

---

Institut für Informatik der Technischen Universität München

---

# Conversation-and-Control

Extending Speech-controlled Graphical User Interfaces

Andreas Löhr

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Jürgen Schmidhuber

Prüfer der Dissertation:

1. Univ.-Prof. Bernd Brügge, Ph.D.
2. Prof. Daniel P. Siewiorek, Ph.D.  
Carnegie Mellon University, Pittsburgh/USA

Die Dissertation wurde am 21.6.2006 bei der Technischen Universität München eingereicht und durch die Fakultät am 11.9.2006 angenommen.



## Zusammenfassung

Die heutigen Standardeingabegeräte für graphische Benutzeroberflächen, Maus und Tastatur, können von Benutzern, deren motorische Fähigkeiten beeinträchtigt sind, nicht verwendet werden. Beispiele hierfür sind Benutzer mit krankheitsbedingt eingeschränkter Motorik und Benutzer in Anwendungsdomänen mit manuellen Tätigkeiten. Die vorliegende Arbeit beschäftigt sich mit dem Forschungsgebiet *sprachgesteuerte graphische Benutzeroberflächen*, welches durch den Einsatz von Sprache als Eingabemodalität für graphische Benutzeroberflächen nach Lösungen für dieses Problem sucht.

Der Stand der Forschung wird zunächst in zwei Gruppen eingeteilt: *sprachbasierte Mausemulation* und *Kommando-und-Steuerung*. Sprachbasierte Mausemulation verwendet Sprachkommandos zur Emulation der Mausfunktionen, bzw. der Tastatur. Der Benutzer kann mittels Sprachkommandos indirekt über die Funktionen des Mauszeigers graphische Objekte manipulieren. Kommando-und-Steuerung verwendet Sprachkommandos zur direkten Steuerung der interaktiven Funktionalität von graphischen Objekten.

Sprachbasierte Mausemulation ist durch die betriebssystemweit verfügbaren Funktionen des Mauszeigers universell anwendbar, während die Sprachkommandos für Kommando-und-Steuerung vom jeweils zu steuernden graphischen Objekt explizit unterstützt werden müssen. Sprachbasierte Mausemulation erfordert mehr Interaktion durch den Benutzer als Kommando-und-Steuerung, somit können Aufgaben mittels Kommando-und-Steuerung schneller ausgeführt werden. Die mittels Kommando-und-Steuerung erreichten Ausführungszeiten sind jedoch derzeit noch um mindestens 50% höher als die Ausführungszeiten, die mit mausgesteuerten graphischen Oberflächen erreicht werden.

Um die Ausführungszeiten mittels Kommando-und-Steuerung zu verkürzen, schlägt diese Dissertation den Ansatz *Konversation-und-Steuerung* vor, welcher auf einer Erweiterung von Kommando-und-Steuerung durch drei neue Interaktionstechniken, genannt *qualifizierte Aktivierung*, *automatische Behebung von Informationsfehlern* und *qualifizierte Rückmeldung*, beruht. Qualifizierte Aktivierung und automatische Behebung von Informationsfehlern reduzieren die durchschnittliche Anzahl der benötigten Sprachkommandos, während qualifizierte Rückmeldung auf eine Reduzierung der durchschnittlichen Länge der Sprachkommandos abzielt. Dadurch wird eine Reduzierung der durchschnittlichen Interaktionszeit erreicht, was sich in einer reduzierten durchschnittlichen Ausführungszeit niederschlägt.

Die theoretischen Grundlagen für Konversation-und-Steuerung werden von einem in dieser Arbeit entwickelten formalen Modell für sprachgesteuerte graphische Oberflächen abgeleitet und formal verifiziert. Um die Machbarkeit von Konversation-und-Steuerung zu zeigen wird auf Basis des formalen Modells ein Implementierungskonzept entworfen und dessen Realisierung in Form eines Frameworks für Konversation-und-Steuerung beschrieben. Mit Hilfe des Frameworks wird ein Experiment durchgeführt, womit die formal prognostizierte Reduzierung der durchschnittlichen Ausführungszeiten empirisch nachgewiesen wird. Das Framework diente ferner als Grundlage für die Erstellung von prototypischen Anwendungen mit graphischen Oberflächen aus der Anwendungsdomäne *Mobile Wartung*, welche Konversation-und-Steuerung unterstützen.



## Abstract

Today's standard input devices for graphical user interfaces, mouse and keyboard, cannot be utilized by users whose motor functions are limited. Examples include users suffering from motor restrictions due to illnesses and users in application domains with manual activities. This dissertation engages in the research area *speech-controlled graphical user interfaces* which aims at finding a solution to this problem by utilizing speech as input modality for graphical user interfaces.

We begin with defining a taxonomy which differentiates the state of the art into two groups: *speech-based mouse emulation* and *command-and-control*. Speech-based mouse emulation uses spoken commands to emulate the functions of the mouse device (the keyboard, respectively). The user manipulates graphical objects indirectly by the functions of the mouse cursor, which are triggered by spoken commands. Command-and-control uses spoken commands to facilitate a direct control of interactive functions of graphical objects.

Speech-based mouse emulation is universally applicable due to the operating system-wide available functions of the mouse cursor, whereas the spoken commands for command-and-control must be supported explicitly by each specific graphical object. Speech-based mouse emulation requires more user interaction than command-and-control. Thus, command-and-control requires less time to complete tasks than speech-based mouse emulation. However, the task completion times which can currently be achieved using command-and-control are at least 50% higher than the task completion times which can be achieved using mouse-controlled graphical user interfaces.

For achieving lower task completion times with command-and-control we propose the approach *conversation-and-control* which is based on an extension of command-and-control by three new interaction techniques called *qualified activation*, *automatic information error recovery* and *qualified feedback*. Qualified activation and automatic information error recovery aim at reducing the average number of necessary spoken commands, and qualified feedback aims at reducing the average length of necessary spoken commands. Thus, we achieve a reduction of the average interaction time which results in a reduction of the average task completion time.

We derive the theoretical concepts for conversation-and-control from a general model for speech-controlled graphical user interfaces, which we have developed during this work. In the first instance, we use a formal approach to verify these concepts. To show the practical feasibility of conversation-and-control, we develop an implementation concept for conversation-and-control on the basis of the formal model for speech-controlled graphical user interfaces. We realize the implementation concept in the form of a framework for conversation-and-control. Using this framework we conduct an experiment, which provides empirical confirmation of the formally predicted reduction of the average task completion time of conversation-and-control compared to command-and-control. We furthermore create prototypes of applications with graphical user interfaces supporting conversation-and-control in the application domain *mobile maintenance*.



---

## Acknowledgements

It is a pleasure to express my gratitude to those to whom I am indebted, directly and indirectly, in writing this thesis. First of all, I am deeply thankful to my advisor Professor Bernd Brügge, Ph.D., who supported me from the first day on we met. He was always sure that I was capable of such a work, especially then, when I was not. I am very grateful to Bernd Brügge for supporting my work in every aspect, for giving me the opportunity and freedom to realize my ideas, for providing guidance and advice when I needed it, and for teaching me how to do research and how to express my thoughts.

Further thanks are due to Professor Daniel Siewiorek, Ph.D., who contributed valuable feedback, suggestions and insights to this work. A great thank also goes to Jim Beck, Ph.D., to Professor Richard Martin, Ph.D., and to Dr. Horst Mauersberg, who encouraged me and who made it possible to get involved in the world of research far beyond I ever imagined. Thanks to Bernd Brügge, to Professor Gudrun Klinker, Ph.D., and to all the comrades-in-arms for interesting discussions and fair comments during doctoral seminars. Many thanks to you, Ralf Pfléggar and Tobias Weishäupl, for not getting annoyed by me yelling at my PC.

My parents and my family have played a major role in successfully finishing the endeavor of writing this dissertation. Sincere thanks to you, Mom and Dad, for all the continuative support, not only throughout the last years. My special thank goes to Stephanie Ludwig for extraordinary patience and appreciation, and for taking care of secular affairs, especially during the last weeks of writing. Also, many thanks to my sister and all my friends – you gave me the opportunity to step back and gather new strengths in bad times.

This thesis was partially supported by the High-Tech Offensive der Bayerischen Staatskanzlei (Projekt UMTS – Mobile Wartung). Thank you, Fuat Atabey, Bernd Brügge, Bernhard Gruber, Volker Hafner, Korbinian Herrmann, Recep Kayali, Asa MacWilliams, Horst Mauersberg, Michael Nagel and Martin Wagner for productive working sessions in which the implementation of my work took shape. Thanks also go to Thomas Funk and Christian Sipek who contributed to the practical aspects.

I am much obliged to all the reviewers of draft versions: thank you, Bernd Brügge, Daniel Siewiorek, Christian Bachmaier, Hans and Wolfgang Löhr, Michael Guppenberger and Kilian Kreul for the time you invested and for your fair and fruitful feedback.

Andreas Löhr





# Contents

|   |            |
|---|------------|
| <b>Zusammenfassung</b>                                      | <b>i</b>   |
| <b>Abstract</b>   | <b>iii</b> |
| <b>Acknowledgements</b>                                     | <b>v</b>   |
| <b>Contents</b>   | <b>x</b>   |
| <b>List of Tables</b>                                       | <b>xii</b> |
| <b>List of Figures</b>                                      | <b>xiv</b> |
| <b>1 Introduction</b>                                       | <b>1</b>   |
| 1.1 Graphical User Interfaces . . . . .                     | 2          |
| 1.2 Speech-controlled Graphical User Interfaces . . . . .   | 6          |
| 1.2.1 Speech Recognition Basics . . . . .                   | 6          |
| 1.2.2 Speech-based Mouse Emulation . . . . .                | 8          |
| 1.2.3 Command-and-Control . . . . .                         | 11         |
| 1.2.4 Summary . . . . .                                     | 12         |
| 1.3 Performance of Speech-controlled GUIs . . . . .         | 13         |
| 1.3.1 Temporal Components of Task Completion Time . . . . . | 14         |
| 1.3.2 Proposed Interaction Techniques . . . . .             | 16         |
| 1.4 Outline . . . . .                                       | 18         |
| <b>2 The Speech-GUI Model</b>                               | <b>19</b>  |
| 2.1 Design Goals . . . . .                                  | 20         |
| 2.2 Control Actions . . . . .                               | 21         |
| 2.3 Speech Functions . . . . .                              | 22         |
| 2.4 Commands . . . . .                                      | 28         |
| 2.5 Speech Recognizers . . . . .                            | 31         |
| 2.5.1 Recognition Delay . . . . .                           | 32         |
| 2.5.2 Word Error Rate . . . . .                             | 33         |
| 2.5.3 Summary . . . . .                                     | 35         |

|          |  |           |
|----------|--|-----------|
| 2.6      | Recognition Result Interpreters . . . . .                                  | 35        |
| 2.7      | Consolidated Speech-GUI Model . . . . .                                    | 38        |
| 2.8      | Discussion . . . . .   | 39        |
| 2.8.1    | Reconsidering the Design Goals . . . . .                                   | 39        |
| 2.8.2    | Limitations . . . . .  | 41        |
| 2.8.3    | Other User Interface Models . . . . .                                      | 42        |
| <b>3</b> | <b>The Interaction Delay Model</b>   | <b>47</b> |
| 3.1      | Requirements . . . . .   | 48        |
| 3.2      | Interaction Delay . . . . .  | 48        |
| 3.2.1    | Definition . . . . .   | 48        |
| 3.2.2    | Nominal Interaction Delay . . . . .  | 50        |
| 3.2.3    | Expected Interaction Delay . . . . .                                       | 51        |
| 3.3      | Characteristics of the Interaction Delay . . . . .                         | 60        |
| 3.3.1    | Critical Word Error Rate . . . . .   | 60        |
| 3.3.2    | Word Error Rate-Monotony . . . . .   | 62        |
| 3.3.3    | Recognition Delay-Monotony . . . . .                                       | 63        |
| 3.3.4    | Command Length-Monotony . . . . .  | 63        |
| 3.3.5    | Command Count-Monotony . . . . .   | 64        |
| 3.3.6    | Summary . . . . .  | 65        |
| 3.4      | Discussion . . . . .   | 66        |
| 3.4.1    | Reconsidering the Requirements . . . . .                                   | 66        |
| 3.4.2    | Interaction Delay and User Interface Metrics . . . . .                     | 67        |
| <b>4</b> | <b>Interaction Delay Calculations</b>                                      | <b>69</b> |
| 4.1      | Preliminaries . . . . .  | 70        |
| 4.1.1    | The Swing Catalog . . . . .  | 70        |
| 4.1.2    | Executions . . . . .   | 77        |
| 4.1.3    | Calculation Procedure . . . . .  | 81        |
| 4.2      | Command-and-Control . . . . .  | 83        |
| 4.2.1    | General Conditions . . . . .   | 83        |
| 4.2.2    | Conventional Command-and-Control . . . . .                                 | 87        |
| 4.2.3    | Command-and-Control with Random Navigation and Direct Activation . . . . . | 88        |
| 4.2.4    | Summary . . . . .  | 89        |
| 4.3      | Speech-based Mouse Emulation . . . . .                                     | 90        |
| 4.3.1    | General Conditions . . . . .   | 90        |
| 4.3.2    | Direction-based Mouse Emulation with Continuous Movement . . . . .         | 92        |
| 4.3.3    | Direction-based Mouse Emulation with Discrete Movement . . . . .           | 94        |
| 4.3.4    | Target-based Mouse Emulation . . . . .                                     | 96        |
| 4.3.5    | Grid-based Mouse Emulation . . . . .                                       | 97        |
| 4.3.6    | Summary . . . . .  | 99        |
| 4.4      | Mouse-controlled GUIs . . . . .  | 101       |

|          |   |            |
|----------|---|------------|
| 4.4.1    | Enhancing the Speech-GUI Model . . . . .                  | 101        |
| 4.4.2    | Interaction Delay . . . . .                               | 103        |
| 4.4.3    | Summary . . . . .   | 105        |
| 4.5      | Discussion . . . . .                                      | 106        |
| <b>5</b> | <b>Conversation-and-Control</b>                           | <b>107</b> |
| 5.1      | Qualified Activation . . . . .                            | 108        |
| 5.2      | Interpretation of Qualified Activation Commands . . . . . | 111        |
| 5.2.1    | Interpretation Model . . . . .                            | 111        |
| 5.2.2    | Sample GUI . . . . .                                      | 114        |
| 5.2.3    | Isolated-Topic-Distribution Heuristic . . . . .           | 115        |
| 5.2.4    | Peering-Topic-Distribution Heuristic . . . . .            | 116        |
| 5.2.5    | Summary . . . . .   | 117        |
| 5.3      | Automatic Information Error Recovery . . . . .            | 118        |
| 5.3.1    | Relative-Maximum-Identification Heuristic . . . . .       | 118        |
| 5.3.2    | Resolution-by-Historical-Topic . . . . .                  | 119        |
| 5.3.3    | Summary . . . . .   | 122        |
| 5.4      | Qualified Feedback . . . . .                              | 122        |
| 5.4.1    | Resolution-by-Clarification-Question . . . . .            | 123        |
| 5.4.2    | Topic-State-Backtracking . . . . .                        | 125        |
| 5.4.3    | Clarify-Parameter . . . . .                               | 128        |
| 5.4.4    | Summary . . . . .   | 130        |
| 5.5      | Dialog Model for Conversation and Control . . . . .       | 130        |
| 5.5.1    | Dialog Model Categories . . . . .                         | 131        |
| 5.5.2    | Dialog State . . . . .                                    | 137        |
| 5.5.3    | Core Algorithm . . . . .                                  | 139        |
| 5.5.4    | Summary . . . . .   | 144        |
| 5.6      | Discussion . . . . .                                      | 145        |
| 5.6.1    | Characteristics of the Dialog Model . . . . .             | 145        |
| 5.6.2    | Limitations . . . . .                                     | 147        |
| 5.6.3    | The Role of Speech Synthesis Technology . . . . .         | 149        |
| 5.6.4    | Related Work . . . . .                                    | 150        |
| <b>6</b> | <b>Validation</b>   | <b>155</b> |
| 6.1      | Conversation-and-Control Framework . . . . .              | 156        |
| 6.1.1    | Requirements . . . . .                                    | 156        |
| 6.1.2    | Design . . . . .  | 156        |
| 6.1.3    | Implementation . . . . .                                  | 159        |
| 6.1.4    | Self-Elimination Heuristic . . . . .                      | 161        |
| 6.1.5    | Prototype Systems . . . . .                               | 162        |
| 6.2      | Interaction Delay of Conversation-and-Control . . . . .   | 165        |
| 6.2.1    | Calculations . . . . .                                    | 165        |

|          |  |            |
|----------|--|------------|
| 6.2.2    | Comparison to Conventional Command-and-Control . . . . . | 169        |
| 6.3      | User Experiment . . . . .                                | 172        |
| 6.3.1    | Research Hypothesis . . . . .                            | 173        |
| 6.3.2    | Test Subjects . . . . .                                  | 173        |
| 6.3.3    | Experiment GUI . . . . .                                 | 174        |
| 6.3.4    | Procedures . . . . .                                     | 176        |
| 6.3.5    | Results . . . . .  | 177        |
| 6.3.6    | Discussion . . . . .                                     | 178        |
| 6.4      | Summary . . . . .  | 182        |
| <b>7</b> | <b>Conclusion</b>  | <b>183</b> |
| 7.1      | Thesis Summary . . . . .                                 | 184        |
| 7.2      | Contributions . . . . .                                  | 185        |
| 7.3      | Future Work . . . . .                                    | 186        |
| <b>A</b> | <b>Appendix: Mathematics</b>                             | <b>189</b> |
| A.1      | Theoretical Limit of $P(Rej)$ . . . . .                  | 189        |
| A.2      | Calculating the Critical Word Error Rate . . . . .       | 190        |
| A.3      | Geometric Series starting at 1 . . . . .                 | 191        |
| <b>B</b> | <b>Appendix: Calculations and Studies</b>                | <b>193</b> |
| B.1      | Time to Interpret a Recognition Result . . . . .         | 194        |
| B.2      | Data for Interaction Delay Calculations . . . . .        | 195        |
| B.3      | Tasks from the User Study . . . . .                      | 195        |
|          | <b>Bibliography</b>                                      | <b>227</b> |

# List of Tables

|      |  |     |
|------|--|-----|
| 1.1  | Summary of speech-controlled GUI technology . . . . .  | 13  |
| 2.1  | Exemplary valid commands and invalid commands . . . . .  | 29  |
| 2.2  | Summary of user interface models . . . . .   | 46  |
| 4.1  | The Swing catalog. . . . .   | 71  |
| 4.2  | Basic executions of graphical objects in the Swing catalog. . . . .  | 80  |
| 4.3  | Common commands with command-and-control. . . . .  | 86  |
| 4.4  | Indirectly triggered activations with conventional command-and-control. . . . .  | 87  |
| 4.5  | Calculated interaction delay of conventional command-and-control . . . . .   | 88  |
| 4.6  | Calculated interaction delay of command-and-control with random navigation and direct activation . . . . .                       | 88  |
| 4.7  | Calculated interaction delay of direction-based mouse emulation with continuous movement . . . . .                               | 94  |
| 4.8  | Calculated interaction delay of direction-based mouse emulation with discrete movement . . . . .                                 | 96  |
| 4.9  | Calculated interaction delay of target-based mouse emulation . . . . .   | 97  |
| 4.10 | Calculated interaction delay of grid-based mouse emulation . . . . .   | 99  |
| 6.1  | Qualified activation commands . . . . .  | 167 |
| 6.2  | Calculated interaction delay of conversation-and-control . . . . .   | 169 |
| 6.3  | Average task completion times. . . . .   | 177 |
| 6.4  | Predicted interaction delay for completing experiment. . . . .   | 180 |
| B.1  | Intentions and interactions for basic executions using conventional command-and-control. . . . .                                 | 196 |
| B.2  | Interaction delay calculation details of conventional command-and-control. . . . .   | 197 |
| B.3  | Intentions and interactions for basic executions using command-and-control with random navigation and direct activation. . . . . | 198 |
| B.4  | Interaction delay calculation details of command-and-control with random navigation and direct activation. . . . .               | 199 |
| B.5  | Intentions and interactions for basic executions using direction-based mouse emulation with continuous movement. . . . .         | 200 |

## List of Tables

---

|      |  |     |
|------|--|-----|
| B.6  | Interaction delay calculation details of direction-based mouse emulation with continuous movement. . . . .             | 201 |
| B.7  | Intentions and interactions for basic executions using direction-based mouse emulation with discrete movement. . . . . | 202 |
| B.8  | Interaction delay calculation details of direction-based mouse emulation with discrete movement. . . . .               | 203 |
| B.9  | Intentions and interactions for basic executions using target-based mouse emulation. . . . .                           | 204 |
| B.10 | Interaction delay calculation details of target-based mouse emulation. . . . .   | 205 |
| B.11 | Intentions and interactions for basic executions using grid-based mouse emulation. . . . .                             | 206 |
| B.12 | Values for <i>ishift</i> and <i>iselect</i> for specific word error rates. . . . .                                     | 206 |
| B.13 | Interaction delay calculation details of grid-based mouse emulation. . . . .   | 207 |
| B.14 | Intentions for basic executions using mouse-controlled GUIs. . . . .   | 208 |
| B.15 | Interaction delay calculation details of conversation-and-control. . . . .   | 209 |
| B.16 | Tasks to be completed in the experiment. . . . .   | 210 |

# List of Figures

|      |  |    |
|------|--|----|
| 1.1  | Ivan Sutherland operating the Sketchpad system. . . . .                          | 2  |
| 1.2  | Historical devices for GUI control. . . . .                                      | 3  |
| 1.3  | Taxonomy of existing approaches for speech-controlled GUIs. . . . .              | 8  |
| 1.4  | Constituents of task completion time. . . . .                                    | 15 |
| 2.1  | Exemplary GUI <i>G</i> . . . . .   | 22 |
| 2.2  | Relationships of control actions, navigations, activations and sessions. . . . . | 22 |
| 2.3  | Control actions of simple GUI <i>G</i> . . . . .                                 | 23 |
| 2.4  | Speech functions and their relationship to control actions. . . . .              | 24 |
| 2.5  | Session <i>SG</i> . . . . .  | 24 |
| 2.6  | Initial situation for example 1. . . . .   | 25 |
| 2.7  | Speech functions of example 1. . . . .   | 25 |
| 2.8  | Intention <i>IG1</i> . . . . .   | 25 |
| 2.9  | Speech functions of example 2. . . . .   | 26 |
| 2.10 | Initial situation for example 2. . . . .   | 26 |
| 2.11 | Intention <i>IG2</i> . . . . .   | 27 |
| 2.12 | The initial situation example 3. . . . .   | 27 |
| 2.13 | Speech functions of example 3. . . . .   | 28 |
| 2.14 | Intention <i>IG3</i> . . . . .   | 28 |
| 2.15 | Commands and their relationship to speech functions. . . . .                     | 30 |
| 2.16 | Examples for word errors. . . . .  | 33 |
| 2.17 | Silent consonants. . . . .   | 35 |
| 2.18 | Speech recognizer. . . . .   | 36 |
| 2.19 | Recognition result interpreter. . . . .  | 37 |
| 2.20 | Dynamic speech-GUI model. . . . .  | 38 |
| 2.21 | Static speech-GUI model. . . . .   | 40 |
| 3.1  | 3D plots of critical word error rate. . . . .                                    | 61 |
| 4.1  | A checkbox in the checked and unchecked state. . . . .                           | 72 |
| 4.2  | Logical constituents of a drop down box . . . . .                                | 73 |
| 4.3  | Logical constituents of a menu . . . . .   | 74 |
| 4.4  | Logical constituents of a tree . . . . .   | 74 |

## List of Figures

---

|      |  |     |
|------|--|-----|
| 4.5  | Logical constituents of a spinner . . . . .  | 75  |
| 4.6  | Different types of sliders. . . . .  | 76  |
| 4.7  | Graphical objects for the calculations . . . . .                                       | 76  |
| 4.8  | Conversa Voice Surfer . . . . .  | 84  |
| 4.9  | Calculated interaction delay graphs of command-and-control approaches. . . . .         | 89  |
| 4.10 | Mouse signals and mouse functions. . . . .   | 91  |
| 4.11 | The compass mouse. . . . .   | 94  |
| 4.12 | Example for grid-based mouse-emulation. . . . .  | 97  |
| 4.13 | Calculated interaction delay graphs of speech-based mouse emulation. . . . .           | 99  |
| 4.14 | Speech-GUI model enhancements for mouse-controlled GUIs. . . . .                       | 102 |
|      |  |     |
| 5.1  | Enhancements of the speech-GUI model induced by qualified activation. . . . .          | 109 |
| 5.2  | Qualified activation commands and qualified speech functions. . . . .                  | 112 |
| 5.3  | Qualifier state transitions. . . . .   | 113 |
| 5.4  | Conversation topic. . . . .  | 114 |
| 5.5  | Sample GUI for illustrating conversation topic distributions. . . . .                  | 115 |
| 5.6  | Dialog state for conversation-and-control. . . . .                                     | 138 |
| 5.7  | Conversation-and-control dialog model, main processing loop. . . . .                   | 140 |
| 5.8  | Conversation-and-control dialog model, clarification mode. . . . .                     | 141 |
| 5.9  | Conversation-and-control dialog model, neutral mode. . . . .                           | 142 |
|      |  |     |
| 6.1  | Bridge pattern. . . . .  | 157 |
| 6.2  | Conversation-and-control framework design. . . . .                                     | 158 |
| 6.3  | Conversation-and-control framework implementation. . . . .                             | 160 |
| 6.4  | Conversation-and-control framework in J2EE. . . . .                                    | 161 |
| 6.5  | Screenshots of Mentor conversation-and-control interface and web interface. . . . .    | 164 |
| 6.6  | Screenshots of Publisher web interface and conversation-and-control interface. . . . . | 166 |
| 6.7  | Graphs of conventional command-and-control and conversation-and-control. . . . .       | 170 |
| 6.8  | Graphs without considering the tree for conversation-and-control. . . . .              | 171 |
| 6.9  | Conversation-and-control interface from the experiment. . . . .                        | 174 |
| 6.10 | Average task completion times, graphically. . . . .                                    | 178 |
|      |  |     |
| A.1  | Maple commands to calculate the theoretical limit of $P(Rej)$ . . . . .                | 190 |
| A.2  | Maple commands to calculate the critical command recognition rate. . . . .             | 190 |
| A.3  | Maple commands to calculate a formula for the critical word error rate. . . . .        | 191 |
|      |  |     |
| B.1  | Output of recognition result interpretation test. . . . .                              | 195 |
| B.2  | Grammar for positive integer numbers between 1 and 999. . . . .                        | 195 |



# 1

## Introduction

*"A display connected to a digital computer gives us a chance to gain familiarity with concepts not realizable in the physical world. It is a looking glass into a mathematical wonderland."*

Ivan Sutherland

### Overview

This dissertation aims at improving the utilization of speech as input modality for graphical user interfaces (GUIs). In this chapter we introduce the historical origins of GUIs, which have become the standard interface type for computing systems of today. GUIs are controlled by mouse and keyboard, which is problematic for handicapped individuals and users in application domains involving manual tasks. A solution to this problem is the utilization of speech to control GUIs, because speech does not involve the usage of the hands. *Speech-controlled GUIs* are GUIs which can be controlled by speech and different approaches for speech-controlled GUIs exist. However, they do not yet reach the performance of GUIs which are controlled by mouse and keyboard. We introduce and discuss existing approaches and propose three extensions which we call *qualified activation*, *automatic information error recovery* and *qualified feedback*. Qualified activation and automatic information error recovery reduce the average amount of spoken commands while qualified feedback aims at reducing the average length of spoken commands.



