first recognized some 300 ms after the word has started, and, on average, it is finally decided on some 150 ms after it has ended. Unlike correctness and edit overhead, timing metrics are *per word* and the resulting distribution of FO and FD timings are insufficiently described by a single number. Figure 5.5 shows histograms of the distributions, as well as box plots. As can be seen, the distributions (especially for FO) are positively skewed, and as a consequence mean and standard deviation are only of limited explanatory power, with median and quantiles being more expressive.

Again, the performance between the corpora is quite similar, with the simplest corpus (*OpenPento acted*) slightly outperforming the others. In fact, in this corpus, the median FD is negative, that is, more often than not a word is finally decided on even before it has been completely spoken. (As a side note, in all corpora the vast majority of words is first hypothesized before it is completely spoken, $\text{FO}_{word\,rel} < 1$, for

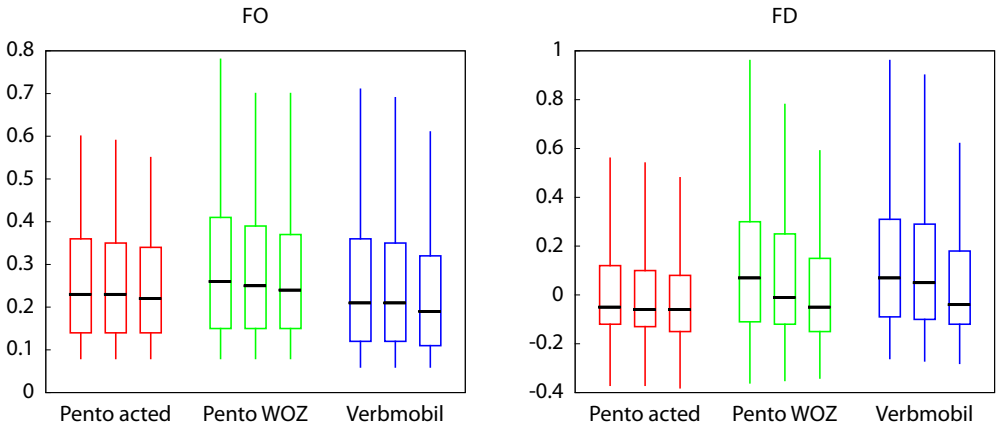


Figure 5.6: Box plots showing the influence of words considered in timing metrics when a transcript is available. For each metric and corpus a group of three box plots is shown: (a) all words recognized on the left, (b) both matches and substitutions relative to the transcript in the middle, (c) only words matching the transcript on the right. Box plots show median, quartiles and 5 %/95 % quantiles.

77 % to 87 % of all words. This finding will be used as a foundation of the application reported in Chapter 6.2.6.)

Chapter 3.3.1.1 argued that the final ASR output should be used as gold standard. However, one might be concerned that timing metrics differ systematically between correctly recognized and incorrectly recognized words. In the worst case, good timing could be rendered useless if timing were especially bad for correctly recognized words, and only wrong words (not spoken by the user) were recognized quickly. It turns out that the opposite is the case. When given a transcript for the loaded corpus, INTELIDA is able to limit F0/FD calculation to only those words that were correctly recognized (as determined by minimum edit distance matches), or to those words that either match or are substitutions, but leaving out inserted or deleted words (which potentially should be especially prone to timing errors).

Box plots summarizing the influence of ASR word errors on timing metric distributions are shown in Figure 5.6. For all corpora, the long end of the distribution is shortened considerably when ignoring wrongly recognized words in the computation and the effect is slightly larger for FD than for F0. The most extreme cases of delay before a recognized word becomes final (max(FD), not shown in the figure) are

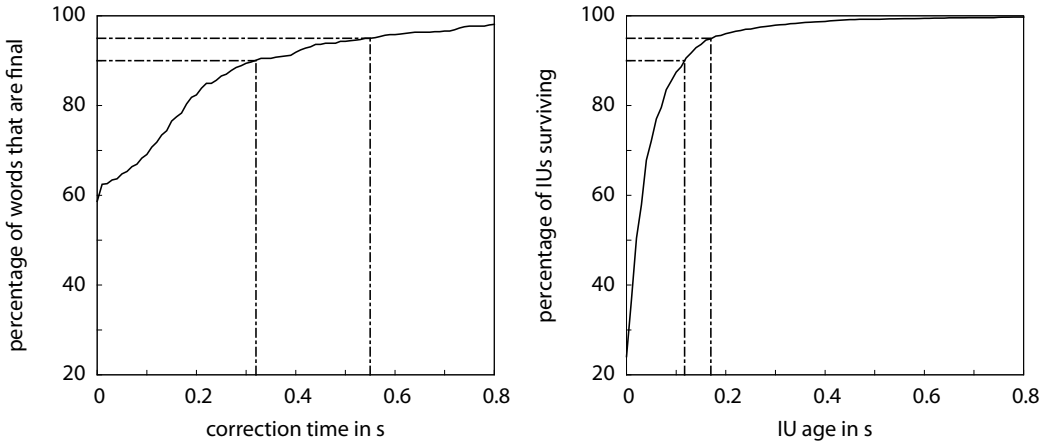


Figure 5.7: Correction time (left side) vs. IU survival time (right side) for the *OpenPento acted* corpus.

reduced by almost two seconds. (In other words: when the ASR takes very long to decide, chances are that the word is going to be mis-recognized anyway.)

Finally, `correction time` was proposed as a simple measure of stability based on `FO` and `FD`. The *correction time* of a word is the time span between first occurrence of the word and the final decision for it: `correction time` = `FD` − `FO`. Figure 5.7 (left side) plots the percentage of words with correction times equal to or lower than the time on the x-axis for *OpenPento acted*. The graph starts at an initial 58.6 % of words that were immediately correctly hypothesized, without further changes. The graph rises above 90 % for a correction time of 320 ms and above 95 % for 550 ms. Inversely this means that one can be certain to 90 % (or 95 %) that a current correct hypothesis about a word will not change anymore once it has been around for 320 ms (or 550 ms, respectively). There is, however, one flaw in this reasoning: at runtime, there is no way to know whether an hypothesis that has been around for 320 ms is a correct hypothesis, and the correction times have only been computed for correct hypotheses (as it is only for these words that `FO` and `FD` exist).

It is thus more meaningful to compute the ages reached by all hypothesized words directly, regardless of whether they later turn out to be part of the final hypothesis.[15] Figure 5.7 (right side) plots the percentage of hypothesized words that have survived (i. e. not been revoked) for the time on the x-axis for the same corpus. As can be clearly seen, the curve is much steeper, meaning that most wrongly hypothesized words 'die

[15]However, this computation is less straightforward than that for `correction time`.

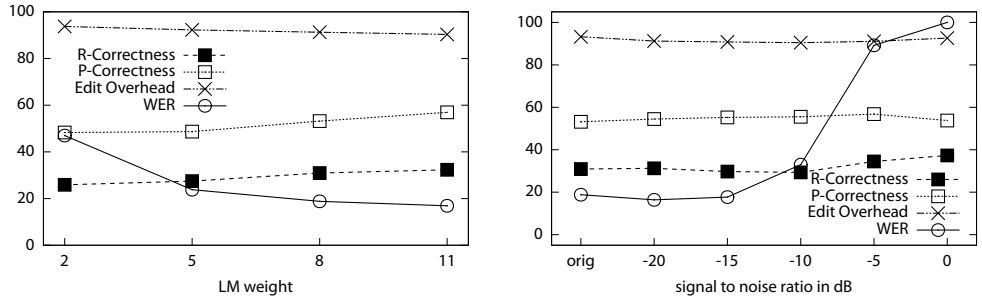


Figure 5.8: Stability against variations in LM weight (left) and with added random (white) noise to the signal (right).

young' (hypotheses that are not taken back anymore are assigned a correction time of 0 as above). More importantly, this curve, once measured for a certain setup, can be used at runtime to give a stability estimate (i. e. the probability of the IU not being revoked) for any word IU (regardless of whether correct or not) by reading off the stability likelihood for the IU's age. (This feature is not implemented in INPROTK but would be a trivial addition. However, further investigation would be necessary to find out whether it generalizes to more IU types beyond words.)

5.4.1 Variations of the Setup and Stability of Results

This subsection analyses the incremental performance of iSR under variations of the setup. While the variation between results for different corpora (and their corresponding setups) has been discussed above, the systematic variations performed here allow more detailed analyses of the influences of specific factors on iSR performance.

The smallest corpus used above (*OpenPento acted*) was analysed under systematic addition of white noise at different levels, and speech recognition was performed with varied language model weights. These parameters were chosen to test the influence of acoustic model and language model performance, respectively, on iSR. One hypothesis tested by a variation of the LM weight is that a lower language model influence results in lower stability of results because language model scores are completely independent of the signal and hence of hypothesis changes caused by signal variation.

Figure 5.8 plots `correctness`, EO, and WER for the different conditions. As can be seen, the metrics remain remarkably stable across the varied conditions. While not shown in the graphs, FO remains largely unchanged under varied LM weights, but FD reduces with higher weights, underpinning the hypothesis above that the lan-

guage model provides stability. Timing metrics do not change much when moderate amounts of noise are added (i. e. when a moderate mismatch between acoustic model and signal occurs). However, for very high noise levels (-5 dB and 0 dB signal-to-noise ratio, SNR) WER approaches 100 % and such an ASR cannot be expected to give meaningful results. As a consequence, timing metrics give erratic readings.

To conclude, the incremental evaluation metrics remain relatively stable across varying WER, indicating that they can be well used to independently describe the differing quality aspects of speech recognition overall (well represented by WER) and its incremental aspects.

The evaluations in this chapter were all performed on recognizers based on statistical language models (as no suitable grammars have been developed for the domain). The author's expectation is that grammar-based recognition – at least with small grammars – will lead to different incremental performance characteristics because best hypotheses can switch around for long subsequences of words, where SLMs would long have decided on one hypothesis. Selfridge et al. (2011), use both SLMs and rule/grammar-based language models, however evaluate them on different data sets, so no direct comparison can be found there, either.

5.4.2 N-Best Processing

In this section so far, only the behaviour of the iSR *current best* hypothesis has been looked at. This subsection, based on (Baumann et al. 2009), investigates how much could be gained from considering other (n-best) hypotheses that are considered by the token-pass algorithm but that are not ranked best at that point in time.

It was mentioned above that the n-best list may hold additional value, at least in the non-incremental case (Chotimongkol and Rudnicky 2001; Lee, Jung, and Lee 2008; Purver, Ratiu, and Cavedon 2006; Williams 2008). Preceding (Baumann et al. 2009) there was little work on n-best lists in incremental speech recognition: Miyazaki, Nakano, and Aikawa (2002) present an extension to a method for incremental NLU (Nakano et al. 1999) to use n-best lists and show some improvement in the NLU task, but apparently no evaluation of n-best iSR took place; it is difficult for the author though to evaluate the claims and to understand the details as the paper is available in Japanese only.

This section tries to evaluate whether n-best lists could also be advantageous regarding the dimensions of incremental performance outlined in Chapter 3 and already investigated in the present section. The analyses performed can help to decide whether using n-best lists in an incremental setting could *in principle* be advantageous.

N-best lists may be especially valuable for dealing with high speech recognition error rates (as they partially solve the problem of high error rates). For this reason,

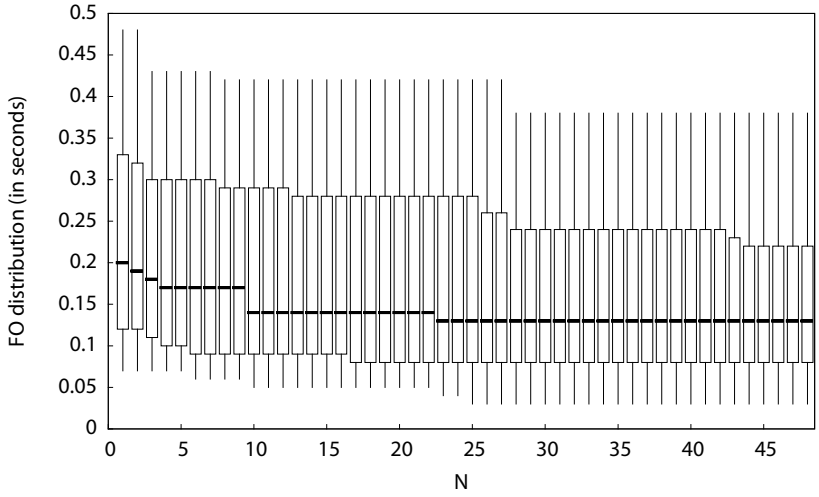


Figure 5.9: FO distribution when considering n-best hypotheses with varying *n* in the *OpenPento WOZ* corpus (limited to utterances with correct final result). Boxplot whiskers show 5 %/95 % quantiles.

the *OpenPento WOZ* corpus is used in the evaluations in this subsection, as it shows the highest error rates among the three corpora.

Baumann et al. (2009) focused the analysis of incremental n-best lists on the overall size of the lists (which vary in size due to relative beam pruning and were found to be considerably larger incrementally than non-incrementally), and on the positions of oracle/anti-oracle word error rates (i. e. the positions of the best and worst hypothesis in the n-best lists). It can be argued that diachronic metrics such as `edit overhead` do not apply equally well to n-best processing, as a system that is based on multiple incremental hypotheses at a time would certainly be organized substantially differently from the add/revoke scheme used in the IU model processors. However, Baumann et al. (2009) also left out the question of timing of results. This section makes up for that omission. The question is: how much earlier might results become available when considering n-best instead of one-best results? The focus will be on FO, as this more easily extends to the n-best case than FD (for which similar issues arise as for diachronic metrics). the definition of FO is straightforwardly extended to the n-best case by considering not only the best hypotheses (over time) when looking for the moment in which a word (and its predecessors) were first correctly recognized, but all hypotheses in the n-best list.

The analysis presented here is restricted to files without sentence error (i. e. files where the final result matches the gold standard). This avoids the problem of matching (many) hypotheses to final results that may themselves be faulty and is made possible by the fact that there is no fundamental difference between timing in files with errors and files without errors (cmp. Figure 5.6). There are 14 utterances which are correctly recognized at best hypothesis (containing 43 words) in the *OpenPento WOZ* corpus.

Figure 5.9 shows F0 distributions of the 43 words relative to the size of the n-best list considered. As can be seen, F0 decreases with larger n-best sizes. Chances are that a word does not enter the token list on the top rank, but that its probability increases over the course of a few frames; the larger n, the earlier it appears on the n-best list on its way to the top-ranking position. The largest improvements occur within the first few entries of the n-best list (as is the case for similarity measures in Baumann et al. 2009).

Of course, a data set of only 43 entries is too small to derive ultimate conclusions. It was mentioned above that the analysis can only help to find out whether n-best lists in an incremental setting could *in principle* be advantageous. N-best lists, however, also have the downside that they merely defer the decision for a 'best' hypothesis to a later stage. Thus, while F0 may be reduced with passing on n-best lists, it remains to be shown how to carry over this advantage to applications in practice. The results in this section indicate that the size of the n-best list need not be overly large as the largest gains are already achieved with a relatively small number of alternative hypotheses.

Overall, it appears that the performance gains (at least in terms of incremental performance metrics) of using n-best hypotheses are too small to justify implementing n-best processing within the incremental architecture. In the remainder of this work, only one-best hypotheses will be considered and a more thorough (re-)analysis of the merit of incremental n-best hypotheses, as well as an investigation to what extent techniques and architectures from the non-incremental n-best and the incremental one-best cases can be transferred, are left to future work. That future work may also have to come up with solutions to measuring a more general form of processing overhead for incremental n-best processing, as the approach of counting introduced and dropped hypotheses at every time step (as outlined by Baumann et al. 2009) appears to be overly simplistic.

5.5 Optimization of Incremental Speech Recognition

The previous sections reported results for iSR in a raw form, in the way that iSR is a by-product of the recognition process. This section presents possibilities of improving iSR results with regards to the metrics defined in Chapter 3.3.2 and as used in the previous section.

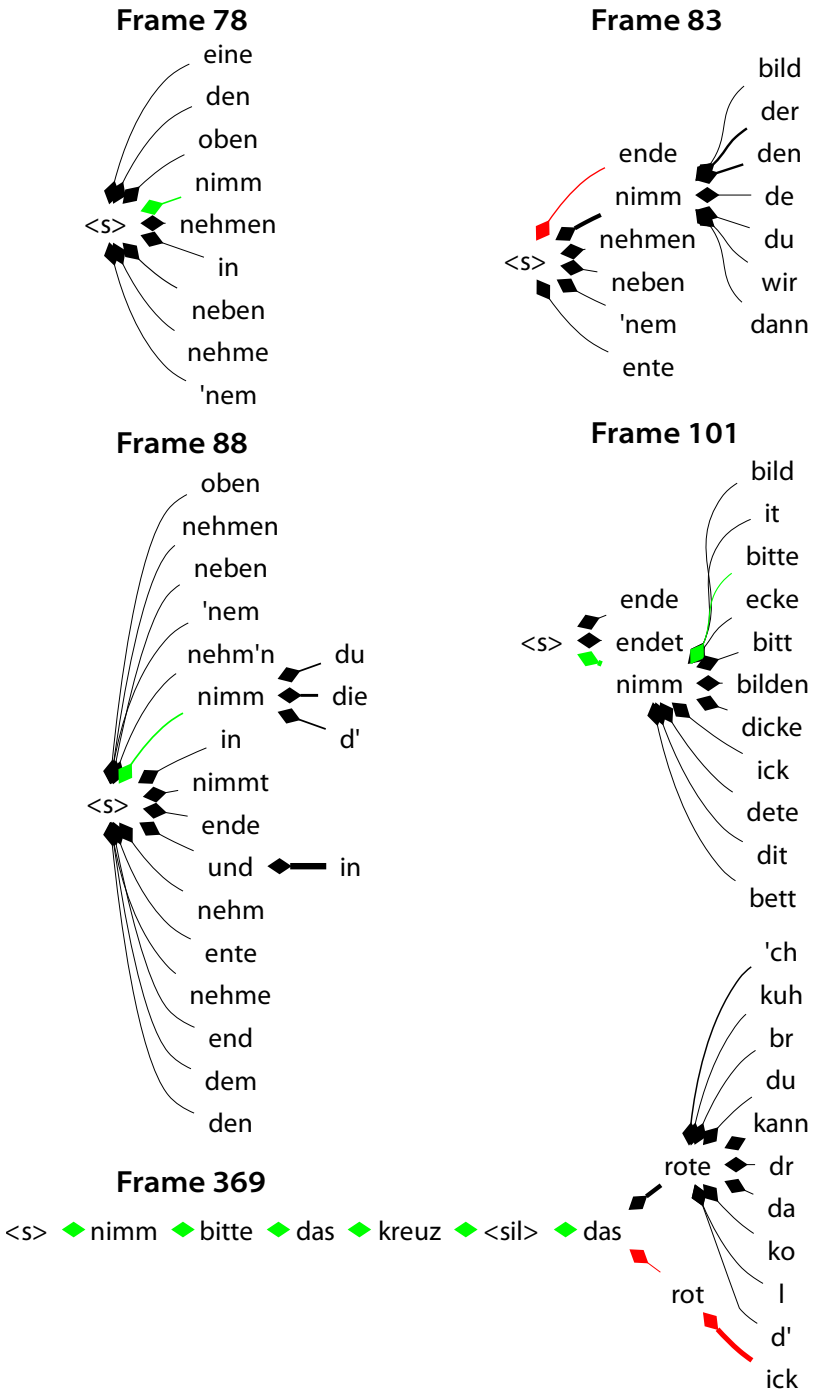


Figure 5.10: Five snapshots of the recognition lattice (i. e. folded n-best lists) at different points in time, while the author is saying “Nimm bitte das Kreuz – das rote [Kreuz – oben links – genau]”. The best-ranked hypothesis is colored – green if it is correct, red if it is incorrect.

It was already mentioned multiple times that there are trade-offs to be considered between the different aspects of incremental processing and hence improving performance in one metric may come at the cost of deteriorating other metrics. However, as the dependencies between metrics are not linear, and as some metric may be more important for a certain application than another, a relevant improvement is still possible.

The most important achievement of the techniques to be described is the reduction of `edit overhead`, as this was one of the major problems that came up in the evaluations. In the first implemented (sub-)systems developed using INPROTK (which integrated incremental speech recognition and incremental semantic chunking; Atterer, Baumann, and Schlangen 2009), the enormous amounts of re-processing induced by a high iSR `edit overhead` would have made the system far too slow for real-time applications, which highlighted the need for low `edit overhead` early on in the research.

Figure 5.10 shows the development of recognition hypotheses during incremental recognition. The sub-figures show the state of the recognition *lattice*, that is, a folded visualization of the recognition n-best lists at four (almost consecutive) points in time early on, and a fifth lattice in the middle of recognition. The lattice representation shows not only the current best hypothesis but also hypotheses that are similarly considered by the recognizer at that point in time. As can be seen, the recognizer intermittently changes the correct sub-hypothesis "`nimm`" (at Frame 78) to "`ende`" (at Frame 83) before later changing back (at Frame 88) and correctly extending the hypothesis with "`bitte`" (at Frame 101). Finally, the fifth lattice (at Frame 369) shown in the figure exemplifies that changes in the hypotheses are mostly occurring towards the right edge, that is, the youngest parts of the hypothesis. The first method to be presented builds on this observation. The lattices in the figure show only indirectly, how long the changes in hypotheses persist (e. g. the change from "`nimm`" to "`ende`" and back to "`nimm`" has occurred within 10 frames in the example). The second method will be based on such timings, that is, it exploit the way that hypotheses evolve.

Both methods described in this section do not alter the derivation of the raw iSR hypotheses from querying the token-pass algorithm's list of best-ranking tokens. Instead, we define ways of manipulating the (sequence of) output hypotheses using simple (incremental) post-processing or *filtering* techniques:

Definition 5.1. An *hypothesis filter* to improve an incremental processor may have an internal *state* $\mathcal{S} \in \mathbb{S}$ and maps an hypothesis (hyp^{in}) onto another hypothesis (hyp^{out}): $(\mathbb{S} \times \mathbb{H}) \longrightarrow (\mathbb{S} \times \mathbb{H})$. We again require $hyp^{out}_{t_{max}} = hyp^{in}_{t_{max}}$, that is, the final output of the mapping is identical to the original final output, maintaining the yieldingness criterion (see Definition 3.3).

Notice that the above definition does not impose any limits on the information that constitutes the state. Any information from the incoming hypotheses, and possibly other information may alter the state. The simplest filter for improving iSR that will be outlined and evaluated in the next subsection, uses a constant state which is not influenced by the incoming hypotheses.

5.5.1 Right Context

One of the problems of iSR (and incremental processing in general) is that if output is expected to be generated as soon as some input is available, often only the very beginning of input is available, on which some (often wrong) output is based, as can be seen in Figure 5.10. Furthermore, as new evidence is integrated, the hypothesis for the most recent frames have to be changed frequently, whereas the hypotheses for older parts of the input are already relatively stable and change less often. We call the frequent changes towards the most recent parts of the input *jitter*. Jitter is the reason for high edit overhead.

A simple strategy to avoid the jitter is hence to allow the processor some *right context* which it can use as input but for which it does not yet have to provide any output. By excluding the most recent part from the hypotheses, instability is reduced. Typical ASR systems use the right context strategy internally at word boundaries (with very short right contexts) in order to restrict the language model hypotheses to an acoustically plausible subset (Ortmanns and Ney 2000). For incremental speech recognition, Wachsmuth, Fink, and Sagerer (1998) first used this idea for early integration of speech recognition and parsing. However, in their architecture, hypotheses could not be changed later on, so that their strategy leads to a decrease in overall performance when compared to non-incremental processing.

In terms of the formalism of hypothesis filtering, the *right context* filter does not make use of a dynamic state; the state simply is a static parameter Δ, and is used as follows:

Definition 5.2. The *right context filter* with context size $\Delta \in \mathbb{N}$ is the mapping of $hyp_t^{in} = w_{1..k}$ to $hyp_t^{out} = w_{1..j}$ with $\forall i \in (1..j) : start(w_i) < t - \Delta$. That is, the right context method removes all words from the hypothesis that are "younger" than Δ according to hyp_t^{in}.

Figure 5.11 presents evaluation results with varied right context sizes for the *Open-Pento acted* corpus. The results for `edit overhead` and `p-correctness` show that Sphinx-4 uses an internal acoustic lookahead as proposed by (Ortmanns and Ney 2000) and described above to restrict language model choices. It is for this reason that the influence of the right context filter only sets in with $\Delta > 30$ ms. For larger Δ, the edit overhead decreases.

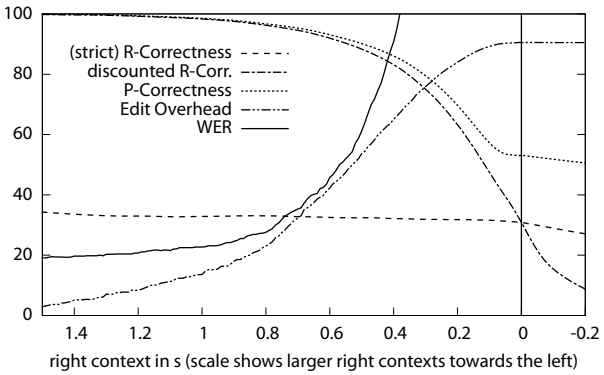


Figure 5.11: Correctness, edit overhead and fixed-WER for varying right contexts Δ in the *OpenPento acted* corpus.

Of course, allowing the use of a right context leads to the current hypothesis lagging behind the gold standard and using only information up to $t - \Delta$ reduces correctness (which expects words to match up to time t. To account for this lag (which is known in advance) a *discounted* r-correctness is also plotted in the graph, which limits the gold standard to be matched to $t - \Delta$. As can be seen in the figure, the right context method is effective in this discounted metric.

The figure does not show timing metrics, as the effect on timing is relatively straightforward: blocking 'new' words from the recognition increases delays (FO and FD) approximately with Δ.

To illustrate the effect of a system that does not support changing previous hypotheses but immediately commits itself, the *fixed* WER is also plotted, which would be reached by such a system when using a right context of Δ. As can be seen in the figure, it is extremely high for low right contexts (i. e. when committing almost immediately, exceeding 100 % for $\Delta \leq 400$ ms) and remains substantially higher than the non-incremental WER even for fairly large right contexts. The WER plot by Wachsmuth, Fink, and Sagerer (1998) looks very similar, highlighting the generality of this observation.

Finally, the concept of *right context* can be extended into negative Δ's: for negative values (plotted to the right of the vertical axis), the plot shows the performance of the iSR in *predicting the future* as it measures the correctness of the hypothesis in the near future. The graph shows that 15 % of hypotheses will still be (discounted r-)correct 100 ms in the future and 10 % will still be correct for 170 ms. Of course, this is a consequence of words often being first recognized before they are over, meaning that such hypotheses will remain r-correct for a certain amount of time. Neither the

right context filter nor speech recognition is able to predict words that have not yet started.

5.5.2 Hypothesis Smoothing

The right context filter could be described as the result of analyzing jitter in a static way: looking at incremental results such as the one depicted for Frame 369 in Figure 5.10 it is easy to draw the (correct) conclusion that jitter occurs at the right end of hypotheses and that skipping this part when producing incremental hypotheses will improve results.

This analysis, however, ignores the very important aspect of diachronic evolution of the incremental hypothesis. For example, a frequent problem in an error analysis of the right context filter was that words such as "`zwei`" are sometimes intermittently extended ("`zweite`", "`zweiter`") before the correct continuation ("`zwei drei`") is found (it could not be found before, because the vowel in "`drei`" had not been realized yet). The size of the right context must be chosen to be large enough that most of these changes are not output.

An analysis of the *dynamics* of hypotheses leads to a different strategy: intermittent mis-recognitions (such as "`zweite`", "`zweiter`") were observed to only last for a few consecutive recognition hypotheses. Exploiting this fact leads to edit filtering, which does not discard a fixed portion of hypotheses but filters parts of the hypothesis based on properties of the *edit* that introduces the change. That is, the smoothing method explicitly deals with edits to reduce edit overhead, avoiding the main flaw of the right context filter.

Definition 5.3. An *edit filter* is a hypothesis filter which stores hyp_{t-1}^{out} as part of its state $\mathcal{S}$, calculates the list of edits $\mathcal{E} = diff(hyp_{t-1}^{out}, hyp_t^{in})$ and applies some or all of these edits to its output depending on a *classify* operation and some set of features $\mathcal{F} \in \mathbb{F}$ which may be derived from the edit in question, the complete hypothesis, or anything else in the filter's state: $classify : \mathbb{F} \times \mathbb{E} \longrightarrow \{pass, \neg pass\}$. Notice that, to ensure applicability, an edit in $\mathcal{E}$ can only be applied if all its predecessors in the list are applied as well. In other words: testing whether to apply edits can stop once *classify* rejects an edit (or a later pass must override a preceding reject).

Simply put, iSR performance can be improved by applying only 'good' edits and inhibiting 'bad' edits. Suppressing an add edit for a wrong word (or a revoke edit of a word that would later turn out to be correct) improves performance. The difference between `correction time` and IU survival time, as can be seen in Figure 5.7 indicates that bad IUs die more quickly than good IUs, and correspondingly, a bad add edit will be revoked quickly. This is jitter. The simplest *classify* operation can hence be based purely on the time that an edit is valid:

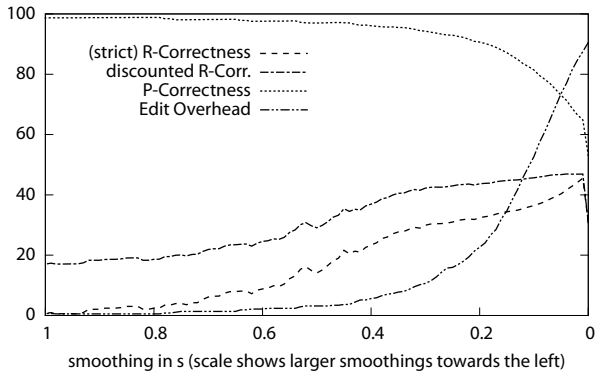


Figure 5.12: Correctness, edit overhead and fixed-WER for varying edit smoothing factors σ in the *OpenPento acted* corpus.

Definition 5.4. The *smoothing filter* is an edit filter with the state $\mathcal{S}$ caching the list of edits from $\mathcal{E}$ which have not been applied in the previous step(s), together with a count of how many times that edit has not been applied. *classify* returns *pass* iff the edit's count reaches a *smoothing factor* σ (which is a constant in the state), which causes the edit to be removed from the cache and *apply*'d to hyp^{out}.

It turns out that part of the method of Imai et al. (2000) is based on smoothing, in that words are considered only when they have shown up in two consecutive hypotheses (i. e. using a fixed smoothness of 2).

Figure 5.12 shows the result of hypothesis smoothing with different σ values, again for the *OpenPento acted* corpus. It shows that `edit overhead` falls rapidly, reaching 50 % (for each edit necessary, there is one superfluous edit, `E0` *parity*) with only 110 ms and 10 % with 320 ms. (The same thresholds are reached by the right context method at 530 ms and 1150 ms, respectively, as shown in Figure 5.11.) Likewise, prefix correctness improvements are better than for the right context method. The results for `r-correctness` are poor: this is due to correct hypotheses being held back for too long, especially when the hypothesis sequence is interspersed with wrong edits (which only last for few consecutive hypotheses but reset the counter after which edits are passed). This is especially true for larger σ and might indicate that a more intricate handling of smoothing counters may hold additional benefits.

We compare timing performance of the two methods at `E0` *parity*, that is, with an `E0` around 50 %. `F0` and `FD` distributions for right context with $\Delta = 530$ ms and smoothing with $\sigma = 11$ (corresponding to 110 ms of smoothing) are shown in Figure 5.13. As can be seen in the figure, smoothing leads to lower delays. One advantage of the

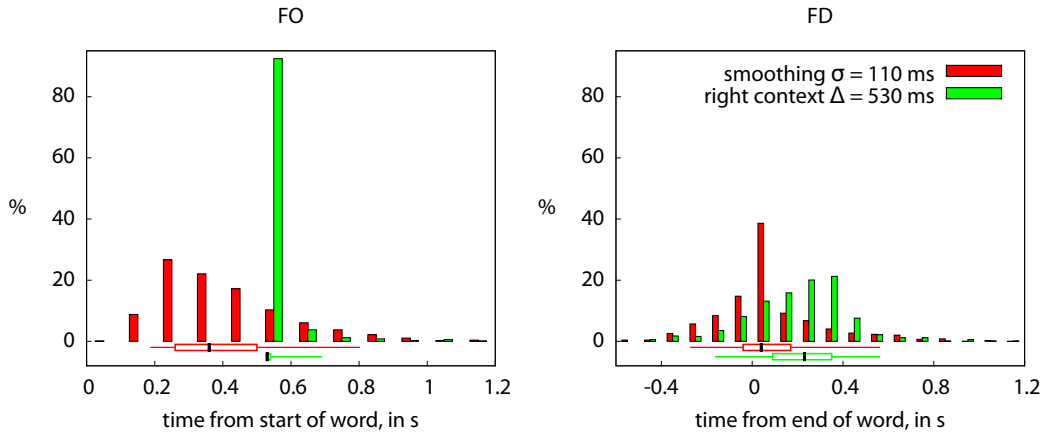


Figure 5.13: Distributions of FO (left) and FD (right) for right context and smoothing at the respective *edit overhead parities* in the *OpenPento acted* corpus.

right context method is the smaller variation, especially in FO. This smaller variation, however, comes at the price of much higher delays, with words only ever being output 530 ms after they started (at the earliest). Given a mean word duration of 378 ms (cmp. Table 5.1), this means that hardly ever is a word output before it has been completely spoken.

Notice that the effect of edit smoothing with fixed σ inhibits all IUs up to the age of σ. As mentioned towards the end of Section 5.4, the graph in Figure 5.7 (right side) could be used to give stability estimates for IUs. In fact, hypothesis smoothing is a form of binary stability estimator that suppresses all IUs until they reach the threshold age of σ.

5.5.3 Advanced Smoothing Methods

The edit filtering method from above relies on a central *classify* operation that determines when an edit should be passed. The smoothing method implemented in the previous subsection relied on age alone but many additional features could also be helpful.

As an example, Figure 5.14 plots the `correction times` of the ten most common words in a small corpus of user utterances in a command-and-control task (to be further described in the next section). As can be clearly seen, the correction times differ radically. The limited size of the corpus does not allow for significance estimation but the observable tendencies could be explained by a range of aspects, from phonetics

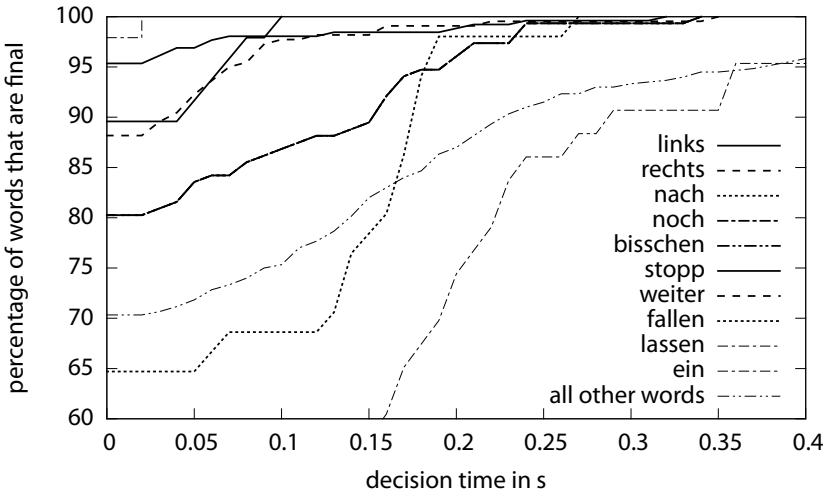


Figure 5.14: The `correction times` of the ten most common words in a small corpus of user utterances in an incremental command-and-control task.

(words sounding similar to other words), words having different lengths, to language modelling (words occurring in similar contexts). In any case, the word identity could be used by the *classify* operation to decide how quickly to pass on an edit.

Another range of features lies in the speech recognition process proper, for example by exploiting the range of other hypotheses in the speech recognition lattice. Finally, the recent history of edits, the type of edit and other aspects of the hypothesis and even the dialogue state could be used to determine whether an edit should be passed on or not by the *classify* operation.

Hamadeh (2012) trained RIPPER classifiers (Cohen 1995) for the edit filter approach using some of the features mentioned above, for the *OpenPento acted* corpus. The resulting classifiers hardly outperformed the standard smoothing approach based on age alone, and age was the main feature chosen by all trained decision trees. However, the experiments showed that it is beneficial pass on revoke edits unconditionally, without delay.

McGraw and Gruenstein (2012) showed that the very similar task of learning stability estimates can be successful, at least when using more advanced machine learning methods (the classifiers used by Hamadeh (2012) ignore the fact that the task requires sequence learning), more features and much more data. Still, the plots in (McGraw and Gruenstein 2012) show only a relatively small improvement over using the simple age feature alone.

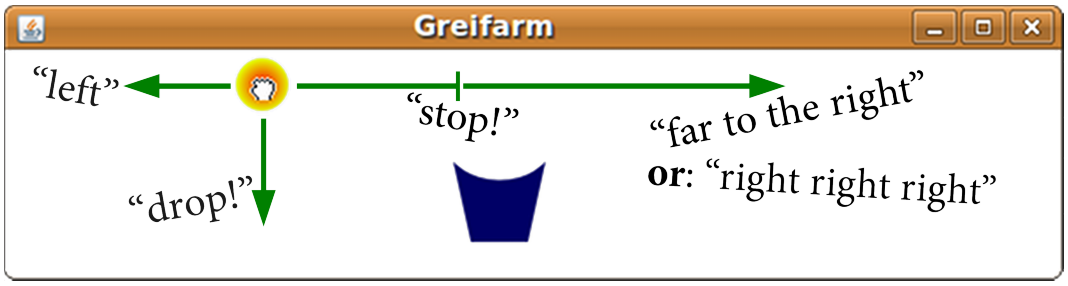


Figure 5.15: The *Greifarm* incremental command-and-control task with a few commands (in quotation marks) and their respective effects (shown as arrows). The goal of the game is to place the robot grappler above the waste bin and to then release the load.

5.6 Example Application: Incremental Command-and-Control

This section shows iSR in use in an application. For concreteness, the application is designed so that it highlights clearly the advantages of incremental speech recognition. The domain for the system, a command-and-control task, is presented in Subsection 5.6.1, and the setup and results of a Wizard-of-Oz experiment within the domain is detailed in Subsection 5.6.3.