**Figure 1.1:** Ivan Sutherland operating the Sketchpad system.

## 1.1 Graphical User Interfaces

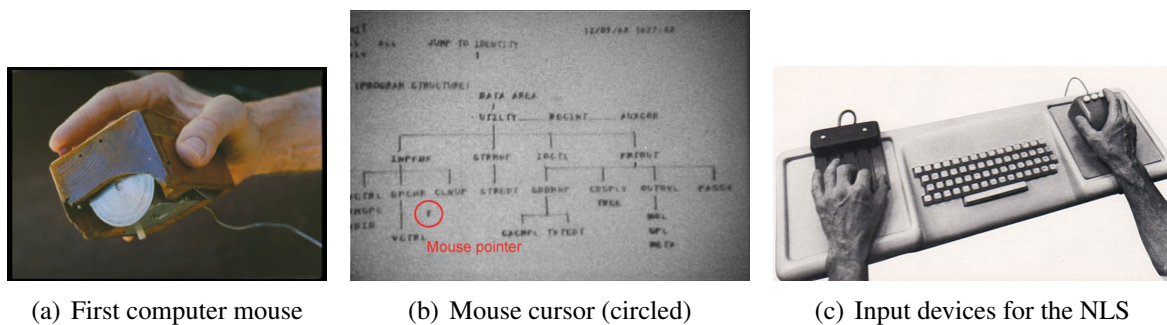
The history of graphical user interfaces (GUIs) reaches back to 1963 when Ivan Sutherland invented the Sketchpad system [182]. Users could draw graphical primitives, such as straight lines or circle arcs, onto a cathode ray tube (CRT) screen using a *light pen*<sup>1</sup>. Users could change the size and position of a specific graphical primitive by using the light pen to point at it and by pushing or turning specific buttons. With the Sketchpad system Sutherland established the basic principles for interacting with a computer using graphics:

1. The user sees a graphical object on the screen.
2. The user points at the graphical object using a pointing device, i.e., the user *focuses* the graphical object. We call changing the currently focused graphical object *navigation*.
3. The user triggers a specific change of the currently focused graphical object, e.g., movement, resizing, morphing, etc., using some other control mechanism which depends on the available input modalities, such as the push buttons of the Sketchpad system. We call the triggering of changes of graphical objects *activation*.

The light pen had a significant drawback: the user was required to hold an arm in front of the CRT screen for long periods of time, which quickly caused the arm to be sore and tired. This problem was solved with the invention of the *computer mouse* by Douglas Engelbart in 1964. The computer mouse was a box-like device which had two gear wheels perpendicular to each other on its bottom (Figure 1.2(a)). Instead of having to be held in front of the screen the mouse was placed onto a solid surface, such as a desk. The wheels contacted the surface

---

<sup>1</sup>A light pen is a pen-like device which must be held in front of a CRT screen. On its tip it has an optical sensor which is able to detect the changes in brightness of a specific pixel when that pixel is refreshed by the electron gun of the CRT screen. Whenever the light pen detects such a change it causes a hardware interrupt in the computing system that it is connected to. The X-Y position of the light pen on the CRT screen can be determined from the X-Y position of the electron beam at the time the interrupt from the light pen occurs.



**Figure 1.2:** Historical devices for GUI control.

and if the mouse was moved the wheels rotated. The rotation of each wheel was transformed into electrical signals which represented movement along one axis in the plane (X axis, Y axis, respectively).

Engelbart showed the feasibility of the mouse as pointing device on the basis of the **oNLine System (NLS)**. This system was demonstrated to the public in 1968 [55]<sup>2</sup> and allowed for the simultaneous creation and manipulation of text and graphics. The NLS introduced a special graphical primitive, called the *mouse cursor* (Figure 1.2(b)). The mouse cursor was permanently shown on the screen and its X-Y position changed with the position of the mouse device, i.e., every movement signal from the mouse caused a proportional movement of the mouse cursor on the screen. With this concept the user was able to point at a specific position on the screen without having to hold a device in front of the screen. Graphics were created and manipulated using the mouse, while text was created and manipulated using a keyboard. Both mouse and keyboard could be placed next to each other on a desk (Figure 1.2(c)).

The NLS is considered to be the first system with a graphical user interface as it coupled the activation of specific pre-existing graphical objects with the invocation of application functionality. An example is the manipulation of text files which were represented by specific graphical objects<sup>3</sup>. In the following we refer to the process of navigating and activating graphical objects of a GUI as the *controlling of a GUI*.

Today, mouse and keyboard are the standard input devices for controlling GUIs. However, the fact that they have to be operated manually is problematic for users who are not able to use their hands, even if only temporarily. For instance, for users with motor impairments (e.g. Carpal Tunnel Syndrome [83]) the usage of manually operated input devices is cumbersome or even painful. Another example are so called *hands-busy* application domains in which users are engaged in manually performed tasks and consequently cannot use their hands to control a GUI – at least not without interruption of the task.

<sup>2</sup>Video clips of the original demo session can be downloaded from <http://sloan.stanford.edu/MouseSite/1968Demo.html>

<sup>3</sup>The NLS also demonstrated other concepts which are an integral part of today's GUIs, such as hyperlinks and multiple windows.

Some research approaches try to mitigate this problem by developing computers, accessories, and input modalities which require less manual interaction. For instance, the CMU Dial Device [172] emulates mouse functions by turning a wheel to the left and right and by clicking on it. Intended to be used in the area of wearable computing, it allows to control a GUI with one hand instead of two (compared to simultaneous keyboard and mouse usage). It requires, however, a specific GUI layout and interaction paradigm, so that the cursor can be moved in a grid environment. Stylus-based computers are typically designed for mobile usage. They mitigate the input problem by replacing the two devices mouse and keyboard by one stylus, which is used to point onto a touchscreen. Using a specifically designed drawing scheme, called *Graffiti*, it is possible to emulate keyboard entry. However, *Graffiti* must be learned and trained. Furthermore, stylus-based computers need to be carried, thus, again, both hands are occupied – one hand holds the stylus and the other hand holds the computer. This problem is aimed to be solved by accessories like the E-Belt [148] or the Mid-Riff Brain [149]; these devices are mechanical arms which can be mounted on the hip and a stylus-based computer can be attached to them. As a result, the hand that formerly held the computer is now free. However, the mechanical arm is cantilevered, which only makes it usable for mobile application domains without spatial constraints.

In contrast to the approaches just discussed, this dissertation focuses on research which deals with the development of *hands-free* input modalities – input modalities which do not require the usage of the hands for the generation of input. Thus, no manual operations at all are needed to control a GUI. In the following we introduce three such research areas: *bio signal interfaces*, *head orientation and position tracking* and *speech recognition*.

**Bio Signal Interfaces** These interfaces capture electrical signals generated by the human body and interpret them for controlling computers. *Brain-computer interfaces* are bio signal interfaces which are based on the observation that the pure imagination of a physical action causes distinct measurable changes in the electrical activity of the human brain. This activity can be captured by electrodes attached to the head and interpreted as input for a computing system. Current research topics in the area of brain-computer interfaces are the definition of models for the brain-computer interface design (Mason and Birch [121]) and the identification of suitable application domains (Moore [136]). Functional prototypes of brain-computer interfaces are available, such as a remote control (Xiaorong et al. [63]) and a system that allows to select from a given set of graphical symbols (Serby et al. [166]). Other types of bio signal interfaces interpret electrical signals emerging from muscle contraction. As such, it has been observed that the clenching of the teeth generates unique electrical signals (Yoshiyuki et al. [183]). This can be exploited to emulate basic mouse functions. For instance, Jeong et al. [86] show, by a prototype system, that by combining two different clenching patterns, seven instructions including rest, up, down, left and right, as well as click and double click can be emulated. Bio signal interface prototypes show the feasibility of this technology, however, it is currently slow (Penny and Roberts [147]) and lacks accuracy (Pino et al. [144]) – which results in bad usability. Users have to undergo intensive training sessions in order

to be able to generate the bio signal activity that the system requires. Furthermore, these interfaces require gears or electrodes to be attached to the human body, which is not well suited for mobile and spatially limited environments.

**Head Orientation and Position Tracking** This technology is based on (a stream of) data from which the current orientation and position of the head is inferred. One variant is to process a stream of digital images of the human head. Image processing algorithms then track specific facial features or artificial artifacts. For instance, Chiu and Chu [36] extrapolate into which direction the eyes look in order to generate X-Y coordinates for positioning the mouse cursor on the screen. Eye tracking does, however, not work well in the presence of large changes in head position and orientation. Therefore, Ashdown et al. [9] track the orientation of the entire head and use the obtained X-Y coordinates to move the cursor over large distances, e.g., when multiple monitors are used. Kitajima et al. [98] track facial features to obtain X-Y coordinates for supporting window-specific tasks, such as moving, zooming and scrolling. Experiments show that head tracking based on image data is feasible, but it requires possibly bulky cameras. Furthermore, image processing is complex and the noise in the image data introduces a noticeable delay and inaccuracy in the movement of the cursor. This can, however, to a certain extent be mitigated by applying an (extended) Kalman filter (Welch and Bishop [195]) to the image data stream. Another possibility for head orientation and position tracking emerges from mounting tilt sensors onto a band that is worn around the head (e.g., the *HeadWay* or *Lazee Mouse Pro* product). The head can be tilted to the left, right, forward and backward in order to emulate move instructions. Mouse clicks can be emulated by shortly blowing into a tube equipped with a pressure sensor. Similar devices can be constructed using ultrasound (e.g., *HeadMaster Plus*), gyroscopes (e.g., *Boost Tracker*) or infrared sensors (e.g., *HeadMouse Extreme*), which also capture the tilting movements of the head. Common to essentially all head tracking solutions and technologies is that they are physically demanding. Additionally, infrared sensors do not work well in outside environments due to the sunlight. Ultrasound- and gyroscope-based devices are bulky and require cabling, which limits their suitability for environments other than a desktop environment, such as a mobile environment. Pfrommer [150] provides a further comparison of head tracking technologies.

**Speech Recognition** Speech recognition technology transforms spoken language, assumed to be available as an audio signal, into text. The recent advances in speech recognition technology (McTear [124]) have facilitated the development of *speech-controlled computing systems* – computing systems which use speech as input modality. Speech-controlled computing systems have long emerged from academically used systems, e.g., CMU Communicator (Rudnicky et al. [160]), to productively used applications, e.g., the automatic time table information system of Deutsche Bahn<sup>4</sup>. Today, it is possible to create speech-controlled computing systems which engage users into a natural dialog,

---

<sup>4</sup>The system is reachable by phone under +49 800 1507090.

where the system accepts natural language as input and gives feedback with naturally formulated responses, e.g., *SENECA* [135] and *How May I Help You* [69]. Furthermore, speech for input purposes can be standardized: Rosenfeld et al. [158] have transitioned the Graffiti interaction paradigm, known from stylus-based touchscreen computers, to the speech-controlled computing systems domain. Thus, they call it *Speech Graffiti*, also known as the *Universal Speech Interface*. A small set of standardized natural language command templates, populated with a domain-specific vocabulary, is sufficient for controlling arbitrary interactions and appliances (Harris and Rosenfeld [73]). The feasibility of this approach has been shown by the means of prototype systems, including *MovieLine* [159] and *ApartmentLine* [171]. Experiments further showed, that users adapt well to the artificial language imposed by Speech Graffiti (Tomko and Rosenfeld [186]).

The research area of *speech-controlled GUIs*, which we discuss in the following section, is a subdivision of speech-controlled computing system research. It deals with the usage of speech recognition technology as hands-free input modality for GUIs.

## 1.2 Speech-controlled Graphical User Interfaces

We now give an overview of the basics of speech recognition technology (section 1.2.1) and introduce two schools of thought for using this technology to control a GUI.

The first school of thought, described in section 1.2.2, is called *speech-based mouse emulation* and aims at emulating the functionality of mouse and keyboard by speech. In this work we do not consider the emulation of keyboards by speech, because a keyboard can be represented by a special GUI which becomes part of the original GUI. This special GUI consists of several graphical objects, each of them represents a specific key on the physical keyboard and emulates a key stroke from the physical keyboard. This facilitates alphanumerical input with the mouse and examples for available systems include built-in tools for Windows XP TabletPC Edition.

The second school of thought, described in section 1.2.3, is called *command-and-control* following the principle of interpreting speech directly as input for navigation and activation of graphical objects, i.e., without manipulating the mouse cursor.

### 1.2.1 Speech Recognition Basics

A *spoken command* is the vocalization of a sequence of words. Speech recognition technology transforms spoken commands, assumed to be available as audio signals, into the words which have originally been vocalized. We call software components which implement speech recognition technology *speech recognizers*. They are based on three data structures: the *phoneme database*, the *vocabulary* and the *language model*. The phoneme database represents the basic sound elements of a spoken languages, i.e., the phonemes, as audio signals. The vocabulary contains all words which are known to the speech recognizer and represents them as sequences

of phonemes.<sup>5</sup> It may contain several phoneme sequences for a specific word as there might be several ways to pronounce the word. An example is the English article "a" which is pronounced differently depending on the context in a sentence, additionally there are differences in British and American English. The language model represents the structure of a language. A popular language model in the context of speech recognition is the *N-gram model* (Brown et al. [29]) – a statistical model which, given N-1 previous words, provides the probability that the next word will be *X* (*X* represents a specific word of the recognizer's vocabulary). Another language model for speech recognition are *probabilistic context free grammars* (Kita [97], Matsuzaki et al. [122]) – context free grammars which are enhanced by probabilistic features.

The speech recognizer takes the audio signal and divides it into segments according to a mathematical analysis process. The details of this process are beyond the scope of this dissertation<sup>6</sup>. By matching the segments against the phoneme database, the audio signal can be converted into the sequence of phonemes, which most likely represents the vocalization of the spoken command. The speech recognizer then splits the phoneme sequence into subsequences and looks up each subsequence in the vocabulary in order to determine the words which have been spoken. As there are multiple possibilities to split the phoneme sequence the speech recognizer uses the language model to determine which resulting word sequences are likely and which ones can be rejected. The word sequence with the highest probability of occurrence in the language model is returned as the result of the speech recognition process, the so called *recognition result*.

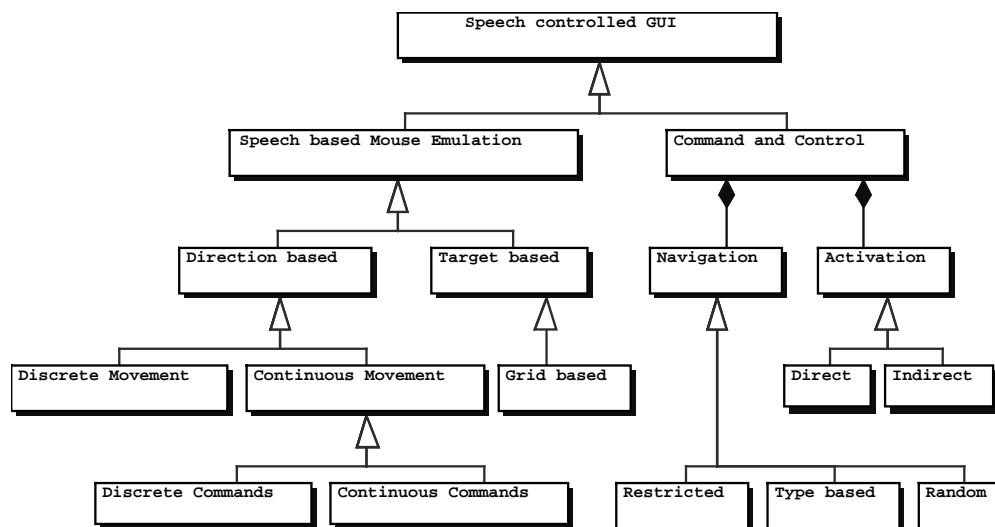
The time which passes between the user finishing the spoken command and the speech recognizer calculating the recognition result is called the *recognition delay*. The more words are contained in the vocabulary the more computationally intensive is the speech recognition process and consequently the higher is the recognition delay. In order to minimize the recognition delay mathematical operations and models, such as *Hidden Markov Models* (Huang et al. [79]), are used to reduce the complexity of the internal matching and look up processes. Additionally, the speech recognizer makes a trade-off between processing time and finding the optimal recognition result: if the optimal recognition result could not be found within a specific amount of time, the best result which has been found up to that time is returned as the recognition result. As such, it is likely that a recognition result is not equal to the original spoken command. We call such a recognition result a *recognition error*.

With increasing size of the vocabulary the probability for recognition errors increases, as the computational intensity increases. The speech recognizer offered by SPIRIT DSP [177] claims to achieve a word error rate  $\leq 1\%$  for a vocabulary of up to 100 words. Deng et al. [49] achieve a word error rate of  $< 5\%$  with a vocabulary of 5000 words. In Nouza et al. [140] word error rates of up to 30% with a vocabulary of up to 200,000 words are reported. Experiments show that the probability of recognition errors decreases with the time that users

---

<sup>5</sup>In literature the term *lexicon* is frequently used in place of vocabulary.

<sup>6</sup>This analysis process represents audio signals as *cepstral coefficients*. The mathematical foundation of cepstral coefficients has been established in 1963 (Tukey et al. [189]).



**Figure 1.3:** Taxonomy of existing approaches for speech-controlled GUIs.

work with a system. Apparently, users learn how they need to articulate commands, so that the speech recognizer recognizes them better (Falavigna et al. [56]). While users of speech recognition-based systems tend to regard the occurrence of recognition errors as immature technology, some researchers believe that recognition errors are inevitable (e.g. Danis and Karat [45]). With state-of-the-art technology it is possible to achieve a recognition error rate below 5% (Huang et al. [78], Mohan et al. [153]).

The introduction into speech recognition technology given in this section is sufficient for the scope of this dissertation. For further studies, a comprehensive introduction into speech recognition technology can be found in Huang et al. [80] and Kotelly [103].

In the following we will subsequently discuss different speech-controlled GUI approaches, finally resulting in the taxonomy depicted in Figure 1.3, which depicts this taxonomy graphically using the Unified Modeling Language (UML)<sup>7</sup>. We begin with speech-based mouse emulation in section 1.2.2 and then continue with command-and-control in section 1.2.3. Finally, section 1.2.4 provides a brief summary.

## 1.2.2 Speech-based Mouse Emulation

Speech-based mouse emulation aims at emulating the functions of the mouse device by speech. We identify two approaches for speech-based mouse emulation, the *direction-based* and the *target-based* approach. Both approaches emulate the mouse button using the same technique: the user speaks commands like "press left button" or "release left button" to emulate the pressing and releasing of the physical mouse button. The approaches differ in how they emulate the

<sup>7</sup><http://www.uml.org/>



movement of the mouse, respectively the movement of the mouse cursor.

### Direction-based Approach

With a direction-based approach the user utters the direction along the X or Y axis of the screen into which the mouse cursor should be moved. Two flavors exist and they differ in the characteristics of the movement. We call these flavors *continuous movement* and *discrete movement*.

Continuous movement implies that the mouse cursor continuously moves into the direction spoken by the user. Some variants require discrete commands, i.e., commands consisting of a finite amount of words, to start the continuous cursor movement. The movement stops if either the user utters a specific discrete stop command or a border of the screen is reached. Manaris et al. [118] and Karimullah and Sears [95] use verbal representations for the directions (e.g. "move left" or "move up"), whereas Gori et al. [46] and Bilmes et al. [20] represent directions as vowels (e.g., "a" for left or "u" for up). Olwal and Feiner [142] use arbitrary English words or phrases as commands. Their system analyzes, amongst other parameters, the duration and loudness of these commands and maps them to characteristics of cursor control, such as the direction and the speed of the cursor movement. Other variants of continuous movement use continuous commands to move the cursor, i.e., the cursor moves as long as the user vocalizes. An example is the work of Igarashi and Hughes [81]: the user speaks the direction into which the cursor should be moved, immediately followed by the vocalization of a vocal which lasts until the cursor has reached the desired destination (e.g., "move up aaaaaaaaaaaaaaaaa"). A continuous movement allows for cursor positioning over long distances with minimal user interaction – once the cursor moves no further interaction is required, except if continuous commands are utilized. However, continuous movement lacks precision due to the recognition delay: the cursor movement will not immediately stop after the user speaks the stop command or the user stops vocalizing – it keeps moving as long as it takes to recognize the stop command, respectively, until the speech recognizer detects the silence. As such, the faster the cursor moves the lower the precision and vice versa (Karimullah et al. [94]). A *predictive cursor* is an image of the cursor projected in the line of movement, indicating where the cursor would halt if the stop command was uttered. It does not significantly improve accuracy because users can estimate a halting point with or without the predictive cursor (Sears et al. [163]) – however, this estimate is not accurate enough.

Discrete movement requires that the user, additionally to the direction, utters the distance which the cursor should be moved relative to the cursor's current position. Manaris et al. [118] display a coordinate system on the screen so that the user can express the relative distance by the number of units that the mouse cursor should be moved (e.g. "move three down" or "move four left"). Yoshiyuki et al. [134, 133] see the current position of the mouse cursor as the point of origin of a virtual X-Y coordinate system. Along the axes of this virtual coordinate system they display *ghost cursors*, that is, images of the actual cursor, in specific intervals. The user specifies the distance as the number of ghost cursors that lies between the current and the new position. Discrete movements are more precise than continuous movement because

the user specifies the distance that the mouse cursor should be moved, instead of guessing where the continuous movement would stop. However, the distance is specified in virtual units which range over several pixels. The pixels between the virtual units cannot be addressed and consequently the mouse cursor cannot be positioned there. Brøndsted and Aaskoven [27] propose a solution to this problem: the current position of the mouse cursor represents the intersection point of eight rulers which are oriented like a compass, i.e., one ruler to the north, one to the north-east, etc. – therefore this approach is referred to as the *compass mouse*. Each ruler marks a point every 100 pixels from the center and the user repositions the mouse cursor by uttering commands of the form "move north-east two hundred and ten pixels". Brøndsted and Aaskoven show that with their approach the mouse cursor can be positioned to every pixel on the screen by uttering maximal two commands. User experiments, however, show that even trained users need more than two commands which is due to the fact that users cannot precisely estimate the number of pixels between the center of the compass and the desired location on the screen – which leads to corrections.

### Target-based Approach

A *target* is a specific area on the screen which has been assigned a *speokable identifier*, such as a label. The identifier is displayed on the screen somewhere close to the target. If the user utters an identifier the mouse cursor is positioned to a specific position within the corresponding target; the exact position depends on the target.

Graphical objects of GUIs can be used as targets (e.g., *QPointer VoiceMouse* [117] and *Dragon NaturallySpeaking Preferred*<sup>8</sup>). The identifiers are derived from properties of the graphical objects, such as textual attributes (e.g., the name of a button), and added dynamically to the speech recognizers vocabulary<sup>9</sup>. Using graphical objects as targets is a fairly straight forward approach, however, an exact positioning of the cursor within the target is not possible. This causes problems with modern GUIs which provide graphical objects which behave differently depending on where inside the object the cursor is positioned. For instance a drop down box object allows for entering a new menu item or opens a drop down menu depending on where inside the object's area the mouse cursor is positioned.

Dai et al. [44] propose a solution to this problem. They use a grid to divide the entire screen into a specific number of regions (e.g. 3x3 regions). It is assumed that at least one region overlaps with the graphical object to be targeted, with a specific position within a graphical object, respectively. Each region is associated with a number. The user utters the number of a region that overlaps with the target. This causes the mouse cursor to be placed to a dedicated position within the respective region, such as the region's center. If the mouse cursor is not yet at the desired position within the target the enclosing region can recursively be divided into smaller regions. The smaller regions are again associated with numbers and the cursor can again be positioned to a dedicated position inside them. Additionally to recursing into

---

<sup>8</sup><http://www.nuance.com/naturallyspeaking/preferred/>

<sup>9</sup>Dynamically changing the vocabulary is supported by current technology, e.g., Java Speech API [127] or Microsoft's Speech Development Kit [126]

regions the user can shift the entire grid along the X and Y axis. Dai et al. show that with their grid-based approach every pixel on the screen can be reached within a finite sequence of grid recursions and shifts which is logarithmic in the size of the screen. However, the average number of user interactions is higher than with direction based approaches.

### 1.2.3 Command-and-Control

The key idea of command-and-control is to interpret spoken commands directly for controlling a GUI, i.e., for navigating and activating graphical objects, without manipulating the mouse cursor. In the following we discuss navigation and activation separately.

#### Navigation

We distinguish three types for navigation called *restricted navigation*, *type-based navigation* and *random navigation*.