Subsection 5.6.4 comments on the implementation of the full incremental command-and-control application based on InproTK, and Subsection 5.6.5 reports results of a small-scale evaluation of the system.

5.6.1 The *Greifarm* Domain

The domain of the example application is a speech-controlled, game-like command-and-control application. It follows the micro-domain principle (Edlund et al. 2008) in that it exposes only those aspects of dialogue interaction that the system is meant to resolve, and simplifies other problems that are outside the focus. The task for users is to order a simulated robot grappler to move above a recycle bin and to drop the load of the grappler (glowing radioactive waste) into the bin. More abstractly stated, the user controls a 1-dimensional motion plus a final commit signal that cannot be undone. A screen-shot of the user interface is shown in Figure 5.15.

In the domain, there are just four possible actions: moving to the RIGHT, moving to the LEFT, STOPPING an ongoing motion, and finally, DROPPING the load. (In the domain, people often further specify a move action with the help of modifier words

such as "little", or "far".) These four actions have highly different associated costs (in terms of error-costs and timing-costs): stopping the motion can always be undone (by continuing) and is most useful if executed as promptly as possible. (In fact, STOP is only useful if executed quickly, which differentiates this task from conventional command-and-control.) In contrast, dropping cannot be undone *at all*, and exposes the risk of dropping a load off target. Hence, a falsely executed STOP action can be assigned a very low cost, while falsely dropping has to be assigned a very high cost. Finally, wrongly moving in some direction can be undone, but will be distracting to the user, establishing an intermediary cost for wrongly executed actions.

Figure 5.15 shows some examples of user commands as they were uttered in the WOZ-experiment (see below). A (non-incremental) system that waits until the end of the utterance before starting to interpret and react will often perform worse in the domain. Especially, users in the domain often repeated commands ("stop, stop, stop!") to express urgency or to express the requested continuation of an action ("links, links"). However, a repeated stop command that takes longer before being executed is certainly counter-intuitive and a non-incremental system would never trigger continuation utterances, as the first action would only be started when the utterance was over.

While the scenario is very reduced, more complex interaction domains contain similar (possibly slightly more complex) positioning tasks as sub-problems. (In fact, a recent system for playing the Pentomino game successfully extends the approach presented here to steering a Pentomino piece to its target position; see Baumann et al. 2013.)

5.6.2 Cost-based Smoothing

The domain described above lends itself to a specific form of advanced smoothing, *cost-based* smoothing: in cost-based smoothing, the cost of misrecognition is taken into consideration for smoothing decisions. Instead of globally optimizing the timeliness/stability trade-off, costs for *classify* decisions are established and the goal is to minimize the costs over the course of the utterance:

- falsely adding a word is assigned a relatively high cost *that is specific to the word* (and, of course, specific to the application),
- deferring any $\oplus$-edit is assigned a small cost to encourage incremental output, resulting in good timeliness, and
- deferring any $\ominus$-edit might also be assigned some small cost, so as to increase hypothesis stability.

Cost-based smoothing hence takes into account high-level information into its smoothing decisions which must be provided either as examples or specified in rules. The *classify* operation could then, in principle, be learned by a machine-learning

Figure 5.16: Graphical wizard interface for the robot arm domain; stopping was realized as moving a tiny distance into the opposite direction.

algorithm for sequence learning. In practice however, this has shown to be difficult (Hamadeh 2012).

The simple domain of the robot arm allows for a simplistic form of cost-based smoothing that is based on some simple rules that also try to take the iSR characteristics of the different words in the domain into account[16]:

- words that result in a DROP semantics are called *stay-safe words* and assigned a high smoothing factor,
- words that result in a STOP semantics are called *urgent words* and assigned a very low smoothing factor,
- all other words are assigned a default smoothing factor.

The variable smoothing factors lead to a differentiation in the timeliness/stability trade-off that correspond to the domain requirements.

As an implementation detail, the variable smoothing counts lead to the smoothing queue not being ordered by counts anymore. When checking the queue, the case that an urgent word is preceded by a normal word must be accounted for. In this case, the implementation ignores the fact that the first word's counter has not yet run out and outputs this word (and the following urgent word) nonetheless.

5.6.3 Wizard-of-Oz Experiment

A Wizard-of-Oz study was performed in the domain to test the assumptions about user behaviour in the domain. 12 subjects participated in the data collection which resulted in 40 minutes of audio and 1500 transcribed words.

The linguistic data show the expected patterns, namely direction commands, often with modifiers for the desired strength of the motion, frequent STOP commands,

[16]Referring back to Figure 5.14: "`stopp`" has a very low average correction time, while "`fallen`" (a word relevant to the DROP action) has a relatively high correction time.

Table 5.3: Concepts derived from the collected data and generated by the iNLU component.

Actions	
concept	example words
LEFT	"links [left]"
RIGHT	"rechts [right]"
CONTINUE	"nochmal [again]"
REVERSE	"zurück [back]"
STOP	"halt [stop]"
DROP	"loslassen [release]"

Modifiers	
concept	example words
WEAK	"bisschen [a little]"
NORMAL	(default modifier)
STRONG	"weit [far]"
MAX	"ganz [to the very]"

and longer pauses before DROP commands, as users were aware of the fact that this action could not be made undone. (Additionally, and remaining unmodelled in the implementation described below, users frequently employed lengthening of direction words ("riiiight") to express distance.)

The analysis showed that the vast majority of user utterances can be described by six simple commands: four directional commands for left, right, continuation in the same direction, and reversal into the opposite direction, a stop and a drop command, which were realized with a relatively small vocabulary. In addition, the strength of directional commands was often adapted using modifier words. Absolute direction commands ("an den linken Rand", "zwei Zentimeter nach rechts") were rare and have been excluded from the command inventory. The action and modifier concepts are summarized in Table 5.3.

In the study, the wizard chose between moving (with three possible strengths), stopping and dropping, using the graphical interface shown in Figure 5.16 and without being able to see the user's visualization of the domain. The exact distance covered by the simulated robot arm was not fixed but also depended on some normally distributed random noise. This noise was introduced to simulate errors but participants noted that the behaviour seemed very human-like. For this reason, this non-deterministic behaviour was also adopted in the implemented system described next.

5.6.4 System Implementation

The system was implemented in InproTK, using custom-built iNLU and action handling components. As this is the first system described in the thesis, and as the system

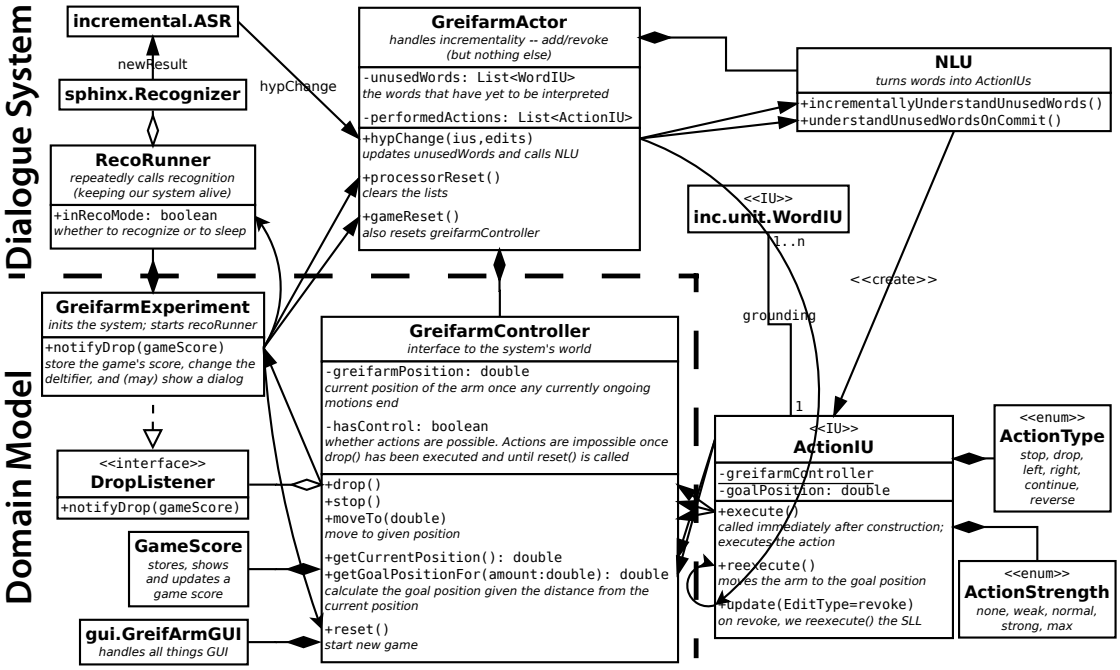


Figure 5.17: UML diagram showing the architecture of the *Greifarm* prototype and the differentiation between domain model and dialogue system proper.

implements *active* IUs, the description will be somewhat more elaborate than in later chapters. In addition, a full UML diagram of the system is shown in Figure 5.17.

The `GreifarmActor` receives incremental hypotheses from iSR, and keeps a list of `ActionIUs` performed so far, as well as a list of `WordIUs` not yet assigned to any action. In the case of revocation of a word that is already assigned to an action, the action is revoked, and its associated words are re-added to the list of unused words for future re-analysis. The core of the system are its iNLU and action management capabilities:

The iNLU greedily creates ActionIUs from the list of yet unprocessed WordIUs, based on a few simple rules that provide for actions and strength modifiers as shown in Table 5.3. The iNLU has to be prudent not to assign words to actions too quickly (for example, "`weiter`" could mean CONTINUE but also be the start of "`weiter rechts`", potentially indicating the opposite direction). A second, more eager pass is effected non-incrementally as soon as the user turn has ended.

ActionIUs initiate their effects autonomously, as explained below, thus there is no central dialogue management component; they are *active* in the sense of Chapter 4.2.2. There are several sub-types for the different types of actions. Implicit direction actions

(CONTINUE and REVERSE) autonomously traverse the IU network to identify their direction. Also, ActionIUs themselves are in charge of handling their own revocation by undoing their effects and re-installing the previous action's effects (except for the un-undoable DROP action). Also, the strength of (unmodified) direction actions may depend on the previous direction actions (this results in "`left left left`" becoming a maximally strong movement). A more traditional, flow-oriented system might use some sort of GrapplerIUModule to consume `ActionIU`s and centrally control the interaction with the domain.

ActionIUs (and only ActionIUs) interact with the `GreifarmController`, a logical model of the domain that is visualized by the `GreifarmGUI` (as in Figure 5.15).

In the system, the tasks of running iSR, handling incremental hypotheses, turning words into actions, executing actions, storing domain data, and visualizing the domain are thus realized in separate, modular components with clearly defined interfaces. As a result, implementing the incremental two-dimensional positioning engine presented in (Baumann et al. 2013) was a matter of replacing the domain controller and visualization components, and only slightly extending the NLU for the additional action types UP and DOWN.

5.6.5 Evaluation

A small-scale evaluation was performed to compare different iSR setups in three variants of the system:

basic incremental the system uses all intermediate results of the basic iSR without any optimizations,

standard smoothing the system uses iSR hypothesis smoothing as described in Section 5.5 with a smoothing time of 150 ms,

cost-based smoothing the system uses cost-based smoothing as described in Subsection 5.6.2, with a smoothing time of 0 for words with STOP semantics, 200 ms for words with DROP semantics, and 70 ms for all other words.

A first experiment was performed to test whether game performance would depend on the conditions. A game score was added to the game to which points are added if the waste hits the recycle bin, subtracted if the target is missed, and which is decreased by 1 every second (to account for task duration and to put a little pressure on the user).

The system was played by a single participant (not involved in the research), with the smoothing condition randomly changing after every 3 rounds. In every round, the start position of the robot and the position of the recycle bin where chosen randomly. The study was limited to a single participant in order to avoid the possibly confounding factor of interparticipant variation. A total of 72 games was played during the half hour that the participant used the system. A statistical analysis shows that the game

score depended on the condition (ANOVA, $F(2, 69) = 14.11$, $p < .005$) and Tukey's range test shows that the basic condition is significantly worse than the other two conditions. This means that iSR that employs a smoothing filter (and hence is able to smooth out false DROP actions) performs significantly better in the incremental command-and-control task than *raw* iSR.

Tukey's range test, however, also shows that there is no significant difference in game score between the standard and cost-based smoothing conditions. This may be for several reasons: firstly, generic smoothing time was chosen to be quite conservative, virtually eliminating false alarms for the DROP action in both conditions. Secondly, while the STOP action does feel snappier in the third condition, there is still frequently the need for corrective actions. The iNLU component allows for different strengths, so that correcting an overshoot in the second condition is similarly expensive (in terms of game score) as a smaller correction for the third condition. Finally, allowing three rounds per condition probably allowed the participant to adapt to the condition's delay for the STOP action.

Despite the fact that game score was similar for the two smoothing conditions, it was possible to consistently *measure* the timing difference: STOP *is* recognized earlier and DROP *is* recognized slightly more reliably in cost-based smoothing than in regular smoothing. A second experiment was hence conducted to test the absolute precision of the STOP action. The same player from above played for another 15 minutes, where the system was set to choose randomly among the two smoothing conditions for every round, and tweaking the setting so that always a large distance had to be covered between initial hand position and recycle bin (increasing the frequency of STOP actions). In this second experiment, there was still no significant difference in game score, but a Student's t-test confirms a significant difference ($t = 2.13$, $p < .05$) in the distances between where the STOP action was executed and the recycle bin position. This indicates that the cost-based smoothing method leads to objectively better positioning in the task. However, this better positioning does not necessarily result in higher task performance (as measured by the game score).

5.7 Summary and Discussion

This chapter evaluated whether incremental speech recognition is possible and whether it fares well in the relevant dimensions of incremental evaluation (timeliness, correctness, and diachronic evolution). Both these questions can be answered positively, with iSR providing words with little delay after they have been spoken (and often with a first guess while the word is still in progress). Additionally, incremental filtering techniques can be applied to reduce the otherwise intolerably high amount

of jitter (hypothesis changes that trigger reprocessing for consumers of incremental hypotheses) to a reasonable amount.

This chapter also investigated the applicability of the metrics developed in Chapter 3 and found their results to be stable across a range of variations of setups, and to similarly apply to the slightly different task of n-best processing. An example application showed that incremental speech recognition renders interactive command-and-control tasks possible that would be hard to achieve without it. (In fact, a much simpler command-and-control task based purely on incremental prosody processing was described by Soeda and Ward (2001); it was described as little intuitive to use and only allowed a very reduced form of interaction.) In the example application, incremental speech recognition allows for well-timed action execution in a highly interactive environment and this has recently been shown to carry over to more complex domains (Baumann et al. 2013).

The example application also showed that the smoothing method is effective to improve application performance as it reduces the jitter associated to iSR that can be detrimental if actions are performed on the basis of intermittent (and wrong) hypotheses. A simple form of cost-based smoothing that takes into account the costs of misrecognition was shown to result in more precise positionings, however could not significantly outperform standard smoothing in the application. It has previously been shown that dialogue is robust to errors (Schlangen and Fernández 2007) which explains why a better fine-positioning not necessarily leads to better game scores. The use of modifier words for corrective actions provided an *attractor* that prevented small positioning errors to have large effects. However, as outlined in Chapter 2.1.3, dialogue is non-linear. and small errors may as well result in large effects in other systems (e. g. if cost-based smoothing were further improved to often make corrective actions unneccessary).

More advanced demonstrations built on iSR will be presented in the next chapter and have been described elsewhere (Baumann and Schlangen 2011; Baumann et al. 2013; Buß, Baumann, and Schlangen 2010) and by others (McGraw and Gruenstein 2012; Selfridge et al. 2012a).

Recently, the slightly extended version of incremental hypothesis smoothing (using machine learned classifiers trained on large amounts of data to estimate hypothesis stability) by McGraw and Gruenstein (2012) has been integrated into Google Voice Search (Ian McGraw, p. c. at Interspeech 2012). Thus, optimized iSR is now available on any Android device and is used by millions every day. Comparisons to the market-leading intelligent personal assistant *Siri* (which runs non-incrementally) show the productivity gains that iSR makes possible.[17]

[17]For an example see http://vimeo.com/52497584.

The notion of stability of InproTK so far (and that of the IU model in general), is purely binary: IUs are committed or not. Stability changes gradually, so that sending update messages (e. g. to inform a consumer that an IU will not be revoked with a certain probability) would not be a good extension. Instead, WordIUs could be appended with an additional (real-valued) operation to indicate their stability estimate; consuming modules could then base decision on the stability, or might be able to register listeners that fire at certain stability levels. Furthermore, the current implementation ignores whether an IU is part of the 'undisputed' part of the search beam: commits only ever happen when recognition has finished. The method of Brown et al. (1982) should be added to allow for early commitment when it is guaranteed to honour the *yieldingness* criterion.

Razik et al. (2008, 2011) have worked on on-the-fly confidence measures, which, in a sense, are similar to stability estimation in incremental speech recognition. A thorough comparison of incremental stability and ASR confidence (including the incremental computation of ASR confidence) would be a valuable contribution for future work, especially when combined with incremental lattice/tree-based hypotheses.

The fact that considering n-best hypotheses does not fundamentally improve word timing highlights the limitation of the employed recognizer to only fully support the operation on full words at a time. It might be worthwhile to consider approaches to iSR based on different approaches to ASR instead. If, for example, sub-word units are available from ASR, these could be used to derive partial syntactic or semantic information even before the full word becomes available. Similarly, if high-level components 'understand' and license certain words (or phenomena) this could be fed back to the search process (as in Wachsmuth, Fink, and Sagerer 1998). As a further example, many users used lengthening ("`riiight`") in the positioning task in the example application (at least when interacting with the wizard). While the speech recognizer might have to be fundamentally re-organized to accommodate for this behaviour, handling this use-case would contribute to more natural interaction in highly interactive environments.

The speech recognition task is a special form of sequence classification (the sequence of audio frames is classified as one of the word sequences allowed by the language model). Recently, theoretical work on the reliable early classification of time series (Anderson, Parrish, and Gupta 2012; Anderson et al. 2012; Xing, Pei, and Yu 2009) has emerged, which, however, has not yet been considered in the analysis of iSR. Investigating whether incorporating this theory can improve iSR must be left as an opportunity for future work.

Finally, the problems that iSR faces are similar to other incremental handling of input modalities (such as gaze, gestures, etc.). It would be interesting whether properties are similar enough so that similar methods (such as hypothesis smoothing) would help these modalities as well.

Reproduced with kind permission by André POLOczek, www.polo-cartoon.de.

6 Short-Term Estimation of Dialogue Flow

The preceding Chapter 5 has shown that incremental speech recognition works: results are reasonably correct, produced reasonably early (often a word can be hypothesized while it is still ongoing), and if hypotheses are too unstable (i. e. change too often) for some task, optimization methods exist to improve the situation at the cost of some timeliness.

Chapter 2 argued that interaction in dialogue relies heavily on the timeliness of feedback exchanged between the interlocutors. The example application in Chapter 5.6 showed that iSR can help to realize direct feedback in the form of immediately performed system actions, resulting in highly interactive behaviour. However, the application shown did not deal with full dialogue: the system feedback was purely visual and the task was simple command and control. The focus was on reacting as early (and at the same time reliably) as possible.

However, immediate reaction is not desirable in all situations. For one, dialogue is organized in turns and only some non-disturbing feedback is licensed while the speaker's turn is ongoing (Clark 1996); secondly, some contributions may require precise timing when they co-occur during an ongoing turn (such as speaking in synchrony a word, co-completing an utterance, or – at least this is the author's impression – giving verbal or mimic feedback). It is evidenced by the turn-taking system, which is the major organizational principle in dialogue, that reacting as quickly as possible is often simply considered improper behaviour. Instead, it is necessary to find opportunities in the dialogue where reactions can be inserted. Conventional dialogue systems have the crude approach to rely solely on the fact that a user has stopped speaking for a certain amount of time to determine that the turn has ended, and only then start to output a system turn (which the system then outputs without being able to adapt its turn).

This chapter approaches turn-taking as a negotiation task that depends to a high degree on well timed behaviour. Post-hoc analyses of when a user turn *was* over are of little interest to an incremental dialogue system; instead, turn-taking needs to be managed incrementally and on the *sub turn* level and it is the prediction of *upcoming* turn-taking relevant events that the turn-management system must be concerned with. The turn-taking management system is not just required to operate in the vicinity of turn-changes but needs to continuously monitor whether turn-changes are occuring, and to initiate a turn-change at a desirable place in the user's speech if the system decides to *grab* the turn, To reflect the fact that turn-taking decisions

are never clear-cut but rather are considerations on a continuous scale, the chapter uses the term *floor tracking* to describe the task of estimating when a user turn is over, or when a user starts to speak, that is, who *owns the floor* and to what degree. Furthermore, this task will be extended to *dialogue flow* prediction, that is, not only determining upcoming floor changes but also the timing behaviour of other events, such as the timing of individual words spoken by the user.

Owing to the complexity of modelling turn-taking, there is a vast area of related work. Ferrer, Shriberg, and Stolcke (2002) repeatedly query decision trees to decide whether an ongoing pause should be considered the end of a turn and a combined turn-taking model of both user and system was introduced by Raux and Eskenazi (2009). Schlangen (2006) started the move from determining the end of a turn *post-hoc* to predicting it, and Atterer, Baumann, and Schlangen (2008) showed that *zero-lag* predictions are possible to detect the end of an utterance; Ward, Fuentes, and Vega (2010) present a model of turn-taking which estimates the remaining duration of a currently ongoing turn.

Most of the experiments cited above were conducted in offline experiments (Raux and Eskenazi 2009 being the notable exception), which do not directly show that better turn-taking results in better systems. In the system by Raux and Eskenazi (2009), the polling frequency (i. e. the frequency at which the system determines turn-changes) is five times a second. Thus, on average, a decision is deferred by 100 ms that is spent waiting for the system to re-determine turn-taking state. The work presented here will again use a centi-second frequency, as for updates to iSR results, and present the integration of dialog flow estimation into incremental spoken dialogue systems.

This chapter takes two steps towards more incrementally interacting spoken dialogue systems, firstly aiming to react to turn-taking opportunities sufficiently quickly to support a *collaboration on utterances*, and secondly extending from reaction to prediction, and from turn-taking to continuous monitoring to allow to speak *in synchrony* with the user.

Section 6.1 describes the InproTK floor tracking component which is used for (multi-modal) sub-turn level contributions to utterances. An example application in Subsection 6.1.2 shows how an incremental system that employs the floor tracking component in a multi-modal collaboration on utterance tasks outperforms a non-incremental baseline system.

Section 6.2 extends the estimation task in two dimensions: (a) from predicting the remaining time of contributions/turns (as in Ward, Fuentes, and Vega 2010) to determining the remaining times of all words spoken by the user, and (b) from reacting to an event as quickly as possible to acting synchronously to the event. An example application in Subsection 6.2.6 shows that predictions are sufficiently precise

and can be made sufficiently early to speak in synchrony a user's words. This shows that *end-to-end* incremental processing is possible without any noticeable delays.

6.1 Floor Tracking in InproTK

This section describes the floor tracking component that is part of InproTK and which was first introduced in (Buß, Baumann, and Schlangen 2010). There are several arguments in favour of establishing a separate component for the floor tracking task (instead of integrating this capability into one of the other components) which will be dealt with in turn:

The integration of a classifier for adaptive voice activity detection timeouts as in (Ferrer, Shriberg, and Stolcke 2002) or similar means into the iSR component could inform the dialogue manager (indirectly) via *commit* edits in the IU model. However, this only works when floor tracking events should always coincide with utterance endings. If floor tracking events should be registered within hesitations during utterances, it is vital to not interrupt speech recognition (which, at least in InproTK, would reset language model probabilities), even though words following a hesitation strongly depend on the words preceding it (and hesitations can even occur during words). Secondly, overloading commit edits with floor tracking semantics would prevent the iSR from revoking and changing hypotheses in the vicinity of hesitations where ASR errors appear to occur more often.

Potentially, the dialogue manager (DM) has the best overview over the dialogue and could additionally perform the floor tracking task, for example using classifiers as in (Raux and Eskenazi 2008, also for endpointing). However, incremental processing already introduces additional complexity into the dialogue manager and the ideal mode for incremental processing in the DM are yet to be found. Thus, a separate floor tracking module helps to keep the dialogue manager simple and can be re-used across different DM implementations.

Finally, the benefit of including other modules' data into the floor tracking decisions (as in Atterer, Baumann, and Schlangen 2008) indicates that floor tracking is in fact a task that is different enough from speech recognition and dialogue management that it warrants a separate module in the system (even though such advanced processing is not currently implemented).

The floor tracking component in InproTK indicates the user's speaking state based on user input (including prosody) and system state. The floor tracker informs components (i. e. the dialogue manager) about anticipated events using signals as described in the following subsection. An interactive system with multi-modal output that uses the component for collaborative utterance construction is afterwards described in Subsection 6.1.2.

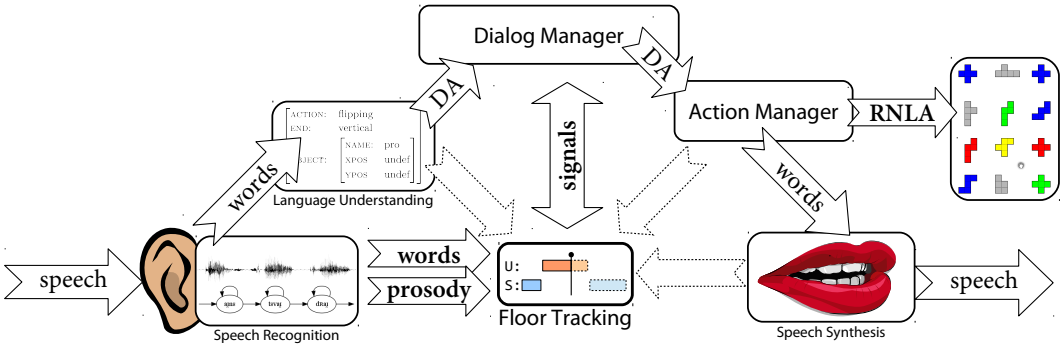


Figure 6.1: System architecture of the *Pentomino Select System*. The floor tracking component receives rich input from the recognizer (and potentially from other modules) and interacts with the dialogue manager through signals about current, upcoming and planned speaker changes. (Dashed arrows indicate possible information flow that is, however, not implemented in the current system.)

6.1.1 Architecture and Implementation

The floor tracking component takes a central position in the system architecture, as can be seen in Figure 6.1, making use of the fact that IU modules in INPROTK are not limited to pure pipelines but can form acyclic graphs. Thus, it is trivial to pass on iSR results to the floor tracker (and it would be easy to extend input to also cover e. g. understanding or output production). Communication with the DM uses *signals* to account for the fact that turn-taking *events* do not evolve incrementally and IUs would not be a fitting data structure. Additionally, communication using INPROTK's inter-module communication based on shared buffers (cmp. Section 4.2.1) would lead to cyclic (and bi-directional) module graphs, resulting in control-flow issues.

The DM informs the floor tracker about high-level expectations about the dialogue progression (e. g. user should start speaking soon, user should be interrupted if possible, system should not react to a barge-in, …) and based on these demands the floor tracker sends signals when the floor state changes accordingly. (For example, the user did not begin speaking within an expected time span). The task of determining *when* to take a decision (in a temporal sense) is thus completely factored out of the dialogue management component (by reducing it to a meaningful inventory of signals), greatly simplifying its implementation.

The floor tracker currently implements end-of-turn (or sub-turn) decisions and time-outs only, as these are the only requirements for the example application in the following subsection. However, the general architecture should be flexible enough to allow for additional decision making input and signalling capabilities.

The floor tracker internally implements several (customizable) rules that set up timers which, when they run out, send out corresponding signals to the DM. If a rule ceases to apply before the timer runs out, it is aborted. The central timer facility in the floor tracker complements nicely the otherwise purely event-based processing in INPROTK (which, in general, is not well suited to reacting to *no* event, such as a time-out, or delayed reactions). No-input time-outs can be registered directly by the DM when it expects the user to start speaking within a certain time-span.

End-of-turn rules are triggered whenever silence is incrementally recognized by iSR[1], that is, at the end of any *inter-pausal unit* (IPU). A simplistic decision rule is implemented that is meant to capture trial intonation based on the pitch track during the word preceding the silence (i. e. the potentially phrase-final word).[2] The implemented floor tracker supports two different timers, a shorter for rising pitch (to capture trial intonation quickly) and a longer timer for non-rising pitch (to not interrupt early on hesitations). The IPU-final word is queried for its pitch marks (which reside in the base data store associated to the word) and a linear regression of the fundamental frequency curve is calculated. The slope of this regression (in semitones per second) is used to classify the phrase-final intonation as rising/non-rising. To handle iSR *jitter* and to allow for iSR results as stable as possible, pitch analysis is only performed when the first timer runs out, thus considering and integrating all changes of the iSR result until then.

6.1.2 Example Application: Collaborating on Utterances

The floor tracking component was put to use by Buß, Baumann, and Schlangen (2010) in a system that collaborates with the user in developing their utterances. In human-human dialogue, utterances are often shaped not only by the speaker but also by the

[1] iSR is a relatively coarse method for silence detection and the acute reader may notice that timing evaluations in Chapter 5 explicitly excluded silence timings. In fact, analysis of the *OpenPento WOZ* corpus shows that F0 for silences is significantly ($p < .001$) later than for words; the median F0 for silences is 300 ms.

The final system presented below used a smoothing factor of 10, incurring an additional 100 ms delay. However, the smoothing delay is known beforehand, and the pause's duration can be subtracted from the timer. (Pauses have a median duration of 200 ms when they are *add*ed.) The bottom line is: timers should be set 200 ms shorter than the desired level and timers shorter than 200 ms may be late.

[2] This rule is only meant to approximate turn-taking and back-channel inviting cues which are far more varied (Gravano and Hirschberg 2009; Gravano and Hirschberg 2011), and should be covered in more depth (Atterer, Baumann, and Schlangen 2008) in full systems.

addressee: for example, the speaker of a referring expression may monitor whether the addressee appears to be understanding and adapt his utterance accordingly. One of the devices used for this are *try markers* (Sacks and Schegloff 1979), "questioning upward intonational contour[s], followed by a brief pause." Try markers are an efficient solution to the problem posed by uncertainty on the side of the speaker whether a reference is going to be understood (Clark 1996). This check for understanding works *in situ* within a tight feedback loop (Buß, Baumann, and Schlangen 2010).

Conventional SDSs cannot realize the close coupling that is necessary between input and output for utterance collaboration to work; especially, the different reactions based on understanding status (i. e. grounding at level 3 in terms of Clark 1996) require incremental semantic and pragmatic understanding. The example system presented here informs about its understanding immediately if this can be performed visually (similarly to Aist et al. 2007b) or verbally, using short feedback utterances/back-channels to evoke a continuation with the information necessary to solve the task, as soon as appropriate (i. e. when the user pauses). The delay for feedback depends on whether a try marker is recognized or not, similarly to (Skantze and Schlangen 2009), which, however, was limited to grounding at level 2 in terms of Clark (1996).

Of course, whether one sees the "collaboration on utterances" just described as the reactions during one utterance (that includes pauses), or as a quick progression of several utterances (or even turns) does not make any difference. The point lies in the differentiation of behaviour across intonation and degree of pragmatic completeness of the speaker's contribution and only shows that the term "turn" is somewhat blurred by these phenomena.

6.1.2.1 Domain and Setup

The domain of the system (as presented by Buß, Baumann, and Schlangen 2010) is the manipulation (selection/deletion) of Pentomino puzzle pieces. The user is shown a game canvas with several Pentomino pieces and a goal state where one piece is either selected or marked for deletion (as shown in Figure 6.2). The user's task is to order the system to select (or delete) the corresponding puzzle piece. The cursor changes its shape/moves to a corresponding piece as soon as the system has identified the action to be performed, or the referred puzzle piece, respectively.

Human-human recordings had been performed in a more complex variant of this domain (Fernández et al. 2006) and indicated the frequent use of "packaging" of instructions, immediate visual feedback, and back-channel feedback, often in connection with try markers (↗), as shown in Example 6.1 (adapted from Buß, Baumann, and Schlangen 2010):

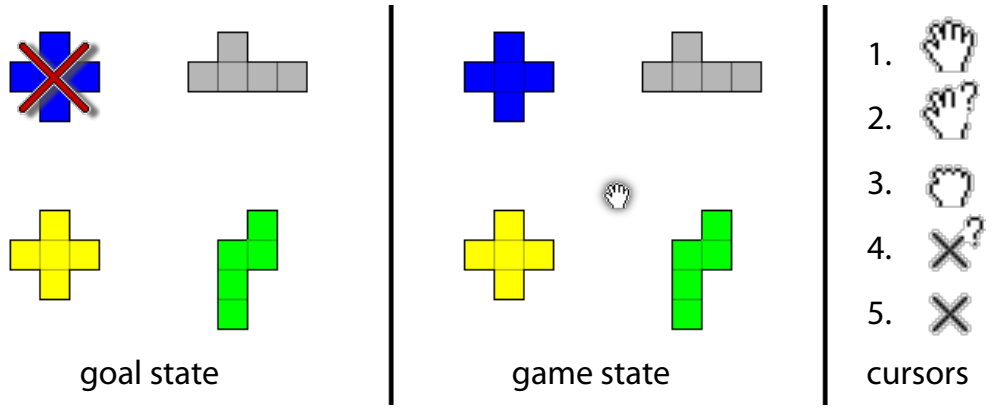


Figure 6.2: The domain of the Pentomino Select System. The user's task is to make the system change the game state (shown in the middle) to reflect the given goal (shown at the left). The system gives immediate visual feedback by changing the cursor on the game canvas; possible cursor states are shown on the right.

(6.1) IG-1: The cross in the corner↗
IF-2: hmm
IG-3: the blue one↗
IF-4: *moves cursor*
IG-5: yeah,
IG-6: delete that.

Another data gathering with simulated human-machine interactions was performed and confirmed the behaviour described above: instruction givers would often solicit feedback (e. g. using try markers) and instruction followers display their understanding by performing (sub-)actions required for the task.

Thus, the implemented system focuses on reacting quickly to intra-utterance hesitations (with specific handling of trial intonation), immediate execution of visual actions, and taking these into account when resolving displays of understanding. At the same time, the micro domain (in the sense of Edlund et al. 2008) is limited to a single dialogue state, and, aside from feedback utterances, only requires short opening and closing utterances. The settings (pieces shown on the board) in the experiments are designed to make complex and underspecified references likely (e. g. by distractors, such as the yellow cross in Figure 6.2).

The dialogue manager that takes decisions based on partial understanding results using an incrementalized question-under-discussion approach (Ginzburg 1996), the unification-based language understanding component that provides these from iSR, and the action manager that decides what should be uttered and what should be performed visually are not part of this thesis. They are described in detail in (Buß, Baumann, and Schlangen 2010; Buß and Schlangen 2010). Visual actions are always performed immediately (but the cursor takes a moment to reach puzzle pieces as its speed is limited); in contrast, verbal actions only occur after a timer in the floor tracker runs out. Timers were set in some informal trials. The time-out is 200 ms for pauses after rising pitch and for non-rising pitch the time-out is 500 ms (these reaction times are similar to the rules used by Skantze and Schlangen 2009).

6.1.2.2 Experiment and Results

Evaluating the contribution of one of the many modules in an SDS is difficult as system performance and user perception can be dominated by a single module performing badly (Walker et al. 1998). The system, at time of testing, suffered from bad ASR performance with some speakers, and occasional system instability (most likely related to concurrency issues which have since been resolved). To be able to focus the evaluation on the incremental dialogue strategies (immediate visual feedback, quick turn-taking when possible), an overhearer evaluation was performed, where participants rated the quality of the system based on observed interactions with the system (played to them as videos).[3]

A non-incremental system that reacted (visually and verbally) after a time-out of 800 ms (comparable to typical settings in commercial dialogue systems) was used as a baseline for the incremental system as described above.

Two players (familiar with the system internals) recorded 30 minutes of interactions with the two versions of the system (with systems being changed randomly). 10 % of the recordings were classified as "outlier" interactions, that is, interactions with technical problems, or severe speech recognition failures. The selection process was meant to be fair to both system versions and excluded similar numbers of interactions. Thus remained 51 interactions with the incremental and 46 with the non-incremental system.

The distribution of the durations of interactions for both systems is shown in Figure 6.3. The durations between the two conditions differed significantly (t-test, $p < .005$) with the incremental system resulting in interactions that were on average 3.5 s (or 35 %) shorter. The faster task duration for the system that collaborated in utterance construction was to be expected as the immediate reactions allow the system

[3] A similar approach was also used by Aist et al. (2007b) to test their incremental system.

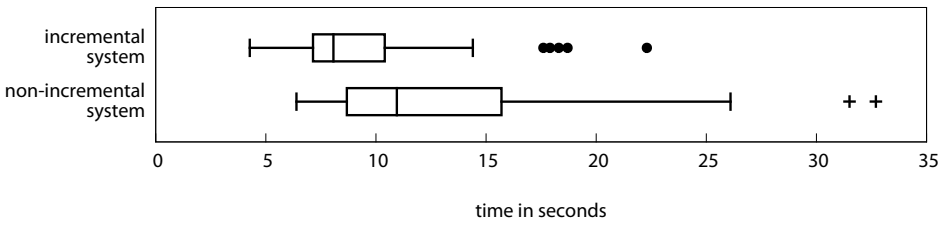


Figure 6.3: Box plots showing the task durations for the non-incremental system and incremental system (which collaborates on utterance construction).

to finish its actions more quickly (execution time is *folded* into the delivery time of the user utterance); faster behaviour for incremental systems vs. non-incremental systems has also been shown multiple times in the literature (e. g. Aist et al. 2007a; Skantze and Schlangen 2009). However, an incremental system that *interacts* differently than its non-incremental counterpart is by no means guaranteed to outperform it: the tight feedback loop of the incremental version could result in systematically more understanding/interaction problems between the system and user. Utterance collaboration and tight feedback do not seem to have a negative impact in the *Pentomino Select System*.

In the overhearer experiment, 8 participants (university students, not involved in the research) were presented a total of 36 of the interactions (chosen randomly and balanced between both conditions and speakers) as video recordings. The first two interactions were used for the participants to get to know the task and were not evaluated. The participants rated the interactions (directly after watching each video) for success (used as a control question), helpfulness, human-likeness, and reactivity on a seven-level Likert scale.