Restricted navigation applies a specific order to graphical objects, called the *navigation order*, which assigns each graphical object a predecessor and a successor. Only the predecessor or the successor of the currently focused graphical object can be navigated to. An example is a speech tool that comes with *Windows XP TabletPC Edition*. It allows for navigation of graphical objects by speaking "next" and "previous" using a navigation order that is determined by the operating system (e.g. from the layout of the graphical objects) or by the programmer of the GUI<sup>10</sup>.

Type-based navigation extends restricted navigation by using the type of graphical objects as navigation filter. Graphical objects of which the type does not match the current filter setting are skipped when predecessor and successor of the currently focused object are determined. Type-based navigation is faster than restricted navigation but requires domain knowledge to determine the type of a graphical object. An example for type-based navigation is the work of Olsen et al. [91] who have developed a speech-controlled GUI for a calendar application. The GUI consists of graphical objects for months, weeks and days. Week objects are aggregated from day objects and month objects are aggregated from week objects. The user can navigate month-wise ("next month"), week-wise ("next week") or day-wise ("next day"). Another example is the work of Arnold et al. [7] who have developed a speech-controlled graphical editor for high-level programming languages. The syntactic elements of a program are represented by graphical objects and the user can navigate the elements by their syntactic type. For instance "move down statement" would navigate to the next object that represents a statement, whereas "move up if" would navigate to the first previous object that represent an if clause.

Restricted and type-based navigation are slow because objects between the currently focused object and the desired object have to be navigated to – regardless of whether type-filtering is applied or not.

Random navigation solves this issue by allowing navigation to any graphical object at any

---

<sup>10</sup>This specific order is commonly referred to as the *Tab-order*.

time, provided that the respective graphical object has a speakable identifier (similar to target-based mouse emulation). An example for random navigation is *Conversay Voice Surfer* [43] which is a plug-in for *Microsoft Internet Explorer* that facilitates speech-controlled web browsing. It provides random navigation by generating labels for each interactive object contained in the currently viewed web page (e.g. hyperlink or form element). The user navigates to a specific interactive object by speaking its label. Another example is the work of James et al. [85] who have created a speech-controlled GUI for *SAP Workplace*<sup>11</sup>, where users navigate tree menus by speaking the labels of respective menu items.

### Activation

We distinguish two types of activation called *direct activation* and *indirect activation*. Common to both approaches is the fact that each graphical object defines specific commands for each specific activation. The approaches differ in how the activations are triggered.

With direct activation the triggering of an activation is only possible if the respective graphical object is focused. As such, the user first has to navigate to the desired graphical object and then has to utter the desired activating command(s).

Indirect activation triggers a specific predefined activation of a graphical object at the moment the object is focused. An example is the invocation of a button action at the time the button is navigated to. Thus, the activation occurs with the user speaking a navigation command. For activations which are not coupled with a specific navigation, direct activation applies.

Indirect activation is faster than direct activation because the user, at least for specific activations, does not have to utter commands explicitly. Existing systems, however, tend to mix direct and indirect activation. For instance *Conversay Voice Surfer* [43] opens the drop down menu of a drop down box if the user speaks the corresponding label (indirect activation). In order to enter a value into an input field the user explicitly has to spell the respective characters (direct activation). Another example of a system that mixes direct and indirect activation is a prototype of a speech-controlled GUI for the maintenance domain developed by Sipek [174]. The user is presented a list of tasks which need to be completed. Each task is represented by a numbered list item. The user can view the details of a task by speaking the respective number (navigation) and then uttering "view" which causes the task details to be displayed (direct activation). The task details display contains a "Save" button, which, if the command "save" is spoken, causes changes to the task details to be persisted (indirect activation).

### 1.2.4 Summary

Speech-based mouse emulation and command-and-control are two schools of thought for speech-controlled GUIs. The mouse can be emulated by the direction-based approach which lets the user move the mouse cursor continuously or discrete into a specific direction. The target-based approach lets the user position the mouse cursor to specific targets on the screen,

---

<sup>11</sup>SAP Workplace is a management application for the SAP runtime environment.

### 1.3. PERFORMANCE OF SPEECH-CONTROLLED GUIS

| Approach   | Synopsis   | Pro   | Con   |
|--|--|---|---|
| Dir.-based mouse emulation., continuous movement | Commands control continuous movement of mouse cursor                                       | Simple commands; easy to learn; universally applicable                          | Slow; inaccurate due to recognition delay   |
| Dir.-based mouse emulation., discrete movement   | Absolute placement of mouse cursor using real or virtual screen coordinates                | Precise positioning; faster than continuous movement                            | Preciseness depends on granularity of coordinates; complex commands                         |
| Target-based mouse emulation                     | Placement of mouse cursor into dedicated and named areas (targets)                         | Intuitive, e.g., graphical objects as targets                                   | Positioning within target is arbitrary; approach does not support complex graphical objects |
| Grid-based mouse emulation                       | Commands control grid which defines targets on screen                                      | Preciseness achieved by grid recursion and shifting                             | Higher number of commands necessary compared to direction-based mouse emulation             |
| Restricted navigation                            | All graphical objects are ordered; commands change focus to next graphical object in order | Analogy to keyboard-based navigation via Tab-key                                | Slow  |
| Type-based navigation                            | Similar to restricted navigation, but filters objects in navigation order based on type    | Faster than restricted navigation for heterogeneous and hierarchical structures | Slow for homogeneous structures   |
| Random navigation                                | Commands change focus to any graphical object  | Fast and intuitive  | Increased recognition complexity due to large vocabulary                                    |
| Direct activation                                | Specific commands trigger specific activations of currently focused object                 | Separation between navigation and activation                                    | Slow  |
| Indirect activation                              | Automatically trigger specific activation if graphical object is focused                   | Faster than direct activation   | Increased risk of misunderstandings   |

**Table 1.1:** Summary of speech-controlled GUI technology

e.g., to graphical objects. The grid-based approach is a special target-based approach that uses grid cells as targets. Command-and-control uses spoken commands to emulate navigation and activation of graphical objects. While activation is typically performed by spoken commands which verbally describe the intended activation, navigation can be performed restricted (only specific graphical objects can be navigated), type-based (only graphical objects of a specific type can be navigated) or randomly (any graphical object can be navigated). Table 1.1 summarizes our survey on speech-controlled GUI technology.

## 1.3 Performance of Speech-controlled GUIs

A common metric for measuring the performance of applications is the metric *task completion time* [50]. It measures the total time a user needs to complete specific predefined tasks with the application, starting with the point in time where the user performs the first interaction with the user interface of the application, and ending with the application being in the desired state. The shorter the task completion time the better the quality of the application.

Early studies with speech-controlled applications from the medical domain showed that

speech recognition as input modality increases the task completion time by 400% as compared to a conventional application with graphical menus (Leeming et al. [105]). Recent studies of applications with speech-controlled GUIs showed that the percentage by which speech control increases the task completion time, could be reduced to approximately 50% compared to mouse-controlled GUIs – at least for specific application domains, specific speech-controlled GUI approaches and GUI instances (Dai et al. [44], Van Buskirk and LaLomia [32], Christian et al. [37], Arnold et al. [7]).

In this dissertation we aim at further improving the achievable performance of applications with speech-controlled GUIs, i.e., we aim at reducing the percentage that speech-control adds to the task completion time compared to mouse-controlled GUIs. We focus on command-and-control for two reasons: first, as the discussion in section 1.2 shows, a lot more research has been conducted and a lot more different approaches and systems exist in the area of speech-controlled mouse emulation than in the area of command-and-control. Second, we argue that speech-controlled mouse emulation adds an additional – unnecessary – level of indirection to the interaction: the user controls the mouse cursor, which is the metaphor of the physical position of a non-present mouse device, in order to control graphical objects, which are a graphical metaphor of application functionality. It is a generally accepted hypothesis that speech by the virtue of being natural is a natural input modality for computing systems<sup>12</sup>. As such we argue that command-and-control is the approach which is better suited for controlling a GUI by speech, as it does not involve a metaphor of a physical device. Command-and-control, although the system is not responding verbally, has a flavor of performing a conversation with the GUI.

In the following we refer to a speech-controlled GUI adhering to a command-and-control approach as a *command-and-control interface*. We motivate our proposed improvements of command-and-control interfaces (section 1.3.2) by an examination of the different temporal components of task completion time in the following section. The examination will reveal that only a few specific components of the task completion time depend on the characteristics of the command-and-control interface – and thus provide possibilities for improvements.

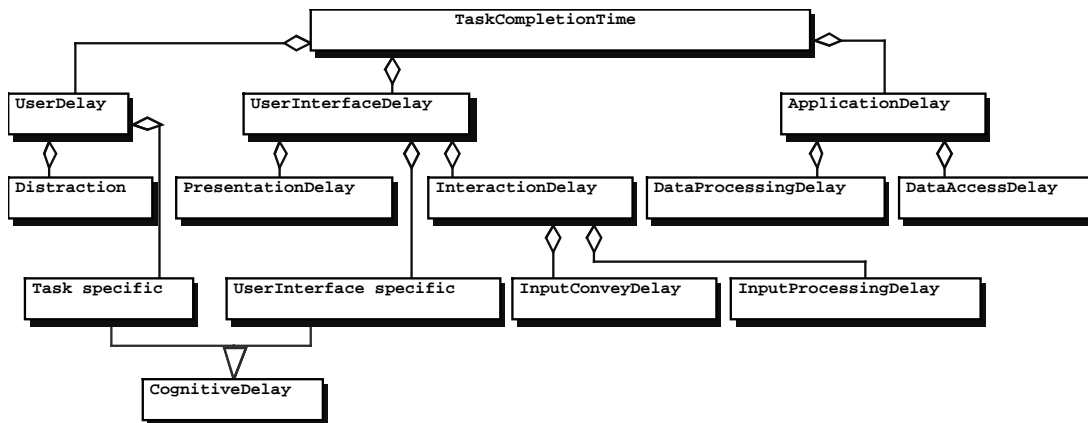
### 1.3.1 Temporal Components of Task Completion Time

The time to complete a task with an application equipped with a command-and-control interface is a cumulation of three main temporal components which we call the *application delay*, the *user delay*, and the *user interface delay*, as depicted in Figure 1.4 on the facing page.

The application delay denotes the time which is consumed by the application processing or accessing data. It influences the task completion time, but it is independent of the inherent performance of the command-and-control interface: for instance, consider an application with a command-and-control interface running on one computer, accessing a database running on another computer. The database also has clients different from our application. If this database

---

<sup>12</sup>There is, however, empirical evidence which contradicts this hypothesis for special tasks/domains, such as non-trivial situations in text editing Karl et al. [96].



**Figure 1.4:** Constituents of task completion time.

is under heavy load then data access is slow, causing our application to be slow – which results in a high task completion time. If the database is idle then data access is fast, causing the application to be fast, which results in a low task completion time. However, the inherent performance of the command-and-control interface remains the same – independently of whether the database (the application layer, respectively) is slow or fast. As such it is not feasible to increase the performance of command-and-control by reducing the application delay.

The user delay denotes the time which the user spends on actions other than interacting with the command-and-control interface. For instance, it includes times where the user is distracted while being in the middle of performing a task (e.g., due to a phone call). Another example is task-specific cognitive load, such as reviewing and evaluating data presented by the command-and-control interface. The user delay influences the task completion time, but it is, like the application delay, independent of the performance of a command-and-control interface. For instance, the user reviewing data for a long time results in a high task completion time, whereas the user reviewing data quickly results in a low task completion time. However, the inherent performance of the command-and-control interface remains the same. As such it is not feasible to increase the performance of command-and-control interfaces by reducing the user delay.

The user interface delay represents the time spent on the plain interaction with the command-and-control interface. It consists of the *presentation delay*, the *interaction delay*, and the *user interface-specific cognitive load*.

The presentation delay denotes the time needed to present the user interface itself (data, respectively). Regarding a command-and-control interface it denotes the time needed for rendering graphical objects. We assume that a command-and-control interface is based on today's available GUI toolkits (e.g. Java Swing [132, 200] or MFC [151]) and argue that the presentation delay induced by command-and-control interfaces has reached an optimal level. As such it is not feasible to increase the performance of command-and-control interfaces by reducing their presentation delay.

The user interface-specific cognitive load denotes the time that the user spends on thinking about how to interact with a specific user interface. We argue that continuous training can reduce this cognitive load to a specific minimum: if a user repeatedly performs specific tasks with a specific user interface then the task completion time for these tasks will decrease down to a certain (at least user-specific) level. While researches generally observe that the cognitive load of speech-controlled user interfaces is higher than the cognitive load of mouse-controlled user interfaces (e.g., Christian et al. [37]) we conclude that the user of a command-and-control interface at some point in time knows "instinctively" which commands to speak. This has been observed for other speech-controlled user interfaces as well, e.g., ShortTalk by Klarlund [99]. This requires, however, that there is a fixed set of available commands that can be trained (e.g., restricted or type based navigation). Alternatively, there must be a simple and easy to learn schema for dynamically changing commands (e.g., random navigation) which users must be trained to use. We will show in the remainder of this work that all discussed speech-controlled GUI approaches, especially command-and-control, use a fixed set of valid commands or an easy to learn schema. While we generally consider it a good idea to aim at reducing the cognitive load induced by command-and-control we argue that it is not a feasible approach, since cognitive load cannot be measured directly: only symptoms which indicate cognitive load can be measured (DiDomenico [51])<sup>13</sup>.

The remaining temporal component of task completion time is the interaction delay. It denotes the time that the user spends on the pure interaction with the user interface. It can generally be decomposed into the *input convey delay* and the *input processing delay*. The input convey delay denotes the time needed to convey input using a specific input modality, whereas the input processing delay denotes the time to process the input and to determine further actions. Applied to command-and-control the interaction delay denotes the time that the user spends on uttering commands, the time that is needed to recognize the commands and the time for the interpretation of commands. In the following section we propose how the interaction delay of command-and-control can be reduced.

### 1.3.2 Proposed Interaction Techniques

In this dissertation we show that the interaction delay of a specific speech-controlled GUI approach, especially of command-and-control, can be determined from quantitative and qualitative characteristics of a model for speech-controlled GUIs, which we specify in chapter 2. In order to further improve the achievable task completion time using a command-and-control interface, we propose to reduce its interaction delay using the new techniques *qualified activation*, *automatic information error recovery* and *qualified feedback*. We explain these techniques in the following.

Qualified activation defines that spoken commands only trigger activations of graphical objects and that the respective command must be qualified with information necessary to iden-

---

<sup>13</sup>The empirical validation of our proposed improvements suggest both a low increase in cognitive load and the ability to compensate this cognitive load by further training.



tify the activation – explicit navigations are not performed. Necessary information includes the speakable identifier of the graphical object and an identifier for the type of the activation. If a command does not include all required information, we consider it an *information error* and will provide a collection of automatic mechanisms which aim at resolving the information error. For example, if the user wants to select the menu item "Munich" from the menu "Cities" the command "select Munich from Cities" would achieve this, provided that the user knows which items are available. If the user changes his mind and wants to change the previously selected item (e.g., from "Munich" to "Pittsburgh") he could utter the command "Pittsburgh". This is an information error in the first place, as the speakable identifier is missing. But considering that the user just has activated the "Cities" menu, the information error can automatically be resolved by selecting the "Pittsburgh" option from the "Cities" menu. Assuming that the "Cities" menu would be the only menu on the screen that has a menu item "Munich", its initial selection could also be achieved by just the command "Munich": this is also an information error in the first place, but the system can automatically infer that the "Cities" menu must be meant, since there is only one graphical object with the menu item "Munich". We claim that qualified activation and automatic information error recovery reduces the average amount of spoken commands which are necessary to interact with a GUI by speech, because a single command contains more information, respectively, is evaluated under consideration of the current context, such as the command history or the GUI state. Thus, qualified activation and automatic information error recovery reduce the interaction delay.

There are, however, information errors which cannot be resolved automatically. In these cases we propose that the speech-controlled GUI ought to give *qualified feedback* as to why the information error cannot be resolved. For instance, if the user says "select Munich from Cities" but the speech recognizer recognized "select from Cities" the following feedback would be generated: "Which item in menu Cities do you mean?". The user would respond with "Munich" and under consideration of the command history the "Munich" item of the "Cities" menu could be determined. In order to realize qualified feedback we represent activations as sets of name-value pairs. The semantic information of a command is used to populate the name-value pairs according to a specific algorithm (which we present later in this work). Depending on which name-value pairs are populated, the activation is triggered or a feedback question regarding the missing property(ies) is generated. This mechanism can also be used to detect and handle ambiguous situations. For instance, assume that there is a menu "Origin" and a menu "Destination" and both menus would contain a menu item "Munich". The command "Munich" would fill a specific name-value pair on both the activation for "Origin" and "Destination", leading to an ambiguity: should "Munich" be selected from the "Origin" or the "Destination" menu? To resolve the ambiguity the feedback question "Do you mean Origin or Destination?" could be generated. The user would for instance respond with "Origin" and under consideration of the command history the "Munich" item on the "Origin" menu would be selected.

We claim that qualified feedback reduces the average length of commands which are needed to interact with a speech-controlled GUI. Currently, recognition results are simply rejected if a recognition error occurs. The user has to repeat the very same spoken command. Qualified

feedback induces commands which are shorter than the command that was mis-recognized – thus, it reduces the interaction delay.

As discussed in section 1.2.3 command-and-control with random navigation and indirect activation is currently the fastest command-and-control approach, to which we refer in the following as *conventional command-and-control*. By extending conventional command-and-control with qualified activation, automatic information error recovery and qualified feedback we arrive at a new approach for speech-controlled GUIs which we call *conversation-and-control*. Using a formal approach we show that conversation-and-control has a lower average interaction delay than conventional command-and-control. In order to show the feasibility of conversation-and-control we have developed a framework, called the *conversation-and-control framework*. Based on this framework we designed, implemented and conducted an experiment to validate conversation-and-control empirically. As predicted by our formal approach, the average task completion time of conversation-and-control is lower than the average task completion time of conventional command-and-control. We also show that the framework is feasible to create conversation-and-control interfaces for applications for the *mobile maintenance* domain.

### 1.4 Outline

In chapter 2 we develop a model for speech-controlled GUIs which serves as the basis for a formal model for the interaction delay in chapter 3. Chapter 4 describes interaction delay calculations of several speech-controlled GUI approaches, including conventional command-and-control. The results motivate the theoretical ideas for conversation-and-control, which we present in chapter 5. Chapter 6 evaluates conversation-and-control regarding its implementability and its interaction delay. The chapter further describes a user experiment a conversation-and-control interface and reports about its results. Chapter 7 gives a summary of this work and provides an outlook onto future work which we regard as necessary in order to transition the concepts presented in this dissertation into productive applications.

# 2

## The Speech-GUI Model

*"All models are wrong. Some models are useful."*

George Box [25]

### Overview

This chapter presents the *speech-GUI model* – a model which abstracts the core components of speech-controlled GUIs: *control actions*, *speech functions*, *commands*, *speech recognizers* and *recognition result interpreters*. Control actions generalize activation and navigation allowing the specification of GUI-specific tasks independently of utilized input modalities. Speech functions abstract the functionality of a specific speech-controlled GUI approach that is used to perform control actions. Speech functions abstract the functionality which is provided by the input modality speech. Speech functions are invoked by the means of uttering commands, which are recognized by a speech recognizer. Speech recognizers model the speech recognition process as an abstract function which transforms commands into recognition results. Recognition result interpreters encapsulate the semantic analysis of speech recognition results. They identify the speech function which the given recognition result – and as such the initially uttered command – addresses.

The speech-GUI model is useful in two regards: first, it is used as the foundation for deriving a model for the interaction delay for speech-controlled GUIs (chapter 3) based on which we perform interaction delay calculations (chapter 4). Second, it is the conceptual basis for conversation-and-control (chapter 5) and the conversation-and-control framework.

## 2.1 Design Goals

The speech-GUI model is based on the following design goals.

**Encapsulation of Speech Recognition Process** Speech-controlled GUIs are based on speech recognition, however, the inherent characteristics of this technology are out of the scope of this dissertation. The speech-GUI model must therefore provide an abstraction which encapsulates the details of the speech recognition process. The abstraction must, however, specify and expose characteristics which affect the performance of speech-controlled GUIs, such as the recognition delay and the probability by which recognition errors occur. The speech-GUI model must furthermore be applicable to the characteristics of currently available speech recognition development kits, such as Microsoft's Speech Development Kit [126], and speech recognition APIs, such as Java Speech API [127, 131].

**Duration of Input** Speech is a temporal input modality denoting that the conveying of input to a computing system via speech takes a significant amount of time. This time corresponds to the input convey delay as defined in section 1.3.1. Regarding different speech-controlled GUI approaches the input convey delay may vary depending on which and how many commands need to be uttered in order to achieve a specific goal. As such, the input convey delay is a specific characteristic of each speech-controlled GUI approach and must be reflected by the speech-GUI model.

**Coverage** One purpose of the speech-GUI model is to provide the basis for the derivation of a model for the interaction delay of speech-controlled GUIs. Therefore, it must be possible to describe existing approaches with the speech-GUI model, i.e., the speech-GUI model must at least cover the approaches which have been introduced in this work (see section 1.2).

**Implementability** Another purpose of the speech-GUI model is to serve as the foundation for the design of the conversation-and-control framework. As such, the characteristics and properties of the speech-GUI model must be checked for practical implementability.

**Extensibility** The speech-GUI model must propose a general concept for the design of speech-controlled GUI approaches. Therefore it must be possible to extend the model at a later point of time to such new approaches, unless they can already be covered with the existing model stage.

In the following sections we will introduce the core components of speech-controlled GUI approaches. For each core component we will provide a detailed textual explanation as well as a specification in the form of a UML class diagram. We will furthermore relate each core component, where applicable, to the *Goals, Operators, Methods, and Selection rules* (GOMS) model, which has (in its initial form) been introduced in 1980 (Card et al. [34]). The GOMS

model, which we explain in the following, has become accepted as a means for analyzing routine human-computer interactions (John [89]), and has been extended to a family of variants (John and Kieras [88]) since its initial introduction. The first version of GOMS is referred to as the *Keystroke-Level Model* (KLM-GOMS). It allows making predictions regarding the expected task completion time with user interfaces involving the keyboard (and the mouse). KLM-GOMS describes the task that a user intends to perform as a *goal* and distinguishes between *high-level goals* (e.g., *WRITE-THESIS*) and *low-level goals* (e.g., *TYPE-WORD*). It has soon been recognized that the possibility to decompose goals into a hierarchy of sub goals is required, which has been formalized in the *Card-Morn-Newell GOMS* (CMN-GOMS) in 1983 (Card et al. [33]). *Methods* describe how to achieve goals or sub goals and are composed from a sequence of *operators*, whereas an operator is an atomic perceptual, motor or cognitive action. The *duration* of an operator models the time to execute it, thus, from the overall duration of the involved operators in accomplishing a goal, the task completion time can be inferred. If there are multiple methods for performing a task, then a set of *selection rules* models the user's behavior of selecting a method.

Having explained the basics of GOMS modeling we now go into the details of the speech-GUI model. Section 2.2 begins with control actions followed by speech functions in section 2.3, and commands in section 2.4. Speech recognizers are introduced in section 2.5 followed by recognition result interpreters in section 2.6. After discussing the core components separately we provide a consolidated UML class diagram of the speech-GUI model in section 2.7, which also provides a high-level UML sequence diagram to visualize the dynamic aspects. The chapter closes with a discussion of the speech-GUI model regarding the fulfillment of the design goals and inherent limitations, and compares the speech-GUI model to other user interface models (section 2.8).

## 2.2 Control Actions

Control actions are generalizations of navigation and activation<sup>1</sup>, i.e., navigation and activation are subclasses of control actions. Each graphical object defines a specific finite set of control actions, which represent its interactive functionality. Consider for example the simple GUI *G* depicted in Figure 2.1 on the next page, which consists of a graphical object representing a push button which we call *Save-button* in the following (the black rectangle that encloses the *Save-button* shall represent the boundaries of the screen). For the *Save-button* two control actions exist: a navigation that sets the focus to the *Save-button* (*NSAVE*) and an activation which triggers an application-specific function associated with the *Save-button* (*ASAVE*), e.g., a saving process of data to a file.

Control actions are, just like navigation and activation, independent of the input modality that is used to control the GUI. They can therefore be regarded as a GUI-specific instance of a low-level GOMS goal. Consider for example the two control actions defined for the *Save-*

---

<sup>1</sup>These terms were introduced in section 1.1.



Figure 2.1: Exemplary GUI G.

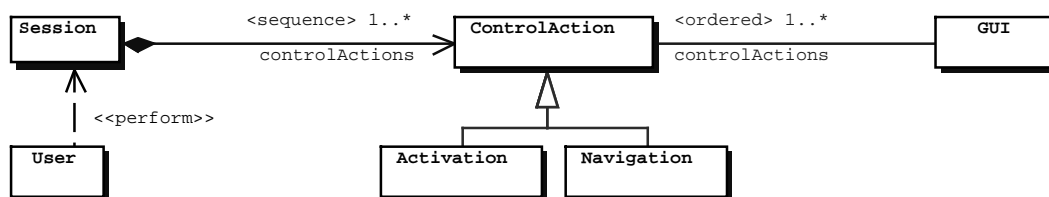


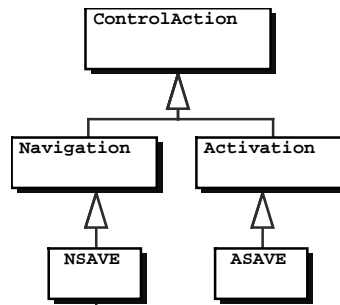
Figure 2.2: Relationships of control actions, navigations, activations and sessions.

button. If a mouse is available then the user would perform a sequence of mouse movements to position the mouse cursor over the button and then perform a click to activate the button. If a keyboard is available the user might use the arrow keys to position the mouse cursor over the button and then hit the space bar to activate the button. As each graphical object defines a specific finite set of control actions, the interactive functionality of a GUI is a finite set of control actions, which results as the union of all control actions of all graphical objects in the GUI. Control actions denote a model which allows for an input modality-independent specification of GUI-specific tasks. We call a sequence of control actions which represents a specific task a *session*. It can therefore be considered to be a GUI-specific instance of a GOMS high-level goal. Correspondingly, by *performing a session* we understand the performing of all control actions contained in that session. The UML diagram depicted in Figure 2.2 specifies the relationships between control actions, navigations, activations, sessions and GUIs. Figure 2.3 on the next page describes the control actions of the simple GUI G in UML, based on the classes defined in Figure 2.2. In the following section we describe how control actions are performed by the means of speech functions.