The control question (success of the interaction) was given consistent marks by all participants. (It turned out that one user had actually chosen the wrong piece, as compared to the goal state, in two videos; this was noticed by all participants in the evaluation.) Both human-likeness and reactivity were rated significantly higher for the incremental version (Wilcoxon rank-sum test, $p < .05$ and $p < .005$, respectively); the difference in the mean ratings was 0.59 and 0.69 scale points. The effect on helpfulness did not reach significance ($p = .06$) and the effect size was smaller (0.52 scale points).

6.1.3 Discussion

This section described the floor tracking component that is part of InproTK. Actions based on the quick determination of the end of a turn are relatively uninspiring from an an incremental systems developer's point of view: when the user stops speaking, any non-incremental system will come up with a result as well. Incremental processing can only help to take turn-taking decisions somewhat more quickly (e. g. Raux and Eskenazi 2008), but this does not change the interaction paradigm itself: the ping-pong game is only sped up, but it remains a ping-pong game.

Floor tracking is more interesting when used for the "collaboration on utterances" where the terms phrase, utterance, and turn meld, as they are *co-determined* by both speaker and listener. The floor tracker in InproTK tries to facilitate working with these less strict distinctions.

The floor tracker as currently implemented is in some respects very basic, as it simply uses two time-outs and one prosodic rule. Its strengths lie mainly in the integration into the modular incremental architecture. Also, it should be conceptually easy to extend it with more advanced notions of turn-taking management (e. g. Raux and Eskenazi 2009; Ward, Fuentes, and Vega 2010) to improve floor-tracking capabilities, as well as the model to be described in the next section which is currently not integrated into the floor tracker.

The example application goes beyond the systems by Aist et al. (2007b), which focused on visual feedback alone, and Skantze and Schlangen (2009), which focused on spoken feedback in a semantically empty domain, and shows incremental behaviour that was rated favourably and resulted in faster interactions, despite the simplicity of the implemented floor tracker.

One observation from the user interactions is that system reactions often occur 'too early' (at least in a technical sense): many words are recognized before the word has been completely spoken and (visual) reactions are started not in relation to the word's beginning or end (or stressed syllable or any other property of the word) but only in relation to when iSR hypothesized them. While this is acceptable for visual reactions (that do not need to be perfectly aligned to a speaker's words), this is probably insufficient for concurrent spoken output (e. g. feedback utterances that are plausible to occur even if the user does not pause and should be aligned to the speaker's rhythm). The next section concerns itself with timing estimation that can be used for the precise timing of system actions.

6.2 Micro-Timing Prediction

One aspect of incremental speech recognition is that words are often recognized before they have been completely spoken (cmp. Chapter 5) and the previous section

has raised the problem that actions grounded in such words may be executed too early. Specifically, the timing of the action depends on the timing of the recognition process, whereas ideally, it would depend on the timing of the word itself. This section, based on (Baumann and Schlangen 2011), introduces the task of *micro-timing prediction*, that is, predicting the temporal difference between when a word is first recognized and when it will actually end. This *end-of-word* (EoW) prediction generalizes the task of predicting the end-of-turn (EoT) that was treated in this chapter so far, to determining the timing of individual words while they are uttered by the speaker.

EoW prediction can even be extended to predicting the timing of upcoming speech, words that the user doesn't even have started to speak so far. This may seem extremely difficult, but humans master this task when they speak in synchrony, shadow, or co-complete someone else's turn. Speaking in synchrony was chosen in this section as a test case for small-scale micro-timing prediction. It can be seen as the holy grail of end-to-end, real-time incremental spoken dialogue, as synchronous speech requires that all timing delays in the system are balanced by corresponding predictive processing in other modules. This section sets out to show that real-time end-to-end incrementality is feasible for a spoken dialogue system (at least in some situations, and here again ignoring the challenges on 'higher' reasoning layers).

Subsection 6.2.1 motivates further the need for micro-timing prediction in highly-interactive systems and Subsection 6.2.2 discusses some work related to the co-verbalization task. Subsection 6.2.3 details how the micro-timing component fits into the overall system architecture and Subsection 6.2.4 details two prediction/estimation models for micro-timing. The estimators are evaluated in Subsection 6.2.5 and an example application is presented in Subsection 6.2.6 which shows that the system achieves the goal of real-time end-to-end incrementality.

6.2.1 Motivation for the Task

Turn co-completion, that is, speaking in synchrony a user's ongoing utterance, can be considered an ideal test-case of just-in-time incremental spoken language processing, as it requires that all levels of language understanding and production are carried out in real time, without any noticeable lags and with proper timing and even with the ability to predict what will come, to counterbalance inevitable internal delays.

The previous section has discussed that spoken dialogue systems, especially incremental ones, have come a long way towards reducing lags at turn changes (e. g. Buß, Baumann, and Schlangen 2010; Raux and Eskenazi 2009; Skantze and Schlangen 2009), or even predicting upcoming turn changes (Baumann 2008; Schlangen 2006; Ward, Fuentes, and Vega 2010).

Compared to regular turn changes, where short pauses or overlaps occur frequently (Weilhammer and Rabold 2003), turn completions in natural dialogues are

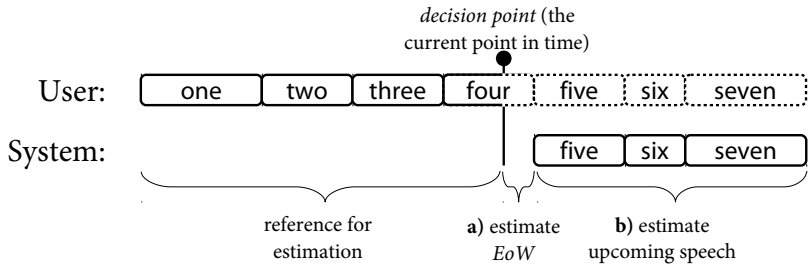


Figure 6.4: The task: When notified that the ongoing utterance should be completed with "five six seven" after the word "four", the first three words are used to (a) estimate the remaining duration of "four" and to (b) estimate the speech rate for the completion.

typically precisely aligned and prosodically highly integrated with the turn that is being completed (Local 2007). With ever more incremental (and hence quicker) spoken dialogue systems, the phenomenon of completion comes into reach for SDSs, and hence questions of micro-timing during the turn become important.

While completing someone else's turn – especially for a computer – may be considered impolite or even annoying, *being able* to do so can be a useful capability. Some tasks where it might be helpful are negotiation training to induce stress in a human trainee (DeVault, Sagae, and Traum 2009), or pronunciation aids for language learners, in which hard to pronounce words could be spoken simultaneously by the system.

Simultaneous speech occurs systematically, e. g. in greetings or when saying goodbye, without being perceived as a problem (Lerner 2002). These situations are important sub-tasks in deployed spoken dialogue systems, as they set the scene for a successful customer contact, and influence strongly the lasting impression of the interaction, respectively. Higher success and naturalness may help to improve not only the overall rating of the system, but can also help to improve the dialogue success itself. A system should certainly not try to complete all or even many user turns, but having the capability to do so means that the system has a very efficient interactional device at its disposal.

Another area where tight temporal integration may become important is simultaneous transcription or simultaneous interpreting (Rashid 2012). If translations are already available (possibly from a given script) the system must be prudent not to outrun the speaker that is being interpreted. Finally, monitoring the user's timing, as is required for the temporal prediction of turn continuations, can also be used for

other conversational tasks such as producing back-channels that are precisely aligned to the user's back-channel inviting cues (Gravano and Hirschberg 2009), to enable micro-alignment of turn-onsets, or to quickly react to deviations in the user's fluency. For example, if a system such as the one presented in the previous section notices a disfluency shortly after it initiated a (non-verbal) action, it might revert the action (if it induced that the action is wrong), or decide to not initiate actions during the user's speech (if it induced that this disturbs the user).

The micro-timing task is visualized in Figure 6.4: all the input that is available from the user up to the current point in time (the *decision point*) is available for the system to base its estimation on. There are then two sub-tasks that can be distinguished: (a) the micro-timing predictor should be able to estimate the time remaining in the currently ongoing word, and (b) the predictor should be able to estimate the speech rate at which the upcoming speech is going to be uttered. Combined with a component that predicts what the user is going to say, the system is able to speak in synchrony to the user as it can output the words that the user will speak, aligned to when the user will start speaking the words, and at the speech rate that the user will use.

6.2.2 Related Work on Simultaneous Speech

The general phenomenon of turn completion can be broken down into cases where the completion is spoken simultaneously with the original speaker (*turn sharing*, Lerner 2002) and where the floor changes in mid-utterance (*collaborative turn sequences*, Lerner 2004; or *split utterances*, Purver et al. 2009). For the present purposes, a differentiation between the two cases is not important, as both rely on the capacity to speak with a high degree of prosodic and temporal integration (Local 2007). In fact, it is beyond the system's control whether the other speaker will stop uttering, which would 'split' the utterance, or whether the turn is shared. The system developed below only deals with the question of when (and how) to start speaking and not the question of whether the current turn owner will stop speaking.

Lerner (2004) distinguishes turn *co-optation*, in which a listener joins in to come first and win the floor, and turn *co-completion*, in which the completion is produced in chorus. Both of these phenomena relate to the current speaker's speech: either to match it, or to beat it. The focus is on matching in this work, but the methods described similarly apply to co-optation.

As Lerner (2002) notes, attributing this view to Sacks, Schegloff, and Jefferson (1974), simultaneous speech in conversation is often treated exclusively as a turn taking problem in need of repair. This is exactly the point of view taken by current spoken dialogue systems, which avoid overlap and interpret all simultaneous speech as *barge-in*, regardless of content. However, Lerner (2002) also notes that simultaneous

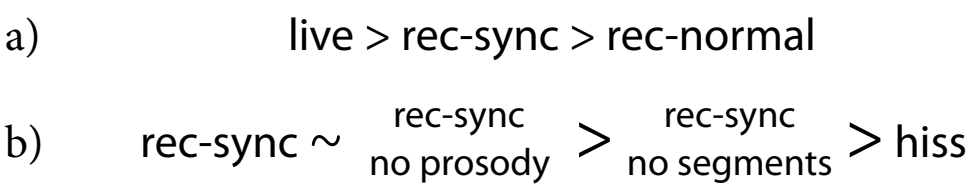


Figure 6.5: Ranking of factors influencing human synchronous speech performance for various conditions, as found by Cummins (2009).

speech systematically occurs without being perceived as a problem, e. g. in greetings, or when saying good bye (see also Clark 1996).

Two corpus studies are available that investigate split utterances and their frequency: Skuplik (1999) looked at *sentence cooperations* in the *Bielefeld Toy Plane Corpus* (BTPS), which is a corpus of task-oriented spoken German (Poesio and Rieser 2010), and find 3.4 % of such utterances.[4] Purver et al. (2009) find 2.8 % of utterance boundaries in the BNC (as annotated by Fernández and Ginzburg 2002) to meet their definition of utterances split between speakers. Thus, while the absolute frequency may seem low, the phenomenon does seem to occur consistently across different languages and corpora.

Local (2007) describes phonetic characteristics at utterance splits (he calls the phenomenon *turn co-construction*). He notices that they differ from regular turn handovers namely in their temporal alignment and close prosodic integration with the previous speaker's utterance. The work presented here focuses on the temporal aspects (both alignment and speech rate) when realizing turn completions, but leaves out pitch integration to future work. Full prosodic integration would likely be hard to evaluate numerically and would likely require more complex modelling than for the temporal alignment, as implemented below.

Cummins (2009) analyses speech read aloud by two subjects at the same time (which he calls *synchronous speech*) and presents an ordering of the influencing factors on the resultant degree of synchrony (measured by dynamic-time-warping, DTW costs). As shown in Figure 6.5 (a), synchrony is (slightly) better in a live setting than with a subject synchronizing to a recording of speech which was itself spoken

[4]The exact number of utterances split between speakers ("sentence cooperations") in the BTPS is not entirely clear: Skuplik (1999) as well as Poesio and Rieser (2010) report 126 sentence cooperations among 3675 contributions in that corpus (exclusive of non-verbal contributions), Poncin and Rieser (2006) report "at least 126" (Poncin and Rieser 2006, p. 737), and Poesio and Rieser (2004) give the number of 160 sentence cooperations (4.3 %), noting that "in most of them cooperation is other-initiated (95 %)" (Poesio and Rieser 2004, slide 8).

in synchrony and this is easier than to a recording of unconstrained speech. Cummins (2009) also experiments with reduced stimuli and the results are shown in Figure 6.5 (b): unmodified stimuli were synchronized to similarly as stimuli where intonation information was removed by re-synthesizing at a fixed fundamental frequency A carrier signal without segmental information (but including the original f_0-contour) fared worse than the above conditions, but was still synchronized to significantly better than when speaking to an uninformative hiss. (The first sentence of each recording was always left unmodified, allowing subjects to estimate speech rate even in the HISS condition.) Thus, pitch information does not seem necessary for the task. However, it may help in the absence of segmental information. As pitch information appears to be redundant to segmental information (at least for humans), the approaches presented below ignore pitch and rely on segmental information only.

Cummins (2002) found that synchronous speech is simplified in prosodic characteristics, presumably to make it easier to synchronize with. Additionally, there is no leader-follower relationship, but lead changes between speakers, indicating that synchrony between speakers is achieved through feedback being exchanged and not through one speaker locking in on the other. Finally, Cummins (2003) also found that human performance for synchronous speech does not improve with (moderate amounts of) practice.

The work presented in this section relies on 'higher-level' incremental components to form a meaningful turn completing application (or other application where synchronous speech actually improves an application) and such components are being developed: incremental understanding during a user's utterance is well underway (Heintze, Baumann, and Schlangen 2010; Sagae et al. 2009), as is decision making on whether full understanding of an utterance has been reached (DeVault, Sagae, and Traum 2009). Purver, Eshghi, and Hough (2011) even present an incremental semantics component aimed explicitly at split utterances.

In fact, DeVault, Sagae, and Traum (2009) provide exactly the counterpart to the completion timing component that is developed in this section, describing a method that, given the words of an ongoing utterance, decides when the point of maximum understanding has been reached and with what words this utterance is likely to finish. However, their system demonstration uses short silence time-outs to trigger system responses (Sagae, DeVault, and Traum 2010). The work presented here eliminates the need for such time-outs, and their work would be an ideal replacement for the 'dummy' continuation component that is used (and described) below.

Finally, Hirasawa et al. (1999) present a study where immediate, overlapping back-channel feedback from the system was found to be inferior to acknowledging information only after the user's turn. However, they disregarded the back-channels' micro-temporal alignment as explored in this study (presumably producing back-channels as early as possible), so their negative results cannot be taken as demonstrating a

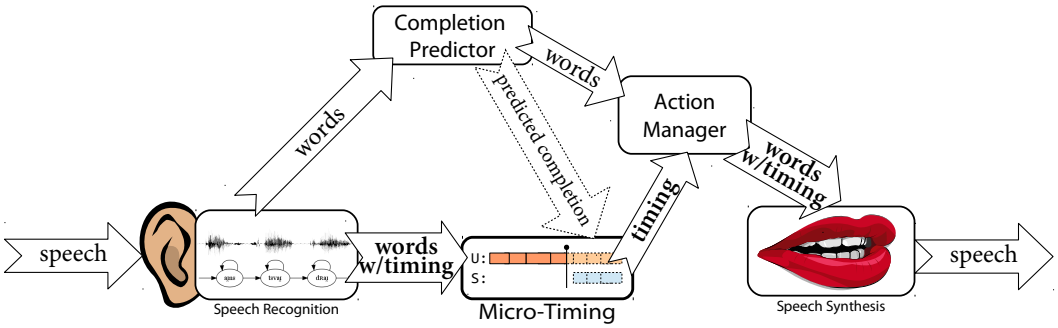


Figure 6.6: Data flow in the co-completion system. The micro-timing component calculates the EoW timing from the (rich) iSR hypothesis (and also from the predicted completion in the full version of the system). The predicted completion can then be timed to be spoken in synchrony with the user.

general shortcoming of the interactional strategy. Instead, they can just as well be seen as highlighting the need for precise micro-temporal alignment of back-channel feedback.

6.2.3 System Architecture

The overall architecture of the co-completion system is shown in Figure 6.6. Words from speech recognition are passed on to a completion predictor, which is a mock implementation of the component presented by DeVault, Sagae, and Traum (2009) in order not to duplicate their work: the *completion predictor* knows the full utterance to be spoken (from a transcript file) and aims to co-complete after *every* word spoken. Some constraints were built into the prediction module that are meant to be representative of real implementations of such a module: it can only find the right completion if the previous two words are recognized correctly and the overall WER is lower than 10 %. (Coming back to Figure 6.4, if the system had falsely recognized "`on two three`" instead of the correct "`one two three`", no completion would take place: even though the last two words "`two three`" were recognized correctly, the WER between "`on two three`" and "`one two three`" is too high.)

The central micro-timing component estimates the timing of the user's words. The component is triggered into action when an iSR result arrives and the understanding module signals that (and with which words) a turn should be completed. At this *decision point*, the micro-timing component estimates (a) *when* the current word

ends (EoW) and (b) *how* the user's speech will continue, as was already shown in Figure 6.4.

Both of these aspects of the user's speech could be put to use in a variety of ways in the action manager of the incremental spoken dialogue system. In the co-completion system, the goal is to output the same words at the same time that the user will speak them. Ideally, the system will start speaking the continuation precisely when the next word starts and match the user's speech as best as possible. Thus, the component must estimate the time between decision point and ideal onset (which will be called *holding time*) and the user's *speech rate* during the following words.

In order for the system to be able to produce a continuation ("`five six seven`" in Figure 6.4) in time, of course the decision point must come sufficiently early (i. e. during "`four`") to allow for a completion to be ready in due time. This important precondition must be met by-and-large by the employed iSR. However, it is not a strict requirement as the timing component's estimated holding time should be negative if iSR results are lagging behind. Depending on the estimated lag, a completion can be suppressed or, if it is small, fairly good completions can still be realized by shortening the first phoneme, or backing off to setting in one or more phones or syllables later (actually, back off until the holding time turns positive). While initial shortening/back-off works for co-completion, it would probably be less adequate for feedback utterances for which the onset timing may be critical.

6.2.4 Two Models for Micro-Timing

At the core of the micro-timing component is a duration model that is able to estimate the timing of a word that is currently ongoing (or even for words that are about to be spoken). Two strategies for the timing module have been implemented and will be described in turn, after first discussing a simple baseline approach.

Baseline: Speak Immediately In the absence of a timing component, a system will start speaking right away, whenever iSR recognizes a word. Thus, for the task of synchronous completions, completions will often already be output while the user is still speaking the preceding word. This seems to have been the strategy taken by Hirasawa et al. (1999) for outputting back-channel utterances and the evaluation in the next subsection will show that it is not very good.

Of course, performance of the baseline depends on the timeliness of the iSR, specifically its $\texttt{FO}_{word\,end}$. It was argued in Chapter 3 that iSR should be able to produce a first hypothesis about a word before it ends (but may require to change that hypothesis). As a result, the better the iSR, the smaller (possibly below zero) $\texttt{FO}_{word\,end}$ will be (which is also indicated by Table 5.2 in Chapter 5). Thus, a better iSR developed in the future will make the baseline strategy even worse than it is today.

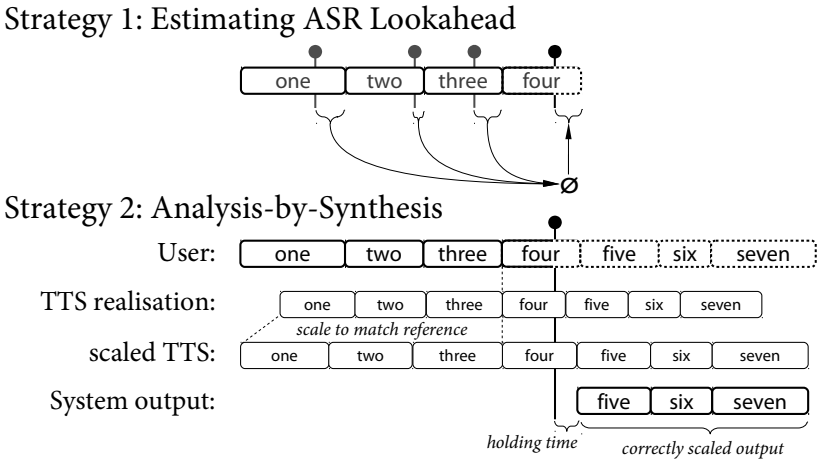


Figure 6.7: Our models to estimate holding time (*when* to speak), and speech rate (*how fast* to speak; only Strategy 2).

Strategy 1: Estimating iSR Lookahead In the ASR-based strategy (illustrated in Figure 6.7, top) the system estimates $\text{F0}_{\textit{word end}}$ which can also be called its *lookahead*, i. e. the average time between when a word is first recognized by iSR and the word's end in the signal. This lookahead is known for the words that have been recognized so far and the average lookahead can be used as an estimate of the remaining duration of the word that is currently being detected (i. e. its *holding time*).

The strategy just described, as well as the baseline strategy, only solve half of the task, namely the end-of-word estimation (sub-task (a) in Figure 6.4). This is sufficient, for example, to generate temporally aligned feedback but is not sufficient to solve the synchronous speech task as outlined above, which also raises the question of *how* to speak (sub-task (b) in Figure 6.4).

Both sub-tasks can be solved simultaneously by estimating the speech rate of the current speaker, based on what was already said so far, and considering this speech rate when synthesizing the completion of the utterance.

Speech rate estimation using some kind of duration model thus forms the second strategy's main component. Work on duration models can be found in the context of speech recognition, e. g. as a pre-processing step to perform model adaptation (Pfau and Ruske 1996), or integrated with the ASR to improve performance (Johnson 2005), or as a post-processing step to re-sort N-best lists (Anastasakos, Schwartz, and Shu 1995). In the ASR context, speech rate is often used synonymously with phone

rate but Pfitzinger (1998) shows experimentally that perceptual speech rate is more closely correlated to a linear combination of phone rate and syllable rate than either of the two alone. Text-to-speech synthesis systems employ duration models to assign durations to the words and phones to be synthesized. Rule-based approaches (Klatt 1979) as well as methods using machine learning have been used (primarily CART, Breiman et al. 1984; for a recent overview see: Lazaridis et al. 2010); for HMM-based speech synthesis, durations are often generated from Gaussian probability density functions (PDFs, Yoshimura et al. 1998). This author is not aware of any work that used duration models to predict the remaining time of an ongoing word or utterance previous to (Baumann and Schlangen 2011).

In the present task, the duration model is needed to make estimations based on limited input (instead of providing plausibility ratings as in most ASR-related applications). A simple approach to duration modelling could be based on average phone durations. However, good estimates require context-dependent phone duration models. Data sparsity would be problematic and require some kind of backoff or smoothing. Finally, phonesets between ASR and TTS may differ (they do here).

As it turns out, a TTS system in itself is an excellent duration model because it potentially ponders all kinds of syntactic, lexical, post-lexical, phonological and prosodic context when assigning durations to words and their phones. Also, the task of collaborative completion already involves a TTS system to synthesize the turn completion – in the present case MaryTTS (Schröder and Trouvain 2003). The durations can be accessed in symbolic form in MaryTTS's XML format (Schröder and Breuer 2004), and the system allows to manipulate this information prior to acoustic synthesis. Depending on which voice is used, MaryTTS uses machine-learned duration models (CART or PDFs) or an optimized version of Klatt's (1979) rules which have been shown to perform only marginally worse than the CART-based approach (Brinckmann and Trouvain 2003).

Strategy 2: Analysis-by-Synthesis As just described, this second strategy employs the TTS' duration model in an analysis-by-synthesis approach, as illustrated in Figure 6.7 (bottom): when triggered to complete an ongoing utterance, the TTS is queried for the durations it would assign to a production of the predicted full utterance, i. e. the prefix that was heard plus the predicted continuation of the turn. (In Figure 6.7, the TTS would be queried for the complete utterance "`one two three four five six` seven".) In that way, the TTS can take the full utterance into account when assigning prosodic patterns which may influence durations. The model then computes the factor that is needed to scale the TTS's duration of the words already finished by the user (in the example in Figure 6.7: "`one two three`") to the duration of the actual

utterance[5] and apply this scaling factor to the remaining words in the synthesized completion. The expected duration of the currently spoken word can then be read off from the scaled TTS output and, by subtracting the time that this word is already going on, indicates the *holding time*. Similarly, the completion of the turn which is now scaled to match the user's speech rate can be fed back to the synthesis system in order to generate the acoustic waveform which is to be output to the speakers once the system should start to speak.

The MaryTTS' duration model depends on the synthesis voice. The experiments (and measurements) reported below were performed with MaryTTS version 4.1.1 (the current version at the time of the experiment), using the `bits3-hsmm` voice (which worked slightly better than other voices).

6.2.5 Evaluation

The micro-timing prediction strategies were evaluated in a series of controlled offline experiments (that is, not in the context of a live system that attempts to co-complete live user utterances). This subsection first describes the evaluation corpus and experiment setup before the two subtasks – EoW prediction for the ongoing word, and speech-rate prediction for the next word – will be evaluated in turn, and means to improve these based on the estimated reliability of predictions are discussed.

The evaluation methodology is simple as the predicted word timings can simply be compared to the actual gold timing of the words. *Gold* word timings could either be taken from (manual) alignments, or the final ASR output. Similar to the iSR evaluation in Chapter 5, the ASR's word alignments were used. A micro-timing predictor that successfully predicts the system's time alignment should be good enough, and occasional small differences in ASR alignment (as compared to a manual alignment) are likely irrelevant.

The evaluation will then focus on the *error distribution*, that is the difference between estimate and reality. Apart from the overall mean of that distribution which indicates a bias of the system (and which would ideally be zero), the variance of estimation errors (which indicates *jitter*) should be as low as possible. Jitter makes the system unpredictable whereas a constant error might be easy to accustom to.

6.2.5.1 Corpus and Experiment Setup

Recordings of the German version of the story *The North Wind and the Sun* (Aesop 1991; IPA 1999) from the *Kiel Corpus of Read Speech* (IPDS 1994) are used as evaluation corpus. The story (including title) consists of 111 words and is read by 16 speakers,

[5] In a way, this extends Pfitzinger's (1998) speech rate measurements beyond phones and syllables towards estimating against words in full context.

giving a total of 1776 words in 255 inter-pausal-units (IPUs), altogether totalling about 12 minutes of speech. (In the following, "turns" will be equated with IPUs, as a corpus of read speech does not contain true turns.) Words and phones in the corpus have a mean/median/std dev duration of 319/290/171 ms and 78/69/40 ms, respectively.

The evaluation supposes that every word within a phrase can be a possible completion point in a real system,[6] hence the performance of the micro-timing component is evaluated for all words in the corpus. This gives a general picture of the micro-timing capabilities, not restricting the results to the application in collaborative completions. (This generalization may have an influence on the results: real collaborative completions are sometimes invited by the speaker, probably by giving cues that might simplify co-completion; if that is true, the version tackled here is actually harder than the real task.)

Good turn completions (and good timings) can probably only be expected in the light of high ASR performance. An acoustic model trained for conversational speech was used which was not specifically tuned for the task and a domain-specific, overly optimistic, language model was trained (based on the test corpus). The resulting WER is 4.2 %.[7] While the results could hence be considered too optimistic, the analyses in Chapter 5.4.1 showed that incremental metrics are relatively stable in the light of varying ASR performance, and – more specifically – the analysis in Figure 5.6 shows that this is especially true if timing metrics are considered only for words that were recognized correctly (and only for these words completions are attempted). The author expects that lower ASR performance would not radically change prediction quality itself; rather, it would have an impact on how often continuations could be predicted, since that is based on correct understanding of the prefix of the utterance, limiting the amount of data points for the statistics.

While it is strictly speaking not part of the timing component, a precondition to being able to speak *just-in-time* is to ponder this decision sufficiently early. Only if ongoing words are hypothesized somewhat before they are finished can a completion be uttered starting from the next word. Figure 6.8 shows a statistic of when iSR first hypothesizes a correct word relative to the word's end ($\mathrm{FO}_{word\,end}$, which can be determined post-hoc from the final recognition result) on the corpus. Most words are hypothesized before their actual endings, with a mean of 134 ms (median: 110 ms) ahead (these numbers are only slightly lower than the other, broader corpora examined in Chapter 5). This leaves enough lookahead to synthesize a completion and for some

[6] To allow some context for timing estimation, the first two words after a pause are not considered for completion.

[7] WER only slightly increases to 5.9 % when the optimistic model is interpolated (.6/.4) with a language model for the Pentomino domain. This robustness to the language model probably reflects the clear speaking style in the *Northwind* corpus.

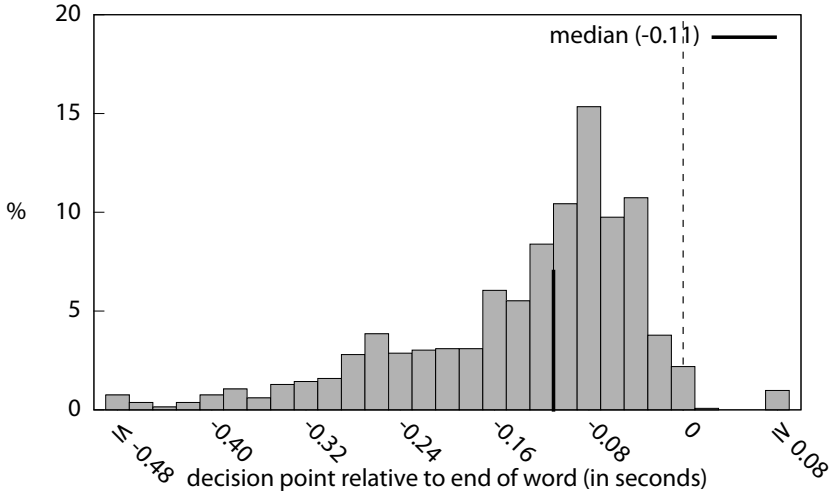


Figure 6.8: Distribution of F0 relative to the word's end ($F0_{word\ end}$; determined post-hoc) in the *Northwind* corpus.

delays that must be taken into account for input and output buffering in the sound card, as well as synthesis delays, which together take around 50 ms in our system.[8]

Interestingly, $F0_{word\ end}$ differs widely for the speakers in the corpus with means between 97 and 237 ms suggesting that incremental performance depends on the overall interplay between iSR and speaker. As can be seen in Figure 6.8, some words are only hypothesized *after the fact*, or at least too late to account for the inevitable lags – a system, however, could still output a completion by adapting its beginning as outlined above, if the timing component successfully estimates the delay.

6.2.5.2 End-of-Word Prediction: When to Start Speaking

The two micro-timing strategies are evaluated by comparing the predicted EoW with the actual EoW. The mock completion predictor described in Subsection 6.2.6 is limited to generating micro-timing results only if the previous two words were recognized correctly and the overall WER is lower than 10 %. Under these constraints, the micro-timing component generated data for 1100 IPU-internal and 223 IPU-final words in the corpus.

[8]No special means were taken to minimize synthesis or sound card delays in the co-completion system.

Table 6.1: Descriptive statistics of the error distributions over estimated onset times for different duration models.

model	error distribution metrics (in ms)			
	mean	median	std dev	MAE
baseline: all	-134	-110	107	110
baseline $-\mu$	0	23	107	63
ASR-based : all	-2	19	105	60
IPU-internal	26	33	82	51
IPU-final	-148	-143	87	142
TTS-based : all	-3	4	85	45
IPU-internal	12	11	77	41
IPU-final	-78	-76	83	79

The main target of this section is turn co-completion and completions can only take place if there is something left to complete (i. e. only after turn-internal words). However, the micro-timing models attempt to be useful more generally. For example, being able to predict the duration of turn-final words is important for tightly integrated speaker changes. For this reason, EoW estimates for both turn-internal and turn-final words were included in the analyses, as well as for all words in combination.

Figure 6.8 can also be taken as depicting the error distribution of the baseline strategy which starts every completion immediately: on average, the completion will be early by 134 ms if it is uttered immediately and the standard deviation is relatively high at 107 ms (as shown in Table 6.1).

An unbiased baseline strategy can be obtained by always assuming the EoW to be 134 ms (the global mean $\text{F0}_{word\,end}$) away. While this results in a mean error of zero, the variance of the distribution remains the same. Furthermore, this enhanced baseline requires the global mean to be known in advance and is hence inflexible: the mean may very well be different for other data sets as it already differs between speakers in the present corpus.

The two micro-timing models' error distributions are shown in Figure 6.9 and the distributions' mean, median, and standard deviation,[9] as well as the *median*

[9]Mean and std dev are reported separately and describe bias and jitter, respectively. A combined metric often used to describe error distributions is *root mean squared error* (RMSE). Notice that RMSE= $\sqrt{\mu^2 + \sigma^2}$.

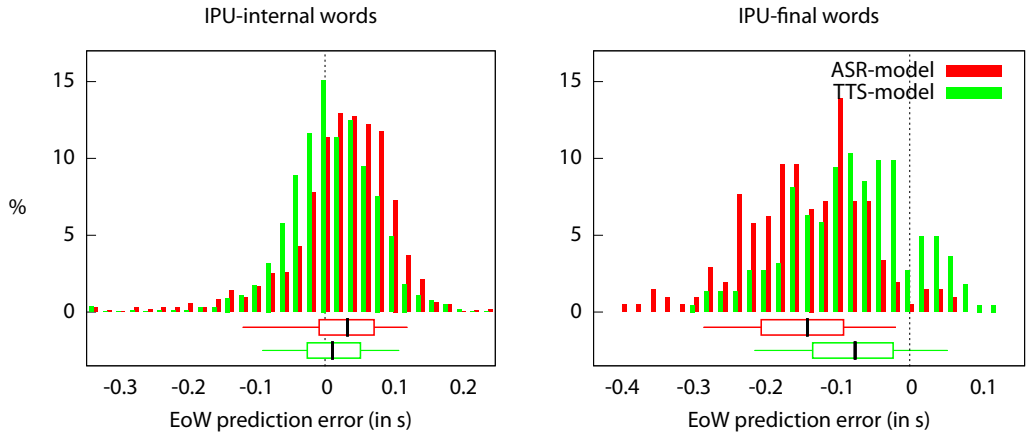


Figure 6.9: Error distributions for EoW prediction using the two models based on ASR and TTS, respectively, for IPU-internal words (left) and IPU-final words (right). Box plots show quartiles (boxes) and 5 %/95 % quantiles (whiskers).

absolute error (MAE[10]) are shown in Table 6.1. As can be seen in the table, both strategies are similarly effective in predicting the average remaining time of a currently uttered word, reducing the mean error close to zero, a significant improvement over starting a completion or next turn immediately (ANOVA with post-hoc Tukey's honest significance differences test). However, the two approaches do not differ significantly, at least when looking at all words in the corpus together.

As evident in the table and as can be seen when comparing the left and right part of Figure 6.9, both methods' bias in EoW prediction differs radically between IPU-internal and -final words: it is positive for IPU-internal but negative for IPU-final words (and in both cases, the TTS-based outperforms the ASR-based model). This is probably due to *final lengthening*: phones are about 40 % longer in IPU-final words in the *Northwind* corpus. Final lengthening is not modelled at all by the ASR-based strategy and the lengthening may be stronger than what is predicted by the duration model of the TTS that was used.[11]

[10]MAE is defined as the median error magnitude (ignoring its sign).

[11]Theune et al. (2006) notice longer pauses, particularly between sentences, for storytelling when compared to newsreading, for which TTS systems are often optimized. Longer pauses might also lead to stronger pre-final lengthening in the storytelling task.

While the two approaches perform similarly when comparing the estimation bias for all words, there actually are differences when looking separately at IPU-internal and IPU-final words. In both cases the TTS-based approach has a significantly lower bias than the ASR-based approach (paired Student's t-tests, $p < .01$).

A low standard deviation of the error distribution is probably even more important than a low mean error, as it is variability, or *jitter*, that makes a system unpredictable to the user. While there is no significant improvement of the ASR-based approach over the baseline, the TTS-based approach significantly outperforms the other approaches (Browne-Forsythe's modified Levene's test, $p < .001$) with a 20 % reduction of jitter down to about the average phone's length.

Regarding human performance in synchronous speech, Cummins (2002) reports an MAE of 30 ms for the live (interactive) synchronous condition. However, MAE increased to 56 ms when synchronizing to an (unsynchronously read) recording, a value which is in the range of the TTS-based system (which also uses unsynchronously read speech).

The TTS-based strategy, of course, requires a predicted completion for the utterance, as the built-in duration model expects full utterances as input. In an experiment where only the iSR input and no completion was fed to the TTS model, the overall mean error was much higher (66 ms; for IPU-internal words: 89 ms) than for the ASR-based or the full TTS-based strategies, indicating that in this case the TTS duration model applied rules for final lengthening to every word. Jitter for this experiment was higher than for the full TTS-based system, and similar to the ASR-based system.

6.2.5.3 Predicting the Micro-Timing of the Upcoming Word

The EoW estimate from above can be directly used to determine the onset of a co-completion, as the next word's onset is identical to the end of the current word. However, as explained in the task description, knowing when to speak is only one side of the medal, as a turn completion itself must be integrated with the previous speech in terms of duration, prosodic shape and loudness.

When evaluating how well a timing component predicts the following word's duration, that word needs to also be correctly recognized by ASR. This holds for 1045 words in the corpus, for which upcoming speech alignment results are reported. As co-completion can be performed word-by-word, the error distribution for each upcoming word, as well as the error distribution of EoW for each *upcoming* word is computed (which is simply the addition of EoW of the current word and the upcoming word's duration).

Only the TTS-based micro-timing model is capable of outputting predictions for a future word; the ASR-based approach does not provide this information. However, both duration and onset estimation together determine the error at the word's end.

Table 6.2: Descriptive statistics of the error distributions for the upcoming word's micro-timing. The onset is identical to the current word's EoW (cmp. Table 6.1) and combined with the estimated duration gives the error of the upcoming word's EoW.

task	approach	error distribution metric (in ms)			
		mean	median	std dev	MAE
onset	ASR-based model	26	33	82	51
	TTS-based model	12	11	77	41
duration	baseline : word μ	-20	-1	177	139
	baseline : *raw* TTS	29	39	89	60
	TTS-based model	-5	4	75	45
combined to end-of-word	ASR + μ baseline	-11	-2	188	134
	ASR + TTS	25	30	100	81
	TTS + TTS	7	10	94	74

Hence, the error at the next word's end for the TTS strategy's duration estimate combined with both strategies' onset estimates is reported in Table 6.2.