## 2.3 Speech Functions

Speech functions model the atomic functions which a specific speech-controlled GUI approach provides for controlling a GUI by the means of speech. For instance, given direction-based mouse emulation with continuous movement (such as described by Karimullah and Sears [95]), the following speech functions would be available:

1. Start continuous mouse cursor movement to the left (*SFML*).
2. Start continuous mouse cursor movement to the right (*SFMR*).



**Figure 2.3:** Control actions of simple GUI *G*.

3. Start continuous mouse cursor movement upward (*SFMU*).
4. Start continuous mouse cursor movement downward (*SFMD*) .
5. Stop mouse cursor movement (*SFSTOP*).
6. Perform a mouse button click (*SFCLICK*).

Speech functions are an analogy to the functions which a mouse device provides to control a GUI, such as movements, clicks and double clicks. In a mouse-controlled GUI the user utilizes the mouse functions to control the GUI, i.e., to perform a session. As such, the user maps a session to a specific sequence of mouse functions. The situation with speech-controlled GUIs is analog: here, the user utilizes the speech functions to perform a session and correspondingly maps a session to a specific sequence of speech functions. We call the sequence of speech functions to which a user maps a session the user's *intention* – as the user intends to perform the session by this specific sequence of speech functions. It is tempting to consider speech functions and intentions to be an analogy of operators and methods in terms of the GOMS model. However, speech functions and intentions represent an intermediate abstract level between goals and operators/methods. Anticipating the discussion in section 2.4, *commands* will correspond to operators, they represent the actual interaction with the system. This abstraction layer allows us to model alternative commands for the same functionality. The UML diagram in Figure 2.4 on the following page depicts speech functions and their relationship to control actions.

The number of speech functions of an intention does not need to be equal to the number of control actions in the session, i.e., there is not necessarily a 1:1 mapping between control actions and speech functions. In order to illustrate this coherence we will give examples which emanate from the simple GUI *G* (Figure 2.1 on the preceding page, Figure 2.3, respectively). We assume that the user is given the task to activate the *Save*-button with the precondition that the *Save*-button is currently not focused. Therefore, the user has to navigate to the *Save*-button (*NSAVE*) before it can be activated (*ASAVE*). We refer to this sequence of control actions as the session  $SG := (NSAVE, ASAVE)$  which is described in UML diagram in Figure

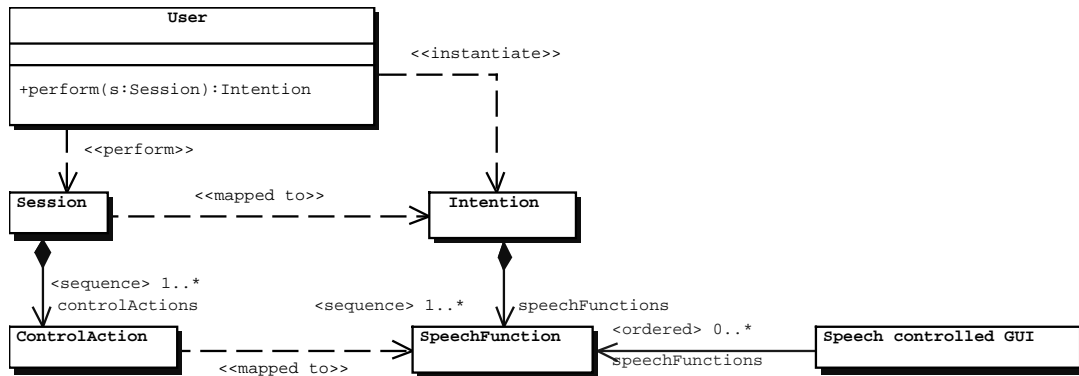


Figure 2.4: Speech functions and their relationship to control actions.

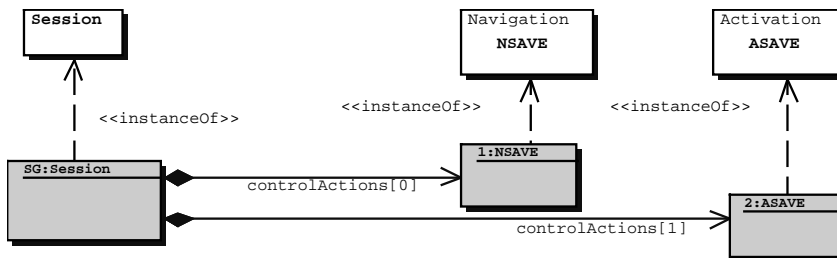


Figure 2.5: Session SG.

2.5. In the following UML diagrams objects, i.e., instances of classes, will be depicted with a gray background.

In the following we give three examples for performing the session SG using different speech-controlled GUI approaches. The first example assumes that command-and-control with random navigation and direct activation is used. It shows that the number of control actions can be equal to the number of speech functions. In the second example direction-based mouse emulation with continuous movement is available, showing that the number of control actions can be lower than the number of speech functions. The third example shows that the number of control actions can be greater than the number of speech functions considering the approach command-and-control with random navigation and indirect activation.

### Command-and-Control with Random Navigation and Direct Activation

In this example we assume that command-and-control with random navigation and direct activation is available, which assigns graphical objects a speakable identifier. Users navigate to graphical objects by uttering the identifier. Figure 2.6 on the next page depicts the initial situation for this example: there is no mouse cursor and the `Save`-button is assigned the identifier "Save".

For the context of this example we define two speech functions: one speech function causes





Figure 2.6: Initial situation for example 1.

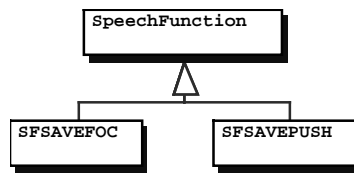


Figure 2.7: Speech functions of example 1.

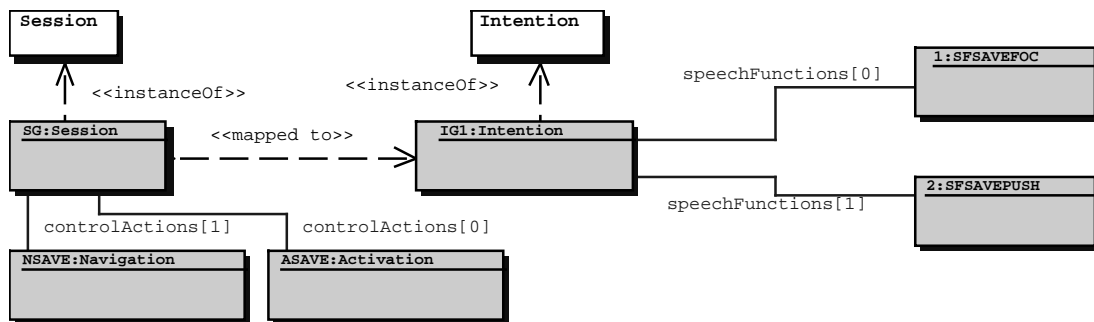
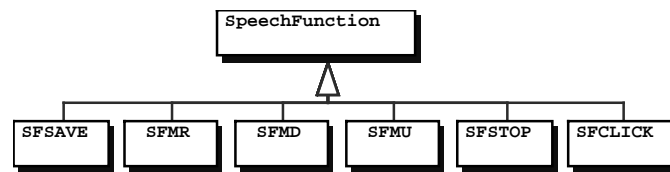


Figure 2.8: Intention *IG1*.

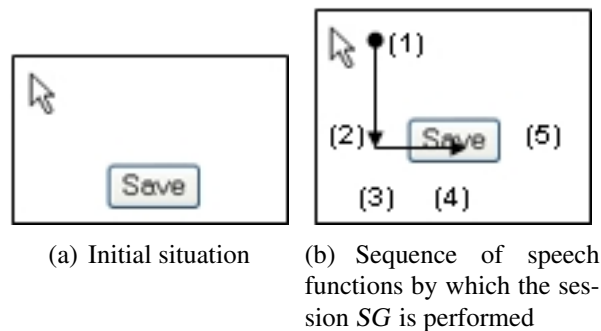
the GUI to set the focus to the *Save*-button (*SFSAVEFOC*) and one speech function that causes the *Save*-button to be pushed, given that it has been navigated to before (*SFSAVEPUSH*). These speech functions are expressed in UML in Figure 2.7.

The user is able to perform the navigation to and the activation of the *Save*-button by invoking the speech function *SFSAVEFOC* followed by the speech function *SFSAVEPUSH*. Consequently, in the context of this example, the session *SG*, consisting of two control actions, is mapped to the intention  $IG1 := (SFSAVEFOC, SFSAVEPUSH)$  as illustrated in Figure 2.8.

This example shows that there are cases where the number of control actions is equal to number of speech functions in the corresponding intention. This denotes that a (sub-)sequence of control actions of length  $n$  can be mapped to a (sub-)sequence of speech functions with the length  $m$ , where  $n = m$ .



**Figure 2.9:** Speech functions of example 2.



**Figure 2.10:** Initial situation for example 2.

### Direction-based Mouse Emulation with Continuous Movement

We will provide an example where the number of speech functions is greater than the number of control actions. For this example we assume that continuous direction-based mouse emulation is used and the available speech functions, as introduced in the beginning of this section on page 22, are described in UML in Figure 2.9.

In the following we refer to Figure 2.10. Figure 2.10(a) depicts the initial situation for this example: the mouse cursor is positioned in the north-west of the *Save*-button. Continuous direction-based mouse emulation only permits orthogonal movements of the mouse cursor along the axes of the screen. Therefore, given this initial situation, *SG* can be performed by

1. invoking the speech function that sets the mouse cursor in downward motion along the screen's Y axis (*SFMD*),
2. invoking the speech function that stops the movement of the mouse cursor when it is positioned in the west of the *Save*-button (*SFSTOP*),
3. invoking the speech function that sets the mouse cursor in rightward motion along the screen's X axis (*SFMR*),
4. invoking the speech function that stops the movement of the mouse cursor when it is positioned over the *Save*-button (*SFSTOP*),

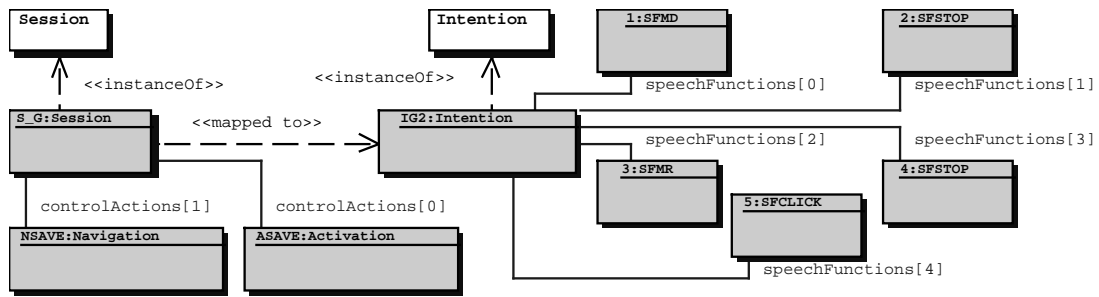
Figure 2.11: Intention *IG2*.

Figure 2.12: The initial situation example 3.

5. invoking the speech function that emulates the mouse button click (*SFCLICK*); as illustrated in Figure 2.10(b).

In the context of this example the session *SG*, consisting of two control actions, can be mapped to the intention  $IG2 := (SFMD, SFSTOP, SFMR, SFSTOP, SFCLICK)$  which consists of five speech functions as illustrated in Figure 2.11.

This example shows that the number of control actions in a session can be lower than the number of speech functions in the corresponding intention. This denotes that a (sub-)sequence of control actions of length  $n$  can be mapped to a (sub-)sequence of speech functions with the length  $m$ , where  $n \leq m$ .

### Command-and-Control with Random Navigation and Indirect Activation

We will now provide an example for the mapping of sessions to intentions where the number of control actions is greater than the number of speech functions. For this we assume that command-and-control with random navigation and indirect activation is available. With this approach graphical objects are assigned an identifier which the user speaks in order to trigger navigation and, possibly, an indirect activation (e.g., Conversay Voice Surfer [43]). Figure 2.12 depicts the situation for this example: there is no mouse cursor and the *Save*-button is assigned the identifier "Save". We assume that in this example the navigation to the *Save*-button (*NSAVE*) is coupled with its activation (*ASAVE*), i.e., the navigation to the *Save*-button indirectly performs its activation. We abstract this as the only available speech function *SFSAVE*, depicted in Figure 2.13 on the following page. As such, the user is able to perform the navigation to and the activation of the *Save*-button by invoking a single speech

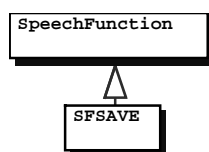
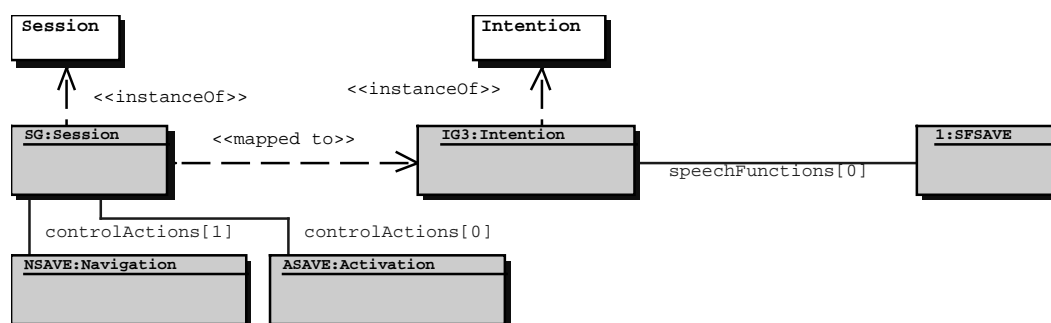


Figure 2.13: Speech functions of example 3.

Figure 2.14: Intention  $IG3$ .

function. Consequently, in the context of this example, the session  $SG$ , consisting of two control actions, is mapped to the intention  $IG3 := (SFSAVE)$  as illustrated in Figure 2.14.

This example shows that there are cases where the number of control actions is greater than the number of speech functions in the corresponding intention. This denotes that a (sub-)sequence of control actions of length  $n$  can be mapped to a (sub-)sequence of speech functions with the length  $m$ , where  $n \geq m$ .

To summarize, the relationship between speech functions of a specific speech-controlled GUI approach and control actions of a specific GUI in general is a many-to-many relationship. It might depend on the current state of the GUI, e.g., the current position of the mouse cursor. We define that a specific speech-controlled GUI approach defines a finite set of GUIs and based on this definition, we explain in the following section how speech functions are invoked.

## 2.4 Commands

Commands are sequences of words which the user utters in order to trigger the invocation of speech functions. We assume that for each speech function contained in an intention one command has to be spoken. We require that a specific command does not invoke more than one speech function. We do, however, permit that a specific speech function can be invoked by more than one command (e.g. to provide commonly used command alternatives), however, the set of commands which invoke a specific speech function must be finite. We also permit that there might be no command which invokes a specific speech function, for instance, to

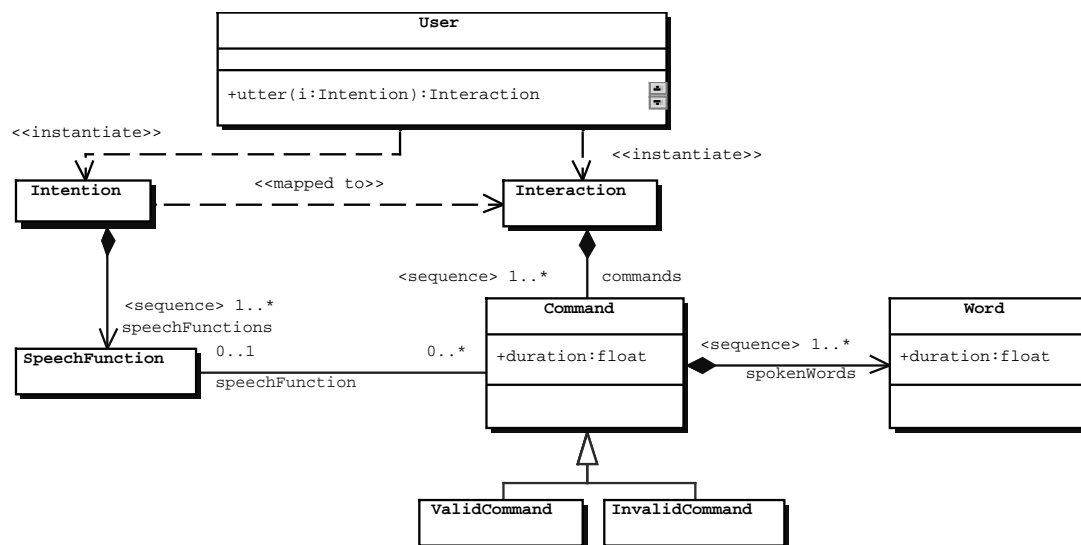
| <b>Valid Command(s)</b>                      | <b>Speech Function</b> |
|--|------------------------|
| "left", "move left"                          | <i>SFML</i>            |
| "right", "move right"                        | <i>SFMR</i>            |
| "up", "move up"                              | <i>SFUP</i>            |
| "down", "move down"                          | <i>SFMD</i>            |
| "stop", "halt"                               | <i>SFSTOP</i>          |
| "click"                                      | <i>SFCLICK</i>         |
| <b>Invalid Command(s)</b>                    | <b>Speech Function</b> |
| "left right", "press up",<br>"move", "hello" | -                      |

**Table 2.1:** Exemplary valid and invalid commands for direction-based mouse emulation with continuous movement, as described by Karimullah and Sears [95].

model a situation where the state of the speech-controlled GUI currently does not allow the invocation of this specific speech function. As such, the relationship between commands and speech functions is many-to-one.

Commands which do invoke a speech function are called *valid commands*, whereas commands which do not invoke a speech function are called *invalid commands*. Since the number of speech functions of a speech-controlled GUI approach is finite and since a finite number of commands invokes a specific speech function, the number of valid commands is finite as well. In Table 2.1 we present exemplary valid and invalid commands for direction-based mouse emulation with continuous movement. Hereby it is important to note that the speech recognizer would distinguish between the separate commands "left" and "right", and the single command "left right" by detecting a significantly long enough pause between the single words. This pause is usually configurable. Each command requires a certain amount of time to be uttered and we call this time the *duration of the command*. It can be determined by summing up the times required to utter the words of which the commands consists. Correspondingly, we call the time to utter a word the *duration of the word*. We assumed in the introduction of this section that per speech function one command has to be uttered, correspondingly, users map their intention to a corresponding sequence of commands. We call this sequence of commands an *interaction*. Referring back to the discussion about GOMS operators in section 2.3 it is now clear why commands correspond to GOMS operators: commands have a duration which, however, depends on the particular composition from words. Commands are uttered by the user in a sequence called interaction, thus, interactions, and not intentions, correspond to GOMS methods.

Early commercially available speech recognition systems were discrete speech recognition systems which required the user to pause between the spoken words. These systems had a vocabulary of about 20,000 words and in order to achieve a word recognition rate of about 95% users had to speak no faster than 50 words per minute (Stuckless [180]). The speech-GUI model, however, assumes the availability of a continuous speech recognizer, i.e., it as-



```

context Word
inv: duration = 571
context Command
inv: duration = spokenWords.iterate(elt; acc : T = 0 | acc + elt)
context ValidCommand
inv: speechFunction.notEmpty()
context InvalidCommand
inv: speechFunction.isEmpty()
    
```

**Figure 2.15:** Commands and their relationship to speech functions.

sumes that the user does not make artificial pauses between words. While researches at Stanford found that early continuous speech recognizers were lacking performance (Detmer et al. [197])<sup>2</sup>, later systems supported a speaking speed of about 150 words per minute with a word recognition rate of about 95% (Zafar et al. [13]). The speaking speed supported by speech recognizers does just reach the lower bound of the speed range by which individuals communicate, which lies between 130 and 250 words per minute depending on the domain (e.g. Reed [152] and Arons [8]). However, if people talk to a computer then they tend to reduce their speaking speed to about 105 words per minute (Karat et al. [93], Lewis [107]). For the speech-GUI model we therefore assume a speaking speed of 105 words per minute which denotes an average duration of a word of 571ms. This value can, if needed, easily be adapted to model users which speak faster or slower.

The UML diagram depicted in Figure 2.15 specifies the relationships between intentions, interactions, commands and speech functions. In the following section we discuss the recognition of commands by speech recognizers.

<sup>2</sup>Low performance denotes low speaking speed and low word recognition rate.

## 2.5 Speech Recognizers

In section 1.2.1 we have introduced speech recognizers as software components which process digital audio signals – digital representations of the vocalization of a command. Digital audio signals are obtained by capturing the wave form that the human sound apparatus generates. In the context of speech-controlled GUIs this is usually achieved by a sound card to which a microphone is connected. The microphone captures the wave form and transforms it into an analog audio signal. The sound card transforms the analog audio signal into a digital audio signal. From the digital audio signal a recognition result is calculated under consideration of the phoneme database, the vocabulary and the language model (as discussed in section 1.2.1).

The characteristics of audio signals, their creation processes, as well as the details of the speech recognition process are beyond the scope of this dissertation, and are therefore not included in the speech-GUI model. We model a speech recognizer as a black box component which transforms one sequence of words, i.e., the spoken words of a command, into another sequence of words, i.e., the recognized words of a recognition result. We do, however, not specify how this transformation takes place.

Having defined a speech recognizer as a black box component requires us to define valid input and the expected output – otherwise the output of a black box component with respect to a specific input would be undefined, and as such it would be ill-defined. We define the entirety of the valid commands of a specific speech-controlled GUI approach as the valid input for the speech recognizer. All other commands, i.e., the invalid commands, are invalid input for the speech recognizer and as such the output of the speech recognizer for invalid commands is undefined. We base this definition on the following two assumptions.

First, we assume that the user is aware of the available valid commands of the respective speech-controlled GUI approach, i.e., we assume that the user knows which valid commands exist. This can for instance be achieved by training the user prior to using the speech-controlled GUI (as discussed in section 1.3.1).

Second, we assume that the user adheres to the *Cooperative Principle for Conversations* (Grice [71]). This principle requires, amongst other requirements, that all parties involved in a conversation – in our case the user and the speech-controlled GUI – must talk about the same conversation topic(s); otherwise the conversation is likely to break down. We consider valid commands to be the conversation topics, and later in chapter 5 we will provide a more detailed abstraction of conversation topics.

Consequently, our definition of valid input for the speech recognizer is feasible: the user knows about the valid commands and the speech-controlled GUI approach determines the valid commands – as such both parties know about them. For avoiding a conversation breakdown, both user and speech-controlled GUI have to utter/understand the valid commands. The reason for constraining valid input to valid commands will become obvious in chapter 3 when we specify the interaction delay model.

We call the output of the speech recognizer the *recognition result*. If the recognition result is equal to the input of the transformation process, i.e., the valid command, we say that the command has been recognized correctly and call the recognition result a *recognition success*.

Otherwise, if the input to the transformation process is not equal to the recognition result, we call the recognition result a *recognition error* and we say that the command has not been recognized correctly.

The speech recognizer has two attributes which denote commonly used quality measures of the transformation of valid commands to recognition results: the *recognition delay* and the *word error rate*. We describe the integration of these quality measures into our model in the following two sections.

### 2.5.1 Recognition Delay

The recognition delay denotes the time after which a recognition result is available after the user has stopped uttering a command (as discussed in section 1.2.1). In the following we will derive an estimation for the recognition delay that can be used for concrete calculations based on our model.

Today it is possible to create speech recognizers which perform speech recognition in about computational real time (Sprex, Inc. [10]). Computational real time denotes that the time to calculate a result from a specific input takes as long as the length of the input<sup>3</sup>. Applied to speech recognition this means that calculating a recognition result takes as long as the duration of the command.

Thus, we would expect that after the user has finished uttering the command, the recognition result would be available after a recognition delay that is (approximately) as long as the duration of the command. However, the recognition delay is independent of the duration of the command (Glass et al. [66]). This can be explained by the stream-oriented processing which speech recognizers apply to audio signals. A speech recognizer starts calculating the recognition result as soon as it receives the first bit of the digital audio data. In other words, the speech recognizer starts computing the recognition result as soon as the user starts speaking and while the user is speaking – not after the user has stopped speaking. We would consequently expect, considering that speech recognition can be performed in computational real time, that the recognition result is available right after the user has stopped speaking; that is, with a recognition delay of 0. This is, however, not reality. Computational real time can only be achieved under optimal conditions in a laboratory<sup>4</sup>. These conditions do by far not reflect the conditions which a speech recognizer is exposed to when it is used in an application.

By evaluating empirical studies of the recent past (as discussed in the following) it becomes clear that the recognition delay lies in the magnitude of seconds. A chronological comparison of these studies reveals that the recognition delay could be reduced significantly. On the one hand this is due to the technological advances in speech recognition theory and improvements of the underlying models. On the other hand, since speech recognition is a computational intensive task, the reduction of the recognition delay is also due to the increase in available computational power. In the following we report about empirical studies with the goal to

---

<sup>3</sup>The length of the input is measured specifically to the application domain.

<sup>4</sup>Small vocabulary, low environmental noise, and good microphone quality



|   |
|---|
| <p>Correct: Did mob mission area of the Copeland ever go to m4 in nineteen eighty one</p> <p>Recognized: Did mob mission area * * the <b>copy</b> <i>land</i> ever go to m4 in nineteen <b>east</b><br/>one</p> |
|---|

**Figure 2.16:** Examples for word errors.

Substitutions are printed bold, insertions are printed in italics and deletions are denoted as \*.

derive and justify a constant value for the recognition delay that will be used in our model <sup>5</sup>.