Table 6.2 shows two simple baselines for predicting the next word's duration: one is to assume the average word duration in the corpus (mean 319 ms in the Northwind corpus); the other is to use the TTS's word duration without applying the linear scaling method.[12] The first baseline results in a very high jitter (σ = 177 ms), while the second baseline does not adapt to the average corpus speech rate, which is about 10 % higher than the employed TTS's default, resulting in a bias of 29 ms. In contrast, duration prediction for the next word with the TTS-based micro-timing strategy works similarly well as for ongoing words (as above), with an MAE of 45 ms (which is again in the range of human performance). The mean error is significantly lower than for either of the baselines (correlated ANOVA with post-hoc Tukey's honest significance differences test).

Errors in calculating the estimated onset time correlate with speech rate estimation errors for the completion (Pearson's r = 0.607, $p < .05$), probably due to the linear scaling in the TTS-based micro-timing model. Thus, when onset estimation is combined with duration prediction, errors add up and hence the error for the next word's

[12]The unscaled results for this baseline were determined with Mary 4.3.0, which, however, leads to results that only minimally differ from those obtained with Mary 4.1.1, used in all other experiments.

end is somewhat higher than for either of the tasks alone, with a standard deviation of 94 ms and an MAE of 74 ms for the full TTS-based model, which again outperforms the combination with the ASR-based model.

6.2.5.4 Estimating the Reliability of the Predictions

System actions during a user turn (e. g. co-completing synchronously) are often optional and should only be performed if the system is sufficiently certain about its estimates. Thus, the micro-timing model should ideally single out those estimates which are especially error-prone, trading coverage against quality of results, or even better, return expected error margins or even an estimated error distribution.

A full solution for co-completion might then take into account the costs and merits (as was attempted for non-verbal actions in the *Greifarm* example, cmp. Chapter 5.6.2), of speaking in (relative) synchrony the next word vs. waiting for another word in the hope that the estimate will be more precise then.

This subsection is limited to a rudimentary approach to reduce the estimation error of the completion's onset. As mentioned above, onset estimation errors and speech rate errors are correlated and hence singling out errors in onset estimation also helps to reduce errors in the completion's speech rate.

A large amount of features from ASR could be used to train models that predict the estimation error (or to improve the estimation itself). The TTS-based micro-timing model furthermore integrates independent, non-ASR knowledge into the process which should be especially helpful. However, to keep things simple, only some correlations of estimation error with some individual features was performed, instead of a an elaborate machine-learning experiment.

The TTS-based micro-timing predictions were found to be especially unreliable when they predicted a negative holding time (i. e. they predicted that the word should already have started), or when they predicted a holding time of more than about 300 ms, as can be seen in the scatter plot shown in Figure 6.10.

The implemented system is not able to shorten onsets to account for negative holding times (e. g. by shortening the onset of co-completion as proposed as a solution to this phenomenon above), so that a simple rule was based on the estimated holding time: completions are only generated for estimated holding times t if $50\,\text{ms} < t < 300\,\text{ms}$. (50 ms was chosen as lower bound to account for system audio delays.) This simple rule removes decisions for 16 % of the words while improving the *jitter* (in terms of standard deviation) of the remaining estimates by 14 % (especially, singling out all decisions to co-complete that are impossible to perform as requested in the implemented system).

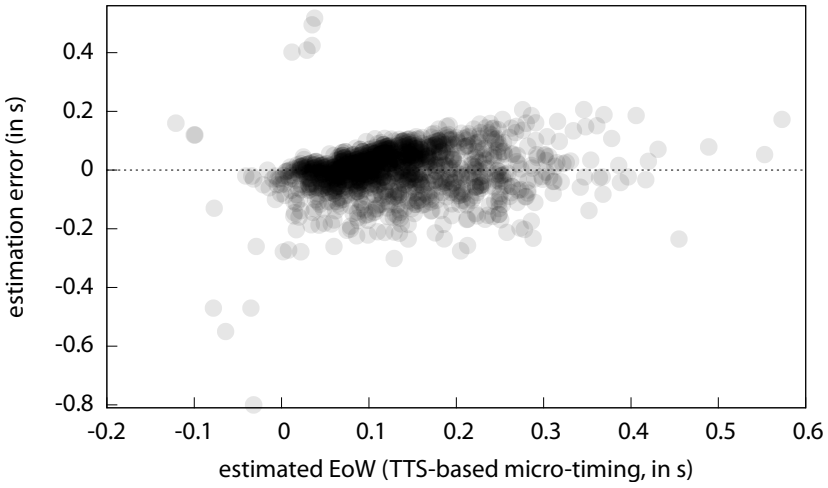


Figure 6.10: Scatter plot of TTS-based micro-timing prediction for EoW vs. estimation error, indicating the reliability of predictions.

6.2.5.5 Summary

The evaluation shows that short-term micro-timing prediction is feasible and works with a degree of precision that compares well to human performance (human MAE of 56 ms for synchronizing to non-synchronously spoken recordings vs. prediction MAE at 41 ms for onset timing and 45 ms for word duration, giving a combined MAE of 74 ms).

The TTS-based linear-scaling approach worked best in the experiments, as it combines several sorts of linguistic information for duration modelling. It can, however, only be used if a an utterance completion is available.

Reliability of predictions can be estimated and a simple rule can be used to improve results if the system is allowed to skip words for which the prediction is estimated to be unreliable, resulting in a 14 % decrease of jitter.

Only duration, but not the alignment of other prosodic aspects has been evaluated for the predicted completions. This is, of course insufficient for a full system. However, the interdependence between ongoing and upcoming prosody, as well as the prosodic interaction between synchronizing speakers are more complex (Dellwo and Friedrichs 2012) than either the simple linear models used here could model, nor the simple evaluation based on error distributions could cover.

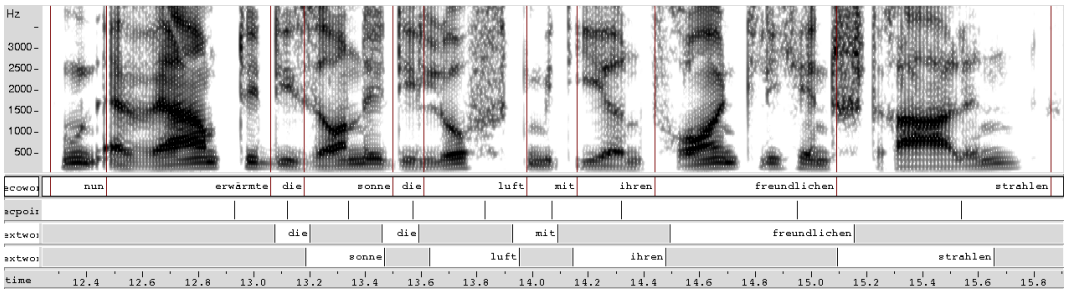


Figure 6.11: Example of shadowing for a file in the *Northwind* corpus (k73nord2). The first line of labels shows the final ASR output, the second line shows the decision points for each word and the third and fourth lines show the system's output (planned output may overlap, hence two lines; in the system, an overlapped portion of a word is replaced by the following word's audio).

6.2.6 Example Application: Speaking in Synchrony With the User

To get a feeling for the complete system and to demonstrate that the micro-timing component works on live input, an application for synchronous co-completion of user utterances word-by-word was developed. Given the prediction for the next word's onset time and duration it prepares the output of that next word while the user is still speaking the preceding word; the full system only implements the TTS-based micro-timing model, which was found to be most efficient in the evaluation above. Initially, the full expected user utterance to be synchronized with is pre-processed by MaryTTS (Schröder and Trouvain 2003) and used in the duration model. The completions are produced word-by-word, by re-scaling the TTS's durations and handing the scaled speech back to MaryTTS for synthesis. The application needs to know what the user is going to speak (because there is no real completion prediction component), so that the user is currently limited to telling the story of *North Wind and the Sun*.

Two examples of shadowing are shown in Figures 6.11 and 6.12. As can be seen in the screenshots, the decision points for all words are sufficiently early before the next word, allowing for the next word's output generation to take place. Overall, shadowing quality is good, with the exception of the second "`die`" in the second example. However, there is an ASR error directly following ("`aus`" instead of "`luft`") and the ASR's alignment quality for "`sonne die`" is already sub-optimal. Notice that the two words following the ASR error are not shadowed as per the error recovery strategy.

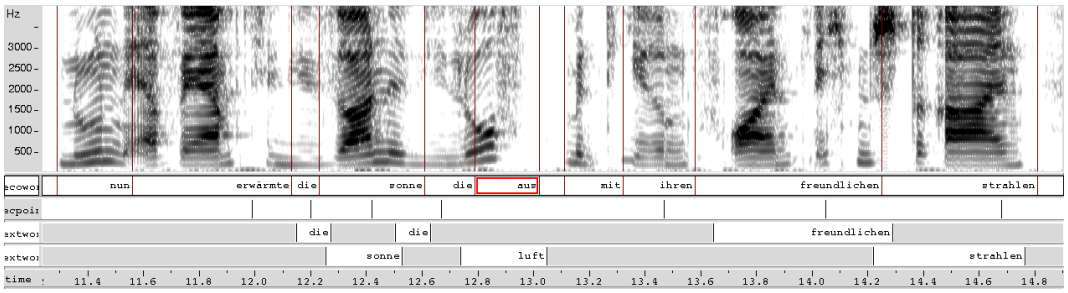


Figure 6.12: Example of shadowing with live input (verena2nord2). Notice that "Luft" is predicted and synthesized although it is (later) misunderstood by ASR as "aus", resulting in a missing shadowing of "mit" and "ihren". In order to not disturb the speaker, the system's audio output was muted.

The synchronous co-completion system is meant as a pure technology demonstrator, showing that real-time incremental predictive processing is possible in INPROTK. It has hence not been formally evaluated with actual users of the system how they judge the quality of the generated collaborative completions.

6.2.7 Discussion

This section described the task of predicting the user's micro-timing, and micro-aligning a system response to the user's speech (in the present case a synchronous turn co-completion).

It was shown that a completion is possible after most words, as iSR in a small-enough domain can have a sufficient lookahead ($\text{FO}_{word\,end}$). For the task, the TTS-based duration model outperforms both the baseline and the ASR-based model. Both the next word's onset and duration can be predicted relatively well, and within the margin of human performance in synchronously reading speech.

The system was not aimed at predicting and matching prosodic characteristics such as loudness and intonation (and these aspects were not evaluated in the present implementation). Simple matching (as was implemented for onset and speech rate) is probably not as good a starting point for these as they are more complex. Instead, these phenomena mostly depend on communicative function, e. g. a co-optation having a wide pitch-range and relatively high loudness regardless of the current speaker's

speech. Additionally, pitch-range would have to be incrementally speaker-normalized which results in some implementation difficulties.[13]

The analysis has been somewhat focused on the task of micro-aligning a turn-completion and speaking in synchrony with the speaker. However, the micro-timing prediction capability can be useful in a broad range of tasks, e. g. in combination with word-based end-of-turn detection (Atterer, Baumann, and Schlangen 2008) to allow for swift turn-taking.[14] Micro-timing models can be used to quickly detect deviations in speech rate (which may indicate hesitations or planning problems of the user) *as they happen* (rather than post-hoc), allowing to take the speaker's fluency into account in understanding, and turn-taking coordination (as outlined by Clark 2002), or to assess whether the user accepts (and appreciates) system actions during the utterance.

Finally, even if in the long run precise timing of system actions turns out to be unnecessary, this can only be found out systematically if one has tried systems that do time their utterances very well and that have deliberate control over this feature. The present system enables such tests. Additionally, precise micro-alignment of turn handovers could be used for controlled testing of linguistic/prosodic theory such as the oscillator model of the timing of turn-taking (Wilson and Wilson 2005).

Clearly, the present duration modelling is rather simplistic and could likely be improved by combining ASR and TTS knowledge, more advanced (than a purely linear) mapping when calculating relative speech rate, integration of phonetic and prosodic features from the ASR, and possibly more. As currently implemented, improvements to the underlying TTS system (e. g. more "conversational" synthesis) should automatically improve our model.[15]

The current model considerably degrades if no full completion is available. However, often some idea about the future context may be available (such as: user will continue speaking vs. user is almost done), even if the full completion remains unknown. An incremental prosody model, that is able to integrate partial knowledge about the future would be able to integrate such information and lead to a more graceful degradation of performance for partially known user speech.

While the example application has shown that micro-timing works sufficiently well to support synchronously co-completing the user, it has also highlighted a deficiency

[13] Edlund and Heldner (2007) report that for a reliable pitch-range estimation 10 to 20 seconds of *voiced* speech and hence – in the author's view – twice the amount of audio is necessary. This would have reduced the corpus size by too much.

[14] Additionally, both implemented models consistently under-estimate the duration of IPU-final words. It should be possible to turn this into a feature by monitoring whether a word actually has ended when it was predicted to end. If it is still ongoing, this may be an additional indicator that the word is turn-final.

[15] There was a dramatic improvement after switching from a previous version of MaryTTS.

of current speech synthesis technology, namely the fact that it performs non-incrementally. While words *can* be synthesized one after the other (however, the prosodic characteristics have to be pre-determined as they require more than the word-context), there is no provision of speeding up an ongoing word, or lengthening it. This would be needed to correct a previous estimation error and to align more closely to the user's speech. In fact, Dellwo and Friedrichs (2012) show that a synchronizing speaker and the target speaker interact in a complex way, showing that humans are well able to perform such online adaptations. This stronger interaction and faster reaction may also be the cause why the individual errors for onset and duration prediction add up in the system while human speakers are able to correct their speech rate during a word.

6.3 Summary and Discussion

This chapter was aimed at putting incrementality to work towards spoken dialogue systems that interact incrementally below the granularity of turns. The floor tracking component that is integrated in InproTK was presented in Section 6.1, which is able to support (sub-)turn-taking decisions based on IPU-final prosody to allow a collaboration on utterances between the user and the system. An example application that uses this feature to act on partial knowledge is more time-efficient and also rated as more human-like and more reactive than a non-incremental baseline.

Going one step further, a micro-timing model was presented in Section 6.2 that can be used to estimate the remaining duration of an ongoing word. An example application used the micro-timing model to also estimate the durations of words following the ongoing word and was able to speak in synchrony to the input, showing that end-to-end real-time incrementality is possible.

Of course, the results from the second step could be integrated into the general floor tracking component, either to broaden the interface towards the dialogue manager, allowing for micro-aligned system actions, or to improve turn-end estimation in the floor tracker by integrating micro-timing considerations such as expected EoW times, or deviation tracking.

Where the first example application only outputs very short feedback utterances, the second (co-completing) example application highlighted the fact that longer user-coinciding system utterances may need to be adapted while they are produced. The output shown in Figures 6.11 and 6.12 show that non-incremental word-by-word output results in gaps and overlaps that correct estimation errors in a highly unnatural way.

This chapter has increased the monitoring granularity of dialogue flow estimation from the level of full turns via sub-turn units to single words. However, humans

continuously monitor their interlocutors, and, more importantly, also monitor their own speech continuously. In contrast to this, the implemented system could not be monitored or adapted while produced. The experience in this chapter has shown that it is especially the lack of proper incremental output generation that constrains the implemented system.

After having focused on input incrementalization into the dialogue system, and supporting the decision making with micro-temporal information so far, the next chapter will turn to incremental speech synthesis that will allow to instantly adapt output in terms of speech rate, prosody, and content, completing the set of "building blocks" for incrementally interacting spoken dialogue systems.

Drawing by Milo Winter, 1919, (taken from Aesop 1991, public domain).

7 Incremental Speech Synthesis

The previous chapters have shown how incremental dialogue processing can help to build a partial understanding of the user (Chapter 5) and to help estimate the upcoming progression of the dialogue flow during a speaker's turn (Chapter 6). These capabilities together allow for advanced interactive behaviour in SDSs, allowing the system to shape the interaction, for example by starting to act during a user's turn (Sections 5.6 and 6.1.2), more closely aligning system actions to the interlocutor based on the speaker's behaviour and the system actions' nature (Section 6.1.2), or even predicting word timings, allowing to speak in synchrony with the user (Section 6.2.6). All the systems that were presented so far, however, show only a limited amount of incremental behaviour in the way that they generate their output. All systems so far were able to take back their (non-linguistic) actions if an action's support is lost; however they had no advanced control over their *spoken* output.

Being able to partially take back actions and to adapt ongoing actions enables *speculative execution*, and hence a higher degree of vitality in the interaction. Actions that turn out to be wrong in light of updated information can simply be adapted which reduces the 'cost' of these wrong actions. In contrast, if actions cannot be taken back or adapted, the system should rather wait until its hypothesis becomes sufficiently stable. The system in Section 6.2.6 performed speculative execution without being able to adapt ongoing actions (here: words that were being synthesized). This resulted in gaps in-between words or words being cut off when timing hypotheses turned out to be wrong. Much better results could be expected if the system had been able to *adapt* the ongoing speech to meet the desired timings in a smooth way.

In short, speculative execution of (partial) actions that can later be aborted or changed enables the system to increase its degree of vitality in the interaction, for example resulting in more naturally sounding co-completion capabilities. A system can also extend its inventory of dialogue contributions, if these contributions can be adapted to account for the evolution of events in a highly dynamic environment.

So, after having focused on incrementalization of the input side and analysis of dialogue flow, this chapter turns to incremental spoken output. Again staying on the lower layer, this chapter focuses on incremental speech synthesis (iSS) as a means to enable more naturally interacting dialogue systems. Section 7.1 presents and discusses use-cases for incremental speech output from which some requirements will be deduced, Section 7.2 gives some background information on speech synthesis in general. Readers familiar with HMM-based speech synthesis may want to skip ahead

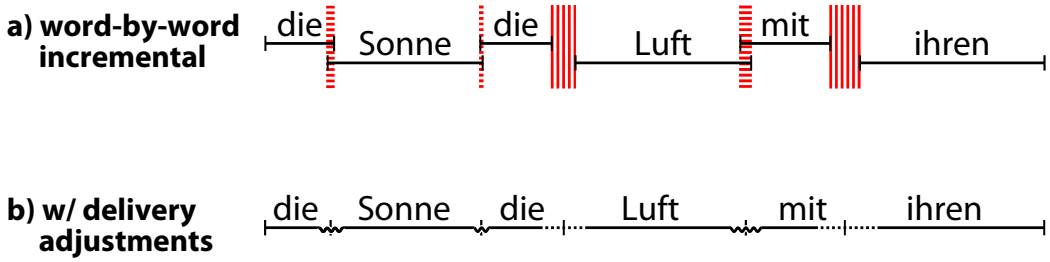


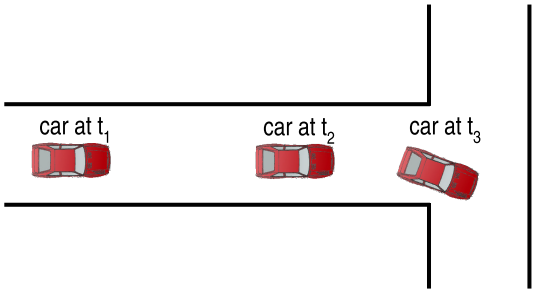
Figure 7.1: Detail of word-by-word incremental synthesis from Figure 6.11: (a) highlighting the overlaps and gaps that result from changing incremental hypotheses, (b) incremental delivery adjustments (here: tempo adjustments) squeeze or stretch ongoing words to account for the changes.

to Section 7.3 which explains how speech synthesis is incrementalized in the InproTK incremental speech synthesis (iSS) component.

The latter part of the chapter investigates iSS in a series of experiments: Section 7.4 evaluates the merit of using iSS in a highly dynamic environment as compared to standard, non-incremental speech synthesis. iSS is shown to be tremendously useful. Section 7.5 shows an actual incremental speech output pipeline that combines incremental NLG with iSS. The realized system again outperforms the non-incremental baselines. Section 7.6 finally evaluates the quality trade-offs in realized prosody associated with using incremental instead of standard speech synthesis, concluding that one phrase of lookahead is sufficient even for the relatively crude prosody processing of the implemented system. Section 7.7 concludes the chapter and points out future work.

7.1 Rationale for Incremental Speech Synthesis

The co-completion system presented in the previous chapter attempted to output words as they were spoken by the user, however relied on a coarse, incremental approach to speech synthesis: every word was simply uttered when the system deemed this ideal, not taking into account any presently spoken word. As can be seen in Figure 7.1 (a), this leads to words being cut off when a next word is started before the previous has ended, or gaps when a word is started only after the previous word has ended. Of course, a system should rather speed up the delivery of a word to make room for the next, than cutting it off, or to stretch an ongoing word if the next word is only to be uttered later (and possibly start early and stretch the onset of that next



time	event description	ongoing utterance (spoken part in **bold**)
t_1	car on Main Street	**The** car drives along Main Street.
t_2	car will have to turn	...**along Main Str**eet and then turns <hes>
t_3	car turns right	...**along Main Street and the**n turns right.

Figure 7.2: Example of incremental utterance production as a car drives along a street and turns. The ongoing utterance is extended as information becomes available.

word). This latter strategy is shown in Figure 7.1 (b). However, it requires the speech synthesis component to be able to adjust the timing of ongoing words, which is only possible if synthesis is conducted on-the-fly, and just-in-time.

Where synchronous speech mostly requires on-the-fly timing adaptation, other interaction phenomena may require quick adaptation of pitch and loudness, as for example turn-taking contests, which result in higher pitch and louder speech (Schegloff 2000).

A more common case that shows the need for incremental speech synthesis in a spoken dialogue system is the need to start speaking before all information required for the utterance is available, or to extend or even change the content of an ongoing utterance (of course, only its yet unspoken part). Take as an example the situation depicted in Figure 7.2 where a car can be observed at time t_1 going along a street, nearing a junction at time t_2, and turning right at time t_3. As shown in the figure, a system that comments on these events should be able to start the comment before all information to be delivered is available, and gradually extend the utterance while it is ongoing as more information comes in. That is, a system commenting on events in this domain should be incremental in order to adapt its utterance to the observed events. As the utterance is extended or changed, prosodic aspects of the already planned

parts may have to be reworked, for example, a sentence-end intonation should be reverted when more content is added.

The overall requirements (or at least desiderata) for incremental speech synthesis will be summarized after a brief survey of related work on incremental output production.

7.1.1 Related Work

Experiments have shown that human speakers usually plan their utterances somewhat ahead, typically at least in chunks of major phrases (Levelt 1989). However, if the need arises and the original plan becomes moot, human speakers can quickly abandon it and re-plan. This re-planning sometimes results in a disfluency (hesitation or self-correction), in other times may not be noticeable.

In contrast, current interactive systems that make use of speech synthesis typically place this synthesis last in a pipeline (McTear 2002). It is triggered when system prompts are to be played and often control does not even return before the utterance has finished playing. VoiceXML allows system utterances to be aborted in case a barge-in is detected by the ASR component. It does not provide for any more sophisticated adaptation of, or feedback on delivery (Oshry et al. 2009). Only some systems can, if their utterance is interrupted, stop playing the utterance and provide information about where the interruption occurred (Edlund 2008; Matsuyama et al. 2010).

The problems of genuinely *incremental* output generation have rarely been discussed in the literature. Edlund (2008) outlines some requirements for incremental speech synthesis: to *give constant feedback* to the dialogue system about what has been delivered, to *be interruptible* (and possibly continue from that position), and to *run in real time*. Edlund (2008) also presents a prototype, which however is limited to diphone synthesis that is performed non-incrementally before utterance delivery starts. While this system gives feedback and is temporarily interruptible, additional information cannot be taken into account to change an utterance during delivery.

Skantze and Hjalmarsson (2010) finally describe a system that generates utterances incrementally in a Wizard-of-Oz setting, allowing the Wizard to incrementally produce and revise its output. In the absence of other output, the system can automatically play hesitations to 'buy time'. An evaluation that they conducted shows that users prefer such an incremental system over a non-incremental version, even though the incremental system produced longer dialogues (because the hesitations were not aborted as soon as the remainder of the utterance became available but were finished as originally planned). The approach presented here is conceptually similar to (Skantze and Hjalmarsson 2010), but targets a lower layer of speech production incrementalization, namely the realization or synthesis layer. Where their system relies on 'regular' speech synthesis that is conducted on relatively short utterance

fragments (which leads to a trade-off between synthesis quality, favoured by long fragments, and system responsiveness, which is higher for shorter fragments), the work presented here aims to incrementalize speech synthesis itself.

Alignment of speech and gesture in virtual humans has been partially achieved by inserting pauses in speech and gesture production, respectively, allowing the 'slower' of the two to catch up (Welbergen 2011, Chap. 5). Of course, being able to adapt the timing of speech synthesis would help to better solve this alignment problem as well.

7.1.2 Requirements

Some requirements towards iSS that are needed to support the use-cases mentioned above are given below. The list extends Edlund's (2008) requirements specification,[1] adding the important characteristic of being able to change one's mind, and to work just-in-time.

An incremental, just-in-time speech synthesis component must be capable of:

1. handling changes/extensions made to (as-yet unspoken) parts of the utterance;
2. starting to speak before utterance processing has finished;
3. enabling adaptations of delivery parameters such as speech rate or pitch;
4. autonomously making appropriate delivery-related decisions;
5. providing system-internal feedback about progress in delivery; and, last but not least,
6. running in real time.

The primary requirement is that an incremental speech synthesis component *be able to extend and change an ongoing utterance*. With this capability, a system can be built that starts speaking as soon as it has enough information to formulate the beginning of an utterance – if done right, there will still be time to expand or change the remainder of the utterance later. For example, a system might want to comment on a world event such as a car appearing to be turning but it being not yet clear whether right or left (as in Figure 7.2). The system may go ahead and let its speech output component begin to realize the utterance "`The car turns <dir>.`" even though the direction remains uncertain.[2] As it is able to exchange "`right`" with "`left`" during delivery, it can start speaking as soon as it is certain *that* the car will turn, but before it is certain *where* the car will turn (and use the duration of the initial part of the utterance to find out where the car will actually go – as long as it is sufficiently certain that the direction will become known before the initial part of the utterance is over). Adding more

[1] Concerning the list of requirements, only items 1 (limited to utterance extension, not covering changes), 5, and 6 are mentioned by Edlund (2008).

[2] `<dir>` here stands as an abstract placeholder for right *or* left to help the synthesizer generate an intonation appropriate for the full utterance. The implementation below simply fills in "`right`" as it is lacking support for abstract placeholders.

words to the end of an ongoing utterance is a similar case. If the system is able to add words, it can seamlessly add information that was not available utterance-initially. For example, a system could append "`onto Straight Street`" in the example above.

The second requirement, *starting to speak before processing is over*, makes the system faster to react even if more than the absolutely necessary beginning is provided by higher-level components (which is actually desirable as it can then be considered when planing the overall utterance prosody).

The third requirement, *adapting delivery parameters* such as tempo, loudness, intonation, stress and other supra-segmental properties of speech, can be important to account for timing issues with incremental changes, to align speech with the environment or with other output modalities, and to allow for smooth concatenation of utterance continuations. For example, it should be possible for incremental speech synthesis to change the tempo at which speech is delivered seamlessly and with very little delay (to the point of interrupting speech flow as proposed by Edlund 2008), possibly allowing to change the duration of every speech sound individually. When an ongoing utterance is changed (the first requirement), supra-segmental properties of the utterance may have to be adapted as well (e. g. an utterance-final drop in pitch must be cancelled if that portion of speech is no longer utterance-final) and this should happen automatically.

Fourth, incremental output components operate under real-time pressure, that is, they must continue to deliver further output in time after they have started. Incremental speech synthesis can help to lower this burden by *autonomously making delivery-related decisions*, such as hesitating, if a higher-level component has not provided the remainder of the utterance, yet, and synthesis approaches this point (i. e., neither "`left`" nor "`right`" has been specified yet in the example above). The synthesis component could handle this situation, for example by first lengthening the remaining few phonemes, and/or inserting a hesitation. Similarly, if it is slightly too late to change a word because it is already being realized, the component could autonomously decide to insert an overt repair (like "`uh`" or "`no`"). Finally, an ongoing hesitation should be aborted quickly once the missing content becomes available.

Finally, especially if an incremental synthesis component takes decisions like hesitations autonomously, other components in a full system may require detailed *feedback about the progress of utterance delivery* at any moment. Also, as for any type of incremental processing, incremental speech production is of rather limited use if computationally it does not run in real time.

The next section gives some details about speech synthesis in general, before the incremental speech synthesis component that is part of InproTK will be described and shown to meet the requirements set out above in Section 7.3.

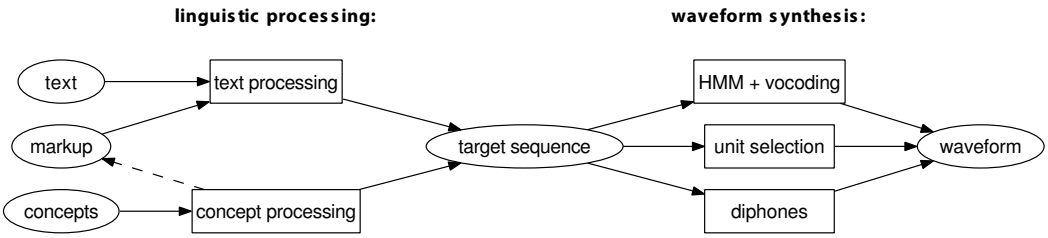


Figure 7.3: Processing paths in speech synthesis systems; various input is transformed into a target sequence which can then be turned into a speech waveform.

7.2 Speech Synthesis in a Nutshell[3]

Speech synthesis turns the linguistic description of an utterance into audible speech. An overview of this process is shown in Figure 7.3. Processing is conventionally split into two parts, linguistic processing and waveform synthesis. The interface between the two, the *target sequence*, describes the utterance using speech sounds, their durations and designated pitch values. Linguistic processing is followed by one of several waveform synthesis techniques that output the final synthesis result.

Two paradigms of linguistic processing exist: *Text-to-Speech* (TTS) uses the textual form of an utterance as input which is analyzed and turned into the target sequence. However, text does not explicitly encode all the details necessary for the production of an utterance. For example, the utterance "`Peter throws the ball.`" should be accentuated in different ways – depending on the importance of the pieces of information to be conveyed – and failing to do so may confuse the user. One approach to dealing with this is to annotate the text using some markup language (e. g. SSML, Burnett, Hunt, and Walker 2004). The annotations may describe aspects of the speech output that are not marked in its textual form, overcoming many of the problems of the textual representation.

Another approach is *Concept-to-Speech* (CTS, see e. g. Taylor 2000) in which the input is described in some logical form on a high level of abstraction and NLU and synthesis are combined into one task. Using the high-level description, CTS may have the advantage of *knowing* which of several possibilities should be synthesized, as can be seen in Example 7.1, where the focus of the utterance is explicitly marked in the logical form and can hence result in more adequate prosodic assignment.

[3] This introduction to speech synthesis builds on the excellent descriptions by Taylor (2009) and Jurafsky and Martin (2009, Ch. 8), which however give little detail on HMM-based synthesis.
Readers familiar with HMM-based speech synthesis may want to skip ahead to Subsection 7.2.5.

(7.1) "throw(peter, ball) ∧ focus(peter)" ⟶ *Peter* throws the ball.
"throw(peter, ball) ∧ focus(ball)" ⟶ Peter throws the *ball*.
"throw(peter, ball) ∧ focus(throw)" ⟶ Peter *throws* the ball.

CTS systems either generate target sequences directly, or output marked-up text that clarifies and specifies the desired input interpretation, and is passed on to text-based processing. This latter approach thus incorporates the task of *natural language generation* which in dialogue systems is often implemented as a separate component. However, most NLG components output pure text (without annotations), effectively disregarding the available information necessary for optimal speech synthesis.

This work focuses on TTS because its interface (text) is more generic and makes our results more widely applicable. Especially since CTS can be implemented by using marked-up textual representations, focusing on TTS does not result in a principled limitation. Section 7.5 shows how iSS can be combined with an iNLG component.

7.2.1 Text-based Linguistic Processing

This subsection describes the processing steps necessary to turn textual input into a target phoneme sequence. Linguistic processing uses a top-down approach, starting on the utterance level and descending onto words and phonemes (Taylor 2009).

The first task is to *segment* the incoming text into individual sentences (or, in spoken language, utterances) and to then *tokenize* the words. Both of these tasks are not as trivial as they may seem, because neither do terminal punctuation marks clearly define sentence boundaries ('Mr. Smith', 'Yahoo!') nor does whitespace unambiguously define word boundaries ('New York', 'you're'). Text *normalization* brings all words into a readable form: numbers and dates are recognized and expanded (often differently depending on the assumed meaning: '1984' vs. '12345'), abbreviations are either spelled out ('USA', 'NHS'), decapitalized ('NATO', 'AIDS'), or expanded ('etc.', 'et al.'), and symbols are replaced with a pronounceable form ('£', ':-)'). Tokenization and normalization often pose problems if the TTS must be able to deal with real-world texts. For SDSs, system utterances most often come from a moderately restricted language which helps to avoid potential problems.

Next, the input text is structurally analysed to determine *intonation phrases*, to assign *stress* marks to words, and to determine *tones* of phrases and stressed words (e. g. following the ToBI system, Silverman et al. 1992), most often using one or several rounds of decision-making by rules or learned decision trees. To support decision making, part-of-speech tagging or some form of parsing or chunking is performed as an initial processing step. The intonation assignment is a problem inherent to the TTS approach, as the prosodic realization is not explicitly marked in textual input. For example, prosodic assignment may depend on information status established in

previous utterances, on discourse context, or even on things like speaker and listener gaze – aspects which may be known to the dialogue system, highlighting the fact that text alone is a sub-optimal interface.

Next, in *phonemization*, the words are transformed from their graphemic to a phonemic representation, through letter-to-sound rules, decision trees, or simple dictionary lookups (cmp. Section 5.1.1.2), which may involve syllabification. Finally, the phonemes need to be assigned duration, pitch, and possibly loudness values (resulting in the *target sequence*) depending on intonation tones and other aspects. Durations are often assigned using decision trees, and final, continuous pitch contours may be generated using the HMM method (see below) from dis-continuous decision making (e. g. by decision trees).

7.2.2 HMM-based Waveform Synthesis

The waveform synthesis step turns a given target sequence into a speech waveform. While there are different competing methods, this section focuses on HMM-based synthesis and only briefly discusses this choice and the alternatives available in Subsection 7.2.3.

Chapter 5.1 argued that speech can be described by the source-filter model and, building on this, Hidden Markov Models. For recognition purposes, the source was described as resembling an impulse train filtered by the vocal tract. The source was disregarded, and analysis focused solely on the filter (described using Mel-frequency cepstral coefficients, MFCCs) which reflects the articulated phoneme (whereas source parameters mostly reflect individual speaker characteristics). While speech recognition *reduces* the full speech signal's complexity, speech synthesis needs to do the reverse: expand and recreate the full signal's variability from a limited parametric representation so that the speech output sounds natural. This is why synthesis also needs to model the 'source' aspect of the source-filter model for synthesis.

HMM-based synthesis is a form of *vocoding* where the target sequence is first transformed into a sequence of parameter frames that are then used to synthesize samples by a vocoder. To explicitly include voicing aspects of the signal source in the model, a pair of parameter sets is used, one to describe the signal source (STR) and one to describe the filter (*Mel-cepstral parameters*, MCP). Both are learned from training data using cepstral analysis techniques. As these parameter sets are independent, they can be modelled independently, as HMM feature *streams*; a third stream emits pitch values for every data frame.

The following subsections first describe how to use HMMs to generate synthesis parameters and then explain how these are vocoded into speech.

7.2.2.1 Parameter Estimation with HMMs

In the most simple form, HMMs could be used for generation by randomly emitting parameter vectors and transitioning through the network according to the emission and transition probabilities ($\mathcal{A}$ and $\mathcal{B}$, cmp. Section 5.1.2.1). This is, however, not only inefficient but also it completely disregards the continuity constraints of speech: the gradual spectral change in speech that was *captured* in recognition using Δ- and ΔΔ-features needs to be explicitly *enforced* in synthesis.

To restrict the model to gradual change, the desired sequence of synthesis coefficients ($\mathbf{C} = c_1, c_2, \ldots, c_n$) is appended with dynamic features Δc_t which compute the change between c_{t-1} and c_{t+1} (similarly ΔΔ-features for acceleration), forming observations $o_t = (c_t, \Delta c_t)$. When using Gaussian distributions as probability density functions $\mathcal{B}$, the most likely emission sequence given an aligned state sequence can then be determined by solving a large system of linear equations that includes both the synthesis coefficients as well as the derived dynamic Δ-features (Taylor 2009) .

However, finding the optimal observation for an aligned state sequence does not solve the problem of finding the optimal alignment.[4] Tokuda et al. (2000) developed a method to iteratively improve alignments, the first break-through for HMM-based synthesis. However, it was found later that fixing the alignment by explicitly modelling state durations (e. g. using decision trees), effectively resulting in a *Hidden Semi-Markov Model* (HSMM), performs just as good and saves iteration costs (Zen et al. 2007a).

The maximum likelihood estimation technique described above finds a sequence that is 'most likely', that is, it optimizes the resulting sequence relative to the means of the observation probabilities. While the resulting speech is clearly understandable, it is lacking in naturalness and 'depth' because the *variance* of the signal is not as large as it is in natural speech. A technique called *global variance boosting* (GV, Toda and Tokuda 2007) can be used to iteratively optimize both the means and the variances, resulting in far more natural speech (Taylor 2009). The resulting parameters from the independently optimized HMM streams are then passed on to vocoding.

7.2.2.2 Vocoding

The *vocoder* turns the parametric emissions of the HMM optimization into a speech signal (the reverse task to the ASR frontend's, see Section 5.1.1.3).

[4]Speech recognition uses Viterbi decoding to find the state sequence which contains the best-matching alignment (even though all possible alignments for each state sequence should ideally be taken into account). The rationale was that the best-matching alignment's probability dominates all other alignments' probabilities. Here, one is left without any 'good' alignment at all, and picking one at random will likely be far from optimal.

Vocoding implements the source-filter model. In its simplest form, the source is modelled as either voiceless (in which case white noise is generated) or voiced. In the latter case, a periodic signal (e. g. an approximation to an impulse train or a sawtooth signal) is generated with a specified fundamental frequency f_0. The source signal is then filtered, e. g. using cepstral coefficients that describe how much to attenuate each Mel-spaced frequency band. Thus, for every frame (in this work: 5 ms) the *voicing* status, the *fundamental frequency* f_0 (if applicable), and band-pass filter parameters expressed as Mel-cepstral parameters (MCP) are required.[5] Voicing can be determined per HMM state and optimization is used for the other parameters as described above.