In an experiment performed by Glass et al. [66] the utilized speech recognizer was equipped with a vocabulary of roughly 2000 words and achieved a recognition delay of < 1s in 85%, and a recognition delay of < 2s in 99% of all monitored utterances. In [11] Mastan reports about a beta test of Microsoft Speech Server 2004 <sup>6</sup> in which the speech recognizer has achieved a recognition delay of < 1.5s in 95% of all monitored utterances. SPIRIT DSP [177] claims that their speech recognizer <sup>7</sup> reaches a recognition delay < 500ms – however, this only applies to a vocabulary of up to 10 words which can for instance be used to recognize digits for a speech-controlled phone in a car. We conclude that a recognition delay of < 2s, depending on the environment of the experiment, the size of the vocabulary, and the specifically utilized speech recognizer, is realistic. As we will later show in section 3.2.1 the vocabularies used for speech-controlled GUI approaches have a maximum size < 70. Thus, our model is based on a recognition delay of 1.5s (Glass et al. [66]) – it can, however, easily be adapted to other values. In the following section we discuss the word error rate of a speech recognizer.

## 2.5.2 Word Error Rate

The word error rate denotes the average frequency by which word errors occur during the recognition process. According to Huang et al. [80] there are three types of word errors, called *substitution*, *deletion* and *insertion*.

**Substitution** an incorrect word was substituted for a correct word

**Deletion** a correct word was omitted in the recognition result

**Insertion** an extra word was included in the recognition result

In Figure 2.16 we depict an example from [80] that illustrates substitutions, deletions and insertions. For calculating the word error rate one has to align the correct text string and the

<sup>5</sup>In literature the recognition delay is also referred to as the *recognition latency*.

<sup>6</sup><http://www.microsoft.com/speech/default.aspx>

<sup>7</sup>Data sheet available at

<http://www.spiritdsp.com/pdf/SPIRITAutomaticSpeechRecognitionDataSheet.pdf>

recognized text string and then compute the number of substitutions (*subs*), deletions (*dels*) and insertions (*ins*)<sup>8</sup>. The word error rate can then be calculated as

$$WordErrorRate = 100\% \cdot \frac{subs + dels + ins}{n}$$

where  $n > 0$  denotes the number of words in the correct text (there must be at least one word in the correct text). It is important to note that the word error rate might become greater than 100%. Consider for instance a case where each word in the correct text is substituted and after each substituted word another word is inserted. This would make up  $n$  substitutions and  $n$  insertions leading to a word error rate of 200%.

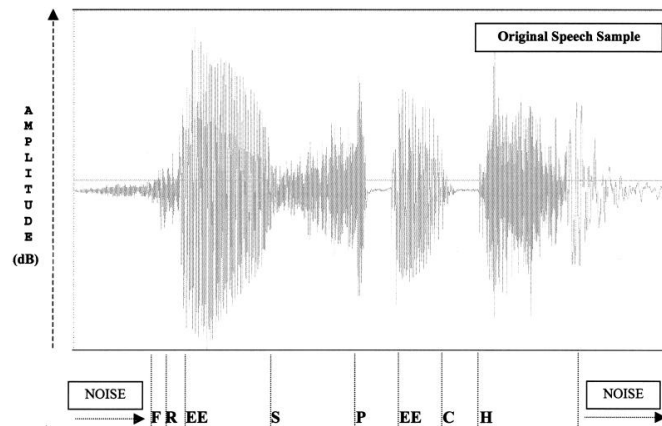
In our model we define that the word error rate is an inherent property of a speech recognizer. It denotes the average rate by which word errors occur per spoken word, given a specific set of environmental conditions, e.g., a measure for the amount of environmental noise. As such, a word error rate of 50% means that, on the average, a word error happens with every second word. A word error rate of 100% means that a word error happens, on the average, with every word. We neglect word error rates above 100% in our model because it is irrelevant if more than one word error happens with one word – already with a word error occurring with every word of a command, the average probability of a word of a command being correctly recognized is 0. It remains 0 even if the word error rate increases above 100%. As a matter of fact we will later show in section 3.2.3 that the average word error rate of a speech recognizer for speech-controlled GUIs must be below the so called *critical word error rate* to be usable – which is typically below 100%.

Our interpretation of the word error rate is a simplification compared to the actual definition. We assume that the probability of a word error with a specific word is equally distributed over all words in the speech recognizer's vocabulary. As such, we model the speech recognition process as a random experiment with constant probabilities for each of the possible events. This includes the assumption that only one word error can happen per word. The reality with speech recognizers, however, is different, as the recognition probability distribution varies between different words. The reason for this is that different words are composed from different phonemes. There are phonemes (phoneme sequences, respectively) which are "easy" to recognize, such as vowels, and phonemes which are "more difficult" to recognize, such as silent consonants as depicted in Figure 2.17 on the next page (refer to Huang et al. [78]). Depending on their composition from phonemes the recognition probability of words in the vocabulary is therefore in fact not equally distributed.

We make this simplification for the sake of facilitating further calculations as the real probability distribution for word errors for words in the vocabulary is a priori unknown. It is furthermore virtually impossible to specify it precisely, as the speech recognition process is affected by a priori unknown factors, including environmental noise or limitations in the models used for speech recognition. Our interpretation represents a black-box view of speech recognizers and we believe that it is appropriate for speech recognizers with small vocabularies (Cole et

---

<sup>8</sup>This process is called the *maximum substring matching problem*, described in Huang et al. [78].



**Figure 2.17:** Example of silent consonants "p" and "c" speaking the words "free speech" taken from Zafar et al. [13].

al. [40]), such as speech-controlled GUI approaches.

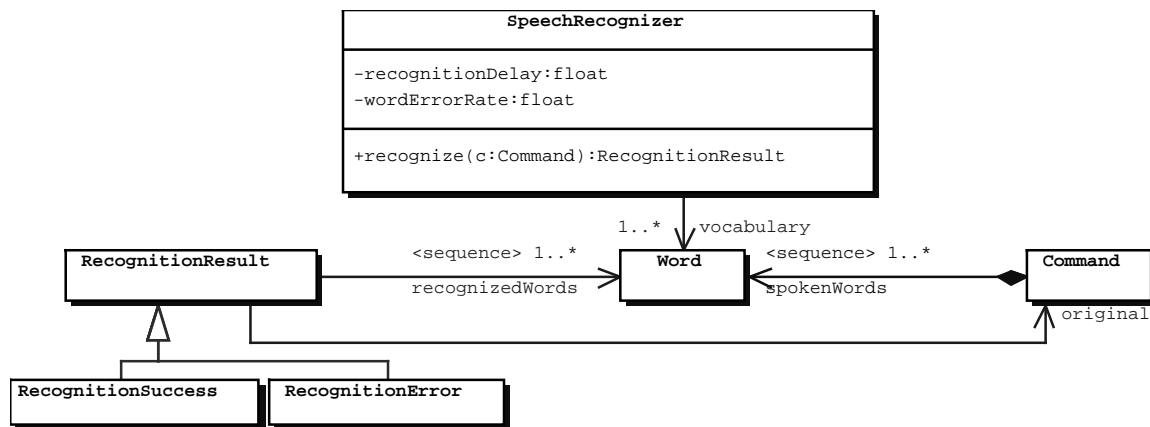
### 2.5.3 Summary

The UML diagram depicted in Figure 2.18 on the following page specifies a speech recognizer and the concepts discussed in this section. The association from recognition error to command is of theoretical nature to be able to specify invariants for recognition success and recognition errors in Object Constraint Language (OCL)<sup>9</sup>. We found it surprising that the GOMS model, up to the time of writing this thesis, has not been extended to (speech) recognition based user interfaces. There have, indeed, been efforts to model the act of speaking in the scope of analyzing the performance of human telephone operators (John [90], Stuart and Gabrys [179]), but the characteristics of a speech recognizer have not yet been integrated. As such, we see our concept of a speech recognizer and especially the interaction delay model (chapter 3) as an extension of GOMS in this direction: our model allows predicting the execution time for a speech function under consideration of the recognition delay and the word error rate. In the following section we explain the interpretation of recognition results.

## 2.6 Recognition Result Interpreters

Whenever a new recognition result becomes available – be it a recognition error or a recognition success – it is processed by the recognition result interpreter. It has knowledge about the valid commands of a specific speech-controlled GUI approach. It determines if the recognition result matches a valid command, i.e., if there exists a valid command of which the sequence of spoken words is equal to the sequence of recognized words.

<sup>9</sup><http://www.uml.org/>



```

context SpeechRecognizer
inv: recognitionDelay = 1500
inv: (wordErrorRate >= 0) and (wordErrorRate <= 1)
context RecognitionSuccess
inv: original.spokenWords = recognizedWords
context RecognitionError
inv: original.spokenWords <> recognizedWords
  
```

**Figure 2.18:** Speech recognizer.

We do not specify how this matching should take place. For instance, a specific recognition result interpreter might represent valid commands as strings (e.g. by concatenating the spoken words) and determine a match by walking through the strings and comparing them against the concatenation of the recognized words. Another recognition result interpreter might use context free grammars or semantic grammars to represent valid commands, and use a corresponding parser to determine which valid command the recognized words match. In fact, specialized context free grammars are commonly used to parse recognition results. For instance, *probabilistic context free grammars* (e.g., Huang et al. [78]) allow the parser to consider the probability of a specific parse path to occur in a real world language. *Semantic grammars* (e.g., Gavalda ([64])) allow for directly generating a representation of the semantic information in a recognition result from the parse tree.

If the recognition result interpreter was able to determine a match, we assume that it provides a reference to the speech function which is associated with the matching command. If no match could be determined, the recognition result interpreter provides a reference to a value that indicates that no speech function could be determined, such as  $\perp$ . Some other component of a speech-controlled GUI, for instance a controller object, can then invoke the returned speech function, or, if no match could be determined, generate a feedback message for the user, stating that the input could not be processed – such as "Sorry, I could not understand

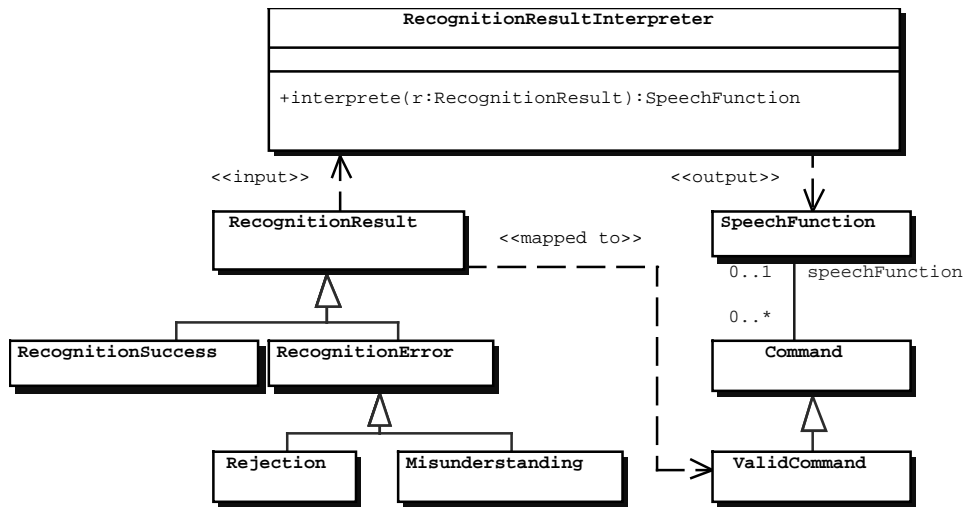


Figure 2.19: Recognition result interpreter.

your request. Please repeat!"

As such, we model a recognition result interpreter as a black box component which accepts recognition results from a speech recognizer as valid input. The output of the recognition result interpreter given a specific valid input is either a speech function or  $\perp$ .

Depending on the value which the recognition result interpreter outputs we are able to specialize recognition errors. The specialization reflects the user's experience regarding the reaction of a speech-controlled GUI on a recognition error. In the following we introduce *rejection* and *misunderstanding* as specialized recognition errors.

A rejection is a recognition error for which the recognition result interpreter outputs  $\perp$ . It occurs if the recognition error does not match a valid command that the recognition result interpreter knows of. For instance, if the user uttered "move left" (which would be a valid command according to Table 2.1), but the speech recognizer returned the recognition error "move" (which would be an invalid command) then no valid command matches the recognition result and the recognition result interpreter returns  $\perp$ . In other words, a rejection is a recognition error which matches an invalid command.

A misunderstanding is a recognition error for which the recognition result interpreter returns a reference to a speech function, although the recognized words are not equal to the originally spoken words. This happens if – per coincidence – the recognized words of the recognition error match a valid command. For instance, if the user uttered "move left" but the speech recognizer returned "move right" the recognition result is not equal to the command that the user uttered, however, "move right" is a valid command that invokes the speech function *SFMR*. The UML diagram depicted in Figure 2.19 summarizes the concepts discussed in this section.

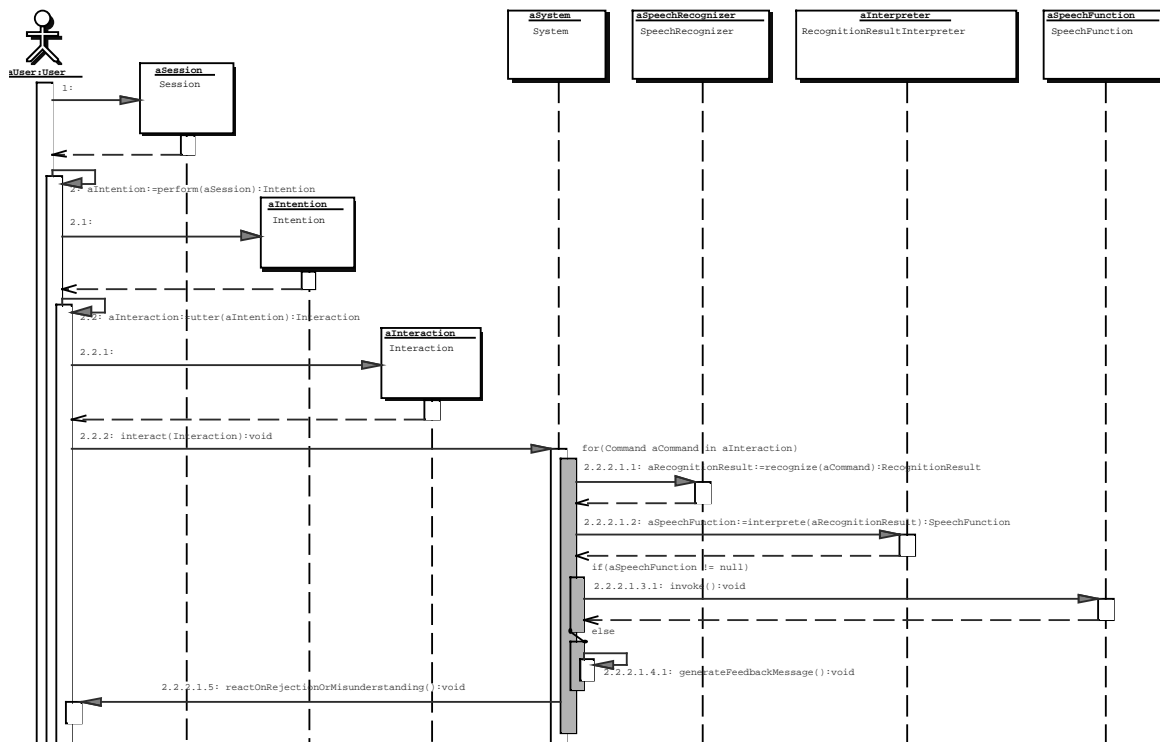


Figure 2.20: Dynamic speech-GUI model.

## 2.7 Consolidated Speech-GUI Model

Figure 2.20 depicts a dynamic model of a speech-controlled GUI approach, based on the classes defined in the preceding sections. Note that the **System** class models the actual speech-controlled GUI, i.e., it is an abstraction of the "glue" between the components of a speech-controlled GUI. We explain the dynamic model in the following.

The user begins with instantiating a session (1). The session can be seen as a mental model of the task that the user performs with the GUI, e.g., a predefined task from a user interface evaluation test. Next, the user performs the session (2) in order to instantiate a new intention (2.1). The new intention contains a sequence of speech functions by which the given session can be performed. In (2.2) the newly instantiated intention is uttered causing the instantiation of a new interaction (2.2.1). The new interaction contains a sequence of commands which invoke the speech functions of the previously instantiated intention. The user passes the interaction to the system (i.e., the vocalization of the command(s) gets digitalized). For each command which is contained in the interaction the system calls the speech recognizer to recognize it (2.2.2.1.1). The system receives the resulting recognition result and passes it to the recognition result interpreter to interpret it (2.2.2.1.2). After that the system receives the result of the interpretation. If the interpretation result is a reference to a speech function the system invokes this speech function (2.2.2.1.3.1), otherwise, if the interpretation result is  $\perp$ ,

a feedback message is generated (e.g. "Sorry, I could not understand you!") in (2.2.2.1.4.1). By (2.2.2.1.5) we abstract the user's reaction to rejections (the interpretation result is  $\perp$ ) or misunderstandings (the interpretation result is a speech function which was not intended). We will deal with this reaction in more details later on in chapter 3.2.3 when we discuss the implications of recognition errors to an interaction.

Finally, Figure 2.21 on the next page depicts the static portion of the speech-GUI model as a UML class diagram (consolidation of the class diagrams in Figures 2.3, 2.4, 2.15, 2.18 and 2.19).

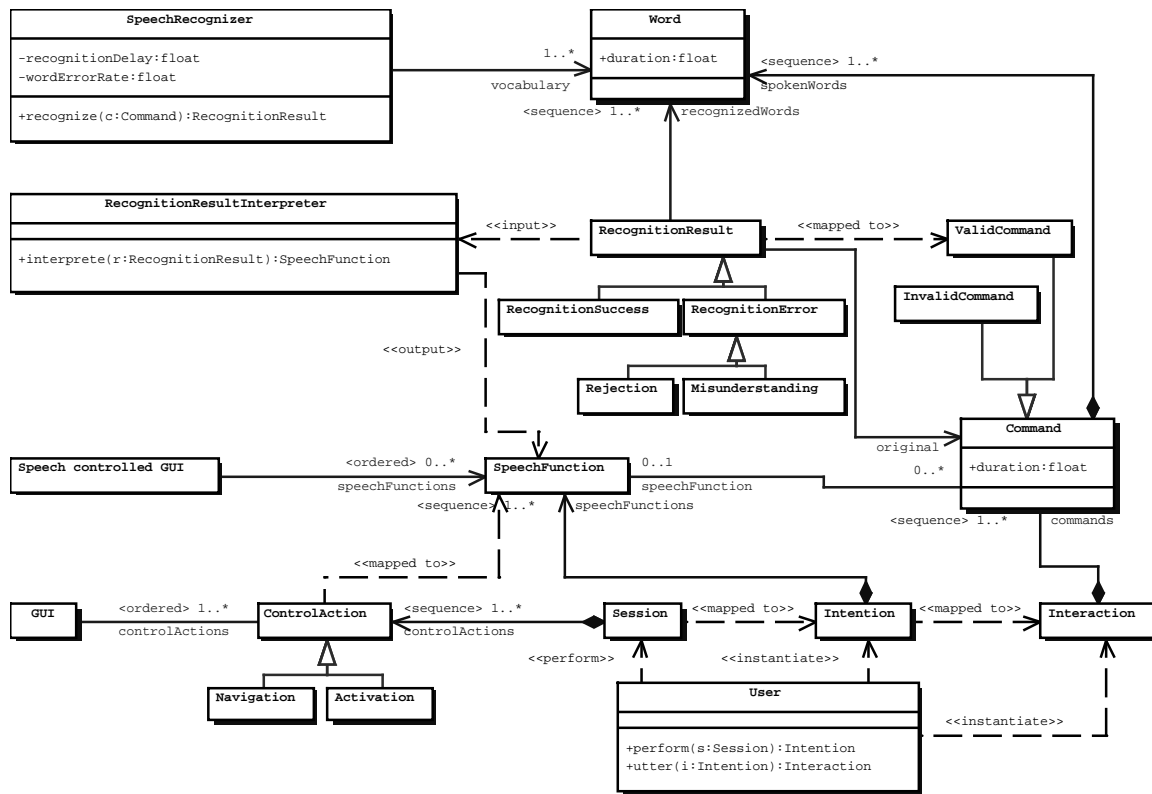
## 2.8 Discussion

In this chapter we have presented the speech-GUI model. It abstracts a speech-controlled GUI as control actions, speech functions, commands, speech recognizers and recognition result interpreters, and defines interaction rules between them, including the data objects which are exchanged. We have compared the speech-GUI model to the GOMS model, a commonly used means for analyzing routine human-computer interactions. The speech-GUI model partially overlaps with GOMS, e.g., control actions and sessions can be considered to be low-level and high-level GOMS goals. There are, however, two differences: first, the speech-GUI model introduces an intermediate layer between goals and operators/methods in order to allow for abstracting the actual speech-controlled functions from the commands by which they are triggered. Second, we include model aspects to cover the characteristics of speech recognition, which are specified in more detail in chapter 3. We now discuss the speech-GUI model regarding the realization of the initial design goal (section 2.8.1) and inherent limitations (section 2.8.2). We conclude this chapter by comparing our model to other existing models for (graphical) user interfaces (section 2.8.3).

### 2.8.1 Reconsidering the Design Goals

The goals which have driven the design of the speech-GUI model were the encapsulation of the speech recognition process, the consideration of the duration of the input, the coverage, the implementability, and the extensibility.

**Encapsulation of Speech Recognition Process** The speech recognizer of the speech-GUI model represents a black box component which encapsulates the speech recognition process. The interface of this component represents the externally visible aspects of the speech recognition process and consists of two parts: the first part is a method which models the transformation of a command (its vocalization, respectively) into a recognition result. The second part of the interface consists of attributes recognition delay and (average) word error rate. They give information about the quality of the transformation, i.e., the first part of the interface, without revealing details of the implementation of the transformation. As such, the speech-GUI model encapsulates the speech recognition process as required.



```

context SpeechRecognizer
inv: recognitionDelay = 1500
inv: (wordErrorRate >= 0) and (wordErrorRate <= 1)
context RecognitionSuccess
inv: original.spokenWords = recognizedWords
context RecognitionError
inv: original.spokenWords <> recognizedWords
context Word
inv: duration = 571
context Command
inv: duration = spokenWords.iterate(elt; acc : T = 0 | acc + elt)
context ValidCommand
inv: speechFunction.notEmpty()
context InvalidCommand
inv: speechFunction.isEmpty()
    
```

Figure 2.21: Static speech-GUI model.



**Duration of Input** The duration of the speech-based input has been considered by the duration of words, commands, respectively. The duration of a word reflects the time needed to utter a word and the duration of a command is the sum of the duration of all words that the command is composed of. As there is no other speech-based interaction than uttering commands the speech-GUI model considers the duration of the speech-based input.

**Coverage** The coverage of the speech-GUI model will be shown in chapter 4 where we perform an analysis of the speech-controlled GUI approaches which we have introduced in this dissertation. For each approach we provide a mapping to the speech-GUI model showing its coverage.

**Implementability** The conversation-and-control framework, presented in chapter 6, is based on the speech-GUI model for speech-controlled GUIs. We show the required implementability by several conversation-and-control prototypes using the conversation-and-control framework.

**Extensibility** The speech-GUI model can be extended to other speech-controlled GUI approaches. We show this in chapter 5 where we introduce *conversation-and-control* as a new speech-controlled GUI approach. Conversation-and-control has not been considered during the design of the speech-GUI model. Particularly in sections 5.1 and 5.2 we design and apply changes to the speech-GUI model in order to reflect the needs of this new approach. However, a general statement about the extensibility to other, possibly new approaches of speech-controlled GUIs cannot be made, as the characteristics of such new approaches are a priori unknown.

## 2.8.2 Limitations

The speech-GUI model, which we have presented in this chapter, accomplishes the design goals of section 2.1 to a certain extent, as discussed in section 2.8.1. However, our model has some limitations which we discuss in the following.

**Implications of Recognition Errors** The model treats an interaction as an atomic object. Thus, the model does not consider the implications which are induced by recognition errors, such as for instance the necessity to repeat a command if it was rejected. A model for these implications is provided in chapter 3 where we derive a model for the interaction delay of speech-controlled GUIs. It can be seen as an enhancement of the speech-GUI model for covering the side effects of recognition errors.

**Temporal vs. Instantaneous Interaction with Graphical Objects** We abstract the interactive functions of graphical objects by control actions. A specific control action represents an instantaneous change of a graphical object or a navigation. Examples for such instantaneous functions include the click on a button, the selection of a menu item,

or the checking of a check box. There exist, however, graphical objects which provide interactive functions that range over a specific period of time (Myers [137]), such as resizing a graphical object, drawing a frame for cutting out a specific region of an image, or moving a graphical object. Such interactive functions are called *temporal interactions* as they begin at a specific point in time and end in a later point in time. They cannot be described by control actions as control actions do not incorporate a temporal model. It is currently unclear how temporal interactions can be described by control actions, respectively, in what regards the model of control actions needs to be enhanced.

**Confidence** Specific existing implementations of speech recognizers augment recognition results with so called *confidence values*. A confidence value can be regarded as a percentage which indicates "how certain", i.e., "how confident", a speech recognizer is about the correctness of the respective recognition result. As such the speech recognizer aims at evaluating the correctness of its internal models. Confidence values can be used by recognition result interpreters to reject recognition results before attempting to interpret them, with the goal to minimize misunderstandings: if the confidence of a recognition result is below a specific confidence threshold then the recognition result is rejected by default. Otherwise it is attempted to be interpreted.

While the evaluation of confidence values can retroactively be integrated into a framework design that is based on our model, it complicates further calculations which we perform in chapter 3. The reason is that the probability of a specific confidence value being above or below the confidence threshold is a priori unknown, since the word error rate – upon which the confidence value depends – is not constant during the runtime of a speech recognizer (as explained in section 2.5). Therefore we neglect confidence values in our model.

### 2.8.3 Other User Interface Models

We compare the speech-GUI model to the *model view controller design pattern* (Reenskaug [154]), the *linguistic model* (Foley and Wallace [59]), the *interactor model* (Myers [137]), the *Seeheim model* (Green [70]), and the *Arch model* (Bass et al. [16]).