An advanced approach that leads to more natural sounding speech is STRAIGHT vocoding (Kawahara 1997) which uses a *mixed-excitation* approach (instead of the binary voicing decision) and describes the source signal with additional features (STR) beyond fundamental frequency alone (Zen et al. 2007b).

Mixed-excitation vocoding using MCP and STR features requires a fair amount of signal processing which, however, is effectuated in the time-domain. This means that the audio signal is generated sample-by-sample. Additionally, the input parameters are consumed in a piece-meal fashion, that is, vocoding is incremental by nature. This is, of course, convenient for our incrementality endeavours.

7.2.3 Discussion of Alternative Synthesis Techniques

This subsection presents some alternatives to HMM-based synthesis. Section 7.3 will later explain why HMM-based synthesis was chosen as the basis for incrementalization. The major alternative to HMM-based synthesis is concatenative synthesis which comes in two flavours. *Unit-selection synthesis* works by stitching together short stretches of speech from a large single-speaker database. The database is phoneme-aligned and during synthesis is searched for the best fitting combination of phoneme sequences in the database to the target sequence and concatenate them with as little signal processing as possible, resulting in high quality speech. The search that is necessary to select the best units is based on similarity to the requested targets and pitch contours (matching costs), as well as concatenation costs determined by (spectral) similarity at the joints.

A simplistic form of concatenative synthesis is *diphone synthesis* where every phoneme transition only exists once in the database. As a consequence, there is no search, but acoustic quality is lower as more concatenations and more signal processing is necessary. Diphone synthesis (which has been used incrementally in the past: Edlund 2008) results in low synthesis quality compared to other synthesis methods.

[5] Using 5 ms frames (as opposed to 10 ms for ASR) allows more fine-grained variation allowing for higher naturalness.

A completely different paradigm is *articulatory synthesis* in which the human articulatory organs are simulated and animated in a simulation engine and the resulting waveform is derived from physical formulae. While providing fascinating insight into speech production, current systems[6] do not allow high-quality synthesis of continuous speech, nor do the required computations by the physics model indicate that articulatory synthesis works at the 'right' level of abstraction.[7]

7.2.4 Evaluation of Speech Synthesis

The evaluation of speech synthesis differs fundamentally from that of speech input components (such as ASR), as the evaluation criteria are more complex: we are interested in user perceptions which cannot simply be compared to a gold standard transcription as was the case for ASR.

One major requirement of any speech synthesis system is that it be *intelligible*. Intelligibility can best be measured based on human listening. Even with listening tests, 100 % intelligibility can probably not be reached, as human listeners make mistakes, too.[8] While intelligibility testing was of major importance for decades, speech synthesizers have reached a very high level of intelligibility, with other factors such as naturalness now predominating.

A dialogue system has to handle non- and mis-understandings in any case (e. g. to account for external distractions), so perfect intelligibility is not even very important. However, SDSs for natural interaction require speech synthesis to also sound natural. *Naturalness* determines the proximity of the synthesis to a human's speech production capabilities and can be measured for example by a *mean opinion score* (MOS) in listener ratings (ITU 2006). Large scale naturalness evaluations take place in the Blizzard Challenge (Black and Tokuda 2005) every year with the goal of advancing corpus-based synthesis techniques. MOS testing will be used in Section 7.4 to compare non-incremental and incremental speech synthesis.

An important aspect of synthesized speech is the *prosodic realization*, measurable as segment durations and intonation contours ('pitch tracks'). Prosodic parameters are continuous and changes to a system's prosody generation component will often be gradual. Thus, evaluation by listening tests for all possible changes becomes infeasible and some sort of numeric comparison to some 'ideal' contour via automatically extracted measures is essential. Clark and Dusterhoff (1999) found that *root-mean-square error* (RMSE) between pitch contours best correlated with user-ratings from

[6] For example, Gnuspeech (Hill, Manzara, and Taube-Schock 1995, `http://www.gnuspeech.org`) or ArtiSynth (Vogt et al. 2005, `http://www.artisynth.org`)

[7] It is for a reason that airplanes get along without flapping their wings.

[8] The problem of 'imperfect' listeners in evaluation can be handled by including human speech in listening experiments, which can be used to establish an upper boundary of the system's intelligibility.

perceptual evaluation among a number of measures. The analysis of incremental prosody generation in Section 7.6 will rely on this result.

Finally, one aspect of speech synthesis is hardly ever included into formal evaluations: the fact that synthesizing speech takes time, which can be a significant burden on interaction quality in dialogue systems (and is one reason why applied systems most often revert to *canned speech*, trading flexibility for speed). Especially for longer utterances, synthesis time can be several seconds. This chapter will analyze a strategy for incremental speech synthesis which salvages some synthesis quality in favour of lower processing delays, allowing for better interactions and potentially improving overall system performance.

7.2.5 MaryTTS

MaryTTS (Schröder and Trouvain 2003) has been developed at the German Research Center on Artificial Intelligence (DFKI) since the early 2000s. Being a research-geared system, it was built with modularity, flexibility and extensibility in mind. This especially shows in the internal data representation which is based on XML (Bray et al. 2008) and allows easy output, external manipulation and re-integration of all data at every processing stage. For example, this can be used to bypass any of Mary's internal modules to try out alternative, external modules, as long as their output can be transformed to MaryXML (Schröder and Breuer 2004).

The flexibility of MaryTTS however comes at a price: processing modules exchange XML documents (accessed and manipulated via the Document Object Model, DOM, Byrne et al. 1998) with each module fully scanning and processing the full input before generating output which is only then passed on to the next processing module. As documents have to be complete to be accessible via DOM, Mary's processing scheme is inherently non-incremental and this even though modules are arranged in a plain pipeline. Another disadvantage of the DOM-based data model is that some operations (like tree traversals) incur significant processing overheads and result in slow performance of the implementation.

To find the processing bottle-necks, MaryTTS's execution times were analyzed for some exemplary utterances by recording processing time in the different modules (as reported by debug messages). Figure 7.4 shows the result of profiling the runtime spent in the different modules while processing five test utterances. As can be seen, the early stages of (symbolic) processing such as tokenizing or POS tagging are very fast and all together take roughly 10 % of the overall time. Acoustic modelling (finalizing pitch and duration decisions, and selecting HMM states for later synthesis using classification and regression trees, CARTs) takes 20 % of the share and the majority of time (roughly 70 %) is spent in HMM synthesis. Not shown in the figure, HMM

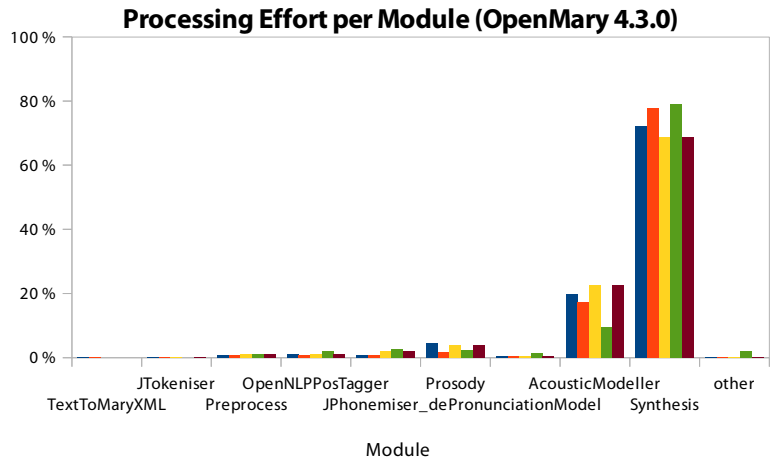


Figure 7.4: Shares of processing time by MaryTTS's processing components for five utterances of various lengths and complexities (OpenMary 4.3.0 with `bits3-hsmm` voice, mean values of three execution runs).

synthesis itself is split into the sub-tasks of parameter optimization and vocoding, which take roughly equal amounts of time.

The following section discusses the integration of MaryTTS' processing modules into the incremental processing framework InproTK to obtain an incremental speech synthesis (iSS) module.

7.3 Incrementalizing Speech Synthesis

This section describes the general approach taken to 'incrementalize' HMM-based synthesis on the basis of MaryTTS, and then describes the ways in which iSS can be used in InproTK.

HMM-based speech synthesis was chosen for incrementalization for several reasons (and even though concatenative synthesis is still more common in commercial TTS systems, Jurafsky and Martin 2009):

First, an HMM-based synthesis nicely separates the production of vocoding parameter frames from the production of the speech audio signal which allows for more fine-grained concurrent processing (see below). This can be seen as a similar distinction as between motor planning and actual articulation in human speech production (Levelt 1989).

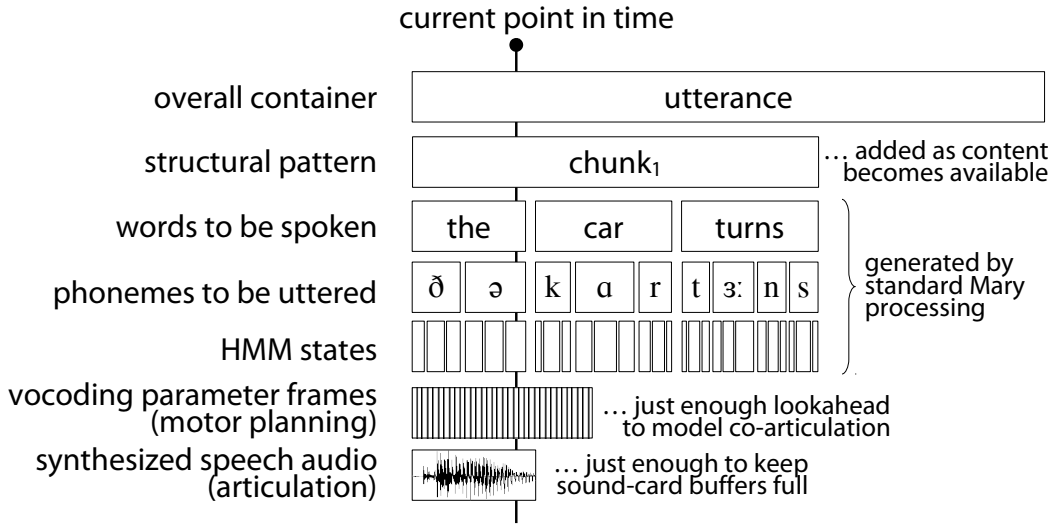


Figure 7.5: Schematic view of the data model and the interplay between MaryTTS and InproTK. Some partial text is passed to MaryTTS which provides words, phonemes, and HMM states, and their durations. HSMM optimization then takes place incrementally with limited contexts, and vocoding provides audio just-in-time.

Furthermore, even though optimization is global for both HMM-based and concatenative techniques as is, the influence of look-ahead on the continuity constraints is fairly constant (and small) while unit-selection with limited look-ahead will jump erratically between unit sequences. Parameters are partially independent in the vocoding frames. This allows to independently manipulate e. g. pitch without altering other parameters or deteriorating speech quality (in unit-selection, a completely different unit sequence might be optimal even for slight changes of pitch).

Finally, HMM synthesis has been shown to work incrementally by Dutoit et al. (2011),[9] however without proper integration into an incremental architecture. The full integration of incremental speech synthesis into the incremental dialogue system architecture is the contribution of the work presented here.

The overall strategy of the integration was to rely on as much of MaryTTS's standard capabilities as possible and to focus on reworking (time-)critical aspects only. As could already be seen in Figure 7.4, the vast majority of processing time is spent in the last of MaryTTS' processing modules, the HMM synthesizer. For this reason, only

this module was made 'fully' incremental, with the preceding symbolic processing pipeline remaining unchanged.

Figure 7.5 presents a schematic overview of the data model and workflow for iSS. When speech is to be synthesized, a sequence of words – not necessarily a full utterance – is passed to MaryTTS for linguistic pre-processing and HMM state and duration assignment. The resulting XML is parsed and turned into word and phoneme IUs.[10]

As can be seen in the figure, the processing scheme does not completely reach the ideal of 'triangular' incremental processing outlined in Section 4.1.2: the results from linguistic pre-processing 'stick out' from the triangle, violating the just-in-time principle.[11] Of course, the current solution for linguistic pre-processing does not really go by the (incremental) book. However, linguistic pre-processing is fast (as shown in Figure 7.4) and hence re-processing in the light of changing hypotheses comes at little cost.

The next step, HSMM optimization, had to be considerably reworked from standard MaryTTS to support incremental processing. Dutoit et al. (2011) have previously shown that the (intrinsically global) HSMM optimization operation can simply be performed phoneme-by-phoneme, using a context of only a few phonemes left and right to model co-articulation. They show that quality (in terms of spectral distortion and subjective ratings) degrades only marginally when using two phonemes of future context instead of the full utterance. Dutoit et al. (2011) appear to have used standard emission optimization without global variance optimization (judging from the voice quality of their system). In contrast, the work presented here implements GV optimization within local contexts. No formal evaluation has been performed, but informal contrastive listening experiments confirm that this compares well with non-incremental GV optimization and is a great improvement over previous incremental HMM synthesis.

Even though the incremental optimization incurs some overhead (because of overlapping local contexts that are optimized repeatedly), the amount of processing before vocoding parameters for the *first phoneme* become available is greatly reduced (cmp. evaluation in Section 7.5 below). Furthermore, the independent parameter streams describing source and filter can be optimized concurrently using multi-threading, which further improves performance compared to MaryTTS's implementation.

As explained in Section 7.2.2.2, the vocoding algorithm is incremental by nature. However, in standard MaryTTS synthesis, the loudness of each utterance is scaled

[9] A fact that was unknown to the author at the time of the decision towards HMM-based synthesis.

[10] HMM states are not contained in the XML representation and are extracted by separate means; they are appended to phoneme IUs instead of forming a separate layer of IU information.

[11] MaryTTS uses CARTs with non-local features to determine HMM states, impeding the incrementalization of this processing step.

in a post-processing step after synthesis. This scaling was changed to an ad-hoc incremental re-scaling method so that audio samples become available as soon as the first parameter frame is consumed.

InproTK's synthesis uses a pull-based approach that is driven by the *audio dispatch* component aiming to keep its output buffer full. The dispatcher pulls audio samples from the vocoder which pulls its parameter frames from the phoneme IUs. Phoneme IUs autonomously perform parameter optimization from the attached HMM states (taking into account their neighbours' states as proposed by Dutoit et al. 2011 for co-articulation) when first queried for a parameter frame. This pull-based pipeline below the phoneme layer – which is convenient for just-in-time querying – stands in opposition to InproTK's push-based incremental processing scheme based on incremental modules (which is convenient to push forward new results as they become available). The two approaches can be mediated by IU updates (as introduced in Chapter 4.2.2) as shown in Figure 7.7, where a higher-level processor (e. g. an incremental natural language generation, iNLG, component) registers as update listener to the chunks that it outputs. It will then be informed about the nearing completion of the chunk and can add a continuation in time. The vocoder updates each PhonemeIU's *progress* field as it passes along.

7.3.1 Incremental Speech Synthesis in InproTK

InproTK provides several interfaces to using and manipulating incremental speech synthesis that cater for different needs. This subsection first presents *utterance trees* for incremental synthesis where pre-planning is possible, and then introduces the generic iSS component in InproTK which supports incremental utterance specification (including revocation of yet-to-be started utterance chunks). Thirdly, the low-level interface to prosody adaptation is presented that allows immediate control over pitch and tempo. Finally, some autonomous hesitation capabilities are presented that reduce the real-time pressure on higher-level components.

7.3.1.1 Utterance Tree-based iSS

Utterance trees for incremental speech output were introduced by Skantze and Hjalmarsson (2010). An *utterance tree* is a prefix tree of possible utterance developments, as shown in Figure 7.6 (left side). Skantze and Hjalmarsson's utterance trees use a granularity of phrases which are synthesized in isolation (and non-incrementally); as a result, there is a trade-off between higher incrementality (which require smaller phrases), and higher synthesis quality (which increases with larger phrases, especially in terms of prosody). In contrast, InproTK's utterance trees are more fine-granular, at the word level., and at the same time linguistic pre-processing is performed on

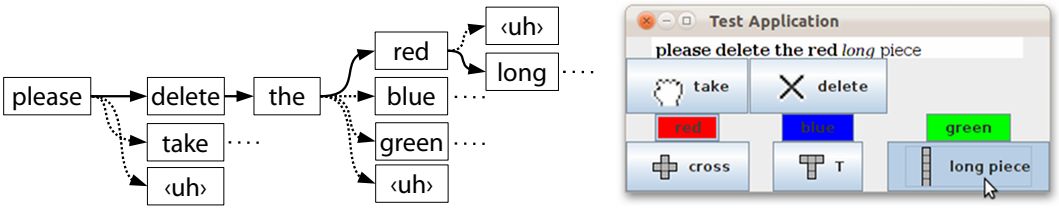


Figure 7.6: iSS based on utterance trees uses multiple forward-pointing same level links, one of which is selected for synthesis (a); selection can be changed until synthesis reaches the link, for example using a demonstration GUI (b).

full paths through the utterance tree, resulting in high-quality connected speech synthesis. As a downside, all utterance variations need to be known (and processed) before the utterance starts (which may incur some utterance-initial delay).[12]

Utterance trees provide for very fast switching between alternative utterance continuations, as all pre-processing has been performed and switching is only a matter of changing forward-pointing same level links (cmp. Chapter 4.1.1). Figure 7.6 (right side) shows a demonstration interface that allows to select utterance branches by clicking appropriate buttons (which change the forward-pointing SLLs) until immediately before the preceding word's last phoneme has been completed.

An early implementation of utterance trees was used in the experiment reported in Section 7.4. Utterance trees can be used whenever all possible utterance alternatives are known beforehand. Trees can be synthesized multiple times, so that even utterances with many alternatives incur only a limited overhead, if they are used repeatedly (e. g. in a deployed SDS).

7.3.1.2 An Incremental Module for Speech Synthesis

Utterance trees have no direct interface to the add/revoke processing scheme of InproTK's incremental modules (cmp. Chapter 4.2.1). Specifically, incremental modules only support one-best hypotheses, whereas an utterance tree holds multiple alternatives. At the same time, incremental modules support later addition of incremental units, as well as revocation and replacement of IUs, which is not supported for utterance trees (especially: a clear interface for this is missing). Hence, an incremental speech synthesis module was developed as an alternative to utterance trees, to better

[12] As a consequence, utterance trees do not scale well to many options, as the tree's paths (and hence pre-processing effort) grows exponentially to the number of variable elements in the utterance.

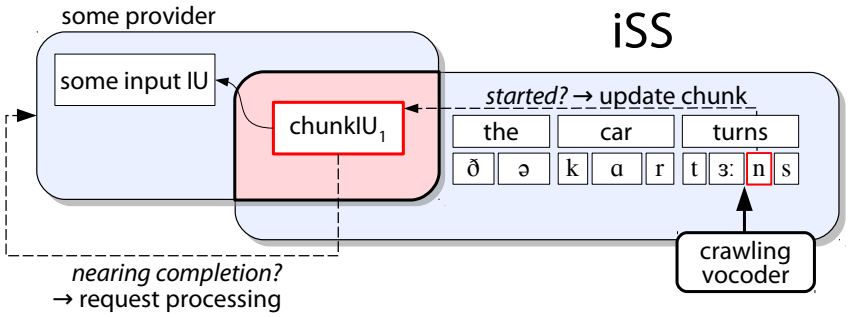


Figure 7.7: IU updates mediate between push-based IU processors and pull-based just-in-time iSS processing.

integrate with the remainder of InproTK and to cater for utterance extension and manipulation.

The iSS module accepts *ChunkIUs* as input. Chunks can contain single words or phrases or even full utterances,[13] and this granularity can be mixed freely. However, revocation of content is limited to full chunks, that is, a providing module needs to decide on the granularity at which chunks may need to be revoked to determine the production granularity. For ease of use, the iSS module also accepts WordIUs, which are automatically converted to chunks.

By default, the iSS module adds an IU update listener to the second to last phoneme of every phrase which updates the ChunkIU as soon as that phoneme is started to be produced. In that way, the providing module can take action before synthesis runs out of material as shown in Figure 7.7.

The phoneme lookahead for updates is configurable and influences prosodic quality; see Section 7.6. A fully incremental output pipeline based on incremental natural language generation and iSS will be presented in Section 7.5.

7.3.1.3 Very Low-Latency Prosody Adaptation

InproTK provides a low-level interface to manipulating planned pitch and tempo. Pitch is one of the parameters that are input into the vocoder at every frame. Hence, pitch can be manipulated until immediately before the corresponding frame is synthe-

[13] It is for this flexibility that the theory-free name 'chunk' was chosen over alternative names, such as 'phrase', which would trigger the question whether intonation phrases, syntactic or pragmatic phrases are meant. In fact, the iSS component is completely agnostic towards this question, though synthesis seems to work best with some sensible phrasing inside chunks (cmp. Section 7.6).

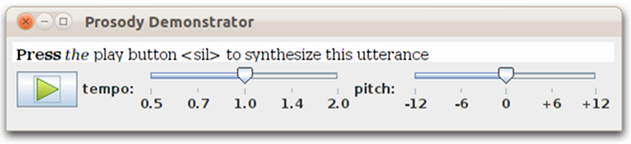


Figure 7.8: A graphical interface to demonstrate and test low-latency prosody adaptations (pitch and tempo) to incremental speech synthesis.

sized. Currently, PhonemeIUs provide an interface to shift the pitch of all contained frames by a given value (in semitones).

Changes to planned durations should ideally re-trigger HSMM optimization to result in high quality synthesis. However, HMM states and their durations are determined in one step by MaryTTS's CARTs (and, as explained above, these CARTs use non-local features, defying incremental processing). Thus, for the time being, a simplistic strategy for duration changes has been implemented: when the duration of a phoneme is changed after pre-processing has been performed, no re-optimization occurs, but the number of frames necessary for the given duration is reached by repeating (or skipping) frames that resulted from the original optimization for increased (or decreased) durations, respectively. This coarse method certainly results in sub-optimal synthesis but allows to easily change the duration even of phonemes that are currently ongoing.

A graphical demonstrator of the low-level interface to prosody adaptation is shown in Figure 7.8 and allows tempo changes up to half/double speed, and pitch changes by one octave up and down that are immediately reflected in ongoing synthesis. While synthesis quality (and naturalness) degrades for large modifications, small adjustments has no impact on synthesis quality and naturalness, according to the author's (informal) testing.

So far, loudness adaptation has not been implemented in InproTK. It could easily be added by scaling the signal magnitude after vocoding (which, however, would not model articulatory changes that correlate with loudness; MaryTTS does not seem to model loudness effects, either). A more thorough approach would be to manipulate vocoding parameters, or to include strength in an incremental HMM state selection, which would also help to model loudness effects on voice quality.

7.3.1.4 Automatic Hesitation

One advantage of incremental speech synthesis is the capability to start speaking before the full utterance is specified. However, as a result the incremental synthesizer

may reach the end of the fully specified part of the utterance before a continuation is available. The standard behaviour of the incremental module in this case is to end the (unfinished) utterance; any later continuation will simply be handled like the start of a new utterance (i. e. without prosodic integration to the preceding part and without notifying the user that some speech production delay is occurring). To avoid this behaviour, special HesitationIUs can be inserted as a precautionary measure at the end of the specified part of the utterance.

Hesitations are realized if no continuation for the utterance becomes available in time (by saying "hm"). Otherwise, if a continuation is available, hesitations are skipped. Skipping works by the hesitation itself automatically changing its duration to 0 as soon as a continuation is same-level-linked to it (the HesitationIU can be said to be an *active* IU). If the continuation becomes available while the hesitation is in progress, it is aborted immediately and synthesis switches to the continuation. Thus, hesitations do not take up any additional time (as in the system by Skantze and Hjalmarsson 2010) but only fill the pause from waiting for the continuation. The realization of a hesitation is currently pre-computed and does not integrate prosodically with the preceding speech. This is, however, not a grave problem as hesitations occur in passages that are disfluent anyways.

7.3.2 Conformance to the Requirements

This subsection summarizes, to what extent the incremental speech synthesis that is part of InproTK meets the requirements set out in Section 7.1.

iSS supports changing the content of ongoing utterances. If all alternative outcomes of the utterance are known beforehand, utterance trees can pre-compute all of them and switching of alternatives remains possible until immediately before one of the options is started to be realized. If alternatives cannot be enumerated beforehand, iSS using the add/revoke-based processing scheme of incremental modules allows to manipulate ongoing utterances. However, this solution requires some lookahead to account for the necessary linguistic re-processing of the utterance and the integration of its results. Furthermore, the later chunks are integrated into the utterance, the more influence this has on overall prosodic quality (cmp. Section 7.6).

As outlined above, synthesis proper has been 'incrementalized' completely in InproTK, with first vocoding frames becoming available after parameter optimization for the first phoneme has been completed and audio samples become available while processing this frame. Subsection 7.5.3.1 gives results for how long this takes in practice. Linguistic pre-processing and HMM state selection remain non-incremental. However, pre-processing can be performed on just an initial chunk of the full utterance with the iSS incremental module.

Both tempo and pitch can be manipulated freely and until immediately before each frame that is to be vocoded (resulting in a maximum latency of 5 ms in addition to audio buffering). Loudness scaling could in principle be performed with even lower latency.

In an experimental system (cmp. Section 7.4), hesitations are inserted that autonomously adapt their own and the preceding few phonemes' duration in the absence of a continuation of the ongoing utterance to account for supply difficulties in the processing pipeline. The standard iSS module does not currently insert hesitations but instead finishes the utterance if no more material is coming in.

The vocoder updates the progress field of every phoneme that it visits (from *upcoming* via *ongoing* to *completed*). Interested components can use IU update listening to be informed about these progress states, be it on the phoneme, word, or phrase level.

Finally, iSS, just like MaryTTS itself, runs at multiple times real time. However, iSS starts to output speech while processing is still ongoing, thus most of the processing time is *folded* into the delivery time, which results in a much more reactive system than when using standard, non-incremental synthesis (cmp. Subsection 7.5.3.1).

7.4 The Merit of iSS

This section investigates the merit of using incremental speech synthesis, as compared to non-incremental speech synthesis, in a highly dynamic environment.

In a *highly dynamic environment* the rate of change in the environment to which the system must reacts occur in intervals that would allow only few utterances to finish as planned. A paradigmatic example of such a domain is sports commentary, which has received some attention in the natural language generation community. For example, Chen and Mooney (2008) present a system that learns from hand-annotated data when and what to comment on. Attention seems to have been placed more on truthfulness of the content, though, as, judging from videos provided on their website,[14] the formulations that are produced are rather monotonic ("`pink7 dribbles towards the goal. pink7 shoots for the goal. pink7 passes to...`"). More importantly for the present discussion, the delivery of a produced utterance does not seem to be tied to the actual temporal occurrence of the events. Repeatedly, utterances are synthesized long after the fact that they describe which sometimes has become obsolete at that point (actually, in the example, the goal is scored while the system talks about some pass).

Lohmann, Eschenbach, and Habel (2011) describe another domain that can be called highly dynamic: a system that adds spoken assistance to tactile maps for the visually

[14] `http://www.cs.utexas.edu/users/ml/clamp/sportscasting/`

impaired. In their settings, users can move around on a computer representation of a map with a hand-held haptic device (which gives force feedback when the device is run into a 'wall'). Users are given spoken advice about the currently traversed streets' names, the relation of streets to each other, and to other map objects in the user's vicinity. Such exploratory moves by users can become rather quick, which in the system they describe can lead to output that comes late, referring to a position that the user has long left. While their prototype system was rated as helpful in a user study, many of their test subjects noted that advice was often uttered very late (Lohmann, Kerzel, and Habel 2012). This can be explained partially from the delays incurred by generating and synthesizing the whole utterance before starting to speak it, and additionally by the fact that ongoing utterances could not be changed (Kris Lohmann, p. c. Hamburg 2012).

7.4.1 Domain and System

The example domain that is used in the system presented here is a highly dynamic commentary domain (which was briefly introduced in Figure 7.2 and which is depicted in Figure 7.9), combining properties of the domains mentioned above (sports commentary and map exploration): the *CarChase* domain. In the domain, a car drives around the streets on the map and a commentator (supposed to be sitting in a helicopter observing the scene from above) comments on where it is driving and what turns it is taking.

The car's itinerary in the domain simulator is scripted from a configuration file which assigns the target positions for the car at different points in time and from which the simulator animates the motion and rotation of the car. The speed of the car is set so that the event density is high enough that the setting cannot be described by simply producing one utterance per event – in other words: the domain is highly dynamic.

The task for the commentator is to generate a natural spoken commentary, where the temporal proximity of events and related spoken realization is important to ensure relevance (i. e. situations such as the one in the system by Chen and Mooney (2008) where the system still talks about a pass while a goal is scored, should be avoided).

The target for this section is to assess the principled advantages that incremental speech synthesis brings about, but not the possibility of incrementally (or non-incrementally, for that matter) generating the natural language expressions for the commentary task. Thus, no NLG component is used to generate commentary based on the car's motions. Instead, the commentary is scripted from the same configuration file that controls the car's motion on the board. However, great care was taken while devising the configurations not to put unrealistic assumptions about NLG into the hand-written scripts: events are only to be commented on, once a specific event (such

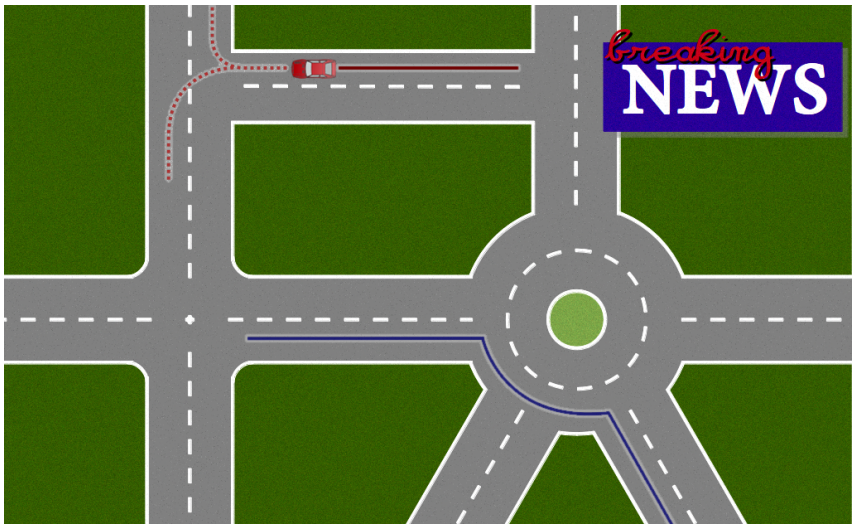


Figure 7.9: The map shown in the CarChase domain, including the car on one of its itineraries (red). At the depicted moment, it can be assumed that the car will take a turn, but it remains unclear whether to the left or right. A second itinerary is shown in blue below.

as turning right) has become visible on the board. However, some reasonable foresight (based on domain knowledge) is assumed, such as figuring out that the car *will* turn (either way) when it approaches a T-junction (as depicted in Figure 7.9, red path). The example in Figure 7.2 showed that incremental speech synthesis can make use of the partial 'turn' information by generating a partial utterance and filling in the direction of the turn event later. In contrast, a non-incremental system can only start to inform about the event once it has actually occurred (at t_3 in Figure 7.2).

Some other settings were included which are less clear cut than the previous example: the lower, blue path in Figure 7.9 results in the incremental system uttering "`The car enters the roundabout and takes the first uh second exit`", that is, it has to correct itself. Furthermore, events can actually come later than expected by the incremental realizer, which, in the implemented system, results in hesitations ("`hm`") being uttered until the event occurs and triggers the continuation of the utterance. In such situations, incremental output realization need not necessarily be advantageous.

The system as used in the evaluation reported below was based on an early version of utterance trees (cmp. Subsection 7.3.1.1) which still suffered from several bugs in parameter frame handling, resulting in spectral distortion at branches in the utterance

tree. Furthermore, prosodic processing was limited to individual chunks of words, so that there was no prosodic integration of the full utterance.

7.4.2 Evaluation

To evaluate the incremental system's realizations, it is compared to a non-incremental baseline system which is unable to alter speech incrementally and hence cannot use the method of extending an ongoing, partial utterance. Instead, the baseline system always produces full utterances (which contain commentary on single events, instead of concatenating descriptions for events to complex utterances as in the incremental system). In order to ensure the temporal proximity of delivery with the causing event, it was assumed that an NLG for the domain could mark utterances as optional: incoming optional utterances are skipped if the component is still speaking the remainder of the previous utterance; non-optional utterances abort and replace any ongoing utterance. All 'turn' events in the domain were marked as optional, all street identification events were marked as non-optional.

4 different configurations were constructed in which the timing of events was varied (by having the car go at different speeds, or by delaying some events), resulting in 9 scenarios in total. Both systems' output for the 9 scenarios was recorded with a screen-recorder, resulting in 18 videos that were played in random order to 9 participants (university students not involved in the research). Participants were told that various versions of commentary-generating systems generated the commentary in the videos and that the commentary was generated purely on the basis of the running picture in the videos. The participants were then asked to rate each video on a five-point Likert scale with regards to how natural (similar to a human) the spoken commentary was (a) formulated, and (b) pronounced. In total, this resulted in 81 paired samples for each question.

The assumption (and rationale for the second question) was that the incremental system's formulations would result in higher ratings in the first question (regarding human-like formulation). At the same time, it was hoped that the acoustic (and prosodic) artifacts resulting from incremental processing would not lead to a significant downgrade in the second question (regarding human-like pronunciation). In order to not draw the subjects' attention towards incremental aspects, no question regarding the timeliness of the spoken commentary was asked for explicitly.

The incremental system generated one or more hesitations in 3 of the 9 scenarios. Of course, hesitations make overt a planning/formulation problem and can be expected to result in lower ratings in the formulation question. Furthermore, synthesis quality is especially weak surrounding hesitations (in the early prototype) so that low pronunciation ratings can be expected as well.

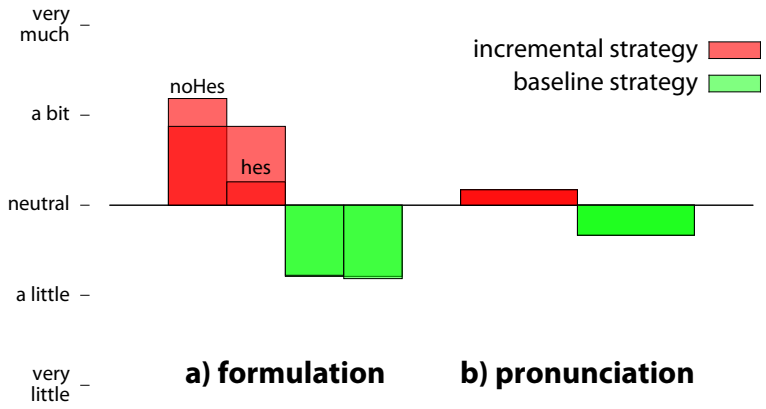


Figure 7.10: Mean ratings of formulation and pronunciation quality for the incremental system and the baseline system. Formulation quality is differentiated between utterances requiring hesitations in the incremental system.

7.4.3 Results

The mean ratings for both formulation quality and pronunciation quality for the incremental and baseline (non-incremental) systems is shown in Figure 7.10. As can be seen in the figure, mean ratings for the incremental system are higher than for the baseline system. The median of the differences between the ratings for the two conditions is 2 points on the Likert scale for question (a) and 0 points for question (b) (means of 1.66 and 0.51, respectively), favouring the incremental system.

The sign test shows that the advantage of the incremental system is clearly significant for question (a) (68+/9=/4-; $p < 5e\text{-}16$) and question (b) (38+/30=/13-; $p < .0007$). Regarding question (b), one could argue that different formulations may have entailed different effects on pronunciation quality and hence paired testing is inadequate. A non-paired t-test for question (b) also shows the highly significant advantage of the incremental system ($p < .0012$).

Thus, it is safe to say that the production strategies enabled by incremental speech synthesis (i. e. starting to speak before all evidence is known and extending the utterance as information becomes available) allows for formulations in the spoken commentary that are favoured by human listeners.

Interestingly, the subjects in the study also rated the incremental system's pronunciation as more natural than the non-incremental synthesis (however, note the larger proportion of equal ratings for that question). It is a fact that the manipulations

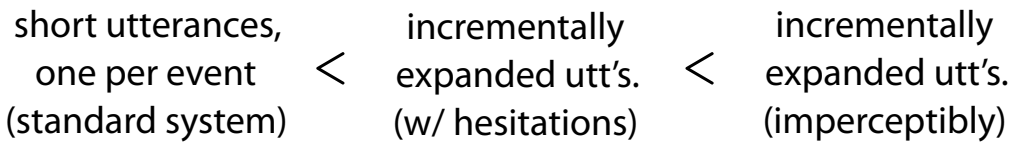


Figure 7.11: User preferences of system behaviour in the *CarChase* domain: incrementally completed utterances that integrate events into the ongoing synthesis are preferred over standard behaviour, even if this requires short hesitations.

to synthesis required for incremental processing (and the bugs that existed in the early prototype that was used in the experiment) can only systematically result in a deterioration of the synthesis quality. It appears that subjects pardoned bad synthesis quality (which occurs in both system versions for certain words) more easily when overall formulation quality is better. This is also evidenced by the fact that ratings for the questions are moderately correlated (Pearson's $r = .537$).

It turns out that subjective ratings for the incremental system in the 3 scenarios containing hesitations were significantly worse than those scenarios without hesitations, for both formulation and pronunciation (t-tests, $p < .001$ and $p < .01$, respectively). This result indicates that subjects do not simply accept system hesitations as inevitable (given that there was simply no evidence yet where the car would turn, for example).

However, when comparing the incremental system with the baseline system for these 3 scenarios that required the incremental system to output hesitations, the incremental system's formulations with hesitations are still rated significantly better than the baseline system's (sign test, 18+/5=/4-; $p < .005$) while there is no effect on pronunciation in these cases.

7.4.4 Discussion

This section has aimed to assess the merit of iSS, that is *whether it's worth the effort*. An early prototype of iSS was used in a highly dynamic environment, where non-incremental synthesis could only be used with simple, short, and hence uninspiring utterances due to the high rate of change in the environment.

This author would argue that naturalistic, conversational settings of all kinds can be considered "highly dynamic", as they all require constant revisions and reactions to external events (such as listener feedback or the absence thereof, Clark 1996). However, it was much easier to make the point here with a proof-of-concept system in the simple, yet highly dynamic commentary domain.

In the commentary domain, a clear user preference for system behaviour emerges, as shown in Figure 7.11: subjects prefer utterances that are incrementally expanded as more information becomes available and can be integrated into the ongoing utterance (i. e. system behaviour that is enabled by incremental speech synthesis) over standard behaviour where every event results in a relatively short system utterance which is not connected (in terms of formulation or prosody) with any preceding or already ongoing utterance. This preference remains even if incremental utterance realization requires hesitations in order to keep the utterance going in cases where utterance timing has been mis-planned (i. e. the system should have spoken more slowly, or started later, because it runs out of material before more content becomes available). However, systems should aim to hesitate as rarely as possible because subjects dislike utterances with hesitations when compared to utterances where incremental realization goes smoothly and is imperceptible to the user.

Finally, and unexpectedly, subjects rated higher the *pronunciation* quality of iSS compared to non-incremental synthesis even though objectively it was lower. However, the speech that was synthesized was more *adequate*. A possible conclusion is that synthesis quality actually matters very little in comparison to interaction quality, and that speech synthesis systems should be evaluated in context, or at least taking into account the sorts of interaction behaviour that they support.

7.5 Example Application: Integration with Incremental NLG

This section highlights the technical feasibility of using iSS as part of an incremental spoken output pipeline for a spoken dialogue system, by show-casing an example application for adaptive information processing (as originally presented by Buschmeier et al. 2012). One indication in the previous section was that subjects appear to prefer the longer, integrated utterances that were made possible by iSS over short, separated, staccato utterances that systems based on conventional synthesis would be forced to fall back to in order to achieve reactive behaviour. Furthermore, the only incremental output generated in one of the interactive example systems so far, were very short feedback utterances in the *Pentomino Select* system in Chapter 6.1.2. This section specifically investigates how an incremental output pipeline can be used to realize longer utterances in an adaptable fashion and with little utterance-initial delay as is required for use in a highly interactive system.

The following subsection introduces and discusses the exemplary use-case for the incremental output pipeline, and Subsection 7.5.2 briefly describes the implemented system which is evaluated in Subsection 7.5.3. Subsection 7.5.4 discusses the results.

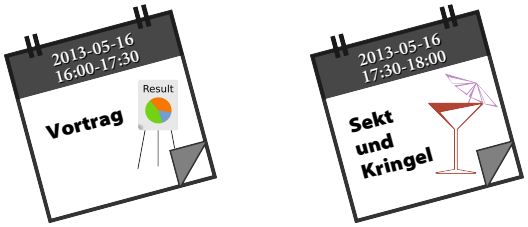


Figure 7.12: Example of two consecutive events in the calendar domain.

7.5.1 Use-Case: Adaptive Information Presentation

The exemplary use-case for adaptive information presentation is the *Calendar* domain. In the domain, calendar events consist of a title, date, and duration. Information presentations that are to be read out by the system can be the reminder of one or multiple, possible sequential events, the rescheduling of events (e. g. caused externally by a secretary), or scheduling conflicts that may need to be resolved. Figure 7.12 shows two consecutive calendar events as an example.

Rescheduling or addition of events could potentially be triggered while information presentation is in progress, but this will be rare in practice. For this reason, an additional noise source is introduced into the domain such as might occur while using the *Calendar* system on the side of a busy street: randomly, every 2 to 5 seconds, a one-second random pink noise event is played that makes understanding any ongoing speech impossible. These noise events turn the domain into a highly dynamic environment, as they prevent the successful non-incremental delivery of almost all information presentations in the domain. Instead, a successful system must adapt to the noise.

Of course, the interspersed noise is just a placeholder for any type of adaptation signal that co-occurs with the system's speech. Other possible signals could be listener feedback (Buschmeier and Kopp 2012) that should results in adaptation to the system's ongoing speech output.

In the *Calendar* domain, noise events occur randomly and hence systems have to react to them on-the-fly. This distinguishes the domain from the *CarChase* domain in the last section, where it was possible to pre-compute an utterance tree to encompass all alternative developments in the given configuration file. The *Calendar* domain hence takes another step to realistic conversational environments.

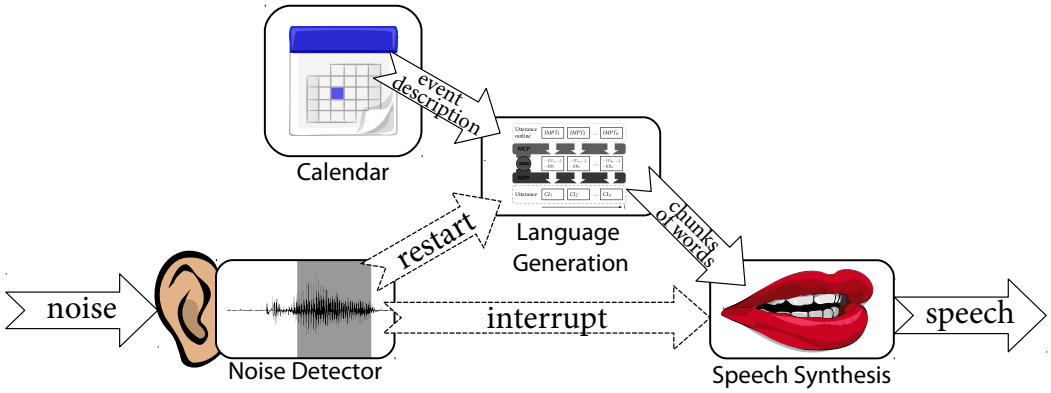


Figure 7.13: Interplay of components in the system for adaptive information presentation: when noise is detected while information is being presented, synthesis is interrupted and iNLG is re-triggered once the noise has ended.

7.5.2 Implemented System

A schematic overview of the system for adaptive information presentation is shown in Figure 7.13: an event description that is to be presented is passed to the incremental natural language generation (NLG) module, starting the processing chain. As explained by Buschmeier et al. (2012), iNLG determines the overall layout of the utterance to be produced (the micro-content plan), dividing the utterance to be produced into shorter sub-units that correspond roughly to intonation phrases. Event descriptions in the domain can be complex and there are usually six to seven sub-units for a full utterance. Sub-units are in the form of *incremental micro-planning tasks* (IMPTs). For every IMPT, the surface realization component of the NLG generates the corresponding words to be produced using the SPUD approach (Stone et al. 2003). Only this latter surface realization task is performed incrementally, at the granularity of IMPTs; it is, however, the computationally expensive part.

The surface realization of the IMPT (a sequence of words) is then sent as one *ChunkIU* to the iSS module, and integrated into ongoing speech as described in Subsection 7.3.1.2 above. Processing occurs *just-in-time*, with the surface realization component being triggered into action by a previous chunk's nearing completion, via IU update messages as was already shown in Figure 7.7 and is repeated for convenience in Figure 7.14. The implemented system uses a relatively large lookahead, as a precautionary measure to keep the prosodic influence of incremental processing as low as possible. Specifically, the system realizes two chunks utterance-initially (as can

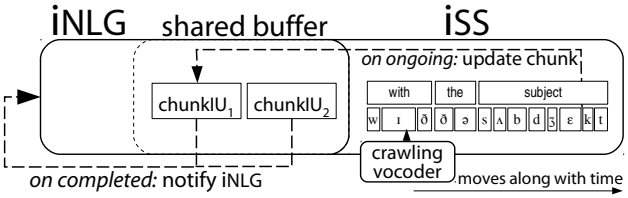


Figure 7.14: IU updates are set up to always keep one fully generated but not yet realized chunk of words as a buffer, to avoid real-time issues and to minimize the influence of incremental prosody generation.

be seen in Figure 7.14 and then adds a next chunk when the currently pen-ultimate chunk is started to be synthesized (see the following section for a thorough discussion of how much lookahead would should be used).

Another module in the system, the noise detector, connects to both iSS and iNLG. On noise onset, it informs iSS to interrupt the ongoing utterance after the current word. This interruption is realized in the iSS module internally by breaking the forward-pointing same-level links leading the crawling vocoder to finish after the currently ongoing word, as no next word can be retrieved. iNLG is informed on the end of the noise burst, triggering it to regenerate the interrupted sub-utterance chunk and to re-send it to iSS. It should be noted that the implemented system does not in fact use a real noise source and noise detector. Instead, the random noise simulator plays back a burst of 1000 ms of pink noise every 2 to 5 seconds and informs the other modules with a delay of 300 ms after noise starts and ends, respectively. A real noise detector should be able to give accurate detection quality with a similar or even lower delay.

In order to react to external events (such as noise having interrupted an ongoing IMPT), the surface-realization component of iNLG is able to adapt the verbosity and redundancy with which chunks are realized (Buschmeier et al. 2012). For example, the title of an event could be realized simply as "`Vortrag`", or "`Betreff: Vortrag`", or "`mit dem Betreff Vortrag`", depending on how verbose the item should be verbalized. Redundancy of presentation applies to dates, for example, where an event can be characterized as "`tomorrow`", or "`May 15th`", or both: "`tomorrow, Mary 15th`". While offending the optimality criterion, redundant information can serve communicative functions and help to increase the probability of the message being understood (Buschmeier et al. 2012, citing Reiter and Sripada 2002).

The implemented system is lacking a model to decide what changes in verbosity and redundancy should ideally be taken when re-phrasing an IMPT. Specifically, it

does not take into account whether an IMPT is aborted early on, or only late in the phrase (or more specifically, does not relate the onset of the noise event with what information has already been conveyed by the synthesizer, cmp. Matsuyama et al. 2010). Instead, the system uses a simple heuristic (which is not based on an empirical study or deep theoretical insight): verbosity is reduced by one level, and redundancy is increased by one level for the first chunk after a noise burst, which results in a noticeably different surface realization. Furthermore, the interruption by noise leads to the iSS generating the first chunk after noise with a new sentence onset intonation.

The system also implements two baseline behaviours: one is to ignore noise altogether (as if the system did not notice it) and handles the domain as if it were not highly dynamic, the other is to pause playback of the ongoing utterance on noise onset (potentially in the middle of a word) and to resume where it left off after the noise burst. This pause/resume behaviour can be considered state of the art (Edlund 2008).

7.5.3 Evaluation

This subsection discusses the performance of the implemented system for the *Calendar* domain both in terms of measured reactivity for the utterance onset, and in user ratings of the full system when compared to simpler (previously state-of-the-art) solutions.

Analyses are performed on 9 settings within the calendar domain (4 sequences of two events, 3 conflicts between events, and 2 reschedulings) that result in 6 or 7 phrases each. Both number of words and total audio duration depend on the interspersed noise. If undisturbed by noise, utterances contain on average 27 words (composed of on average 5.3 phonemes) and take approximately 10-12 seconds each.

7.5.3.1 System Response Time

This subsection evaluates the *system response time*, that is, the time until the first audio sample is delivered to the sound card after the utterance start is requested. As was mentioned multiple times, incremental processing can take advantage of *folding* processing time into delivery time, which is especially relevant for long utterances (that incur longer processing times), and adaptive behaviour (where it becomes impossible to pre-compute all possible realizations).

A non-incremental system's response time is the sum of the times taken by all modules involved to do their work. An incremental system, in contrast, can *fold* large amounts of its processing time into delivery time; what matters is the sum of the *onset times* for each module, i. e. the time until a first output becomes available for the next module to start processing.

Table 7.1: Processing time per processing step before delivery can begin (in ms; averaged over nine stimuli taking the median of three runs for each stimulus; calculated from log messages; code paths preheated for optimization).

	non-incr.	**incr.**
iNLG-microplanning	361	52
Synthesis (linguistic pre-processing)	217	447
Synthesis (both HMM and vocoding)	1004	21
total response time	1582	519

Table 7.1 summarizes the runtime for the three major steps in output production of the system using the 9 exemplary settings (without interspersed noise), for full, non-incremental processing, and for incremental processing of the onset only. As can be seen, especially iNLG microplanning and speech synthesis proper profit greatly from incremental processing. For both processing steps, processing efforts scale with the characteristic processing unit sizes (chunks, and phonemes/HMM optimization frames, respectively).

In contrast, linguistic pre-processing using MaryTTS does not scale with the number of words processed. Instead, constant processing overheads (parsing and regenerating XML documents) appear to be the limiting factor, and occur per call to MaryTTS's linguistic pre-processing pipeline. Furthermore, the iSS module is flawed in that respect that multiple chunks that are added to the input buffer in one step result in multiple calls to linguistic pre-processing, with the second call's result immediately superseding the first. As the system uses two chunks utterance-initially, the time taken for linguistic pre-processing doubles for the incremental system. (This flaw could easily be fixed, however Section 7.6 below shows that the use of two chunks of lookahead is more than would be necessary, making this issue irrelevant for future systems.)

Taken together, incremental processing reduces the system response time by over one second as compared to non-incremental processing. Furthermore, the utterance-initial delay would be further reduced to around 300 ms if iSS would avoid redundant calls to linguistic pre-processing.

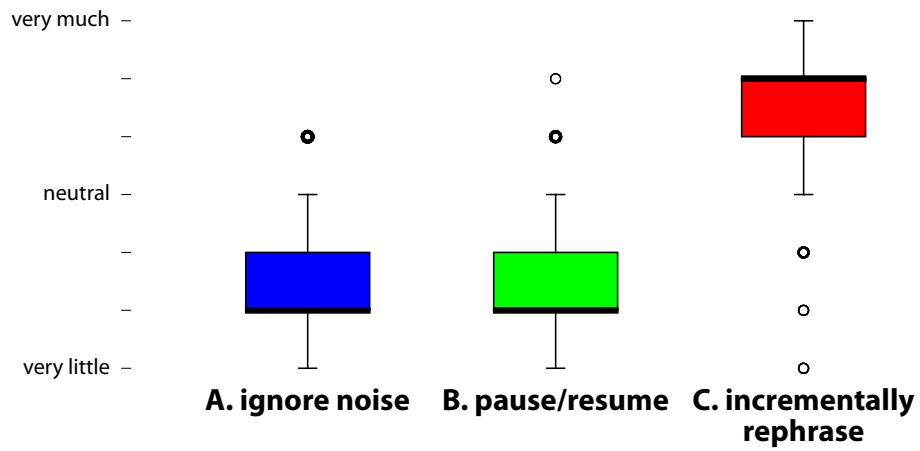


Figure 7.15: Boxplots showing the subjective naturalness ratings for the three systems in the *Calendar* domain.

7.5.3.2 Subjective Evaluation

This subsection reports the results of a user study that evaluated the naturalness of the implemented strategy for adaptive information presentation in the *Calendar* domain.

Utterances for the same 9 exemplary settings as above were recorded with interspersed random noise using the baseline system **A** that completely ignores noise, the pause/resume baseline system **B** (which, however, does not adapt the surface realization to noise), and the full, incremental system **C** that adapts and re-synthesizes after the noise burst, starting from the chunk that was being produced at noise onset.

The resulting 27 stimuli were played to participants in random order. Additionally, before playing the stimuli that contained noise, two examples were played that did not contain any noise interruptions so that participants could get an impression of the system's utterances in the domain.

Twelve PhD students (not involved in the research) listened to and rated the stimuli. Participants listened to each stimulus once and rated it immediately afterwards by noting their agreement to the statement "I found the behaviour of the system in this situation as I would expect it from a human speaker" on a 7-point Likert scale.

Boxplots showing the ratings for the three systems are shown in Figure 7.15. As can clearly be seen in the figure, the incremental system was rated much more natural (median rating of +2 on the scale against −2 for both baseline systems). A Friedman rank sum test supports this impression, revealing a highly significant difference between the perceived human-likeness of the three systems ($\chi^2 = 151, p < .0001$). A

post-hoc analysis with Wilcoxon signed rank tests found *no* significant difference between systems **A** and **B** ($V = 1191.5, p = .91$). The fully incremental system **C** however differed highly significantly from either baseline system ($V = 82, p < .0001$ for system **A** and $V = 22.5, p < .0001$ for system **B**).[15]

7.5.4 Discussion

This section has shown that fully incremental output in a highly dynamic environment using IU modules is feasible and successful. While IU module pipelines based on InproTK had previously been applied to input processing, they were here successfully applied to output processing which is subject to real-time pressure.

The just-in-time incremental output production enables the implemented system to start outputting rather long and complex information presentation utterances with a vastly reduced response time (specifically, a reduction by over a second, which could be further reduced to about 300 ms, see next section) as compared to non-incremental processing.

This speed-up enables adaptive, incremental behaviour that was rated as significantly more human-like than either of two baseline strategies. It is noteworthy that the baseline that represents the prior state of the art – pausing during and resuming as if nothing had happened after noise – was not rated differently than the simple, ignorant baseline behaviour in terms of human-likeness. This is a clear indication that pure "surface-based" reactions are insufficient for receptive behaviour and that a re-generation across processing layers is necessary, which requires a flexible and structured incremental architecture such as the IU architecture.

Results for the *recall* of the information presented to users can only be estimated as recall was not measured in the user evaluation: recall for the ignorant system **A** would certainly be negatively impacted by noise, as information presented during noise would simply be inaudible. While the pause/resume system **B** does not present information during noise, it requires the listener to remember the partial phrase (possibly partial words as interruptions will often occur in the middle of a word) which may be harder than if the interrupted phrase is later repeated (and rephrased) as in the incremental system.

The *Calendar* system in this section always uses a relatively large prosodic lookahead of two phrases (ChunkIUs), that is, it did not focus on delaying higher-level decision making. In the system, the overall layout of the utterance was fixed by NLG once in the beginning of the utterance and could not be revised, as the domain did not call for integration of new information to be presented, but rather to adaptation in delivery as a response to external events (in this case noise). Thus, incrementality was mostly

[15]The author is highly indebted to Hendrik Buschmeier for performing these statistical analyses which were first reported in the joint work (Buschmeier et al. 2012).

used to fold processing time to result in high responsiveness, and not so much to defer decisions to as late as possible as was the case in the *CarChase* system, where the information to be conveyed could be changed until immediately before the fact, at the cost of prosodic quality. The following section investigates the trade-off of late decisions vs. prosodic quality between the two extremes covered so far.

7.6 Evaluating the Prosodic Quality of iSS

Section 7.4 showed that even if iSS produces clearly reduced prosodic quality (and acoustic artifacts), users prefer the interactive behaviours that are enabled by iSS. Users even rated the pronunciation quality as better than that resulting non-incremental processing, even though objectively it cannot have been. Section 7.5 'played safe' the question of prosodic integration by using two iNLG chunks (roughly similar to intonation phrases) of lookahead for prosodic processing, in order to reduce the influence of incremental processing on prosodic quality.

Relating the issue of prosodic integration to the timing metrics introduced in Chapter 3, these two extremes can be seen as a very late *first occurrence* (F0) of material to be included into the utterance for the *CarChase* system and a relatively early F0 for the *Calendar* system. Chapter 3 discussed timing mostly in the context of non-monotonous input processing, specifically that there are trade-offs involved between timing and the *degree* of non-monotonicity (cmp. Section 3.3.2.4). Chapter 5 then developed strategies to decrease *edit overhead* at the cost of some timeliness of hypotheses (cmp. Section 5.5). Input processing *can* be completely non-monotonous (i. e. hypothesis changes may come arbitrarily late) and the InproTK iSR module opts to infinitely change hypotheses if need be to meet the yieldingness criterion (i. e. final hypotheses are identical to a non-incremental processor's results). In contrast, output processing is restricted by real-time pressure: hypotheses can only be changed *as long as they haven't been realized yet*. Thus, for output processing, the question is how the timing of hypotheses influences the quality of the final result.

It should be noted that iSS in general (as implemented in InproTK) is agnostic to the size of units that are added for later synthesis, as well as to the lookahead, that is, how long before they need to be realized units are added. However, both lookahead and granularity are crucial in order to devise a plausible prosody.

This section, based on (Baumann and Schlangen 2012b), sets out to systematically analyze the trade-off between the lookahead used when integrating more speech material and the resulting prosodic quality. The following subsection discusses the design space for incremental prosodic processing and Subsection 7.6.2 reports an experiment in the *Calendar* domain aiming to find plausible sweet spots in the trade-off.

7.6.1 The Design Space for Incremental Prosody Production

Prosody is influenced by *long-range dependencies*, as for example phrase intonation depends on the finality of the phrase, and rhythm clashes potentially many words ahead may influence stress assignment (Levelt 1989). Nonetheless, human performance indicates that most often prosody can be (and is) generated on-the-fly; Levelt (1989) claims that most often rhythm can be assigned with a one word lookahead, and intonation with even less. Current main-stream speech synthesis systems, as outlined in Section 7.2, perform linguistic pre-processing non-incrementally on full utterances, instead. The iSS component described here relies on such processing, which is effected repeatedly. The goal here is to generate a prosodic assignment for the utterance incrementally (with only limited amounts of the utterance available and requesting further material just-in-time) that matches the prosody that would be produced non-incrementally by linguistic pre-processing closely, but with limited lookahead.

Imagine a system incrementally produces the utterance in Example 7.2 (with "`|`" denoting chunk boundaries from iNLG):

(7.2) "`am 14. Mai | zehn bis zwölf Uhr | Einkaufen auf dem Wochenmarkt`"

When synthesis starts with the first chunk ("`am 14. Mai`"), the questions arise of *when* more words need to be added, and of *how many* words should be added at a time. These are the questions of lookahead and granularity, respectively. (Of course, non-incremental processing would add all words immediately.) When more words are processed, there is also the question of how much of the already known words should be taken into account as left context. The full design space for incremental prosody production is illustrated in Figure 7.16.

The implemented system adds material at the *granularity* of chunks (as they are generated by the iNLG component) which roughly correspond to prosodic phrases. A finer granularity (e. g. at the word level) could potentially improve timing behaviour, as it allows to change upcoming speech until later in the utterance (under the assumption that it is hard to change anything that is already ongoing, which is a limitation of the current implementation). However, adding full phrases at a time has the advantage of feeding the non-incremental linguistic pre-processing pipeline with 'sensible' input for prosodic assignment. In other words: querying for "`zehn bis zwölf Uhr`" will likely better match the prosody of the full utterance than querying for "`14. Mai zehn bis`". Thus, the analysis in this section is limited to the granularity that is provided by the iNLG component from Buschmeier et al. (2012).

Using *left context* may provide important clues to symbolic linguistic pre-processing about the overall layout of the utterance. Furthermore, Subsection 7.5.3.1 concluded

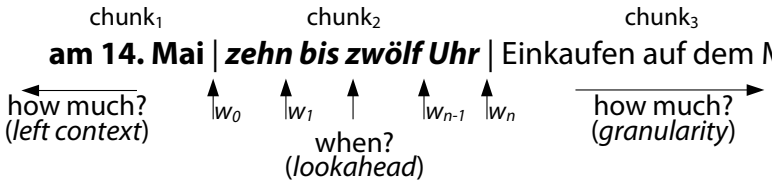


Figure 7.16: The design space for incremental prosody production. When appending the chunk "`Einkaufen auf dem Wochenmarkt`", the questions arise of (a) when to do this (question of lookahead), (b) how much to add at a time (question of granularity), and (c) how much of the preceding material to reconsider (question of left context).

that linguistic pre-processing as implemented is dominated by constant processing overheads. Thus, there is no need to limit the left context.

Regarding *lookahead*, it has to be determined when a next chunk of words is requested for prosodic integration. The just-in-time principle dictates to do this as late as possible. it is the definition of *possible* that matters here: ignoring prosodic continuity, a next chunk should be integrated only immediately before the currently ongoing chunk finishes (at position w_n in Figure 7.16 for the chunk "`Einkaufen auf dem Wochenmarkt`"). However, to fully respect prosody and its long-range dependencies, even just-in-time processing would require the full utterances immediately (to meet this definition of "as late as *possible*").

However, combined with using left context, a smooth transition to a next chunk can be produced by re-computing even the current phrase in light of its now known continuation and adapting the prosody of those parts of it that have not been realized yet. In the example, when chunk_3 is appended, it can be processed together with chunks 1 and 2 as left context. It is very likely that now the part of the utterance corresponding to chunk_2 will be assigned an intonation that is much better suited for its in-utterance position than before. When doing these re-computations in time (by using lookahead), this better prosodic realization can be used instead of the originally computed one for the yet unspoken part of chunk_2.

The trade-off between the timeliness of possible speech manipulation and the realized prosodic quality is explored systematically in the following subsection.

7.6.2 Experiment

This subsection explores the influence of lookahead on prosodic quality in incremental speech synthesis. The following settings for lookahead are used as experimental conditions:

non-incremental standard, non-incremental prosodic processing is used as the control condition and equates to an infinite lookahead.

trivially incremental synthesize every phrase in isolation when the current phrase ends, that is, using a zero lookahead (this strategy is used by Skantze and Hjalmarsson 2010, and can probably be considered state-of-the-art)

only left context use a lookahead of zero, but integrate all previous material as left context, allowing for better prosodic connection, especially at the onset of phrases; this setting corresponds to the arrow labelled w_n in Figure 7.16.

full phrase lookahead require the next phrase before starting production of the current phrase (so that the current phrase can be fully reconsidered taking the next phrase into account); this setting corresponds to the arrow labelled w_0 in Figure 7.16, and was used in the original *Calendar* system.

Additionally, **intermediate settings** are denoted as w_i: w_1 is the setting that processes and integrates the next chunk during production of the first word of the current phrase, w_{n-1} integrates one word before the end of the current phrase, and so on (of course, w_{n-1} can be earlier in the utterance than e. g. w_3 for short phrases). (Hence, 'full phrase lookahead' and 'only left context' conditions can be described as w_0 and w_n, respectively.)

In all conditions, update granularity is kept constant with full phrases being added at a time. Apart from the *trivially incremental* condition, all conditions make use of full left context.

It is expected that providing left context in the w_n setting as compared to the trivial setting will improve quality (as measured by similarity to non-incremental, full-utterance synthesis) of the beginning of each new chunk (as it is realized taking into account the preceding material). Increasing lookahead (i. e. moving the addition of more material 'to the left' in Figure 7.16) should additionally improve quality of the endings of each chunk (as the ending is re-computed and integrated into ongoing synthesis, taking into account the material of the next chunk). These assumptions will be tested below.

7.6.3 Evaluation

For simplicity, the utterances from the *Calendar* domain are re-used (without interspersed noise), and synthesized using the lookahead conditions introduced above. Technically, this is realized by varying the position of the update triggers as was shown

above in Figure 7.7. (For short (two word) phrases the trigger position was adjusted to the last word in the w_3 condition.) As was the case with iSR evaluations in Chapter 5, an incremental system can impossibly reliably outperform its non-incremental sibling. Hence, the focus is again on the similarity of incrementally produced results with non-incremental results from an otherwise identical system.

The synthesizer's generated prosody can be measured in terms of pitch (f_0 of every voiced frame) and phoneme durations. Other factors, such as for example vocal effort are not actively influenced by linguistic pre-processing (and would be much more complex to evaluate). Phoneme durations for two different realizations of the same utterance are compared phoneme by phoneme. A frame-by-frame comparison of pitch, however, is untruthful, as pitch tracks would diverge in light of different phoneme duration assignments. The evaluation below normalizes pitch tracks to the durations in the gold standard, that is, missing or extra pitch marks in phonemes of different duration are ignored, to keep pitch in both tracks aligned.[16]

7.6.3.1 Qualitative Analysis

Figure 7.17 shows part of exemplary pitch tracks generated in different settings (omitting some intermediate settings for clarity; phoneme durations normalized to non-incremental condition) for one of the synthesized utterances.

As can be seen, the prosody of the output in the *trivially incremental* condition (shown in red) deviates rather strongly from the non-incremental condition (shown in black), both at phrase beginnings and endings. Phrase beginnings receive a strong initial pitch excursion as may be appropriate for the beginning of a full utterance, but less so for the middle of the utterance. Likewise, sentence-end intonations with falling pitch and longer durations (not shown in the figure because of temporal normalization) are assigned to the end of every phrase in the *trivially incremental* condition.

The w_n(*only left context*) condition, which is triggered during the last word of the ongoing phrase, successfully reduces the deviation s at phrase beginnings (avoiding onset intonations within the utterance). However, sentence-end intonations remain, as linguistic pre-processing assumes the utterance to be finished after the current phrase. A further problem stems from the integration of new prosodic information which is attempted for as much content as possible by the iSS component. In the w_n condition, integration is performed up to the middle of the final word of the phrase, which can result in a jump in the pitch contour preceding the phrase-final phoneme. Such unnaturally sounding discontinuities could, of course, be avoided,

[16] Of course a more elaborate scheme can be used for the alignment, but is unlikely to radically change the results presented below.

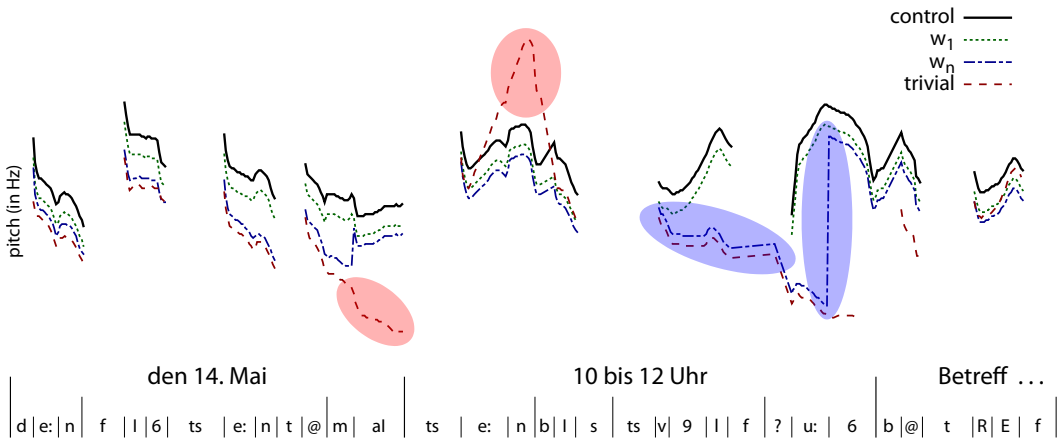


Figure 7.17: Exemplary pitch tracks of part of an utterance under some of the experimental conditions. Deficiencies of the *trivial* and *only left context* settings are highlighted in red and blue, respectively.

e. g. by gradually adjusting pitch to enforce a maximum gradient.[17] Finally, conditions w_1 and w_0 (w_0 is not shown in the figure for clarity) closely follow the non-incremental pitch track.

The author performed an informal assessment of the prosodic quality by listening to some of the produced audio files. Perceptually, the w_0 and w_1 conditions were indistinguishable from the control condition. The w_n setting results in a 'bored', 'sad' perceived attitude of the speaker which can be ascribed to the intermittent sentence-end intonations (realized with low pitch and lengthening) within the utterance. In the *trivial* setting, the additional intermittent onset intonations, realized as intense pitch excursions, together with the intermittent sentence-end intonations give the impression of riding a rollercoaster.

7.6.3.2 Quantitative Evaluation

The same 9 utterances (without noise) as in the calendar evaluation were synthesized in all conditions and the resulting pitch and segment durations (as determined from system logs) were compared with the result from the non-incremental control

[17] This countermeasure was not implemented, however, as it would increase further the deviation from the ideal, non-incrementally produced pitch track as measured quantitatively and evaluated in Subsection 7.6.3.2.

Table 7.2: Deviation in pitch and timing of lookahead conditions from the non-incremental control condition.

condition	timing deviation (in ms)		pitch deviation (in Hz)	
	RMSE	95 % quantile	RMSE	95 % quantile
w_0 (full phrase)	1.77	0	7.11	9
w_1	2.12	0	8.49	17
w_2	3.76	5	11.32	26
w_3	5.48	6	14.87	36
w_{n-1}	5.32	6	17.24	46
w_n (w/ left context)	5.75	10	18.23	50
w_n (trivial)	14.60	34	28.17	65

condition. A few setting/condition combinations resulted in partially faulty logging behaviour for some utterances (which was only discovered during evaluation). In these cases the faulty parts were excluded, leaving approximately 1200 phonemes and 11000 voiced audio frames for comparison. Mean phoneme duration is 81 ms and mean pitch is 172 Hz in the control condition.

Table 7.2 summarizes how prosody (duration and pitch assignments) deviates from the non-incremental control condition under the various lookahead conditions, using root mean squared error (RMSE) as metric.[18] Additionally, the 95 % quantiles of timing and pitch error magnitude are given.

As can be seen in the table, RMSE for both duration and pitch in the w_0 (full phrase) condition is remarkably low at less than 2 milliseconds for duration and about 7 Hz for pitch. Given the mean pitch and phoneme durations, the relative errors are about 2 % and 4 %, respectively. Quené (2007) reports the *just noticeable difference* (JND) for speech tempo to be roughly 5 % and Nooteboom reports a JND for pitch of "a few percent" (Nooteboom 1997, p. 643). Furthermore, deviation is smaller than 10 Hz for more than 95 % of all pitch values. For 98 % of all phonemes, timing differs by at most 4 ms and for 98 % or all pitch values deviation is at most 23 Hz. Thus, it is fair to conclude that one full phrase of lookahead results in prosodic assignments that are not noticeably different (and if, then only rarely) from non-incrementally produced prosody.

Figure 7.18 shows that deviation of both timing (in ms) and pitch (in Hz) increase (almost linearly) the later in the ongoing phrase the next is appended, i. e. the less

[18] The mean error was small in most conditions so that mean and variance are not reported separately.

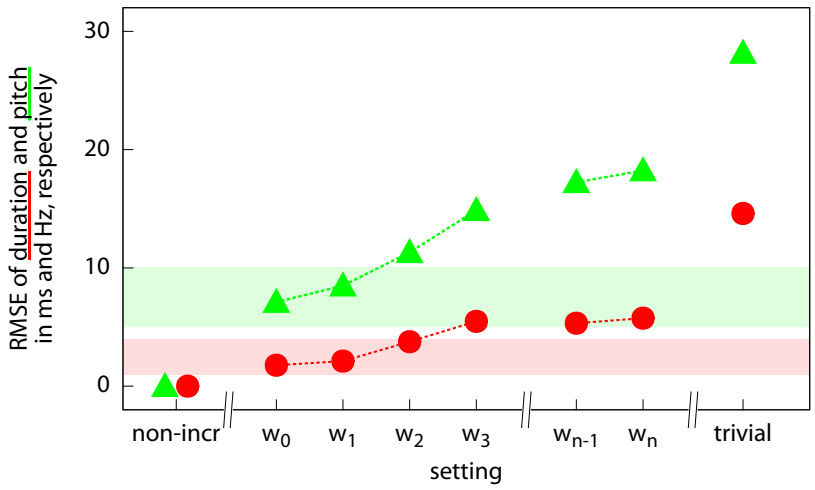


Figure 7.18: Deviation/RMSE of pitch (green) and timing (red) plotted against lookahead. The more lookahead available, the better the results. *Just noticeable differences* (JND) for pitch and timing are shown as green and red shaded areas.

lookahead is available. The figure also shows condition w_1 to be still within the margin of JND while w_3 and above are likely noticeably different from the control condition (and more likely worse than better).

Quite importantly, *left context* alone already drastically cuts down on pitch and timing deviation, even when no lookahead at all is available. Significance testing shows that pitch and timing differs more from the control condition in the *trivial* than in the w_n condition (Brown-Forsythe's modified Levene's test for variance of the deviations, $p < .00001$ for both pitch and timing). In addition, mean durations are significantly longer and pitch is significantly lower (t-tests, $p < .00001$ for both pitch and timing), however with small effect sizes. Thus, if no lookahead is available, synthesis using the w_n condition is clearly superior to the former state of the art of synthesizing phrases in isolation.

7.6.4 Conclusion

This section has analyzed the influence of iSS on realized prosody. Incremental prosody production with *full phrase lookahead* (which integrates a next phrase before the current phrase starts) was found to differ only marginally (within the range of

JND) as compared to standard, non-incremental prosody processing. Furthermore, even if no lookahead at all can be used, the use of *left context* greatly improves the results of iSS as compared to trivial incremental processing that handles each phrase in isolation.

Figure 7.18 showed that more lookahead leads to better prosodic quality, raising the question of how much lookahead exactly should be used. Section 7.5.3.1 above found that processing two phrases utterance-initially (which happens in the w_0 condition) takes twice as much time as compared to processing just a single phrase and processing the next phrase while the first word of that phrase is ongoing – the w_1 condition. Furthermore, book-keeping of the additional ChunkIUs that is involved in the w_0 condition increased code complexity and was a frequent source of problems in the development of the *Calendar* system. At the same time, the relative prosody performance gain of w_0 over w_1 is small. Thus, the w_1 condition, where the next phrase is added while the first word of the current phrase is being produced, can be used as a rule of thumb, providing a good trade-off of quality, computational performance, as well as conceptual simplicity.

This section treated prosodic analysis purely as a numeric comparison of (aligned) tracks, and this is of course insufficient for a full assessment of the realized intonations. Listening experiments would allow to validate and strengthen the results and conclusions drawn. (However, as argued in the discussion of Section 7.4, 'pure' listening experiments do not reflect the whole picture either, as it is the overall system behaviour what matters most.)

This section analyzed the "shallow" incremental prosody production that is implemented in the current version of the InproTK iSS component. A "deeply" incremental prosody model based on underspecified high-level data (following the idea of *triangular data-processing* as outlined in Chapter 4.1.2) might help to reduce lookahead (and correspondingly improve timing properties) and could potentially integrate changes to a plan that is being realized much more smoothly than the current strategy which simply replaces old for new pitch targets, resulting in discontinuities as were seen in Figure 7.17.

7.7 Summary and Discussion

This chapter has focused on incremental spoken output, specifically on incremental speech synthesis which is a necessary low-level component that other, higher-level components like iNLG need, in order to be of use to real systems.

iSS has been shown to enable previously unseen output behaviours in highly dynamic environments, and to integrate seamlessly into the incremental architecture, showing that the IU model as implemented in InproTK equally applies to input

and output processing. The combination of an IU module for iNLG with iSS greatly reduced system response time and again resulted in preferred system behaviour, and, finally, produced utterances that prosodically differ only marginally when compared to non-incrementally produced speech.

The InproTK iSS component still relies on the non-incremental processing components borrowed from MaryTTS, which is somewhat unsatisfactory. One area for future work is HMM state selection which is currently performed non-incrementally (instead of just-in-time) and uses non-local features in the decision trees involved. Future work should investigate the quality loss associated with abandoning non-local features (or at least features that require a large lookahead) to assess whether incremental HMM state selection is feasible. Implementing this feature would improve synthesis quality when tempo and accentuation changes are performed (which may result in acoustic artifacts in the current implementation). Chunwijitra, Nose, and Kobayashi (2012) deal with local variance optimization for HMM synthesis. Their findings could be integrated instead of the currently used simplistic method devised by Dutoit et al. (2011).