**Model View Controller** The model view controller (MVC) pattern has first been described in 1979 in the context of writing GUIs in Smalltalk by Trygve Reenskaug in [154]<sup>10</sup>. It has long emerged into a de-facto standard for the design of complex software systems (Gamma et al. [62]). MVC is originally an object-oriented technique for separating program code for GUIs into three modules called the *model*, the *view* and the *controller*. The model describes the structure of the data that the GUI presents and implements functionality to maintain it. The view aggregates program code which is responsible for presenting the data, the model, respectively. The controller coordinates the communication and the data flow between the model and the view. Each module defines an

---

<sup>10</sup>At that time Trygve Reenskaug called it the *thing-model-view-editor*.

interface for the communication with the other modules. Input from the user is captured by the view which reacts on it, e.g., changes graphical objects, and which propagates the input to the controller. This might then lead to changes in the model which are in turn reflected by the view. MVC can recursively be applied to decompose subsystems and subcomponents.

The *model view presenter pattern* (MVP) (Ullienboom [190]) is a variant of MVC which is optimized for rich clients, i.e., clients which a large amount of graphical objects and a large amount of dependencies between them. With MVC the interactive behavior of graphical objects is handled by the view, e.g. the disabling of a specific set of objects if a specific menu item is selected. MVP separates this behavior into a separate *presenter* object which first handles the input as it is relevant for the presentation and then propagates it to the controller. This reduces the view to its original purpose: a "dumb" component which simply presents the data.

MVC2 (Seshadri [167]), or Model 2, is another variant of MVC which is optimized for stateless web-based applications. In MVC the state of the view, i.e., the state of each graphical object, is encapsulated in the view itself. MVC2 factors out the state of the view into the controller which lives on the server, because views in the context of web-based applications are stateless (e.g., HTML pages).

The relationship between the MVC pattern (its variants, respectively) and the speech-GUI model is twofold. On the one hand the speech-GUI model can be seen as a suggestion for decomposing a controller for speech-controlled GUI approaches, assuming that the graphical objects represent the view and the underlying application represents the model. The user in our model would represent the object that controls the data flow within the controller. Session, intentions, and interactions would describe different representations of input data which the controller processes and which it internally transforms into each other. Speech recognizers and recognition result interpreters could be seen as helper classes which are used to determine suitable actions which the controller has to perform with the obtained data. On the other hand the MVC pattern has been applied to the design of the speech-GUI model for speech-controlled GUI approaches: the user acts as the controller, sessions represent the model and intentions and interactions represent two different views of the model – as a session is mapped to an intention which directly corresponds to an interaction.

**Linguistic Model** The linguistic model for user interface management is described by Foley and Wallace in [59]. The linguistic model is a layered model which focuses on the processing of input by a user interface. It separates a user interface into three layers called the *lexical layer*, the *syntactic layer* and the *semantic layer*, corresponding to the layers from formal language theory.

The lexical layer defines units (functions, respectively) of input, the so called lexemes, from which input to a computing system can be composed. Lexemes are indivisible (i.e., atomic) and represent the smallest units of semantical meaning. For example, the input

functions of a mouse device would be represented by a lexeme for each possibility to convey a movement (such as up, down, left, right). A lexeme for both the pressing and the releasing of the mouse button would exist. Regarding the speech-GUI model each word from the speech recognizer's vocabulary would be represented by one lexeme.

The syntactic layer defines valid sequences of lexemes, so called tokens. For instance, considering the mouse lexemes (as mentioned above), the lexeme for pressing the mouse button followed by the lexeme for releasing the mouse button denotes a "click". Or, the lexeme for pressing the mouse button followed by an arbitrary sequence of movement lexemes followed by the lexeme for releasing the mouse button denotes "drag-and-drop". Regarding the speech-GUI model a token corresponds to a command and its composition from specific words.

The semantic layer of the linguistic model describes the functionality of the user interface as a set of functions and defines which tokens are necessary to invoke a specific function. As such, the semantic layer would define functions for triggering specific changes of graphical object, and would define (sequences of) tokens which invoke the specific functions. The speech-GUI model introduces an additional indirection here. The functions of the linguistic model do not directly correspond to control actions. Instead, they correspond to speech functions, and for each speech function a specific set of commands is defined (analogously to tokens).

As shown, there are analogies between the linguistic model and the speech-GUI model as core components of the speech-GUI model can be mapped to concepts of the linguistic model. However, the linguistic model is a theoretical model which does not propose a particular system architecture that could be implemented. This is in contrast to the speech-GUI model which proposes an implementable architecture of a speech-controlled GUI approach.

**Interactor Model** The interactor model, introduced by Myers [137], abstractly describes the interactive functions of pointing-device operated GUIs by so called *interactors*. An interactor is an object which represents the interactive behavior of an entire class of graphical objects, such the class of menus or the class of buttons. Each graphical object has an interactor that corresponds to its class. The interactor captures and handles the input that is generated by a pointing device and modifies the graphical object accordingly. Myers defines six types of interactors: the menu interactor, the move-grow interactor, the new-point interactor, the angle interactor, the text interactor and the trace interactor. Each interactor can be parameterized by a specific set of general properties which allows for describing different types of graphical objects of the same class. Myers shows that six interactors are sufficient to describe the interactive behavior of pointing device-based GUIs (GUIs as of 1990, respectively).

The interactor model is specific to the input modality pointing device. It is, however, independent of the specific type of pointing device, e.g., mouse, trackball, light-pen, etc. Control actions, by which the speech-GUI model abstracts interactive functions,

are independent of the input modality by which the GUI is controlled. As mentioned in section 2.8.2 the interactor model is able to describe both temporal and instantaneous interaction, whereas the speech-GUI model only allows for describing instantaneous interaction.

**Seeheim Model** The Seeheim model was first introduced by Green in 1985 [70]. It is, like the linguistic model, a logical model, i.e., it does not provide a proposal of how to implement a GUI. It proposes, however, similar to the linguistic model, to separate an application with a GUI into three layers, called the *presentation layer*, the *dialog layer* and the *application layer*.

The presentation layer describes the "static" part of the GUI, i.e., the screens which make up the GUI and the graphical objects as they are composed into screens. The dialog layer models the "dynamic" portion of the GUI, such as the reaction of graphical objects on user input, the invocation of application-specific functionality and controlling the data flow between the presentation layer and the application layer. The application layer represents the functionality that the application implements as an interface to which the dialog layer has access.

We argue that the Seeheim model is closely related to MVC: the presentation layer compares to the view, the dialog layer corresponds to the controller, and the application layer can be mapped to the model. MVC, however, is more general than the Seeheim model. While the Seeheim model describes the separation of applications with GUIs into separate components, MVC can be applied to any computer program that manipulates and presents data. As such we consider the Seeheim model as an instance of MVC. The speech-GUI model can be seen as an architecture proposal for the dialog layer of a Seeheim model instance that describes speech-controlled GUIs.

**Arch Model** The Arch model (Bass et al. [16]) is a specialization of the Seeheim model which introduces mediators between the application layer and the dialog layer, respectively between the dialog layer and the presentation layer. Consequently, instead of three layers, the Arch model consists of five layers, called *component categories*, into which the constituents of a graphical user interface are grouped: *interaction toolkit components*, *presentation components*, *dialog components*, *domain adapter components*, and *domain-specific components*.

Domain-specific components represent domain-specific data or domain-specific functions. They correspond to the application layer of the Seeheim model. Interaction toolkit components implement the (physical) interaction between the computing system and the user, i.e., the communication between input modality and software which captures the input from the input modality. They compare to platform specific graphical object of currently available GUI toolkits (such as Java Swing [132] or MFC [151]) and thus correspond to the presentation layer of the Seeheim model. Dialog components correspond to the dialog layer of the Seeheim model and are responsible for managing function calls

|                  | #Core Items | Synopsis   | Architecture-oriented | Input-oriented |
|------------------|-------------|--|-----------------------|----------------|
| MVC              | 3           | User interface decomposed into a <i>model</i> component, a <i>view</i> component and a <i>controller</i> component; inter-component interaction possible, controller is core interaction coordinator                         | yes                   | no             |
| Linguistic Model | 3           | Data from input modality is abstracted into <i>lexical layer</i> , <i>syntactic layer</i> and <i>semantic layer</i>  | no                    | yes            |
| Interactor Model | 1 (6)       | Abstract <i>interactor</i> models mouse and keyboard interaction with graphical objects; defines 6 concrete interactor for state-of-the-art interactions   | no                    | yes            |
| Seeheim Model    | 3           | User interface decomposed into <i>presentation layer</i> , <i>dialog layer</i> and <i>application layer</i> ; strictly layered interaction   | yes                   | no             |
| Arch Model       | 5           | User interface decomposed into component classes: <i>interaction toolkit components</i> , <i>presentation components</i> , <i>dialog components</i> , <i>domain adapter components</i> and <i>domain-specific components</i> | yes                   | yes            |

Table 2.2: Summary of user interface models

and data flow between domain-specific components and interaction toolkit components. Presentation components act as mediators between interaction toolkit components and dialog components. Compared to the Seeheim model they reside in a layer between the dialog layer and the presentation layer. Presentation components are platform independent abstractions of interaction toolkit components. For instance the presentation object "selector" can abstract the interaction toolkit components "menu" or "radio button". Speaking in terms of MVC the interaction toolkit components represent the view whereas the presentation objects represent the model of a graphical object (refer to the Pluggable-Look-And-Feel framework of Java Swing <sup>11</sup>). Domain adapter components provide a user interface specific abstraction of domain-specific components intended to be accessed by dialog components. For instance, if data that should be presented is distributed over several domain components a specific domain adapter component hides this fact to the dialog components and encapsulates the originally distributed information as one domain adapter object.

The speech-GUI model can be seen as an architecture proposal for the dialog layer of a Seeheim model instance that describes speech-controlled GUIs (see previous paragraph). The Arch model is a specialization of the Seeheim model. Thus, the components of the speech-GUI model can be seen as domain adapter components, dialog components and presentation components.

Table 2.2 summarizes the core features of the user interface models discussed.

<sup>11</sup><http://java.sun.com/products/jl1f/ed2/book/>

# 3

## The Interaction Delay Model

*"Do not worry about your problems with mathematics, I assure you mine are far greater."*

Albert Einstein

### Overview

In this chapter we present the *interaction delay model* which allows for calculating the interaction delay of speech-controlled GUI approaches. The interaction delay, in the context of speech-controlled GUI approaches, consists of the time needed for uttering and processing the spoken commands of an interaction. We begin with discussing the requirements that drive the design of the interaction delay model. Then, we derive the interaction delay model from properties of the speech-GUI model, as presented in chapter 2, such as the number of commands in an interaction, the recognition delay, and the word error rate. We discuss mathematical characteristics and limitations of the interaction delay model. The characteristics suggest, as a direction of future work derived from this dissertation, that calculating the interaction delay could be used as a metric for speech-controlled GUI approaches. In this regard we conclude the chapter with a comparison of the interaction delay model against existing user interface metrics.

## 3.1 Requirements

We discuss the requirements that led the design of the interaction delay model.

**Coverage** The interaction delay model must be applicable to speech-controlled GUI approaches which are based on the speech-GUI model, i.e., the interaction delay model must at least cover the approaches which have been introduced in this work (see section 1.2).

**Quality of Speech Recognition** Today's speech recognition technology is available in the form of components or APIs (e.g. Java Speech API [131], Microsoft's Speech Development Kit [126]). This allows the speech recognizer, which is used for a particular speech-controlled GUI approach implementation, to be exchangeable. Therefore, we require the interaction delay model to consider the quality of speech recognizers, which, according to the speech-GUI model, is measured by the (average) recognition delay and the (average) recognition rate. The recognition delay directly influences the interaction delay, as it denotes the time after which a recognition result becomes available after having stopped speaking. The recognition rate indirectly influences the interaction delay, as rejected or misinterpreted commands requiring the user to utter additional commands.

**Non-empirical Data** A common issue with evaluating user interfaces is the necessity of performing an empirical experiment to obtain data to which a metric can be applied (we discuss some of these metrics later on in section 3.4.2). Empirical experiments are costly as they consume time to be designed and are, to a certain extent, not reproducible. To avoid the need for empirically gathered data it must be possible to calculate the interaction delay from quantitative and qualitative properties of the speech-GUI model.

## 3.2 Interaction Delay

This section is partitioned into three parts. In the first part we provide a definition of interaction delay based on properties of the speech-GUI model (section 3.2.1). In the second part we derive an initial iteration of the interaction delay model, called the *nominal interaction delay*, which allows for calculating the interaction delay without considering the implications of recognition errors (section 3.2.2). In the third part we examine the implications of recognition errors. We integrate our findings into the nominal interaction delay which results in an enhanced iteration of the interaction delay model which we call the *expected interaction delay* (section 3.2.3).

### 3.2.1 Definition

The interaction delay of a session is the time needed for uttering and processing the commands necessary to invoke all speech functions in the corresponding intention. From the speech-GUI model we conclude that the interaction delay is comprised of



1. the duration of each command contained in the interaction,
2. the recognition delay that occurs for the recognition of each command in the interaction, and
3. the time that the recognition result interpreter needs to determine a match for each respective recognition result, i.e., the time that is needed to interpret a recognition result.

The speech-GUI model defines the duration of a command and the recognition delay as properties of commands, speech recognizers, respectively. The speech-GUI model specifies a recognition delay of 1.5 seconds (as derived in section 2.5.1). The average duration of a command spoken in the English language is also in the magnitude of seconds (as discussed in section 2.4). Regarding the time for interpreting a recognition result we have conducted an empirical study of which we present a summary here. For more details refer to appendix B.1.

The goal of this study was to show that the time to interpret a recognition result does not carry weight in comparison to the duration of commands and the recognition delay. We based our study on three conclusions which we have drawn from examining the various research papers which have been published for the speech-controlled GUI approaches (see discussion in section 1.2). First, we conclude that a speech recognizer used for a specific speech-controlled GUI approach needs a per-screen vocabulary with a maximum size of maximal 70 words. Second, we conclude that a specific speech-controlled GUI approach defines a maximal number of roughly 8000 valid commands per screen over the speech recognizers vocabulary. It is important to note that the size of the vocabulary and the number of valid commands can (dynamically) be minimized specifically to the current screen, which improves the word error rate and which reduces interpretation complexity (Smailagic and Siewiorek [175]). Since we do not consider this possibility because the screen is a priori unknown, we examine the worst case. Third, we conclude that the valid commands of a specific speech-controlled GUI approach have a length  $\leq 5$ . The speech-GUI model defines a maximum word error rate of 100% (see section 2.5), therefore a recognition result, induced by a command of length 5, has a maximum length of 10 if all occurring word errors are insertions.

For the study we created a Java program which generated 8000 random word sequences with a length of 5 from a vocabulary of 70 words. These commands represented the valid commands; as such we examined the worst case where all valid commands have a length of 5. The valid commands were stored in a hash table using the respective command itself as the key. After that we generated 2,000,000 random commands from the same vocabulary with a maximum length of 10 words to simulate recognition results, and looked them up in the hash table. If the hash table contained the newly generated command we interpreted this as a match, otherwise the newly generated command was rejected.

The study revealed that the interpretation of a command can be done in an average time of  $3 \cdot 10^{-4}$ ms – as such, it is by a factor of  $10^7$  lower than the recognition delay and the duration of a command. We argue that the interpretation time can consequently be neglected for the remainder of this dissertation – no user will notice if the user interface reacted  $3 \cdot 10^{-4}$ ms faster or slower.

### 3.2.2 Nominal Interaction Delay

By  $d_c$  we represent the duration of the command  $C$  (as defined in the speech-GUI model, section 2.4), which lets us provide an initial estimate for the interaction delay. The estimate calculates the interaction delay by walking through all commands in an interaction, summing up their durations and the occurring recognition delays.

**Definition 1** *Nominal Interaction Delay*

Let  $\mathcal{S}$  be a session. Let  $n \in \mathbb{N}$ , let  $(C_x)_{x=1,\dots,n}$  be the sequence of commands in the interaction corresponding to  $\mathcal{S}$  and let  $r \in \mathbb{R}^+$  be the recognition delay. We define

$$\Delta_{nom}(\mathcal{S}) := \sum_{x=1}^n (d_{C_x} + r) = n \cdot r + \sum_{x=1}^n d_{C_x}$$

We call  $\Delta_{nom}(\mathcal{S})$  the nominal interaction delay.

If a recognition error occurs the recognition result interpreter either rejects the recognition result or a misunderstanding occurs. Both rejection and misunderstanding have the effect that the user has to utter commands which have not been included in the nominal interaction delay. In the case of a rejection the recognition result interpreter does not invoke any speech function (refer to the definition of the recognition result interpreter in section 2.6). We assume that in this case the user repeats the command which was rejected<sup>1</sup>. In the case of a misunderstanding the recognition result interpreter invokes a speech function which was not intended by the user. Consequently, the user has to undo the invocation of the incorrect speech function and then has to repeat the command which was misunderstood. For this we assume that every speech-controlled GUI provides a speech function called *SFUNDO*, which undoes the most recently invoked speech function<sup>2</sup>. To summarize, if a recognition error occurs the user has to utter additional commands. We call these additional commands *corrections*. As explained above there is one correction per rejection (the repetition), and two corrections per misunderstanding (the undo followed by the repetition).

If a recognition error occurs during the recognition of a correction the recognition result interpreter either rejects or misunderstands the correction – and the user, recursively, needs to react as above: if the correction was rejected it needs to be repeated. Otherwise, if the correction was misunderstood, the user needs to undo it, and then perform the repetition of the correction. It becomes clear that a single recognition error can lead to an entire cascade of corrections of which the duration and the recognition delay significantly increase the interaction delay.

---

<sup>1</sup>The user might as well try to reformulate the command, however, this process depends on the specific user and there is no known model about this process. Developing such a model is out of the scope of this dissertation.

<sup>2</sup>Please note that *SFUNDO* is a theoretical component in our model. It should be seen as a proposal for future implementations of speech-controlled GUIs. It is up to the speech-controlled GUI approach designer if *SFUNDO* is provided. It further depends on the application domain if the invocation of speech functions can be undone at all, e.g., if the speech function which was misunderstood triggered an atomic transaction.

We call a command that is contained in an interaction an *intended command*. Corrections, which are caused by recognition errors during the recognition of an intended command, are called *level-1* corrections. Correspondingly, we call corrections which are caused by recognition errors during the recognition of a level-1 correction *level-2* corrections – and so on. In the following section we will provide an estimate of the interaction delay under consideration of the additional delay induced by corrections.

### 3.2.3 Expected Interaction Delay

The *expected interaction* delay denotes the interaction delay of a session under consideration of the duration and recognition delay induced by corrections. In the following we derive a formula for the expected interaction delay in four steps. In the first step we approximate the probabilities for rejections and misunderstandings. These probabilities are crucial for determining the expected number of corrections, because a rejection implies one correction (the repetition) whereas a misunderstanding implies two corrections (the undo and the repetition). In the second step we will estimate the expected number of level-1 corrections. Generalizing the results from the second step, we estimate the totally expected number of corrections in step three. In step four we finally arrive at a formula for the expected interaction delay.

#### Step 1: Probabilities of Rejection and Misunderstanding

Let  $\mathcal{V}$  be a set of words which represents the vocabulary of a speech recognizer. Let  $\mathcal{CMD}$  denote the set of commands which can be constructed from  $\mathcal{V}$ , i.e.,  $\mathcal{CMD}$  contains all word sequences which can be constructed from words in  $\mathcal{V}$ . As such,  $\mathcal{CMD}$  contains all possible recognition results. Let  $\mathcal{CMD}_{vld} \subseteq \mathcal{CMD}$  represent the valid commands for a specific speech-controlled GUI approach which is based on the vocabulary  $\mathcal{V}$ .

With the just defined terminology we aim at providing an estimate for the probability of a rejection, which happens, if a recognition error does not match a valid command. Since we modeled a speech recognizer as a black box component we do not know about the characteristics of the recognition process. We consequently assume that a recognition error can match any command with the same probability – be it valid or invalid. The number of invalid commands is determined by  $|\mathcal{CMD}| - |\mathcal{CMD}_{vld}|$ . Therefore, with the above assumption, we estimate the probability of a rejection, under the precondition that a recognition error has happened, by the term <sup>3</sup>

$$\frac{|\mathcal{CMD}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}|}$$

The issue hereby is that  $\mathcal{CMD}$  is an infinite set: given an arbitrary finite set (such as  $\mathcal{V}$ ) an infinite number of sequences of elements from that set can be constructed. Consequently, since  $\mathcal{CMD}_{vld}$  is finite (see section 2.4), the probability of a rejection under the precondition that a

<sup>3</sup>If we were exact we would estimate the denominator of the term as  $|\mathcal{CMD}| - 1$ , since one command would have been correct. We do, however, neglect the correct command to simplify further estimations.

recognition error happened would be 1, because

$$\lim_{|\mathcal{CMD}| \rightarrow \infty} \frac{|\mathcal{CMD}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}|} = 1 \quad (3.1)$$

The correctness of (3.1) is discussed in appendix A.1. Equation (3.1) would mean that our model expects any recognition error to be rejected and no recognition error to be misunderstood. For this behavior to be practical the speech recognizer would have to have the ability to mark a recognition result as a recognition error, so that the recognition result interpreter can decide to reject it. This is, however, not possible: a recognition result is the word sequence which the speech recognizer believes to match best the characteristics of the incoming audio signal according to its internal models. With currently available speech recognition technology there is always a specific inherent probability of this match being incorrect, i.e., this probability is induced by the algorithms of speech recognition process (as discussed in section 1.2.1). Therefore, the speech recognizer cannot determine if the recognition result is a recognition error or not. And consequently, the recognition result interpreter cannot determine which recognition results should be rejected before the interpretation process starts.<sup>4</sup>

In order to provide a more realistic approximation of the probability of a rejection we need to consider a suitable finite subset of  $\mathcal{CMD}$ . We arrive at this subset by reconsidering a constraint in the speech-GUI model that was introduced in section 2.5: the valid input of a speech recognizer is constrained to the valid commands of the respective speech-controlled GUI approach, i.e., constrained to  $\mathcal{CMD}_{vld}$ . Let  $n_{max}$  be the length of the longest valid command, i.e.,  $n_{max}$  is the length of the longest word sequence in  $\mathcal{CMD}_{vld}$ . As discussed in section 3.2.1, given the maximum error rate of 100%, the longest recognition result has a length of  $2 \cdot n_{max}$ . As such, we will only need to consider elements of  $\mathcal{CMD}$  which have a maximum length of  $2 \cdot n_{max}$ . We represent this subset by  $\mathcal{CMD}_{2n_{max}}$  and obviously  $\mathcal{CMD}_{vld} \subseteq \mathcal{CMD}_{2n_{max}}$ . The number of elements in  $\mathcal{CMD}_{2n_{max}}$  is finite and can be calculated as follows:

$$\begin{aligned} |\mathcal{CMD}_{2n_{max}}| &:= \underbrace{|\mathcal{V}|}_{\#seq. \text{ of length } 1} + \underbrace{|\mathcal{V}|^2}_{\#seq. \text{ of length } 2} + \underbrace{|\mathcal{V}|^3}_{\#seq. \text{ of length } 3} + \dots + \underbrace{|\mathcal{V}|^{2 \cdot n_{max}}}_{\#seq. \text{ of length } 2 \cdot n_{max}} \\ &= \sum_{x=1}^{2 \cdot n_{max}} (|\mathcal{V}|)^x = \begin{cases} \frac{|\mathcal{V}|^{2 \cdot n_{max} + 1} - 1}{|\mathcal{V}| - 1} - 1, & \text{for } |\mathcal{V}| \neq 1 \\ 2 \cdot n_{max}, & \text{for } |\mathcal{V}| = 1 \end{cases} \end{aligned} \quad (3.2)$$

Equation (3.2) is a geometric series of the form  $\sum_{k=1}^n x^k = \frac{x^{n+1} - 1}{x - 1} - 1$ , as we derive in section A.3. We are now able to approximate the probability of a rejection ( $Rej$ ), under the precondi-

---

<sup>4</sup>Current speech recognizers might provide a so called confidence value which denotes "how confident" the speech recognizer is about the recognition result really matching what was uttered. As such, the recognition result interpreter could reject any recognition result below a specific confidence threshold, however, the confidence is not included in the speech-GUI model (as discussed in section 2.8.2).

tion that a recognition error happened (*RecoError*), as

$$P(Rej|RecoError) = \frac{|\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}|}$$

$P(Rej|RecoError)$  is well defined, because

$$\begin{aligned} \mathcal{CMD}_{vld} \subseteq \mathcal{CMD}_{2n_{max}} &\Rightarrow |\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}| \geq 0 \\ \mathcal{CMD}_{vld} \subseteq \mathcal{CMD}_{2n_{max}} &\Rightarrow |\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}| \leq |\mathcal{CMD}_{2n_{max}}| \\ |\mathcal{V}| > 0 \text{ (per definition, section 2.5)} &\Rightarrow |\mathcal{CMD}_{2n_{max}}| > 0 \\ &\Rightarrow \\ &0 \leq P(Rej|RecoError) \leq 1 \end{aligned}$$

We consider it important to note that the above estimation of  $P(Rej|RecoError)$  and  $P(Mis|RecoError)$  is unrealistic, if we allowed that  $\mathcal{CMD}_{vld}$ , i.e., the valid commands, would contain a small number of extremely long commands. In that case,  $n_{max}$  would be extremely high, thus,  $|\mathcal{CMD}_{2n_{max}}|$  would also be extremely high and again, similar to (3.1),  $P(Rej|RecoError)$  would move towards 1 and  $P(Mis|RecoError)$  would move towards 0. However, as we will show in chapter 4, our estimation is reasonable from a practical perspective, where  $n_{max}$  is reasonably small ( $\leq 5$ ).