The current solution for linguistic pre-processing and prosody generation does not go by the (incremental) book. Even though the non-incremental solution is sufficiently fast, the fact that access to the internal structures and data used by MaryTTS's linguistic processing is limited from the IU network means that the prosody model's decisions cannot be actively controlled. A "deeply" incremental prosody model that makes its structures available to incremental processing would provide a much improved interface to prosody than is currently implemented, for example allowing to change an utterance's tone to 'question intonation' and all else (i. e. pitch and duration assignments and the corresponding HMM state selection) falling into place automatically, just-in-time and without computational overhead.

Aspects of speech synthesis specific to *conversational* settings, are planned to be integrated into the iSS component and could easily be integrated into the whole system in a way so that changes to emotional parameters (that could be represented in some *mood* state) are kept separate from content-based aspects of prosody (to be handled via the IU network). The two aspects could be combined just-in-time using local rules such as the ones by Schröder (2004) allowing for independent and immediate control of emotional colouring as outlined in (Baumann 2012).

Reproduced with kind permission by BeCK, www.beck-cartoons.de.

8 Conclusion and Outlook

This chapter briefly summarizes the thesis, concludes that the goal of the thesis – investigating incremental processing as a means for more naturally interacting spoken dialogue systems – has been met, and raises some questions and ideas for future work.

8.1 Summary

This thesis investigated incremental dialogue processing in order to enable more naturally interacting spoken dialogue systems. Chapter 2 found modular incremental processing to be desirable for natural and efficient interaction, as it provides for close feedback loops and thus assists in grounding between the dialogue participants.

Chapter 3 developed an evaluation methodology for incremental processing that focuses on the correctness, timing, and diachronic evolution of incremental hypotheses. The metrics were developed primarily for iSR but it was later shown that they generalize well to other types of incremental processors when they were put to use in later chapters.

Chapter 4 presented the software toolkit for incremental spoken dialogue processing based on the IU model that was implemented in the context of this thesis. The toolkit supports absolute and differential hypotheses as a basis for data exchange between processing modules and also supports other processing schemes that help to reduce the complexity in the system by allowing *active* IUs to analyze and extend the IU network autonomously. In addition, update listeners can be used for communicating against the main direction of information flow in the architecture, which spares the complexity of supporting bi-directional communication across module buffers.

Chapter 5 presented (incremental) speech recognition and a workbench for evaluating iSR that implements the metrics developed in Chapter 3. iSR was evaluated on three different corpora and proved to work reasonably well for all of them. Specifically, meaningful results become available with little delay. Additionally, iSR was further tested for stability under varying conditions, and n-best processing was shown to result in improvements already for low values of n. Based on the observation that iSR is highly non-monotonous, i. e. incurs edits that have to be taken back shortly after, simple optimization techniques were developed and improve stability against a small cost in timeliness. Finally, a small command-and-control application was developed which requires incremental speech processing and made use of *cost-based* iSR optimization. In the system, iSR optimization has significant advantages over

standard, non-optimized iSR. Furthermore, the system gave proof that interactive speech-based incremental systems work in practice. While the example domain was very simple, an advanced version in a more complex domain has recently been shown to work similarly well (Baumann et al. 2013).

Chapter 6 used incremental processing to move interaction management from the full turn to more fine-granular units in dialogue systems. We first investigated silence-delimited sub-turn units that the system used to collaborate with the user in creating her utterances, and then moved the level of analysis to individual words, while at the same time crossing the border from reactive to predictive processing. The example application for utterance collaboration was rated as more human-like and more reactive, indicating that full SDSs using sub-turn units may be advantageous. Stemming from the observation that iSR would often produce hypotheses for words before these were finished, the micro-timing of individual words was investigated. The simple models that were implemented predict the ongoing word's end and the next word's duration with high reliability, almost reaching human performance. While the example application of speaking in synchrony with a user may not have high additional value in a full SDS, this technology demonstration exhibits end-to-end incrementality in real-time, with all processing delays being counterbalanced by corresponding predictions into the future. More relevant conversational uses of micro-timing analysis *as it happens* (rather than *post hoc*) have been sketched as well.

Chapter 7 introduced incremental speech synthesis (iSS), building on the observation that the trivially incremental synthesis for synchronous speech in Chapter 6 was auditorily unsatisfactory. The chapter outlined requirements for iSS and showed how these are met by the implemented iSS component. Additionally, the toolkit accommodated these new requirements that it was originally not built for. A series of experiments then demonstrated that iSS is useful for highly dynamic environments, that it is feasible as part of an incremental output pipeline, and investigated the trade-off between timeliness and prosodic quality. iSS (together with iNLG) enabled system behaviour that was rated significantly more human-like than that of standard systems. Even in standard systems, iSS can be useful as it may improve system response time by folding almost all processing time into delivery.

8.2 Conclusion

The thesis set out to extend the interaction capabilities of spoken dialogue systems by proving fine-granular incremental and proactive processing to be technically feasible and successful at enabling more naturally interacting SDSs.

Where standard SDSs use a granularity of full turns, the systems presented here use sub-turn units (Section 6.1) or individual words (Section 6.2) as units for decision

making and the iSR component itself provides even more fine-grained results (both in terms of frequency of updates, as well as sub-word information like syllables and phonemes). The iSS component allows manipulation at different granularities, ranging from prosodic change that takes place within 5 ms, to different types of higher-level interactions (with correspondingly longer lags). At the phrase level, the interdependence of lookahead and prosodic quality was discussed.

The micro-timing of words can be estimated proactively, that is, the end of a word can be predicted before that word is over, allowing to balance the delays that are caused by other system modules, thus resulting in a system that proactively schedules next words for delivery. The iSS component is able to hesitate if a next word does not become available in time, which can also be considered as a sort of proactive processing, minimizing the impact of delays in other system components.

Behaviours that were only ever enabled by incremental processing (collaborating on utterances, speaking in a highly dynamic environment, or reacting to external events while speaking) resulted in systems that were rated as more natural or human-like than baseline systems that did not employ incremental processing (or less sophisticated incremental processing).

Other system behaviours were realized with incremental processing that could not be compared to non-incremental baseline systems at all (acting early while recognition is still ongoing, speaking in synchrony, starting to speak before processing is over, or manipulating prosody during synthesis) making the assessment of their advantage hard. It is likely that when employed in full systems, these aspects will also contribute towards naturalness of behaviour.

Thus, it is safe to conclude that incremental processing enables more natural interaction and will become an important aspect of next-generation SDSs. Incremental processing helps to overcome the ping-pong style of interaction of most present-day systems, paving the way to more *conversational* human-computer interaction.

Finally, this thesis has *not* attempted to build full incremental SDSs featuring both incremental spoken input, incremental spoken output, and – foremost – incremental dialogue management. The application for synchronous co-completion, which both receives and produces speech incrementally comes closest (and could potentially be extended to feature full iSS as developed in Chapter 7) but does not include any turn-taking abilities (or other contributions more intelligent than plain shadowing). Incremental turn-taking and dialogue management capabilities were demonstrated in the application for utterance collaboration. However, while these components were highly reactive, they did not cross the line to predictive processing. At least for the floor tracking component this may be relatively easy to fix, if more input data were integrated into a more complex model (as in Atterer, Baumann, and Schlangen 2008) and used in conjunction with micro-timing estimation.

8.3 Open Questions

Given that incremental spoken dialogue processing works incrementally, end-to-end and in real-time, new questions arise.

One topic that has not been covered in this thesis is incremental dialogue management, which is, however, necessary to fully leverage the findings presented. Work on iDM is ongoing (Buß and Schlangen 2010, 2011; Selfridge et al. 2012c). The crucial question for iDM boils down to making decisions: decisions that become apparent to the interlocutor cannot easily be revoked, that is, decision making 'breaks' the non-monotonic aspects of the IU model. It seems that so far, iDMs have rather avoided than embraced the uncertainty which arises from incremental decision making. Instead, in order to defer making strict decisions and at the same time allow for prompt reactions, iDMs should generate *underspecified* decisions (cmp. Guhe and Schilder 2002), and these could be serialized by iNLG to an utterance prefix that leaves open as many options as possible for as long as possible (cmp. Dethlefs et al. 2012) in order to hide the system's uncertainty in understanding (cmp. Skantze 2008).

The above-mentioned works rely on one-best input hypotheses, which weakens their decision making. Lattice/tree-based incremental processing could help to further bridge the gap with POMDP-based dialogue management (cmp. Roy, Pineau, and Thrun 2000). Tree-based incremental processing would specifically contain the system's uncertainty and would add an additional dimension to the triangular data model (turning the triangles into tetrahedrons: an incoming signal is unambiguous and extends along the time axis, whereas the system's current hypothesis extends along all possible dialogue states).

The work in this thesis has aimed to increase the granularity of processing as much as possible. However, the *ideal* level of granularity (at a specific level of processing) remains open. While iSR now outputs word increments, it remains open whether this is the right level for an understanding component for which some sort of phrase-chunking (possibly based on prosodic/timing features) might be sufficient.[1] However, such chunking would presuppose some semantic knowledge in the iSR component, or might focus too much on prosody (which is unlikely to unambiguously define semantic chunks, just like this does not work for turn-taking). However, both iSR optimization and NLU itself could certainly profit from incremental prosodic information.

Going into the other direction, iSR could also be made *more* than word-incremental and output sub-word units that might already be helpful for NLU analyzes (cmp. the example of "grün/grüne/grünes" in Section 3.3.2.1). The NLU (and possibly pragmatic) analysis could even be fed back to iSR to provide guidance to the recognition process.

[1] Thanks to Gabriel Skantze for raising this idea.

Top-down feedback from higher-level incremental modules towards iSR is an obvious candidate for future work. However, initial results indicate that this is not as straightforward and raises the question whether n-best/tree-based incremental processing is sufficient,[2] or whether the incremental recognition process should provide for partial re-analyses of previous input (given newly added constraints), resulting in a form of non-monotonous incremental re-processing (instead of 'just' non-monotonous output generation).

On the speech synthesis side, a fully incremental prosody model would allow for more tightly integrated prosodic control over ongoing system utterances, as sketched in Chapter 7. Furthermore, a structured and integrated incremental prosody model might also be of use on the input side, where semantic chunking could help to clarify intonation phrases and vice-versa prosody could hint at information status of constituents.

Speech input and speech output have been handled independently from each other (in InproTK and in other systems). A joint speech input and output component, however, might be beneficial in order to autonomously handle phenomena like turn-taking contests (raising the voice as a reflex to an interruption) or to realize turn-keeping operations. While such a double-module would show reflexive behaviour (to be quick, and in order to not bother the higher-level processors with these issues), a rich interface could still be offered so that higher-level reasoning modules and dialogue management could control such behaviour and be informed of its effects.

The software toolkit InproTK has shown to be flexible enough to support both incremental input and output processing. However, for the most part, no alternative implementations of modules exist. Work is ongoing to integrate industry-grade speech recognition (based on *Android*'s built-in capabilities) which would allow to compare the different performance aspects (Android uses distributed speech recognition, that is ASR is performed on a server which likely hurts timeliness to some extent) both based on a quantitative comparison of the individual components, but also in the context of small example systems, shading some light on the practical trade-offs involved with varying incremental performance. InproTK has been used with multiple languages (German and English, so far) and no principled differences in incremental aspects have been found. Setting up InproTK for other languages is easy if the underlying ASR and TTS systems provide for these languages.

InproTK is far from finished and some obvious candidates for improvement (top-down interaction, n-best/lattice-based hypotheses, incremental prosody models for synthesis) will be added to future versions.

[2] We recently found that iSR errors could often not be corrected using iNLU, because all alternatives had already fallen off the search beam.

Finally, the rather technology-centric findings in this thesis will not by themselves result in better dialogue interactions. Conversational *abilities* do not by themselves result in a high conversational *level*. How a system may shape the interaction, what level of conversational abilities are accepted by users, needs to be investigated by dialogue and interaction designers and tested by usability engineers.

This thesis merely provides the *building blocks* that may help to study these issues and hopes to enable more naturally interacting spoken dialogue systems in the future.

Bibliography

Aesop (1991). *Aesop's Fables*. Vol. 21. includes a 'life' of Aesop. Project Gutenberg. URL: ftp://uiarchive.cso.uiuc.edu/pub/etext/gutenberg/etext91/aesop11.zip (cit. on pp. 157, 171).

Aist, Gregory, James Allen, Ellen Campana, Carlos Gomez Gallo, Scott Stoness, Mary Swift, and Michael K. Tanenhaus (2007a). "Incremental Dialogue System Faster than and Preferred to its Nonincremental Counterpart". In: *Proceedings of the 29th Annual Conference of the Cognitive Science Society*. Nashville, USA, pp. 761–766 (cit. on pp. 16, 146).

– (June 2007b). "Incremental Understanding in Human-Computer Dialogue and Experimental Evidence for Advantages over Nonincremental Methods". In: *Proceedings of DECALOG, the 11th International Workshop on the Semantics and Pragmatics of Dialogue*. Trento, Italy, pp. 149–154 (cit. on pp. 16, 143, 145, 147).

Amtrup, Jan Willers (1999). *Incremental speech translation*. Berlin, Heidelberg: Springer. ISBN: 3-540-66753-9 (cit. on p. 17).

Anastasakos, Anastasios, Richard Schwartz, and Han Shu (1995). "Duration Modeling in Large Vocabulary Speech Recognition". In: *Proceedings of ICASSP*. Detroit, USA, pp. 628–631 (cit. on p. 155).

Anderson, Hyrum S., Nathan Parrish, and Maya R. Gupta (Sept. 2012). *Early Time-Series Classification with Reliability Guarantee*. Tech. rep. SAND2012-6961. Sandia National Laboratories. URL: http://prod.sandia.gov/techlib/access-control.cgi/2012/126961.pdf (cit. on p. 136).

Anderson, Hyrum S., Nathan Parrish, Kristi Tsukida, and Maya R. Gupta (2012). "Reliable early classification of time series". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2012*. IEEE. Kyoto, Japan, pp. 2073–2076. DOI: 10.1109/ICASSP.2012.6288318 (cit. on p. 136).

Atterer, Michaela, Timo Baumann, and David Schlangen (2008). "Towards Incremental End-of-Utterance Detection in Dialogue Systems". In: *Proceedings of Coling 2008*. Manchester, UK (cit. on pp. 139, 140, 142, 168, 220).

– (2009). "No Sooner Said Than Done? Testing the Incrementality of Semantic Interpretations of Spontaneous Speech". In: *Proceedings of Interspeech 2009*. Brighton, UK (cit. on pp. 49, 66, 121).

Bangalore, Srinivas, Vivek Kumar Rangarajan Sridhar, Prakash Kolan, Ladan Golipour, and Aura Jimenez (2012). "Real-time Incremental Speech-to-Speech Translation of Dialogs". In: *Proceedings of the 2012 Conference of the North American Chapter*

of the Association for Computational Linguistics: Human Language Technologies. Montréal, Canada: Association for Computational Linguistics, pp. 437–445. URL: http://www.aclweb.org/anthology/N12-1048 (cit. on p. 17).

Baumann, Timo (2008). "Simulating Spoken Dialogue With a Focus on Realistic Turn-Taking". In: *Proceedings of the 13th ESSLLI Student Session*. Hamburg, Germany (cit. on p. 148).

– (2012). "Feedback in Adaptive Interactive Storytelling". In: *Proceedings of the Interdisciplinary Workshop on Feedback Behaviors in Dialog*. Stevenson, USA (cit. on p. 216).

Baumann, Timo, Michaela Atterer, and David Schlangen (2009). "Assessing and Improving the Performance of Speech Recognition for Incremental Systems". In: *Proceedings of NAACL-HLT 2009*. Boulder, USA, pp. 380–388 (cit. on pp. 20, 45, 49, 104, 110).

Baumann, Timo, Okko Buß, and David Schlangen (2010). "InproTK in Action: Open-Source Software for Building German-Speaking Incremental Spoken Dialogue Systems". In: *Proceedings of ESSV*. Berlin, Germany (cit. on p. 20).

– (2011). "Evaluation and Optimisation of Incremental Processors". In: *Dialogue & Discourse* 2.1. Special Issue on Incremental Processing in Dialogue, pp. 113–141. ISSN: 2152-9620. DOI: 10.5087/dad.2011.106 (cit. on pp. 20, 45, 49).

Baumann, Timo and David Schlangen (2011). "Predicting the Micro-Timing of User Input for an Incremental Spoken Dialogue System that Completes a User's Ongoing Turn". In: *Proceedings of SigDial 2011*. Portland, USA (cit. on pp. 21, 135, 148, 156).

– (Sept. 2012a). "Evaluating Prosodic Processing for Incremental Speech Synthesis". In: *Proceedings of Interspeech*. ISCA. Portland, USA (cit. on p. 21).

– (2012b). "Evaluating Prosodic Processing for Incremental Speech Synthesis". In: *Proceedings of Interspeech* (cit. on p. 207).

– (2012c). "INPRO_iSS: A Component for Just-In-Time incremental Speech Synthesis". In: *Procs. of ACL System Demonstrations*. Jeju, Korea (cit. on p. 21).

– (Sept. 2012d). "The INPROTK 2012 Release: A Toolkit for Incremental Spoken Dialogue Processing". In: *Sprachkommunikation 2012: Beiträge zur 10. ITG-Fachtagung*. (Braunschweig, Germany). Ed. by Tim Fingscheidt. Informationstechnische Gesellschaft im VDE (ITG). ISBN: 978-3-8007-3455-9 (cit. on p. 20).

– (2012e). "The INPROTK 2012 Release". In: *Proceedings of SDCTD*. Montréal, Canada (cit. on p. 20).

Baumann, Timo, Okko Buß, Michaela Atterer, and David Schlangen (2009). "Evaluating the Potential Utility of ASR N-Best Lists for Incremental Spoken Dialogue Systems". In: *Proceedings of Interspeech 2009*. Brighton, UK, pp. 1031–1034 (cit. on pp. 20, 49, 110, 117–119).

Baumann, Timo, Maike Paetzel, Philipp Schlesinger, and Wolfgang Menzel (2013). "Using Affordances to Shape the Interaction in a Hybrid Spoken Dialogue System".

In: *Proceedings of ESSV 2013*. Ed. by Petra Wagner. TUDpress, pp. 12–19 (cit. on pp. 84, 129, 133, 135, 219).

Bertalanffy, Ludwig von (Dec. 1972). "The History and Status of General Systems Theory". In: *The Academy of Management Journal* 15.4, pp. 407–426. ISSN: 0001-4273 (cit. on p. 28).

Bisani, Maximilian and Hermann Ney (2008). "Joint-sequence models for grapheme-to-phoneme conversion". In: *Speech Communication* 50.5, pp. 434 –451. ISSN: 0167-6393. DOI: 10.1016/j.specom.2008.01.002 (cit. on p. 93).

Black, Alan W. and Maxine Eskenazi (2009). "The spoken dialogue challenge". In: *Proceedings of the SIGDIAL 2009 Conference: The 10th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. SIGDIAL '09. London, UK: Association for Computational Linguistics, pp. 337–340. ISBN: 978-1-932432-64-0. URL: http://dl.acm.org/citation.cfm?id=1708376.1708426 (cit. on p. 41).

Black, Alan W. and Keiichi Tokuda (2005). "The Blizzard Challenge – 2005: Evaluating corpus-based speech synthesis on common datasets". In: *Proceedings of Interspeech* (cit. on p. 183).

Black, Alan W., Susanne Burger, Alistair Conkie, Helen Hastie, Simon Keizer, Oliver Lemon, Nicolas Merigaud, Gabriel Parent, Gabriel Schubiner, Blaise Thomson, Jason D. Williams, Kai Yu, Steve Young, and Maxine Eskenazi (2011). "Spoken Dialog Challenge 2010: comparison of live and control test results". In: *Proceedings of the SIGDIAL 2011 Conference*. SIGDIAL '11. Portland, USA: Association for Computational Linguistics, pp. 2–7. ISBN: 978-1-937284-10-7. URL: http://dl.acm.org/citation.cfm?id=2132890.2132892 (cit. on p. 41).

Boersma, P. (2002). "Praat, a system for doing phonetics by computer". In: *Glot international* 5.9/10, pp. 341–345. ISSN: 1381-3439 (cit. on p. 107).

Bohus, Dan and Alexander I. Rudnicky (July 2009). "The Ravenclaw Dialog Management Framework: Architecture and Systems". In: *Computer Speech & Language* 3.23, 332–361. ISSN: 0885-2308. DOI: 10.1016/j.csl.2008.10.001 (cit. on p. 41).

Boros, Manuela, Wieland Eckert, Florian Gallwitz, Günther Görz, Gerhard Hanrieder, and Heinrich Niemann (Oct. 1996). "Towards Understanding Spontaneous Speech: Word Accuracy Vs. Concept Accuracy". In: *Proceedings of the 4th ICSLP*. Philadelphia, USA, pp. 1009–1012 (cit. on pp. 66, 102).

Bray, Tim, Jean Paoli, Eve Maler, François Yergeau, and C. M. Sperberg-McQueen (Nov. 2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. W3C. URL: http://www.w3.org/TR/2008/REC-xml-20081126/ (cit. on p. 184).

Breiman, Leo, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone (1984). *Classification and regression trees*. Wadsworth, Monterey, USA. ISBN: 978-0412048418 (cit. on p. 156).

Brennan, Susan E. (1996). "Lexical entrainment in spontaneous dialog". In: *Proceedings of the International Symposium on Spoken Dialogue*. Philadelphia, USA, pp. 41–44 (cit. on p. 80).

Brennan, Susan E. and Herbert H. Clark (Nov. 1996). "Conceptual Pacts and Lexical Choice in Conversation". In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 22.6, pp. 1482–1493. ISSN: 1939-1285. DOI: 10.1037/0278-7393.22.6.1482 (cit. on p. 37).

Brinckmann, Caren and Jürgen Trouvain (2003). "The Role of Duration Models and Symbolic Representation for Timing in Synthetic Speech". In: *International Journal of Speech Technology* 6.1, pp. 21–31. DOI: 10.1023/A:1021043804581 (cit. on p. 156).

Brown, Peter F., James C. Spohrer, Peter H. Hochschild, and James K. Baker (1982). "Partial traceback and dynamic programming". In: *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'82*. Vol. 7. Paris, France, pp. 1629 –1632. DOI: 10.1109/ICASSP.1982.1171441 (cit. on pp. 104, 136).

Burnett, Daniel C., Andrew Hunt, and Mark R. Walker (Sept. 2004). *Speech Synthesis Markup Language (SSML) Version 1.0*. W3C Recommendation. W3C. URL: http://www.w3.org/TR/2004/REC-speech-synthesis-20040907/ (cit. on p. 178).

Buschmeier, Hendrik and Stefan Kopp (2012). "Adapting Language Production to Listener Feedback Behaviour". In: *Proceedings of the Interdisciplinary Workshop on Feedback Behaviors in Dialog*, pp. 14–17 (cit. on p. 200).

Buschmeier, Hendrik, Timo Baumann, Benjamin Dorsch, Stefan Kopp, and David Schlangen (2012). "Combining Incremental Language Generation and Incremental Speech Synthesis for Adaptive Information Presentation". In: *Proceedings of SigDial*. Seoul, Korea, pp. 295–303 (cit. on pp. 21, 199, 201, 202, 206, 208).

Buß, Okko, Timo Baumann, and David Schlangen (2010). "Collaborating on Utterances with a Spoken Dialogue System Using an ISU-based Approach to Incremental Dialogue Management". In: *Proceedings of SigDial 2010*. Tokyo, Japan (cit. on pp. 20, 135, 140, 142, 143, 145, 148).

Buß, Okko and David Schlangen (2010). "Modelling Sub-Utterance Phenomena in Spoken Dialogue Systems". In: *Proceedings of SemDial (PozDial)*. Poznan, Poland (cit. on pp. 145, 221).

– (2011). "DIUM – An Incremental Dialogue Manager That Can Produce Self-Corrections". In: *Proceedings of SemDial 2011 (Los Angelogue)*. Los Angeles, USA (cit. on pp. 17, 79, 221).

Byrne, Steve, Lauren Wood, Vidur Apparao, Chris Wilson, Robert Sutor, Scott Isaacs, Gavin Nicol, Jonathan Robie, Arnaud Le Hors, Mike Champion, and Ian Jacobs (Oct. 1998). *Document Object Model (DOM) Level 1*. W3C Recommendation. W3C. URL: http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001 (cit. on p. 184).

Carroll, John, Ted Briscoe, and Antonio Sanfilippo (1998). "Parser evaluation: a survey and a new proposal". In: *Proceedings of the 1st International Conference on Language Resources and Evaluation*, pp. 447–454 (cit. on p. 66).

Chen, David L. and Raymond J. Mooney (July 2008). "Learning to Sportscast: A Test of Grounded Language Acquisition". In: *Proceedings of 25th International Conference on Machine Learning (ICML-2008)*. Helsinki, Finland (cit. on pp. 193, 194).

Cheveigné, Alain de and Hideki Kawahara (2002). "YIN, a fundamental frequency estimator for speech and music". In: *Journal of the Acoustical Society of America* 111.4, pp. 1917–1930. ISSN: 0001-4966. DOI: 10.1121/1.1458024 (cit. on p. 85).

Chotimongkol, Ananlada and Alexander I. Rudnicky (2001). "N-best speech hypothesis reordering using linear regression". In: *Proceedings of Eurospeech*. Aalborg, Denmark, pp. 1829–1832 (cit. on pp. 102, 117).

Chunwijitra, V., T. Nose, and T. Kobayashi (2012). "A speech parameter generation algorithm using local variance for HMM-based speech synthesis". In: *Proceedings 13th Annual Conference of the International Speech Communication Association*. The International Speech Communication Association (cit. on p. 216).

Clark, Herbert H. (1996). *Using Language*. Cambridge University Press. ISBN: 978-0521567459 (cit. on pp. 31, 45, 138, 143, 151, 198).

– (2002). "Speaking in Time". In: *Speech Communication* 36.1, pp. 5–13. ISSN: 0167-6393. DOI: 10.1016/S0167-6393(01)00022-X (cit. on p. 168).

Clark, Robert A.J. and Kurt E. Dusterhoff (1999). "Objective methods for evaluating synthetic intonation". In: *Proceedings of Interspeech* (cit. on p. 183).

Cohen, William (1995). "Fast effective rule induction". In: *Proceedings of the 12th International Conference on Machine Learning*. Morgan Kaufmann, pp. 115–123 (cit. on p. 127).

Cummins, Fred (2002). "On synchronous speech". In: *Acoustic Research Letters Online* 3.1, pp. 7–11. ISSN: 1529-7853 (cit. on pp. 152, 162).

– (Apr. 2003). "Practice and performance in speech produced synchronously". In: *Journal of Phonetics* 31.2, pp. 139–148. ISSN: 0095-4470. DOI: 10.1016/S0095-4470(02)00082-7 (cit. on p. 152).

– (Jan. 2009). "Rhythm as entrainment: The case of synchronous speech". In: *Journal of Phonetics* 37.1, pp. 16–28. ISSN: 0095-4470. DOI: 10.1016/j.wocn.2008.08.003 (cit. on pp. 151, 152).

DeVault, David, Kenji Sagae, and David Traum (Sept. 2009). "Can I Finish? Learning When to Respond to Incremental Interpretation Results in Interactive Dialogue". In: *Proceedings of the SIGDIAL 2009 Conference*. London, UK, pp. 11–20 (cit. on pp. 17, 42, 79, 149, 152, 153).

DeVault, David and David Traum (2012a). "A Demonstration of Incremental Speech Understanding and Confidence Estimation in a Virtual Human Dialogue System.

System Demonstration". In: *The 13th annual SIGdial Meeting on Discourse and Dialogue (SigDial 2012)* (cit. on p. 42).

– (2012b). "Incremental Speech Understanding in a Multi-Party Virtual Human Dialogue System. System Demonstration". In: *The 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2012)*, pp. 25–28 (cit. on p. 42).

Dean, Thomas and Mark Boddy (1988). "An Analysis of Time-Dependent Planning". In: *Proceedings of AAAI-88*. AAAI. Cambridge, USA, pp. 49–54 (cit. on p. 49).

Dellwo, Volker and Daniel Friedrichs (2012). "Variability of speech rhythm in synchronous speech". In: *Proceedings of Speech Prosody*. URL: http://sprosig.isle.illinois.edu/sp2012/uploadfiles/file/sp2012_submission_209.pdf (cit. on pp. 165, 169).

Descartes, René (1824). *Discours de la méthode pour bien conduire sa raison, et chercher la verité dans les sciences*. Ed. by Victor Cousin. Paris: Tome Premier. URL: http://fr.wikisource.org/wiki/Discours_de_la_mÃl'thode_(Ãl'd._Cousin) (cit. on p. 25).

Dethlefs, Nina, Helen Hastie, Verena Rieser, and Oliver Lemon (May 2012). "Optimising Incremental Generation for Spoken Dialogue Systems: Reducing the Need for Fillers". In: *INLG 2012 Proceedings of the Seventh International Natural Language Generation Conference*. Utica, IL: Association for Computational Linguistics, pp. 49–58. URL: http://www.aclweb.org/anthology/W12-1509 (cit. on p. 221).

Duncan Jr, Starkey and George Niederehe (1974). "On signalling that it's your turn to speak". In: *Journal of Experimental Social Psychology* 10.3, pp. 234 –247. ISSN: 0022-1031. DOI: 10.1016/0022-1031(74)90070-5 (cit. on pp. 31, 32).

Dutoit, Thierry, Maria Astrinaki, Onur Babacan, Nicolas d'Alessandro, and Benjamin Picart (Mar. 2011). *pHTS for Max/MSP: A Streaming Architecture for Statistical Parametric Speech Synthesis*. Tech. rep. 1, pp. 7–11. URL: http://www.numediart.org/docs/numediart_2011_s13_p2_report.pdf (cit. on pp. 186–188, 216).

Edlund, Jens (2008). "Incremental Speech Synthesis". In: *Second Swedish Language Technology Conference*. System Demonstration (cit. on pp. 175–177, 182, 203).

Edlund, Jens and Mattias Heldner (2007). "Underpinning /nailon/: Automatic Estimation of Pitch Range and Speaker Relative Pitch". In: *Lecture Notes in Computer Science* 4441, p. 229. DOI: 10.1007/978-3-540-74122-0_18 (cit. on p. 168).

Edlund, Jens, Joakim Gustafson, Mattias Heldner, and Anna Hjalmarsson (2008). "Towards human-like spoken dialogue systems". In: *Speech Communication* 50, pp. 630–645. ISSN: 0167-6393. DOI: 10.1016/j.specom.2008.04.002 (cit. on pp. 128, 144).

Farrell, Anthony Timothy (Apr. 13, 2004). "Call centre agent automated assistance". Pat. US 6,721,416 B1. International Business Machines Corporation. URL: http://patft1.uspto.gov/netacgi/nph-Parser?patentnumber=6721416 (cit. on p. 17).

Fernández, Raquel and Jonathan Ginzburg (2002). "Non-sentential utterances: A corpus-based study". In: *Traitement automatique des langues* 43.2, pp. 13–42 (cit. on p. 151).

Fernández, Raquel, Tatjana Lucht, Kepa Rodriguez, and David Schlangen (Dec. 2006). "Interaction in Task-Oriented Human–Human Dialogue: The Effects of Different Turn-Taking Policies". In: *Proceedings of the First International IEEE/ACL Workshop on Spoken Language Technology*. Palm Beach, USA (cit. on pp. 110, 143).

Ferrer, Luciana, Elizabeth Shriberg, and Andreas Stolcke (Sept. 2002). "Is the Speaker Done Yet? Faster and More Accurate End-Of-Utterance Detection Using Prosody". In: *Proceedings of the International Conference on Spoken Language Processing (ICSLP2002)*. Denver, USA (cit. on pp. 139, 140).

Finkler, Wolfgang (1997). *Automatische Selbstkorrektur bei der inkrementellen Generierung gesprochener Sprache unter Realzeitbedingungen*. Dissertationen zur Künstlichen Intelligenz. infix Verlag (cit. on p. 53).

Gales, Mark and Steve Young (2007). "The Application of Hidden Markov Models in Speech Recognition". In: *Foundations and Trends in Signal Processing* 1.3, pp. 195–304. DOI: 10.1561/2000000004 (cit. on p. 89).

Gallo, Carlos Gómez, Gregory Aist, James Allen, William de Beaumont, Sergio Coria, Whitney Gegg-Harrison, Joana P. Pardal, and Mary Swift (June 2007). "Annotating Continuous Understanding in a Multimodal Dialogue Corpus". In: *Proceedings of DECALOG, the 11th International Workshop on the Semantics and Pragmatics of Dialogue*. Trento, Italy, pp. 75–82 (cit. on p. 62).

Ginzburg, Jonathan (1996). "Interrogatives: Questions, Facts and Dialogue". In: *The Handbook of Contemporary Semantic Theory*. Ed. by Shalom Lappin. Oxford: Blackwell (cit. on p. 145).

Gravano, A. and J. Hirschberg (2009). "Backchannel-inviting cues in task-oriented dialogue". In: *Proceedings of Interspeech*. Vol. 2009, pp. 1019–1022 (cit. on pp. 142, 150).

Gravano, Agustín and Julia Hirschberg (2011). "Turn-taking cues in task-oriented dialogue". In: *Computer Speech & Language* 25.3, pp. 601–634. ISSN: 0885-2308. DOI: 10.1016/j.csl.2010.10.003 (cit. on pp. 29, 32, 142).

Greenberg, Steven (1996). "Auditory Processing of Speech". In: *Principles of Experimental Phonetics*. Ed. by Norman J. Lass. Mosby. Chap. 10, pp. 362–407 (cit. on p. 24).

Grewendorf, Günther, Fritz Hamm, and Wolfgang Sternefeld (1989). *Sprachliches Wissen. Eine Einführung in moderne Theorien der Grammatischen Beschreibung*. 3rd ed. Suhrkamp (cit. on p. 26).

Guhe, Markus (2007). *Incremental Conceptualization for Language Production*. Mahwah, USA: Lawrence Erlbaum Associates (cit. on pp. 36, 47, 50, 51).

Guhe, Markus and Frank Schilder (2002). "Underspecification for incremental generation". In: *Proceedings of KONVENS 2002* (cit. on p. 221).

Hamadeh, Rabih (2012). "Untersuchung des Einsatzes von maschinelles Lernenbasierten Filtermethoden in der inkrementellen Spracherkennung". Master's thesis. FB Infomatik, Universität Hamburg (cit. on pp. 127, 130).

He, Yulan and Steve Young (2005). "Semantic Processing using the Hidden Vector State Model". In: *Computer Speech and Language* 19.1, pp. 85–106 (cit. on p. 62).

Heintze, Silvan, Timo Baumann, and David Schlangen (Sept. 2010). "Comparing Local and Sequential Models for Statistical Incremental Natural Language Understanding". In: *Proceedings of SigDial 2010*. Tokyo, Japan (cit. on pp. 64, 152).

Heise Zeitschriften Verlag, ed. (July 2012). *c't: Magazin für Computer Technik* 30.16.

Hildebrandt, Bernd, Hans-Jürgen Eikmeyer, Gert Rickheit, and Petra Weiß (1999). "Inkrementelle Sprachrezeption [Incremental language understanding]". In: *KogWis: Proceedings der 4. Fachtagung der Gesellschaft für Kognitionswissenschaft*. Ed. by I. Wachsmuth and B. Jung, 19–24 (cit. on pp. 47, 52).

Hill, David R., Leonard Manzara, and Craig-Richard Taube-Schock (1995). "Real-time articulatory speech-synthesis-by-rules". In: *Proceedings of AVIOS*. Vol. 95. Citeseer (cit. on p. 183).

Hirasawa, Jun-ichi, Mikio Nakano, Takeshi Kawabata, and Kiyoaki Aikawa (Sept. 1999). "Effects of System Barge-in Responses on User Impressions". In: *Proceedings of the 6th European Conference on Speech Communication and Technology (EUROSPEECH 1999)*. Budapest, Hungary (cit. on pp. 152, 154).

Huang, X., F. Alleva, H.W. Hon, M.Y. Hwang, K.F. Lee, and R. Rosenfeld (Jan. 1992). *The SPHINX-II speech recognition system: an overview*. Tech. rep. CMU-CS-92-112. Carnegie Mellon University (cit. on p. 103).

Hunt, Melvin J. (1990). "Figures of merit for assessing connected-word recognisers". In: *Speech Communication* 9.4, pp. 329–336 (cit. on p. 48).

IPA, International Phonetic Association, (July 1999). *Handbook of the International Phonetic Association: A guide to the use of the International Phonetic Alphabet*. Cambridge, UK: Cambridge University Press. ISBN: 9780521637510 (cit. on p. 157).

IPDS, Institut für Phonetik und digitale Sprachverarbeitung, (1994). *The Kiel Corpus of Read Speech*. Ed. by University of Kiel. CD-ROM. Kiel, Germany (cit. on pp. 110, 157).

ITU (July 2006). *Mean Opinion Score (MOS) terminology (ITU-T Recommendation P.800.1)*. International Telecommunications Union. URL: http://www.itu.int/rec/T-REC-P.800-199608-I/en (cit. on p. 183).

Imai, T., A. Kobayashi, S. Sato, H. Tanaka, and A. Ando (2000). "Progressive 2-pass decoder for real-time broadcast news captioning". In: *Proceedings of ICASSP 2000*. Vol. 3. Istanbul, Turkey. DOI: 10.1109/ICASSP.2000.861969 (cit. on pp. 103, 104, 125).

Imai, Toru, Hideki Tanaka, Akio Ando, and Haruo Isono (2003). "Progressive early decision of speech recognition results by comparing most likely word sequences". In: *Systems and Computers in Japan* 34.14, pp. 73–82. ISSN: 1520-684X. DOI: 10.1002/scj.10193 (cit. on p. 104).

Janert, Philipp K. (2009). *Gnuplot in Action: Understanding Data with Graphs*. Greenwich, USA: Manning Publications Co. ISBN: 978-1933988399 (cit. on p. 109).

Jekat, S. J., C. Scheer, and T. Schultz (1997). *VMII Szenario I: Instruktionen für alle Sprachstellungen*. Tech. rep. VM-Techdoc 62. Universität Hamburg, LMU München, Universität Karlsruhe (cit. on p. 110).

Johnson, Michael T. (2005). "Capacity and Complexity of HMM Durational Modeling Techniques". In: *IEEE Signal Processing Letters* 12.5, pp. 407–410 (cit. on p. 155).

Jokinen, Kristiina (2009). *Constructive dialogue modelling: speech interaction and rational agents*. Vol. 11. Wiley Series in Agent Technology. Chichester, UK: Wiley-Interscience (cit. on p. 23).

Jokinen, Kristiina and Michael McTear (2010). *Spoken Dialogue Systems*. Ed. by Graeme Hirst. Synthesis Lectures on Human Language Technologies. Morgan & Claypool (cit. on pp. 23, 36).

Jonsdottir, Gudny Ragna, Kristinn R. Thórisson, and Eric Nivel (2008). "Learning Smooth, Human-Like Turntaking in Realtime Dialogue". In: *Intelligent Virtual Agents*. Ed. by Helmut Prendinger, James Lester, and Mitsuru Ishizuka. Vol. 5208. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 162–175. ISBN: 978-3-540-85482-1. DOI: 10.1007/978-3-540-85483-8_17 (cit. on p. 42).

Joustra, Yme (2011). "Evaluation of Turn Taking Behaviour in Semaine". In: *Proceedings of TSConIT 2011*. (June 20, 2011). Vol. 15. University of Twente (cit. on p. 42).

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2nd ed. Pearson International (cit. on pp. 22, 23, 89, 91, 92, 100, 178, 185).

Kato, Yoshihide, Shigeki Matsubara, and Yasuyoshi Inagaki (July 2004). "Stochastically evaluating the validity of partial parse trees in incremental parsing". In: *Proceedings of the ACL Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*. Barcelona, Spain, pp. 9–15 (cit. on p. 48).

Kawahara, H. (1997). "Speech representation and transformation using adaptive interpolation of weighted spectrum: vocoder revisited". In: *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*. Vol. 2. IEEE, pp. 1303–1306 (cit. on p. 182).

Kenny, P., R. Hollan, V. Gupta, M. Lennig, P. Mermelstein, and D. O'Shaughnessy (Apr. 1991). "A*-admissible heuristics for rapid lexical access". In: *Acoustics, Speech,*

and Signal Processing, 1991. ICASSP-91., 1991 International Conference on, 689 –692 vol. 1. DOI: 10.1109/ICASSP.1991.150433 (cit. on p. 106).

Kilger, Anne and Wolfgang Finkler (1995). *Incremental Generation for Real-time Applications*. Tech. rep. RR-95-11. Saarbrücken, Germany: DFKI (cit. on p. 47).

Klatt, Dennis H. (1979). "Synthesis by rule of segmental durations in English sentences". In: *Frontiers of Speech Communication Research*, pp. 287–299 (cit. on p. 156).

Lamere, Paul, Philip Kwok, William Walker, Evandro Gouvea, Rita Singh, and Peter Wolf (Sept. 2003). "Design of the CMU Sphinx-4 decoder". In: *Proceedings of EUROSPEECH*. ISCA. Geneva, Switzerland, pp. 1181–1184 (cit. on p. 103).

Larsen-Freeman, Diane and Lynne Cameron (2008). "Complex Systems and Applied Linguistics". In: SUB: A 2009 / 7836 (cit. on p. 29).

Lazaridis, Alexandros, Todor Ganchev, Theodoros Kostoulas, Iosif Mporas, and Nikos Fakotakis (Sept. 2010). "Phone duration modeling: Overview of techniques and performance optimization via feature selection in the context of emotional speech". In: *International Journal of Speech Technology* 13.3, pp. 175–188. DOI: 10.1007/s10772-010-9077-x (cit. on p. 156).

Lee, Cheongjae, Sangkeun Jung, and Gary Geunbae Lee (2008). "Robust Dialog Management with N-best Hypotheses Using Dialog Examples and Agenda". In: *Proc. of ACL-HLT*. Columbus, USA (cit. on pp. 102, 117).

Lee, K.F., H.W. Hon, and R. Reddy (1990). "An overview of the SPHINX speech recognition system". In: *Acoustics, Speech and Signal Processing, IEEE Transactions on* 38.1, pp. 35–45 (cit. on p. 103).

Lerner, Gene H. (Jan. 2002). "Turn Sharing: The Choral Co-Production Of Talk In Interaction". In: *The Language of Turn and Sequence*. Ed. by Cecilia E. Ford, Barbara A. Fox, and Sandra A. Thompson. Oxford, UK: Oxford University Press. Chap. 9, pp. 225–256 (cit. on pp. 149, 150).

– (2004). "Collaborative turn sequences". In: *Conversation Analysis: Studies from the First Generation*. Ed. by Gene H. Lerner. Pragmatics & Beyond. Amsterdam, The Netherlands: John Benjamins, pp. 225–256 (cit. on p. 150).

Levelt, William J.M. (1989). *Speaking: From Intention to Articulation*. Mit Pr (cit. on pp. 15, 27, 47, 175, 185, 208).

Levenshtein, Vladimir I. (Feb. 1966). "Binary codes capable of correcting deletions, insertions, and reversals". In: *Soviet Physics – Doklady* 10.8, pp. 707–710 (cit. on p. 68).

Lewis, James R. (2011). *Practical Speech User Interface Design*. CRC Press. ISBN: 978-1-4398-1584-7 (cit. on p. 40).

Local, John (Aug. 2007). "Phonetic detail and the organisation of talk-in-interaction". In: *Proceedings of the 16th ICPhS*. Saarbrücken, Germany (cit. on pp. 149–151).

Lohmann, K., C. Eschenbach, and C. Habel (2011). "Linking spatial haptic perception to linguistic representations: assisting utterances for tactile-map explorations". In:

Spatial information theory. Ed. by M. Egenhofer, N. Giudice, R. Moratz, and M. Worboys. Berlin, Heidelberg: Springer, pp. 328–349 (cit. on p. 193).

Lohmann, Kris, Matthias Kerzel, and Christopher Habel (2012). "Verbally Assisted Virtual-Environment Tactile Maps: A Prototype System". In: *Proceedings of the Workshop on Spatial Knowledge Acquisition with Limited Information Displays 2012*. (Aug. 31, 2012). Ed. by Christian Graf, Nicholas A. Giudice, and Falko Schmid. Seeon, Germany, pp. 25–30 (cit. on p. 194).

Lucas, Bruce (Sept. 2000). "VoiceXML for Web-based distributed conversational applications". In: *Commun. ACM* 43.9, pp. 53–57. ISSN: 0001-0782. DOI: 10.1145/348941.348985 (cit. on p. 40).

Maat, Mark ter (2011). "Response Selection and Turn-taking for a Sensitive Artificial Listening Agent". PhD thesis. University of Twente (cit. on p. 42).

Malsburg, Titus von der, Timo Baumann, and David Schlangen (2009). "TELIDA: A Package for Manipulation and Visualisation of Timed Linguistic Data". In: *Proceedings of SigDial 2009*. London, UK (cit. on pp. 20, 109).

Martin, D., A. Cheyer, and D. Moran (1999). "The Open Agent Architecture: a framework for building distributed software systems". In: *Applied Artificial Intelligence* 13.1/2, pp. 91–128. URL: citeseer.ist.psu.edu/martin99open.html (cit. on p. 83).

Matsuyama, Kyoko, Kazunori Komatani, Ryu Takeda, Toru Takahashi, Tetsuya Ogata, and Hiroshi G. Okuno (2010). "Analyzing User Utterances in Barge-in-able Spoken Dialogue System for Improving Identification Accuracy". In: *Proceedings of Interspeech* (cit. on pp. 175, 203).

McGraw, Ian (Sept. 13, 2012). Personal Communication. Portland, USA (cit. on p. 43).

McGraw, Ian and Alexander Gruenstein (Sept. 2012). "Estimating Word-Stability During Incremental Speech Recognition". In: *Proceedings of Interspeech*. ISCA. Portland, USA (cit. on pp. 43, 104, 127, 135).

McTear, Michael (2002). *Spoken Dialogue Technology. Toward the Conversational User-Interface*. Springer Verlag (cit. on pp. 22, 23, 36, 175).

Mermelstein, Paul (1976). "Distance Measures for Speech Recognition – Psychological and Instrumental". In: *Pattern Recognition and Artificial Intelligence, Proceedings of the Joint Workshop*. Ed. by C. H. Chen, pp. 374–388 (cit. on pp. 94, 95).

Miller, Geore (Nov. 1963). "Speaking in General. Review of J. H. Greenberg (Ed.), Universals of language". In: *Contemporary Psychology* 8.11, pp. 417–418. ISSN: 1554-0138. DOI: 10.1037/007084 (cit. on p. 31).

Miyazaki, Noboru, Mikio Nakano, and Kiyoaki Aikawa (2002). "Robust speech understanding using incremental understanding with n-best recognition hypotheses". In: *Joho Shori Gakkai Kenkyu Hokoku* 10 (cit. on p. 117).

Nakano, Mikio, Noboru Miyazaki, Jun-ichi Hirasawa, Kohji Dohsaka, and Takeshi Kawabata (1999). "Understanding Unsegmented User Utterances in Real-Time Spoken Dialogue Systems". In: *Proc. of ACL*. College Park, USA (cit. on p. 117).

Nooteboom, Sieb G. (1997). "The prosody of speech: Melody and rhythm". In: *The Handbook of Phonetic Sciences*. Ed. by W. J. Hardcastle and J. Laver. 1st ed. Oxford: Blackwell, pp. 640–673 (cit. on p. 213).

Odell, Julian J., Valtcho Valtchev, Phil C. Woodland, and Steve J. Young (1994). "A One Pass Decoder Design For Large Vocabulary Recognition". In: *Proceedings of the ARPA Workshop on Human Language Technology*. Plainsboro, USA (cit. on p. 100).

Oppenheim, A. V. and R. W. Schafer (Sept. 2004). "DSP history - From frequency to quefrency: a history of the cepstrum". In: *IEEE Signal Processing Magazine* 21.5, pp. 95–106. ISSN: 1053-5888. DOI: 10.1109/MSP.2004.1328092 (cit. on p. 95).

Ortmanns, Stefan and Hermann Ney (2000). "Look-ahead techniques for fast beam search". In: *Computer Speech & Language* 14, pp. 15–32 (cit. on p. 122).

Oshry, Matt, Paolo Baggia, Kenneth Rehor, Milan Young, Rahul Akolkar, Xu Yang, Jim Barnett, Rafah Hosn, RJ Auburn, Jerry Carter, Scott McGlashan, Michael Bodell, and Daniel C. Burnett (Dec. 2009). *Voice Extensible Markup Language (VoiceXML) 3.0*. W3C Working Draft. http://www.w3.org/TR/2009/WD-voicexml30-20091203/. W3C (cit. on p. 175).

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu (July 2002). "BLEU: A Method for Automatic Evaluation of Machine Translation". In: *Proceedings of ACL*. Philadelphia, USA, pp. 311–318 (cit. on p. 66).

Parnas, David Lorge (1979). "Designing Software for Ease of Extension and Contraction". In: *Software Engineering, IEEE Transactions on* SE-5.2, pp. 128–138. ISSN: 0098-5589. DOI: 10.1109/TSE.1979.234169 (cit. on p. 33).

Pfau, T. and G. Ruske (Aug. 1996). "Estimating the Speaking Rate by Vowel Detection". In: *Proceedings of the 14th International Conference of Phonetic Science*. San Francisco, USA (cit. on p. 155).

Pfitzinger, Hartmut R. (Dec. 1998). "Local Speech Rate as a Combination of Syllable and Phone Rate". In: *Proceedings of the 5th International Conference on Speech and Language Processing*. Vol. 3. Sydney, Australia, pp. 1087–1090 (cit. on pp. 156, 157).

Pieraccini, Roberto and David Lubensky (2005). "Spoken Language Communication with Machines: The Long and Winding Road from Research to Business". In: *IEA/AIE*. Ed. by Moonis Ali and Floriana Esposito. Vol. 3533. Lecture Notes in Computer Science. Springer, pp. 6–15. ISBN: 3-540-26551-1 (cit. on p. 14).

Placeway, P., S. Chen, M. Eskenazi, U. Jain, V. Parikh, B. Raj, M. Ravishankar, R. Rosenfeld, K. Seymore, M. Siegler, R. Stern, and E. Thayer (1997). "The 1996 hub-4 sphinx-3 system". In: *Proc. DARPA Speech recognition workshop*, pp. 85–89 (cit. on p. 103).

Poesio, Massimo and Hannes Rieser (July 2004). *Completions and continuations in dialogue: a preliminary account*. Ed. by Jonathan Ginzburg and Enric Vallduví. Invited lecture at Catalog, The 8th Workshop on the Semantics and Pragmatics

of Dialogue. URL: http://www.upf.edu/dtf/personal/enricvallduvi/catalog04/invabst/catalog04-poesio.ppt (cit. on p. 151).

Poesio, Massimo and Hannes Rieser (Feb. 2010). "Completions, coordination, and alignment in dialogue". In: *Dialogue and Discourse* 1.1, pp. 1–89. DOI: 10.5087/dad.2010.001 (cit. on p. 151).

Poncin, Kristina and Hannes Rieser (2006). "Multi-speaker utterances and co-ordination in task-oriented dialogue". In: *Journal of Pragmatics* 38.5, pp. 718–744 (cit. on p. 151).

Postel, J. and J. Reynolds (Oct. 1985). *File Transfer Protocol*. RFC 959 (INTERNET STANDARD). Updated by RFCs 2228, 2640, 2773, 3659, 5797. Internet Engineering Task Force. URL: http://www.ietf.org/rfc/rfc959.txt (cit. on p. 32).

Proceedings of DECALOG, the 11th International Workshop on the Semantics and Pragmatics of Dialogue (June 2007). Trento, Italy.

Purver, Matthew, Arash Eshghi, and Julian Hough (Jan. 2011). "Incremental Semantic Construction in a Dialogue System". In: *Proceedings of the 9th International Conference on Computational Semantics (IWCS)*. ACL. Oxford, UK: Association for Computational Linguistics (cit. on p. 152).

Purver, Matthew, Florin Ratiu, and Lawrence Cavedon (2006). "Robust Interpretation in Dialogue by Combining Confidence Scores with Contextual Features". In: *Proc. of Interspeech*. Pittsburgh, USA (cit. on pp. 102, 117).

Purver, Matthew, Christine Howes, Patrick G. T. Healey, and Eleni Gregoromichelaki (Sept. 2009). "Split Utterances in Dialogue: a Corpus Study". In: *Proceedings of SIGdial*. London, UK, pp. 262–271 (cit. on pp. 150, 151).

Pétursson, Magnús and Joachim Neppert (1996). *Elementarbuch der Phonetik*. 2nd ed. Hamburg, Germany: Buske (cit. on pp. 28, 94).

Quené, Hugo (2007). "On the just noticeable difference for tempo in speech". In: *Journal of Phonetics* 35.3, pp. 353 –362. ISSN: 0095-4470. DOI: 10.1016/j.wocn.2006.09.001 (cit. on p. 213).

Rabiner, Lawrence R. and Biing-Hwang Juang (1993). *Fundamentals of Speech Recognition*. Upper Saddle River, NJ, USA: Prentice Hall PTR (cit. on pp. 89, 97, 98).

Ramalingam, Ganesan and Thomas Reps (1993). "A categorized bibliography on incremental computation". In: *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, pp. 502–510 (cit. on p. 51).

Rashid, Rick (Oct. 25, 2012). "Microsoft Research and the Evolution of Computing". In: *Computing in the 21st Century. Computing, Naturally*. Keynote Address. Microsoft Research Asia. Tianjin, China. URL: http://v.youku.com/v_show/id_XNDc1NDAyMTcy.html?f=18558089. Video recording (cit. on p. 149).

Raux, A., B. Langner, A. Black, and M. Eskenazi (2003). "LET'S GO: Improving Spoken Dialog Systems for the Elderly and Non-natives". In: *Proc. of Eurospeech*. Geneva, Switzerland (cit. on p. 41).

Raux, Antoine and Maxine Eskenazi (2008). "Optimizing Endpointing Thresholds using Dialogue Features in a Spoken Dialogue System". In: *Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue*. Columbus, Ohio: Association for Computational Linguistics, pp. 1–10. URL: http://www.aclweb.org/anthology/W/W08/W08-0101 (cit. on pp. 140, 147).

– (2009). "A finite-state turn-taking model for spoken dialog systems". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pp. 629–637 (cit. on pp. 41, 139, 147, 148).

Rayner, Manny, D Carter, V Digalais, and P Price (1994). "Combining knowledge sources to reorder n-best speech hypothesis lists". In: *Proc. of the ARPA Human Language Technology Workshop*. Plainsboro, USA (cit. on p. 102).

Razik, Joseph, Odile Mella, Dominique Fohr, and Jean-Paul Haton (2008). "Frame-synchronous and local confidence measures for on-the-fly automatic speech recognition". In: *Proceedings of Interspeech*, pp. 1517–1520 (cit. on pp. 102, 136).

– (2011). "Frame-synchronous and Local Confidence Measures for Automatic Speech Recognition". In: *International Journal of Pattern Recognition and Artificial Intelligence* 25.2, pp. 157–182. ISSN: 0218-0014. DOI: 10.1142/S0218001411008543 (cit. on pp. 102, 136).

Reddy, D., L. Erman, R. Fennell, and R. Neely (1976). "The Hearsay-I Speech Understanding System: An Example of the Recognition Process". In: *IEEE Transactions on Computers* C-25.4, pp. 422–431 (cit. on p. 47).

Reiter, Ehud and Anja Belz (2009). "An Investigation into the Validity of Some Metrics for Automatically Evaluating Natural Language Generation Systems". In: *Computational Linguistics* 35.4, pp. 529–558 (cit. on p. 66).

Reiter, Ehud and Somayajulu Sripada (2002). "Human Variation and Lexical Choice". In: *Computational Linguistics* 28, pp. 545–553 (cit. on p. 202).

Roy, Nicholas, Joelle Pineau, and Sebastian Thrun (2000). "Spoken dialogue management using probabilistic reasoning". In: *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, pp. 93–100 (cit. on p. 221).

Rudnicky, A., E. Thayer, P. Constantinides, C. Tchou, R. Stern, K. Lenzo, W. Xu, and A. Oh (1999). "Creating Natural Dialogs in the Carnegie Mellon Communicator System". In: *Proc. of Eurospeech*. Budapest, Hungary (cit. on p. 41).

Sacks, Harvey and Emanuel A. Schegloff (1979). "Two Preferences in the Organization of Reference to Persons in Conversation and Their Interaction". In: *Everyday*

Language: Studies in Ethnomethodology. Ed. by George Psathas. New York, USA: Irvington Publishers, Inc., pp. 15–21 (cit. on p. 143).

Sacks, Harvey, Emanuel A. Schegloff, and Gail A. Jefferson (1974). "A Simplest Systematic for the Organization of Turn-Taking in Conversation". In: *Language* 50, pp. 735–996 (cit. on pp. 31, 32, 150).

Sagae, Kenji, David DeVault, and David Traum (2010). "Interpretation of partial utterances in virtual human dialogue systems". In: *Proceedings of the NAACL HLT 2010 Demonstration Session*. Association for Computational Linguistics, pp. 33–36 (cit. on p. 152).

Sagae, Kenji, Gwen Christian, David DeVault, and David Traum (2009). "Towards Natural Language Understanding of Partial Speech Recognition Results in Dialogue Systems". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*. Association for Computational Linguistics, pp. 53–56 (cit. on pp. 17, 42, 48, 64, 152).

Schegloff, Emanuel A (1968). "Sequencing in Conversational Openings". In: *American Anthropologist*. New Series 70.6 (cit. on p. 31).

– (2000). "Overlapping talk and the organization of turn-taking for conversation". In: *Language in society* 29.1, pp. 1–63 (cit. on p. 174).

Schlangen, David (Sept. 2006). "From Reaction to Prediction: Experiments with Computational Models of Turn-Taking". In: *Proceedings of Interspeech 2006*. Pittsburgh, USA (cit. on pp. 139, 148).

Schlangen, David, Timo Baumann, and Michaela Atterer (2009). "Incremental Reference Resolution: The Task, Metrics for Evaluation, and a Bayesian Filtering Model that is Sensitive to Disfluencies". In: *Proceedings of SigDial 2009*. London, UK (cit. on pp. 17, 49, 64, 69).

Schlangen, David and Raquel Fernández (2007). "Speaking through a noisy channel: Experiments on inducing clarification behaviour in human-human dialogue". In: *Proceedings of Interspeech 2007* (cit. on p. 135).

Schlangen, David and Hannes Rieser, eds. (2011). *Dialogue & Discourse* 2.1. Special Issue on Incremental Processing in Dialogue. ISSN: 2152-9620 (cit. on p. 16).

Schlangen, David and Gabriel Skantze (2009). "A General, Abstract Model of Incremental Dialogue Processing". In: *Proceedings of the EACL*. Athens, Greece, pp. 710–718 (cit. on pp. 42, 44, 48, 49, 54, 74, 80).

– (2011). "A General, Abstract Model of Incremental Processing". In: *Dialogue and Discourse* 2.1, pp. 83–111 (cit. on pp. 44, 54, 74–76, 81).

Schlangen, David, Timo Baumann, Hendrik Buschmeier, Okko Buß, Stefan Kopp, Gabriel Skantze, and Ramin Yaghoubzadeh (Sept. 2010). "Middleware for Incremental Processing in Conversational Agents". In: *Proceedings of SigDial 2010*. Tokyo, Japan (cit. on pp. 20, 83, 84).

Schröder, Marc (2004). "Dimensional emotion representation as a basis for speech synthesis with non-extreme emotions". In: *Proc. Workshop on Affective Dialogue Systems*, pp. 209–220 (cit. on p. 216).

Schröder, Marc and Stefan Breuer (2004). "XML Representation Languages as a Way of Interconnecting TTS Modules". In: *Proceedings of ICSLP* (cit. on pp. 156, 184).

Schröder, Marc and Jürgen Trouvain (Oct. 2003). "The German Text-to-Speech Synthesis System MARY: A Tool for Research, Development and Teaching". In: *International Journal of Speech Technology* 6.3, pp. 365–377. ISSN: 1572-8110. DOI: 10.1023/A:1025708916924 (cit. on pp. 85, 156, 166, 184).

Schulzrinne, H., S. Casner, R. Frederick, and V. Jacobson (July 2003). *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550 (Standard). URL: http://www.ietf.org/rfc/rfc3550.txt (cit. on p. 85).

Selfridge, Ethan, Iker Arizmendi, Peter Heeman, and Jason Williams (June 2011). "Stability and Accuracy in Incremental Speech Recognition". In: *Proceedings of the SIGDIAL 2011 Conference*. Portland, Oregon: Association for Computational Linguistics, pp. 110–119. URL: http://www.aclweb.org/anthology/W/W11/W11-2014 (cit. on pp. 41, 53, 67, 104, 117).

– (2012a). "A Temporal Simulator for Developing Turn-Taking Methods for Spoken Dialogue Systems". In: *Proceedings of SigDial*. Seoul, Korea (cit. on p. 135).

Selfridge, Ethan, Peter Heeman, Iker Arizmendi, and Jason D. Williams (Dec. 2012b). "Demonstrating the Incremental Interaction Manager in an end-to-end "Lets Go!" dialogue system". In: *IEEE Workshop on Spoken Language Technology*. System Demonstration. Miami, USA (cit. on p. 41).

Selfridge, Ethan, Iker Arizmendi, Peter A. Heeman, and Jason D. Williams (2012c). "Integrating Incremental Speech Recognition and POMDP-based Dialogue Systems". In: *Proceedings of SIGdial 2012*. Seoul, Korea (cit. on pp. 41, 221).

Seneff, S., E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue (1998). "Galaxy II: A Reference Architecture for Conversational System Development". In: *Proceedings of ICSLP*. Sydney, Australia (cit. on p. 41).

Serébrennikov, A collective of authors led by Boris Aleksandrovich Serébrennikov (1975). *Allgemeine Sprachwissenschaft*. Ed. by Hans Zikmund and Günther Feudel. Vol. II. Akademie-Verlag Berlin (cit. on pp. 25–27).

Seward, Alexander (2003). "Low-latency incremental speech transcription in the synface project". In: *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech), Geneva, Switzerland, 2003: vol 2*, pp. 1141–1144 (cit. on p. 104).

Shannon, Claude E. and Warren Weaver, eds. (1949). *The Mathematical Theory of Communication*. Paperback edition, September 1969. The University of illinois Press (cit. on pp. 23, 24).

Shannon, Claude (1949). "The Mathematical Theory of Communication". In: *The Mathematical Theory of Communication*. Paperback edition, September 1969; reprinted from the Bell System Technical Journal, July and October 1948. The University of illionois Press, pp. 29–125 (cit. on p. 23).

Sharabi, Moshe and Moshe Davidow (2010). "Service quality implementation: Problems and solutions". In: *International Journal of Quality and Service Sciences* 2 (2), pp. 189–205. DOI: 10.1108/17566691011057357 (cit. on p. 14).

Silverman, K., M. Beckman, J. Pitrelli, M. Ostendorf, C. Wightman, P. Price, J. Pierrehumbert, and J. Hirschberg (1992). "ToBI: A standard for labeling English prosody". In: *Second International Conference on Spoken Language Processing* (cit. on p. 179).

Singh, Rita (2004). "Sphinx". In: *Berkshire Encyclopedia of Human-Computer Interaction: When Science Fiction Becomes Science Fact*. Ed. by William Sims Bainbridge. Vol. 2. Berkshire Publishing, pp. 694–695 (cit. on p. 103).

Sjölander, K. and J. Beskow (2000). "Wavesurfer-an open source speech tool". In: *Proceedings of ICSLP*. Vol. 4, pp. 464–467 (cit. on p. 107).

Skantze, Gabriel (2007). "Error Handling in Spoken Dialogue Systems". PhD thesis. Stockholm, Sweden: KTH (cit. on p. 42).

– (2008). "Galatea: A Discourse Modeller Supporting Concept-Level Error Handling in Spoken Dialogue Systems". In: *Recent Trends in Discourse and Dialogue*. Ed. by Laila Dybkjær and Wolfgang Minker. Vol. 39. Text, Speech and Language Technology. Springer Netherlands, pp. 155–189. ISBN: 978-1-4020-6820-1. DOI: 10.1007/978-1-4020-6821-8_7 (cit. on p. 221).

– (2010). *Jindigo: a Java-based Framework for Incremental Dialogue Systems*. Tech. rep. Stockholm, Sweden: KTH. URL: http://www.speech.kth.se/prod/publications/files/3654.pdf (cit. on pp. 42, 84).

Skantze, Gabriel and Anna Hjalmarsson (Sept. 2010). "Towards Incremental Speech Generation in Dialogue Systems". In: *Proceedings of SIGdial*. Tokyo, Japan (cit. on pp. 175, 188, 192, 210).

Skantze, Gabriel and David Schlangen (Apr. 2009). "Incremental Dialogue Processing in a Micro-Domain". In: *Proceedings of EACL 2009*. Athens, Greece (cit. on pp. 19, 42, 73, 143, 145–148).

Skovenborg, Esben and Søren H Nielsen (2004). "Evaluation of different loudness models with music and speech material". In: *Proceedings of the 117th AES convention, San Francisco* (cit. on p. 85).

Skuplik, Kristina (1999). *Satzkooperationen. Definition und empirische Untersuchung*. Tech. rep. 1999/03. Bielefeld, Germany: SFB 360, Universität Bielefeld (cit. on p. 151).

Soeda, Shunsuke and Nigel Ward (2001). "Design for a System able to use Time-Critical Spoken Advice". In: *Proceedings of the 15th Annual Conference of JSAI*. Matsue, Japan (cit. on p. 135).

Stone, Matthew, Christine Doran, Bonnie Webber, Tonia Bleam, and Martha Palmer (2003). "Microplanning with Communicative Intentions: The SPUD System". In: *Computational Intelligence* 19, pp. 311–381 (cit. on pp. 35, 201).

Swadesh, Morris (1937). "The Phonemic Interpretation of Long Consonants". In: *Language* 13.1, pp. 1–10 (cit. on p. 27).

Tanenbaum, A.S. (1981). "Network protocols". In: *ACM Computing Surveys (CSUR)* 13.4, pp. 453–489 (cit. on p. 25).

Tanenhaus, Michael K., M.J. Spivey-Knowlton, K.M. Eberhard, and J.C. Sedivy (1995). "Integration of visual and linguistic information in spoken language comprehension". In: *Science* 268.5217, pp. 1632–1634 (cit. on p. 15).

Taylor, Martin and David Waugh (2000). "Dialogue analysis using layered protocols". In: *Abduction, Belief and Context in Dialogue. Studies in Computational Pragmatics*. Ed. by Harry Bunt and William Black. Natural Language Processing. John Benjamins, pp. 189–231 (cit. on pp. 28, 29).

Taylor, Maurice Martin (1988). "Layered protocols for computer-human dialogue. I: Principles". In: *International Journal of Man-Machine Studies* 28.2-3, pp. 175 –218. ISSN: 0020-7373. DOI: 10.1016/S0020-7373(88)80036-1 (cit. on p. 28).

Taylor, P. A. (2000). "Concept-to-Speech Synthesis by Phonological Structure Matching". English. In: *Philosophical Transactions: Mathematical, Physical and Engineering Sciences* 358.1769, pp. 1403–1417. ISSN: 1364503X. URL: http://www.jstor.org/stable/2666826 (cit. on p. 178).

Taylor, Paul (2009). *Text-to-Speech Synthesis*. Cambridge Univ Press. ISBN: 978-0521899277 (cit. on pp. 89, 94–96, 178, 179, 181).

Ternes, Elmar (1999). *Einführung in die Phonologie*. 2nd ed. Wissenschaftliche Buchgesellschaft. ISBN: 978-3534138708 (cit. on p. 94).

Theune, M., K. Meijs, D. Heylen, and R. Ordelman (2006). "Generating expressive speech for storytelling applications". In: *Audio, Speech, and Language Processing, IEEE Transactions on* 14.4, pp. 1137–1144 (cit. on p. 161).

Thórisson, Kristinn R. (1997). "Gandalf: an embodied humanoid capable of real-time multimodal dialogue with people". In: *Proceedings of the first international conference on Autonomous agents*. AGENTS '97. Marina del Rey, California, USA: ACM, pp. 536–537. ISBN: 0-89791-877-0. DOI: 10.1145/267658.267823 (cit. on p. 42).

Thórisson, Kristinn R. (2008). "Modeling Multimodal Communication as a Complex System". In: *Modeling Communication with Robots and Virtual Humans*. Ed. by Ipke Wachsmuth and Günther Knoblich. Vol. 4930. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 143–168. ISBN: 978-3-540-79036-5. DOI: 10.1007/978-3-540-79037-2_8 (cit. on p. 44).

Thórisson, Kristinn R. and Gudny Ragna Jonsdottir (2008). "A Granular Architecture for Dynamic Realtime Dialogue". In: *Intelligent Virtual Agents*. Ed. by Helmut Prendinger, James Lester, and Mitsuru Ishizuka. Vol. 5208. Lecture Notes in Com-

puter Science. Springer Berlin Heidelberg, pp. 131–138. ISBN: 978-3-540-85482-1. DOI: 10.1007/978-3-540-85483-8_13 (cit. on pp. 42, 44).

Thórisson, Kristinn R., Olafur Gislason, Gudny Ragna Jonsdottir, and Hrafn Th. Thórisson (2010). "A Multiparty Multimodal Architecture for Realtime Turntaking". In: *Intelligent Virtual Agents*. Ed. by Jan Allbeck, Norman Badler, Timothy Bickmore, Catherine Pelachaud, and Alla Safonova. Vol. 6356. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 350–356. ISBN: 978-3-642-15891-9. DOI: 10.1007/978-3-642-15892-6_37 (cit. on p. 44).

Toda, Tomoki and Keiichi Tokuda (2007). "A speech parameter generation algorithm considering global variance for HMM-based speech synthesis". In: *IEICE transactions on information and systems* 90.5, pp. 816–824 (cit. on p. 181).

Tokuda, Keiichi, Takayoshi Yoshimura, Takashi Masuko, Takao Kobayashi, and Tadashi Kitamura (2000). "Speech Parameter Generation Algorithms for HMM-based Speech Synthesis". In: *Proceedings of ICASSP* (cit. on p. 181).

Visser, Thomas, David Traum, David DeVault, and Rieks op den Akker (Sept. 2012). "Toward a Model for Incremental Grounding in Spoken Dialogue Systems". In: *Workshop on Real-Time Conversations with Virtual Agents (RCVA 2012)*. Santa Cruz, USA (cit. on p. 42).

Vogt, F., O. Guenther, A. Hannam, K. van den Doel, J. Lloyd, L. Vilhan, R. Chander, J. Lam, C. Wilson, K. Tait, et al. (2005). "ArtiSynth designing a modular 3D articulatory speech synthesizer". In: *The Journal of the Acoustical Society of America* 117, p. 2542 (cit. on p. 183).

Wachsmuth, Sven, Gernot A. Fink, and Gerhard Sagerer (Sept. 1998). "Integration of Parsing and Incremental Speech Recognition". In: *Proc. European Signal Processing Conference*. Vol. 1. Rhodes, pp. 371–375 (cit. on pp. 48, 53, 104, 122, 123, 136).

Walker, Marilyn A., Diane J. Litman, Candace A. Kamm, and Alicia Abella (1998). "Evaluating spoken dialogue agents with PARADISE: Two case studies". In: *Computer Speech and Language* 12.3 (cit. on p. 145).

Walker, Willie, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel (2004). *Sphinx-4: A Flexible Open Source Framework for Speech Recognition*. Tech. rep. SMLI TR2004-0811. Sun Microsystems Inc. (cit. on p. 85).

Ward, Nigel G., Alejandro Vega, and Timo Baumann (2012). "Prosodic and Temporal Features for Language Modeling for Dialog". In: *Speech Communication* 54.2, pp. 161–174. ISSN: 0167-6393. DOI: 10.1016/j.specom.2011.07.009 (cit. on p. 103).

Ward, Nigel G., Anais G. Rivera, Karen Ward, and David G. Novick (2005). "Root causes of lost time and user stress in a simple dialog system". In: *INTERSPEECH 2005 - Eurospeech, 9th European Conference on Speech Communication and Technology, Lisbon, Portugal, September 4-8, 2005*. ISCA, pp. 1565–1568. DOI: http:

//www.isca-speech.org/archive/interspeech_2005/i05_1565.html (cit. on pp. 15, 45).

Ward, Nigel (2006). "Non-lexical conversational sounds in American English". In: *Pragmatics & Cognition* 14.1, pp. 129–182. ISSN: 1569-9943. DOI: doi:10.1075/pc.14.1.08war (cit. on p. 32).

Ward, Nigel, Olac Fuentes, and Alejandro Vega (Sept. 2010). "Dialog Prediction for a General Model of Turn-Taking". In: *Proceedings of Interspeech*. Tokyo, Japan (cit. on pp. 139, 147, 148).

Weaver, Warren (1949). "Recent Contributions to the Mathematical Theory of Communication". In: *The Mathematical Theory of Communication*. paperback edition, September 1969. The University of illionois Press, pp. 1–28 (cit. on pp. 23, 24).

Weilhammer, Karl and Susen Rabold (2003). "Durational Aspects in Turn Taking". In: *Proceedings of the 15th International Congress of Phonetic Sciences*. Barcelona, Spain. URL: http://www.phonetik.uni-muenchen.de/Publications/WeilhammerRabold-03-ICPhS.pdf (cit. on p. 148).

Welbergen, Herwin van (2011). "Behavior Generation for Interpersonal Coordination with Virtual Humans". PhD thesis. University of Twente (cit. on p. 176).

Wesseling, W., R.J.J.H. Son, and L.C.W. Pols (2006). "On the Sufficiency and Redundancy of Pitch for TRP Projection". In: *Ninth International Conference on Spoken Language Processing*. ISCA (cit. on p. 29).

Williams, Jason D. (2008). "Exploiting the ASR N-Best by tracking multiple dialog state hypotheses". In: *Proceedings of Interspeech*. Brisbane, Australia, pp. 191–194 (cit. on pp. 102, 117).

Wilson, Margaret and Thomas P. Wilson (2005). "An oscillator model of the timing of turn-taking". In: *Psychonomic Bulletin & Review* 12.6, pp. 957–968. ISSN: 1531-5320. DOI: 10.3758/BF03206432 (cit. on p. 168).

Wirén, Mats (1992). "Studies in Incremental Natural Language Analysis". Unpublished doctoral dissertation. Sweden: Linkoping University (cit. on p. 59).

Xing, Zhengzheng, Jian Pei, and Philip Yu (2009). "Early Prediction on Time Series: A Nearest Neighbor Approach". In: *Proceedings of the International Joint Conference on Artificial Intelligence*. Pasadena, USA, pp. 1297–1302 (cit. on p. 136).

Yngve, Victor H (1970). "On getting a word in edgewise". In: *Chicago Linguistics Society, 6th Meeting*, pp. 567–578 (cit. on p. 32).

Yoshimura, Takayoshi, Keiichi Tokuda, Takashi Masuko, Takao Kobayashi, and Tadashi Kitamura (1998). "Duration Modeling for HMM-based Speech Synthesis". In: *Proceedings of the 5th International Conference on Spoken Language Processing* (cit. on p. 156).

Young, Steve J., NH Russell, and JHS Thornton (1989). *Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems*. Tech. rep. CUED/F-INFENG/TR. Cambridge University Engineering Department (cit. on pp. 47, 100).

Zemack, Matti (2007). "Implementing methods for equal loudness in radio broadcasting". Master's thesis. Stockholm, Sweden: Skolan för datavetenskap och kommunikation. Kungliga tekniska högskolan (cit. on p. 85).

Zen, Heiga, Keiichi Tokuda, Takashi Masuko, Takao Kobayashi, and Tadashi Kitamura (2007a). "A Hidden Semi-Markov Model-Based Speech Synthesis System". In: *IEICE Transactions on Information and Systems* 90.5, pp. 825–834. ISSN: 1745-1361 (cit. on p. 181).

Zen, Heiga, Tomoki Toda, Masaru Nakamura, and Keiichi Tokuda (2007b). "Details of the Nitech HMM-based speech synthesis system for the Blizzard Challenge 2005". In: *IEICE Transactions on Information and Systems* 90.1, pp. 325–333. ISSN: 1745-1361 (cit. on p. 182).

Zilberstein, Shlomo (1996). "Using Anytime Algorithms in Intelligent Systems". In: *AI magazine* 17.3, p. 73 (cit. on pp. 49, 52).

Zimmermann, H. (1980). "OSI reference model–The ISO model of architecture for open systems interconnection". In: *IEEE Transactions on Communications* 28.4, pp. 425–432 (cit. on p. 25).