A recognition error is either rejected or misunderstood; therefore rejection and misunderstanding, under the condition that a recognition error happened, are mutually exclusive events. Consequently, the probability of a misunderstanding under the condition that a recognition error happened calculates as

$$P(Mis|RecoError) = 1 - P(Rej|RecoError)$$

If no recognition error happened ( $\overline{RecoError}$ ) then no command is rejected or misunderstood, therefore  $P(Rej|\overline{RecoError}) = P(Mis|\overline{RecoError}) = 0$ . Under consideration of the *total probability theorem* for two mutually exclusive events  $A$  and  $B$  (Papoulis [146]), which is given below

$$P(A) = P(A|B) \cdot P(B) + P(A|\overline{B}) \cdot P(\overline{B})$$

we calculate

$$\begin{aligned} P(Rej) &= P(Rej|RecoError) \cdot P(RecoError) + P(Rej|\overline{RecoError}) \cdot P(\overline{RecoError}) \\ &= P(Rej|RecoError) \cdot P(RecoError) \end{aligned}$$

and

$$P(Mis) = P(Mis|RecoError) \cdot P(RecoError) + P(Mis|\overline{RecoError}) \cdot P(\overline{RecoError})$$

$$\begin{aligned}
&= P(Mis|RecoError) \cdot P(RecoError) \\
&= (1 - P(Rej|RecoError)) \cdot P(RecoError)
\end{aligned}$$

What is missing to be able to calculate  $P(Mis)$  and  $P(Rej)$ , according to the above formulas, is  $P(RecoError)$ . It denotes the probability that a recognition error happens during the recognition of a command. The speech-GUI model does not define  $P(RecoError)$ , however, it defines the average word error rate of a speech recognizer as the average rate by which word errors occur per spoken word. We represent the word error rate by  $w$ . The probability of a word being recognized correctly, i.e., no word error happens, is consequently  $1 - w$ . As such,  $1 - w$  denotes the probability of a command of length 1 being recognized correctly. In section 2.5.2 we modeled the word error rate as being equal for each word in the speech recognizer's vocabulary. Consequently, the probability of a command of length 2 being correctly recognized is  $(1 - w)^2$ . The probability of a command of length 3 being correctly recognized is  $(1 - w)^3$ . And so on.

As we have pointed out in section 2.5 the word error rate, according to its actual definition, can be greater than 100%, in which case this would mean that  $1 - w < 0$ . However, we have restricted the word error rate to be between 0 and 1 (inclusively) – which means that  $0 \leq 1 - w \leq 1$ . We furthermore defined, that not more than one word error can happen per word. What still can happen under these conditions, is, that two word errors of different words neutralize themselves, e.g., an insertion followed by a deletion. As this would still result in a correctly recognized command, we neglect this case in the following. With these considerations we provide the following definition of the command recognition rate.

**Definition 2** *Command Recognition Rate*

Let  $w \in [0; 1]$  be the word error rate of a speech recognizer and let  $C$  be a command of length  $n$  ( $n \in \mathbb{N}, n > 0$ ). The command recognition rate,  $\mathcal{R}_C$ , is defined as

$$\mathcal{R}_C := (1 - w)^n$$

The definition of the command recognition rate is idealized – we assume that the probability for the successful recognition of a word within the command does not depend on which words occur before that word. In other words, we assume that the probability for recognizing a word  $A$  and then, in a different recognition process, the word  $B$ , is stochastically independent of the probability of recognizing the word sequence  $AB$  in a single recognition process. In reality this is not true because, as explained in section 1.2.1, speech recognizers rely on a language model from which they conclude which word sequences are or are not likely to occur. Therefore, the recognition rate of single words  $A$  and  $B$ , which are essentially commands of length 1, is in general different from the recognition rate of  $AB$ . Furthermore, Thus, our definition of command recognition rate is an approximation which we make for the sake of simplifying further calculations. We argue that an exact calculation would require knowledge about the language model and would have to be performed with conditional probabilities – which would make a formal analysis of different speech-controlled GUI approaches very cumbersome and

time consuming <sup>5</sup>.

With the definition of the command recognition rate we can calculate  $P(RecoError)$  for a command  $\mathcal{C}$  as follows:

$$P(RecoError) = 1 - \mathcal{R}_{\mathcal{C}}, \text{ for a command } \mathcal{C}$$

We arrive at formulae for the probabilities of rejection and misunderstanding of  $\mathcal{C}$ :

$$\begin{aligned} P(Mis) &= (1 - P(Rej|RecoError)) \cdot P(RecoError) \\ &= \left(1 - \frac{|\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}|}\right) \cdot (1 - \mathcal{R}_{\mathcal{C}}) \end{aligned}$$

and

$$\begin{aligned} P(Rej) &= P(Rej|RecoError) \cdot P(RecoError) \\ &= \frac{|\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}|} \cdot (1 - \mathcal{R}_{\mathcal{C}}) \end{aligned}$$

According to (3.2)  $|\mathcal{CMD}_{2n_{max}}| > 0$ . The probabilities for rejection and misunderstanding are the basis for estimating the number of level-1 corrections in the next step.

### Step 2: Expected Number of Level-1 Corrections

We estimate the expected number of level-1 corrections for a specific command  $\mathcal{C}$ . Let  $\mathcal{N}_{\mathcal{C},1}$  be a random variable that denotes the number of level-1 corrections for  $\mathcal{C}$  with  $P(\mathcal{N}_{\mathcal{C},1} = x)$  denoting the probability that  $\mathcal{N}_{\mathcal{C},1}$  has the value  $x$ . We expect no level-1 corrections if no recognition error occurs, one, if a rejection occurs and two, if a misunderstanding occurs. Consequently,  $x \in \{0, 1, 2\}$ . This allows us to specify the probability distribution of  $\mathcal{N}_{\mathcal{C},1}$  as follows.

$$P(\mathcal{N}_{\mathcal{C},1} = 0) = \mathcal{R}_{\mathcal{C}} \text{ (command recognized correctly)} \quad (3.3)$$

$$P(\mathcal{N}_{\mathcal{C},1} = 1) = P(Rej) \text{ (command rejected)} \quad (3.4)$$

$$P(\mathcal{N}_{\mathcal{C},1} = 2) = P(Mis) \text{ (command misunderstood)} \quad (3.5)$$

Correspondingly, the expected amount of level-1 corrections for  $\mathcal{C}$  is the expectation of  $\mathcal{N}_{\mathcal{C},1}$ . Let  $l_{\mathcal{C}}$  denote the length of the command  $\mathcal{C}$ , then the expectation of  $\mathcal{N}_{\mathcal{C},1}$ ,  $E(\mathcal{N}_{\mathcal{C},1})$ , calculates as follows:

$$\begin{aligned} E(\mathcal{N}_{\mathcal{C},1}) &= 0 \cdot P(\mathcal{N}_{\mathcal{C},1} = 0) + 1 \cdot P(\mathcal{N}_{\mathcal{C},1} = 1) + 2 \cdot P(\mathcal{N}_{\mathcal{C},1} = 2) \\ &= P(Rej) + 2 \cdot P(Mis) \end{aligned} \quad (3.6)$$

---

<sup>5</sup>Provided that these conditional probabilities could a priori be determined at all, since the recognition process is affected by external factors, such as a priori unknown environmental noise.

$$\begin{aligned}
 &= P(\text{Rej}|\text{RecoError}) \cdot P(\text{RecoError}) + 2 \cdot (1 - P(\text{Rej}|\text{RecoError})) \cdot P(\text{RecoError}) \\
 &= P(\text{Rej}|\text{RecoError}) \cdot P(\text{RecoError}) + (2 - 2P(\text{Rej}|\text{RecoError})) \cdot P(\text{RecoError}) \\
 &= 2P(\text{RecoError}) - P(\text{Rej}|\text{RecoError}) \cdot P(\text{RecoError}) \\
 &= 2P(\text{RecoError}) - P(\text{Rej}) \\
 &= 2(1 - \mathcal{R}_C) - \frac{|\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}|} \cdot (1 - \mathcal{R}_C) \\
 &= (1 - \mathcal{R}_C) \cdot \left( 2 - \frac{|\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}|} \right) \tag{3.7}
 \end{aligned}$$

$$= (1 - (1 - w)^{l_C}) \cdot \left( 2 - \frac{|\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}|} \right) \tag{3.8}$$

According to (3.2)  $|\mathcal{CMD}_{2n_{max}}| > 0$ . To summarize the results of this step, if a user utters a specific command  $\mathcal{C}$ , we expect an average of  $E(\mathcal{N}_{C,1})$  level-1 corrections.

### Step 3: Totally Excepted Number of Corrections

In this step we derive a formula for the totally expected number of corrections by identifying a common pattern during the derivation of expected level-2 and level-3 corrections.

The expected number of level-2 corrections for  $\mathcal{C}$  depends on how many level-1 corrections are expected to be recognized incorrectly. The command recognition rate of a level-1 command that was induced by a rejection of  $\mathcal{C}$  is equal to the command recognition rate of  $\mathcal{C}$  - as in that case the level-1 command is a repetition of  $\mathcal{C}$ . The situation is different for a level-1 command that was induced by a misunderstanding of  $\mathcal{C}$ , as the level-1 command could either be the command that invokes the undo function or the command that repeats  $\mathcal{C}$ . In the latter case, the repetition of  $\mathcal{C}$ , the command recognition rate of the level-1 command is equal to the command recognition rate of  $\mathcal{C}$ . In the former case, the command that invokes the undo function, the command recognition rate is a priori unknown as we have not specified the command for the undo function. However, in order to simplify further calculations we approximate the command recognition rate of the command that invokes the undo function by the command recognition rate of  $\mathcal{C}$ .

With the above considerations we may assume that level-1 corrections are recognized incorrectly with the same probability than the initial command  $\mathcal{C}$ , because two subsequent recognition processes are stochastically independent – unless the speech recognizer implements



machine learning algorithms which we neglect. This means that with a probability of  $P(Rej)$  a level-1 command will be rejected and with a probability of  $P(Mis)$  a level-1 command will be misunderstood. Analogously to a recognition error occurring with a level-1 correction the user needs to utter one level-2 correction for every level-1 correction which is rejected, and two level-2 corrections for each level-1 correction which is misunderstood. This allows us to estimate the number of level-2 corrections by

$$\begin{aligned} & \underbrace{\frac{E(\mathcal{N}_{\mathcal{C},1})}{\#exp. \text{ l.-1 corr. cmds.}} \cdot P(Rej) \cdot 1}_{\#l.-1 \text{ corr. cmds. rejected}} + \underbrace{\frac{E(\mathcal{N}_{\mathcal{C},1})}{\#exp. \text{ l.-1 corr. cmds.}} \cdot P(Mis) \cdot 2}_{\#l.-1 \text{ corr. cmds. misunderstood}} \\ &= E(\mathcal{N}_{\mathcal{C},1}) \cdot (P(Rej) + 2 \cdot P(Mis)) = \overset{6}{E(\mathcal{N}_{\mathcal{C},1})^2} \end{aligned}$$

which makes an expected number of level-1 and level-2 corrections for  $\mathcal{C}$  of

$$\underbrace{E(\mathcal{N}_{\mathcal{C},1})}_{\#expected \text{ level-1 corr. cmds.}} + \underbrace{E(\mathcal{N}_{\mathcal{C},1})^2}_{\#expected \text{ level-2 corr. cmds.}}$$

With a probability of  $P(Rej)$  a level-2 command will be rejected, which requires the user to utter one level-3 correction. With a probability of  $P(Mis)$  a level-2 command will be misunderstood, which requires the user to utter two level-3 corrections. Accordingly, we estimate the number of level-3 corrections by

$$\begin{aligned} & \underbrace{\frac{E(\mathcal{N}_{\mathcal{C},1})^2}{\#exp. \text{ l.-2 corr. cmds.}} \cdot P(Rej) \cdot 1}_{\#l.-2 \text{ corr. cmds. rejected}} + \underbrace{\frac{E(\mathcal{N}_{\mathcal{C},1})^2}{\#exp. \text{ l.-2 corr. cmds.}} \cdot P(Mis) \cdot 2}_{\#l.-2 \text{ corr. cmds. misunderstood}} \\ &= E(\mathcal{N}_{\mathcal{C},1})^2 \cdot (P(Rej) + 2 \cdot P(Mis)) \\ &= E(\mathcal{N}_{\mathcal{C},1})^3 \end{aligned}$$

which makes an expected number of level-1, level-2 and level-3 corrections of

$$\underbrace{E(\mathcal{N}_{\mathcal{C},1})}_{\#expected \text{ level-1 corr. cmds.}} + \underbrace{E(\mathcal{N}_{\mathcal{C},1})^2}_{\#expected \text{ level-2 corr. cmds.}} + \underbrace{E(\mathcal{N}_{\mathcal{C},1})^3}_{\#expected \text{ level-3 corr. cmds.}}$$

Following this pattern the total number of corrections for a specific command  $\mathcal{C}$  can be calculated as

---

<sup>6</sup> $P(Rej) + 2 \cdot P(Mis) = E(\mathcal{N}_{\mathcal{C},1})$ , refer to (3.6) on page 55

$$\begin{aligned}
 & \sum_{x=1}^{\infty} E(\mathcal{N}_{\mathcal{C},1})^x \\
 &= E(\mathcal{N}_{\mathcal{C},1})^1 + E(\mathcal{N}_{\mathcal{C},1})^2 + E(\mathcal{N}_{\mathcal{C},1})^3 + \dots \\
 &= -E(\mathcal{N}_{\mathcal{C},1})^0 + E(\mathcal{N}_{\mathcal{C},1})^0 + E(\mathcal{N}_{\mathcal{C},1})^1 + E(\mathcal{N}_{\mathcal{C},1})^2 + E(\mathcal{N}_{\mathcal{C},1})^3 + \dots \\
 &= -E(\mathcal{N}_{\mathcal{C},1})^0 + \sum_{x=0}^{\infty} E(\mathcal{N}_{\mathcal{C},1})^x \\
 &= \left( \sum_{x=0}^{\infty} E(\mathcal{N}_{\mathcal{C},1})^x \right) - 1
 \end{aligned}$$

The above formula contains the infinite geometric series  $\sum_{x=0}^{\infty} E(\mathcal{N}_{\mathcal{C},1})^x$ . As  $E(\mathcal{N}_{\mathcal{C},1}) \geq 0$  the following conditions are satisfied (Forster [60]):

$$E(\mathcal{N}_{\mathcal{C},1}) < 1 \Rightarrow \sum_{x=0}^{\infty} E(\mathcal{N}_{\mathcal{C},1})^x = \frac{1}{1 - E(\mathcal{N}_{\mathcal{C},1})} - 1 \quad (3.9)$$

$$E(\mathcal{N}_{\mathcal{C},1}) \geq 1 \Rightarrow \lim_{x \rightarrow \infty} \sum_{x=0}^{\infty} E(\mathcal{N}_{\mathcal{C},1})^x = \infty \quad (3.10)$$

From (3.9) we conclude that, if  $E(\mathcal{N}_{\mathcal{C},1}) < 1$ , we can calculate the totally expected number of corrections as a function against  $E(\mathcal{N}_{\mathcal{C},1})$ . From (3.10) we conclude that, if  $E(\mathcal{N}_{\mathcal{C},1}) \geq 1$ , i.e., if for command  $\mathcal{C}$  one or more level-1 corrections have to be expected in average, the total number of expected corrections is infinite. This would mean that the user never finishes the interaction in which  $\mathcal{C}$  is involved. This special case is examined in more detail later on in section 3.3.1. Given the above considerations we arrive at a definition for the totally expected number of corrections for a specific command as the outcome of this step.

**Definition 3** *Totally Excepted Number of Corrections*

We call  $n_{corr}(\mathcal{C})$  the totally expected number of corrections for a command  $\mathcal{C}$  and define  $n_{corr}(\mathcal{C})$  as follows:

$$E(\mathcal{N}_{\mathcal{C},1}) < 1 \Rightarrow n_{corr}(\mathcal{C}) := \frac{1}{1 - E(\mathcal{N}_{\mathcal{C},1})} - 1$$

$$E(\mathcal{N}_{\mathcal{C},1}) \geq 1 \Rightarrow n_{corr}(\mathcal{C}) := \infty$$

#### Step 4: Expected Interaction Delay

We now arrive at a formula for the interaction delay which considers the additional delay that is introduced by corrections. We call this formula the *expected interaction delay*, as it, additionally to the duration and the recognition delay of each command, sum up the duration and the recognition delay for each expected correction.

#### Definition 4 Expected Interaction Delay

Let  $\mathcal{S}$  be a session. Let  $n \in \mathbb{N}$ , let  $(\mathcal{C}_x)_{x=1,\dots,n}$  be the sequence of commands in the interaction corresponding to  $\mathcal{S}$  and let  $r \in \mathbb{R}^+$  be the recognition delay. We define

$$\Delta_{eff}(\mathcal{S}) := \sum_{x=1}^n (d_{\mathcal{C}_x} + r + n_{corr}(\mathcal{C}_x)(d_{\mathcal{C}_x} + r))$$

We call  $\Delta_{eff}$  the *expected interaction delay*.

In Theorem 1 we show that for a word error rate of 0%, i.e., a command recognition rate of 100%, the expected interaction delay is equal to the nominal interaction delay. Hence, we show that the nominal interaction delay is a special case of the expected interaction delay.

**Theorem 1** Let  $w \in \mathbb{R}$  be a word error rate and let  $\mathcal{S}$  be a session. Let  $n \in \mathbb{N}$  and let  $(\mathcal{C}_x)_{x=1,\dots,n}$  be the sequence of commands in the interaction corresponding to  $\mathcal{S}$ , then

$$w = 0 \Rightarrow \Delta_{nom}(\mathcal{S}) = \Delta_{eff}(\mathcal{S})$$

**Proof** From the definitions of  $\Delta_{nom}$  and  $\Delta_{eff}$  we conclude that the claim is fulfilled if

$$w = 0 \Rightarrow \forall \mathcal{C}_x : E(\mathcal{N}_{\mathcal{C}_x,1}) = 0$$

We show this in the following. Let  $x \in \{1, 2, 3, \dots, n\}$  and let  $w = 0$ . According to (3.8) we write

$$\begin{aligned} E(\mathcal{N}_{\mathcal{C}_x,1}) &= (1 - (1 - 0)^{l_c}) \cdot \left( 2 - \frac{|\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}|} \right) \\ &= 0 \cdot \left( 2 - \frac{|\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}|} \right) = 0 \end{aligned}$$

◇

As a consequence of Theorem 1 we use the term *interaction delay* as a synonym for the expected interaction delay. Thus, the term interaction delay model will in the following refer to the formula for the expected interaction delay. We symbolize the interaction delay of a session by  $\Lambda(\mathcal{S})$  and discuss its mathematical characteristics in the following section.

### 3.3 Characteristics of the Interaction Delay

We now point out several mathematical characteristics of the interaction delay model. We will use these characteristics later in chapter 4 to interpret the results of interaction delay calculations. Given the characteristics we hypothesize that the interaction delay model could be utilized as metric for speech-controlled GUIs, possibly for user interfaces in general. We do, however, not further investigate this hypothesis in this dissertation, but regard it as one possible direction of future work (refer to section 7.3).

#### 3.3.1 Critical Word Error Rate

Depending on whether the value for  $E(\mathcal{N}_{C,1})$ , which is the essential parameter of the interaction delay, is  $< 1$  or  $\geq 1$ , an infinite amount of corrections has to be expected. We examine under which conditions  $E(\mathcal{N}_{C,1})$  can get  $\geq 1$ , which leads to the definition of the *critical word error rate*. According to (3.7) we write

$$E(\mathcal{N}_{C,1}) \geq 1 \Leftrightarrow (1 - \mathcal{R}_C) \cdot \left(2 - \frac{|\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}|}\right) \geq 1$$

As  $|\mathcal{CMD}_{vld}| > 0$  and  $|\mathcal{CMD}_{2n_{max}}| > 0$  per definition, the equation is satisfied for

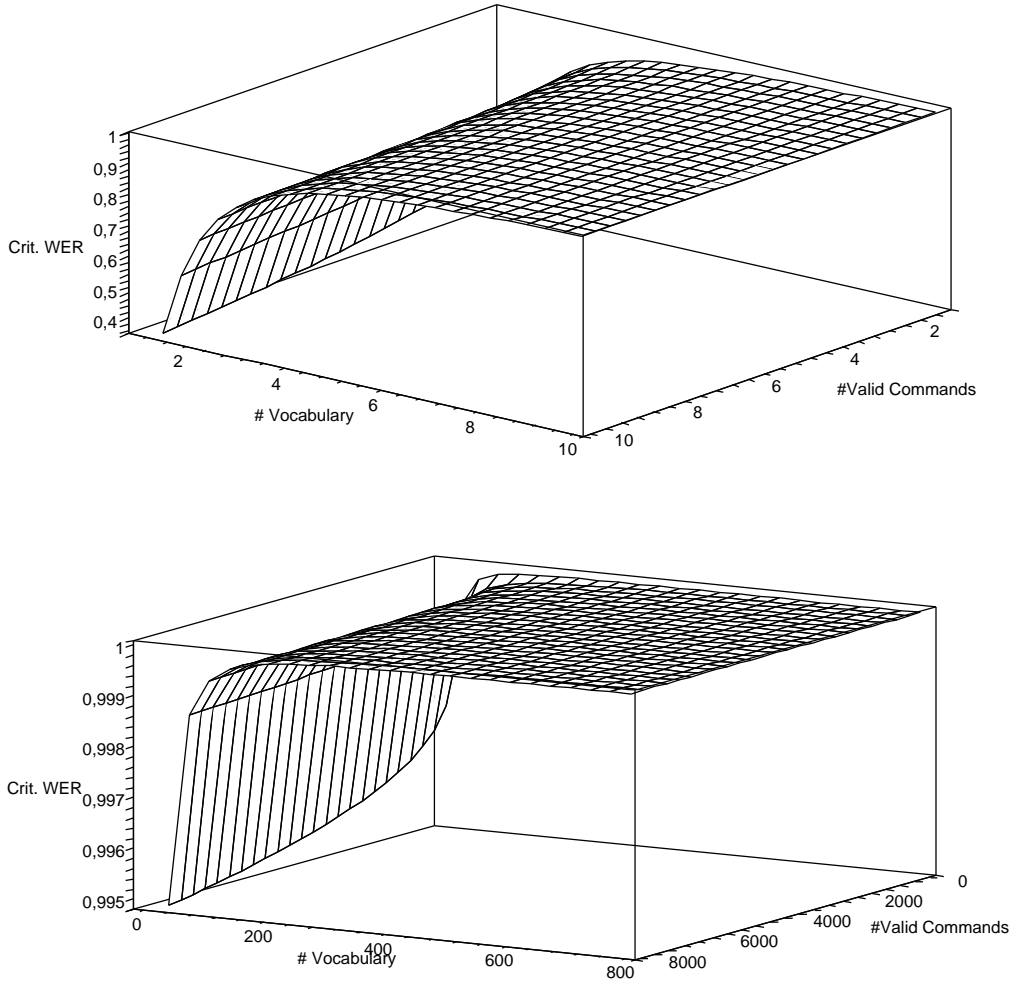
$$\mathcal{R}_C \leq \frac{|\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}| + |\mathcal{CMD}_{vld}|} \quad (3.11)$$

The derivation of (3.11) is discussed in appendix A.2. In order to count on a finite number of corrections for all valid commands it needs to be ensured that the inequality (3.11) holds for any command  $\mathcal{C}$ . Since the command recognition rate decreases with increasing command length, it is sufficient to ensure that (3.11) holds for command with length  $n_{max}$ . We call the recognition rate of the longest command(s)  $\mathcal{R}_{crit}$ . With (3.11) the following condition for  $\mathcal{R}_{crit}$  must be satisfied:

$$\begin{aligned} \mathcal{R}_{crit} &> \frac{|\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}| + |\mathcal{CMD}_{vld}|} \Leftrightarrow \\ (1 - w)^{n_{max}} &> \frac{|\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}| + |\mathcal{CMD}_{vld}|} \Leftrightarrow \\ w &< 1 - \left( \frac{|\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{n_{max}}| + |\mathcal{CMD}_{vld}|} \right)^{\frac{1}{2n_{max}}} := w_{crit} \end{aligned} \quad (3.12)$$

The derivation of (3.12) is discussed in appendix A.2. We call  $w_{crit}$  the *critical word error rate*. If the word error rate of a speech recognizer reaches or rises above  $w_{crit}$  then the user has to expect an infinite number of corrections depending on which specific command is recognized

### 3.3. CHARACTERISTICS OF THE INTERACTION DELAY



**Figure 3.1:** 3D plots of critical word error rate.

incorrectly. As  $|\mathcal{V}| > 1$  (section 2.5) we use equation (3.2) on page 52 to expand  $|\mathcal{CMD}_{2n_{max}}|$  which lets us arrive at

$$w_{crit} = 1 - \left( \frac{|\mathcal{CMD}_{vld}|}{\frac{|\mathcal{V}|^{2 \cdot n_{max} + 1} - 1}{|\mathcal{V}| - 1} - 1 + |\mathcal{CMD}_{vld}|} \right)^{\frac{1}{n_{max}}}$$

Figure 3.1 shows plots of the critical word error rate against the size of the vocabulary and the number of valid commands for  $n_{max} = 5$ . The plots for  $n_{max} < 5$  are not significantly different. From the plots we recognize, that vocabulary sizes above 10 words, the critical recognition rate is above 90%. Since speech-controlled GUIs typically have far more than 10 words in the vocabulary, (refer to the considerations in section 3.2.1) we argue that with a word

error rate above 90% the user would have given up interacting with the speech-controlled GUI long before noticing, that the respective task cannot be completed. The critical word error rate is therefore a theoretical concept allowing us to simplify mathematical proofs of the now following theorems.

### 3.3.2 Word Error Rate-Monotony

A low word rate results in a low value for the interaction delay and vice versa. We call this property the *word error rate-monotony* of the interaction delay, expressed in Theorem 2.

**Theorem 2** *Let  $w_1, w_2 \in [0; 1]$  be the word error rates of two different speech recognizers and let  $\mathcal{S}$  be a session. Both  $w_1$  and  $w_2$  are below the critical word error rate of the respective speech-controlled GUI approach. Let  $\Lambda(\mathcal{S})_{w_1}$  denote the interaction delay using the speech recognizer with the word error rate  $w_1$ ; analogously for  $w_2$  and  $\Lambda(\mathcal{S})_{w_2}$ . Then*

$$w_1 < w_2 \Rightarrow \Lambda(\mathcal{S})_{w_1} < \Lambda(\mathcal{S})_{w_2}$$

**Proof** The expected number of corrections for a command  $\mathcal{C}$ ,  $n_{corr}(\mathcal{C})$ , is the only variable in  $\Lambda(\mathcal{S})$  that depends on the word error rate. It is defined on the basis of  $E(\mathcal{N}_{\mathcal{C},1})$  (refer to definition 3). For  $E(\mathcal{N}_{\mathcal{C},1}) \geq 1$  :  $n_{corr}(\mathcal{C})$  is infinite. For this to occur the word error rate must be equal or greater than the critical word error rate. Thus, as per preconditions of the theorem the critical word error rate is not exceeded, this case can be neglected. Let therefore  $E(\mathcal{N}_{\mathcal{C},1}) < 1$  then

$$n_{corr}(\mathcal{C}) = \frac{1}{1 - E(\mathcal{N}_{\mathcal{C},1})} - 1$$

Let  $E(\mathcal{N}_{\mathcal{C}_x,1})_{w_1} < 1$  denote  $E(\mathcal{N}_{\mathcal{C}_x,1})$  for any command  $\mathcal{C}_x$  in the corresponding interaction ( $E(\mathcal{N}_{\mathcal{C}_x,1})_{w_2}$  defined analogously). What remains to be shown is

$$w_1 < w_2 \Rightarrow E(\mathcal{N}_{\mathcal{C}_x,1})_{w_1} < E(\mathcal{N}_{\mathcal{C}_x,1})_{w_2}, \text{ for all } \mathcal{C}_x$$

Let now  $w_1 < w_2$ . Let  $F := \left(2 - \frac{|\mathcal{C}\mathcal{M}\mathcal{D}_{2n_{max}}| - |\mathcal{C}\mathcal{M}\mathcal{D}_{vld}|}{|\mathcal{C}\mathcal{M}\mathcal{D}_{2n_{max}}|}\right)$ , then, with (3.8) from page 56, we write

$$\begin{aligned} E(\mathcal{N}_{\mathcal{C}_x,1})_{w_1} < E(\mathcal{N}_{\mathcal{C}_x,1})_{w_2} &\Leftrightarrow \\ (1 - (1 - w_1)^{l_c}) \cdot F < (1 - (1 - w_2)^{l_c}) \cdot F &\Leftrightarrow \end{aligned}$$

as  $F > 0$  per definition of its constituents

$$1 - (1 - w_1)^{l_c} < 1 - (1 - w_2)^{l_c} \Leftrightarrow$$

$$(1 - w_1)^{l_c} > (1 - w_2)^{l_c} \Leftrightarrow$$

$$\sqrt[l_c]{1 - w_1} > \sqrt[l_c]{1 - w_2} \Leftrightarrow$$

$$w_1 < w_2, \text{ as } w_1, w_2 \in [0; 1]$$

◇

### 3.3.3 Recognition Delay-Monotony

A low recognition delay results in a low value for the interaction delay and vice versa. We call this property the *recognition delay-monotony* of the interaction delay, expressed in Theorem 3.

**Theorem 3** *Let  $r_1, r_2 \in \mathbb{R}^+$  be the recognition delays of two speech recognizers and let  $\mathcal{S}$  be a session. Let  $\Lambda(\mathcal{S})_{r_1}$  denote the interaction delay assuming  $r_1$  and a specific speech-controlled GUI approach; analogously for  $\Lambda(\mathcal{S})_{r_2}$  and  $r_2$ . Then*

$$r_1 < r_2 \Rightarrow \Lambda(\mathcal{S})_{r_1} < \Lambda(\mathcal{S})_{r_2}$$

**Proof** We rewrite the formula for the interaction delay (definition 4 on page 59):

$$\begin{aligned} & \sum_{x=1}^n (d_{\mathcal{C}_x} + r + n_{corr}(\mathcal{C}_x)(d_{\mathcal{C}_x} + r)) \\ &= \sum_{x=1}^n (r \cdot (1 + n_{corr}(\mathcal{C}_x)) + d_{\mathcal{C}_x} + n_{corr}(\mathcal{C}_x) \cdot d_{\mathcal{C}_x}) \end{aligned}$$

The components  $d_{\mathcal{C}_x}$  and  $n_{corr}(\mathcal{C}_x)$  are  $\geq 0$  and independent of the recognition delay – thus, they are constant in the scope of this theorem. As also  $r \geq 0$  all summands in the sum are  $\geq 0$ . Consequently, with increasing  $r$  the sum increases. ◇

### 3.3.4 Command Length-Monotony

Short commands result in a low value for the interaction delay and vice versa. We call this property the *command length-monotony* of the interaction delay, expressed in Theorem 4.

**Theorem 4** *Let  $\mathcal{S}$  be a session and let  $A$  and  $B$  represent two speech-controlled GUI approaches using the same speech recognizer, i.e., word error rate and recognition delay are constant for  $A$  and  $B$ . The word error rate is below the critical word error rate for  $A$  and  $B$ . Let  $i, j \in \mathbb{N}^+$  and let  $(\mathcal{C}_{A_k})_{k=1, \dots, i}$  and  $(\mathcal{C}_{B_l})_{l=1, \dots, j}$  represent the interaction to perform  $\mathcal{S}$  using approach  $A$ , respectively  $B$ . The term  $l_{\mathcal{C}_x}$  denotes the length of the command  $\mathcal{C}_x$ . Then*

$$(i = j) \wedge (\exists y \in \{1, 2, \dots, i\} : l_{\mathcal{C}_{A_y}} < l_{\mathcal{C}_{B_y}} \wedge (l_{\mathcal{C}_{A_s}} = l_{\mathcal{C}_{B_s}}, s \neq y)) \Rightarrow \Lambda(\mathcal{S})_A < \Lambda(\mathcal{S})_B$$

**Proof** We rewrite the formula for the interaction delay in definition 4 on page 59:

$$\sum_{x=1}^n (d_{\mathcal{C}_x} + r + n_{corr}(\mathcal{C}_x)(d_{\mathcal{C}_x} + r))$$

$$\begin{aligned}
 &= \sum_{x=1}^n (d_{\mathcal{C}_x} + r + n_{corr}(\mathcal{C}_x) \cdot d_{\mathcal{C}_x} + n_{corr}(\mathcal{C}_x) \cdot r) \\
 &= \underbrace{\left( \sum_{x=1}^n r \right)}_U + \underbrace{\left( \sum_{x=1}^n d_{\mathcal{C}_x} \right)}_V + \underbrace{\left( \sum_{x=1}^n n_{corr}(\mathcal{C}_x) \cdot d_{\mathcal{C}_x} \right)}_W + \underbrace{\left( \sum_{x=1}^n n_{corr}(\mathcal{C}_x) \cdot r \right)}_Z
 \end{aligned}$$

The components  $r$ ,  $d_{\mathcal{C}_x}$  and  $n_{corr}(\mathcal{C}_x)$  are  $\geq 0$  per definition, therefore the terms  $U$ ,  $V$ ,  $W$ , and  $Z$  are  $\geq 0$ . The term  $U$  is independent of  $l_{\mathcal{C}_x}$ , therefore it can be neglected. What remains to be shown is that  $V$ ,  $W$ , and  $Z$  increase monotonously with monotonously increasing  $l_{\mathcal{C}_x}$ . With

$$d_{\mathcal{C}_x} = l_{\mathcal{C}_x} \cdot 571$$

the term  $V$  obviously increases monotonously with monotonously increasing  $l_{\mathcal{C}_x}$ . Furthermore, since  $r$  is constant and  $\geq 0$  and  $d_{\mathcal{C}_x}$  also occurs in the term  $W$  it remains to be shown is that  $n_{corr}(\mathcal{C}_x)$  increases monotonously with monotonously increasing  $l_{\mathcal{C}_x}$ .

Let  $F := \left( 2 - \frac{|\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vid}|}{|\mathcal{CMD}_{2n_{max}}|} \right)$ . For the scope of this proof  $F$  is constant due to the preconditions of the theorem. With definition 3, (3.8) and the fact that the critical word error rate is not exceeded per definition, we write:

$$n_{corr}(\mathcal{C}_x) = \frac{1}{1 - (1 - (1 - w)^{l_c}) \cdot F} - 1$$

As  $l_c > 0$  and  $w \in [0; 1]$  per definition the term  $(1 - w)^{l_c}$  decreases with increasing  $l_c$ <sup>7</sup>. Therefore the term  $(1 - (1 - w)^{l_c})$  increases with increasing  $l_c$  which denotes that the term  $(1 - (1 - w)^{l_c}) \cdot F$  also increases with increasing  $l_c$ . Consequently, the term  $1 - (1 - (1 - w)^{l_c}) \cdot F$  decreases with increasing  $l_c$  which has as consequence that  $n_{corr}(\mathcal{C}_x)$  increases with increasing  $l_c$ .

◇

### 3.3.5 Command Count-Monotony

Few commands result in a low value for the interaction delay and vice versa. We call this property the *command count-monotony* of the interaction delay, shown in Theorem 5.

**Theorem 5** *Let  $S$  be a session and let  $A$  and  $B$  represent two speech-controlled GUI approaches using the same speech recognizer, i.e., word error rate and recognition delay are constant for  $A$  and  $B$ . Let  $i, j \in \mathbb{N}^+$  and let  $(\mathcal{C}_{A_k})_{k=1, \dots, i}$  and  $(\mathcal{C}_{B_l})_{l=1, \dots, j}$  represent the interaction to perform  $S$  using approach  $A$ , respectively  $B$ . The term  $l_{\mathcal{C}_x}$  denotes the length of the*

<sup>7</sup>For  $w = 0$  there are no corrections and theorem would have been proofed already.



command  $\mathcal{C}_x$ . Then

$$(i < j) \wedge (\forall y \in \{1, 2, \dots, i\} : l_{\mathcal{C}_{Ay}} = l_{\mathcal{C}_{By}}) \Rightarrow \Lambda(\mathcal{S})_A < \Lambda(\mathcal{S})_B$$

**Proof** Consider the formula for the interaction delay (definition 4 on page 59):

$$\sum_{x=1}^n (d_{\mathcal{C}_x} + r + n_{corr}(\mathcal{C}_x)(d_{\mathcal{C}_x} + r))$$

What needs to be shown is that any summand in the sum is  $\geq 0$ , i.e., any command contributes a non-zero interaction delay. We claim

$$\forall x \in \{1, 2, \dots, n\} : d_{\mathcal{C}_x} + r + n_{corr}(\mathcal{C}_x)(d_{\mathcal{C}_x} + r) > 0$$

Let  $x \in \{1, 2, \dots, n\}$  then

$$\begin{aligned} & d_{\mathcal{C}_x} + r + n_{corr}(\mathcal{C}_x)(d_{\mathcal{C}_x} + r) \\ = & l_{\mathcal{C}_x} \cdot 571 + r + n_{corr}(\mathcal{C}_x) \cdot l_{\mathcal{C}_x} \cdot 571 + n_{corr}(\mathcal{C}_x) \cdot r, \text{ since } l_{\mathcal{C}_x} \geq 1 \text{ and } r, n_{corr}(\mathcal{C}_x) \geq 0 \\ & \geq l_{\mathcal{C}_x} \cdot 571 \geq 571 > 0 \end{aligned}$$

◇

### 3.3.6 Summary

We showed that the interaction delay increases monotonically by monotonically increasing the word error rate, the recognition delay, the command length or the number of commands. Word error rate and recognition delay are characteristics of the utilized speech recognizer whereas the length and number of commands depend on the respective speech-controlled GUI approach. The critical word error rate is an inherent property of a specific speech-controlled GUI approach. It can be used during the design of the approach, e.g., to determine quality requirements for the utilized speech recognizer.

The characteristics suggest that the interaction delay can be used as a metric for speech-controlled GUIs: the better the speech recognizer (i.e., the lower the word error rate and the lower the recognition delay), the lower the (average) length of commands and the lower the (average) number of commands, the better the speech-controlled GUI approach. We consider the question of whether the interaction delay is a suitable metric for speech-controlled GUIs as future work. In order to emphasize the feasibility of this possible direction for future work we compare the interaction delay, under the assumption that it would be a suitable metric, to other user interface metrics in section 3.4.2.

## 3.4 Discussion

The interaction delay model can be seen as an extension to the GOMS model towards speech recognition-based user interfaces. It defines a model of the time required to successfully recognize a command and commands map to GOMS operators (see section 2.4). Since the interaction delay model furthermore allows to calculate the interaction delay of a single control action, i.e., a session of length 1, it extends GOMS with the possibility to assess the time for executing a GOMS goal using speech recognition as input modality (control actions map to GOMS goals, as explained in section 2.2).

We now discuss the fulfillment of the initial requirements by the interaction delay model and conclude the section with a comparison of the interaction delay to currently existing user interface metrics.

### 3.4.1 Reconsidering the Requirements

The requirements for the interaction delay model were the coverage, the inclusion of the quality of currently available speech recognition technology, and the operation on non-empirical data.

**Coverage** We will show in chapter 4 that the interaction delay model can be applied to the speech-controlled GUI approaches which have been discussed in this dissertation. Currently it is unclear if the interaction delay model can be applied to other speech-controlled GUI approaches. We do, however, show in chapter 4 as well that the interaction delay model can also be applied to conventional mouse-controlled GUIs.

**Quality of speech recognition** The recognition delay and the recognition rate are explicit parameters of the formula by which the interaction delay is calculated. If other measures for the quality of speech recognizers become available the interaction delay model must be redesigned, or revised, respectively.

**Non-empirical data** The core data for calculating the interaction delay is the session, i.e., a sequence of control actions that should be performed with the GUI. This sequence can be determined without an empirical experiment, e.g. it could be determined by the designer of a GUI. There is, however, no reason against using a session which has been determined empirically, e.g. by a previously performed user study. The (average) recognition delay and the (average) recognition rate are parameters of the interaction delay which are usually determined empirically. We do, however, assume that these values are known in advance, i.e., before the interaction delay of a specific speech-controlled GUI approach is determined.

### 3.4.2 Interaction Delay and User Interface Metrics

In this section we compare the interaction delay, under the assumption that it would be a suitable user interface metric, to the metrics *subjective satisfaction*, *number of mouse clicks* and *task success rate*.

**Subjective Satisfaction** Subjective satisfaction is a general metric for user interfaces. Test candidates perform a given number of tasks with a user interface and fill out a *USE Questionnaire* – USE stands for *usefulness*, *satisfaction*, and *ease of use* (Lund [111]). In such a questionnaire the test candidates give answers to multiple-choice questions which query the subjective opinion about test candidates' satisfaction with the user interface. The answers are associated with a score and optionally associated with a weight factor. The score of all answers (including the weight factors) is calculated per questionnaire and correlated to the maximal achievable score from which conclusions about the quality of the user interface are drawn. For instance, the higher the (average) score the higher the quality of the user interface. An example for a USE Questionnaire, however, using a different terminology, is given by Duggan in [52].

Subjective satisfaction is an empirical metric as each test candidate physically performs the test and records results. The interaction delay can be determined non-empirically for a specific speech-controlled GUI, at least if the specific session is known in advance.

The scores from the questionnaires reflect the subjective opinion of each specific test candidate at the time the test candidate performed the test. It is likely that if the very same candidate performed the same test a second time with some temporal distance (e.g. several days) the questions are answered differently. As such the subjective satisfaction metric has weaknesses regarding the reproducibility whereas the interaction delay produces exactly the same result for the same session (here a session corresponds to the test which the candidates perform).

**Number of Mouse Clicks** Number of mouse clicks is a metric for GUIs which are operated with a mouse device. It could however be applied to other user interfaces which require a mouse. The metric counts the number of mouse clicks which a user needs to complete a specific task with the GUI in question. The lower the number of mouse clicks after completing the task the higher the quality of the GUI.

Although number of mouse clicks is usually determined empirically it could be measured formally from a specification of the layout and the functionality of the GUI. This is common with the interaction delay which can also be calculated formally.

However, number of mouse clicks just considers activation. Navigation between graphical objects is neglected, because the time to move the mouse cursor from one location to another is not considered. This means that regardless of how the graphical objects are laid out – as long as they are all equally accessible and visible – the number of mouse clicks, and therefore the quality of the GUI, does not change. The interaction delay also

does not consider the time that the user interface inherently needs for navigation – only the interaction delay for navigation is considered. However, other than the number of mouse click metric, it considers the input operations (i.e., the speech functions) which the user needs to invoke in order to perform navigation

Number of mouse clicks can only be applied to user interfaces which can be operated with a mouse device, to user interfaces which know the concept of a click, respectively. The interaction delay has been designed for speech-controlled GUIs independently of whether the respective approach emulates a mouse or incorporates other concepts. The interaction delay can however, as we show later on in section 4.4, be applied to mouse-controlled user interfaces. As such the interaction delay covers a broader range of user interface types than number of mouse clicks.

**Task Success Rate** Task success rate (Nielsen [138]) is another general user interface metric. Similar to subjective satisfaction candidates perform a given number of tasks with the user interface under consideration. Instead of letting the users fill out a questionnaire after the tasks have been performed, task success rate considers if the users were able to perform the tasks at all. The number of successfully completed tasks per test candidate is counted and correlated to the total number of tasks which had to be performed. From this correlation conclusions about the quality of the user interface are drawn, e.g. the more tasks have been completed (in average) the better the quality of the user interface.

The criteria for whether a task has been successfully completed or not depends on the application domain. With task success rate one cannot compare the quality of two different user interface approaches *A* and *B* (e.g., if *A* uses a different input modality than *B* or if *A* and *B* use the same input modality but in a different fashion). For instance if both *A* and *B* had the same task success rate but with *A* the majority of the users need less time to successfully complete the tested tasks, the quality of *A* and *B* would nevertheless be considered equal. This is different with the interaction delay which considers the approach by which the input modality speech is utilized, however, without considering the cognitive load associated with that approach. Additionally, the interaction delay can be applied to other input modalities beyond speech, as we will show later in section 4.4. Consequently, the interaction delay considers characteristics of the input modality which the task success rate does not.

Finally, task success rate requires an empirical experiment to obtain data, whereas the interaction delay can be calculated non-empirically.

# 4

## Interaction Delay Calculations

### Overview

This chapter is about calculating the interaction delay of speech-controlled GUIs which serves three purposes: first, based on the results, we show that conventional command-and-control has the lowest interaction delay amongst the approaches introduced in this thesis. Second, the obtained results for conventional command-and-control serve as the reference values against we compare the interaction delay of conversation-and-control in chapter 6. Third, we show that the speech-GUI model as well as the interaction delay model cover the speech-controlled GUI approaches that are discussed in this dissertation as required (refer to sections 2.1 and 3.1). The chapter is organized into four logical parts as follows.

In the first part we define a specific type of session, called *execution*, which we use as the basis for our calculations. This is mandated by the interaction delay formula requiring a specific session as parameter and we will show that executions represent the building blocks for application domain-specific sessions. Thus, specifying representative executions and calculating their interaction delay is sufficient. In the second part of the chapter we calculate the interaction delay of representative executions for conventional command-and-control, command-and-control with random navigation and direct activation, direction-based mouse emulation with continuous movement, direction-based mouse emulation with discrete movement, target-based mouse emulation, and grid based mouse emulation. For each particular calculation we present a summary in this chapter – detailed results are documented in appendix B.2. The third part is an outlook to emphasize the feasibility of speech-GUI model and the interaction delay model: we enhance the speech-GUI model consistently with the interaction delay model, so that we can describe and calculate the interaction delay of mouse-controlled GUIs. The chapter finishes with a discussion of the obtained results.

## 4.1 Preliminaries

In this section we perform preliminary considerations and preparations, which allow us to calculate the interaction delay of speech-controlled GUIs using the interaction delay model. In section 4.1.1 we introduce a catalog of representative graphical objects and model each graphical object as a set of control actions according to the speech-GUI model. In section 4.1.2 we show that specific control actions, which emerge from a single graphical object, can be aggregated to special types of sessions, called *executions*. An execution models the inherent purpose of a graphical object and, as we will show, is a building block for application domain-specific tasks with a GUI. Finally, in section 4.1.3, we explain the general calculation procedure.

### 4.1.1 The Swing Catalog

GUIs of today are composed from a basic set of graphical objects provided by so called *GUI toolkits* – frameworks and code libraries which facilitate the implementation of GUIs in a specific programming language. The GUI toolkit Java Swing [200] is based on the platform independent programming language Java (Ullenboom [190]). We use Java Swing as the basis for our calculations, because other GUI toolkits, such as MFC [151] for Windows or Cocoa [75] for Max OS X, would restrict our results to specific platforms.



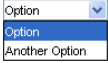
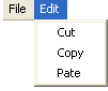


Table 4.1 on the facing page depicts a catalog of graphical objects from Java Swing. We refer to this catalog as the *Swing catalog*, and it includes only a subset of the graphical objects which Java Swing provides. The graphical objects which are omitted are specializations or aggregations of the ones which are included. Examples for specializations include the scrollbar object (specialization of a slider) and the text area object (specialization of the input field). Aggregated objects are for instance the scroll pane (aggregation of pane and scroll bar), the tabbed pane (aggregation of multiple panels), or the table object (aggregation of panels and labels). A detailed introduction into Java Swing can be found in Zukowski [200].

In the following we will explain the interactive functionality of each graphical object in the Swing catalog and model this functionality as sets of control actions. As a matter of fact in Java Swing each graphical object can be focused <sup>1</sup>, however, not every graphical object has a graphical representation of its focused state. For instance, if a specific menu item of a list is focused it is presented with a blue background. If a button is focused then its outline is drawn with a blue shadow <sup>2</sup>. In contrast, if a label or a panel is focused, then there is no graphical indication of that. In terms of the speech-GUI model, however, this coherence denotes that every graphical object can be navigated to. We reflect this as the navigation NAV of which an instance is provided by every graphical object. Furthermore, graphical objects of Java Swing can logically be decomposed into a number of graphical sub-objects. A list, for example, is composed from one graphical sub-objects per option. We will use this observation in the

---

<sup>1</sup>`JComponent.requestFocusInWindow()`

<sup>2</sup>This applies if the Windows UI manager is used.

| Graphics  | Name                 | Description   |
|---|----------------------|---|
|   | Panel                | A container for Java Swing objects, including panel.  |
| Text of label   | Label                | Text which is displayed on the screen.  |
|    | Button               | Represents a button that can be pushed.   |
| <input type="checkbox"/> Running  | Checkbox             | An object that supports two states called 'checked' and 'unchecked'.  |
| <input type="radio"/> Alternative<br><input type="radio"/> Another Alternative      | Radio Button (Group) | Group of radio buttons. Each radio button supports two states called 'selected' and 'deselected'. Usually used in groups where only one radio button of the group can be selected at the same time. |
|    | List                 | A list of options which can be selected. Multiple selection possible.   |
|    | Drop Down Box        | A list of options which can be selected. Only single selection possible. Options are initially in a hidden state and can be revealed by clicking the 'arrow' next to the currently selected option. |
|   | Menu                 | A list of actions that can be invoked. One action can be invoked at a time. List of actions is initially hidden and can be revealed by clicking the name of the menu.                               |
|  | Tree                 | A graphical object that displays hierarchically arranged data as a tree. Each piece of data/hierarchy level is represented as a node that can be expanded or collapsed.                             |
| <input type="text" value="240"/>  | Input Field          | An area into which the user can enter alphanumeric characters.  |
| <input type="text" value="12"/>   | Spinner              | Specialized text field. Accepts numeric input. Provides two buttons which increase or decrease current value.   |
|  | Slider               | A graphical object that allows to select from a range of values.  |

**Table 4.1:** The Swing catalog. The 'Graphics' column shows exemplary graphics.

following discussion, of which we provide a summary in the form of a UML class diagram in Figure 4.7 on page 76.

**Panel and Label** The panel object is a container for Java Swing objects, including the panel itself, and does not provide any interactive functionality. The label object displays static text on the screen with which the user cannot interact. We argue that the user would not navigate to a graphical object that provides no interactivity; hence, we will not include panels and labels in our calculations explicitly.



**Figure 4.1:** A checkbox in the checked and unchecked state.

**Button** Button objects are graphical metaphors for push buttons – physical buttons which can be pushed. We model the pushing of a button by the activation *PUSH* allowing us to model a button as the control action set  $\{NAV, PUSH\}$ .

**Checkbox** Checkboxes are graphical objects which have two states, called *checked* (check mark is shown) and *unchecked* (check mark is hidden), as depicted in Figure 4.1. The state of a specific checkbox is independent of the state of other checkboxes. Also, the setting of the state is idempotent: if a checked checkbox is set to checked it remains checked<sup>3</sup>. This allows us to model the setting of the checkbox states as two control actions which we call *CHECK* and *UNCHECK*. Correspondingly we model a checkbox as the control action set  $\{NAV, CHECK, UNCHECK\}$ .

**Radio Button** Radio buttons support have two different states, called *selected* and *deselected*. Once a radio button is selected the user cannot bring it back to the deselected state<sup>4</sup>. Radio buttons are usually utilized in groups of at least two radio buttons in which only a single radio button may be in the selected state. The user can unselect a radio button by selecting a different one from the same group – but not by directly deselecting the currently selected radio button. Thus, a radio button does only provide one interactive function – being selected. We model this as the activation *SELECT* and correspondingly we model a radio button as the control action set  $\{NAV, SELECT\}$ .

**Item and Pop-up** Item and Pop-up are two helper objects which we introduce to facilitate further modeling. An item is an abstract object that inherits from the label. We will further specialize the item in the following paragraphs. A pop-up is a specialized panel which may only contain items. It has two states of visibility called *visible* and *invisible*. In the visible state the items of the pop-up are rendered on the screen, whereas in the invisible state the items are not rendered. The pop-up does not provide control actions for setting its visibility, instead, other graphical (sub-)objects which use the pop-up will provide for respective control actions. We will show the usage of the pop-up in the following.

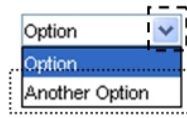
**Drop Down Box** A drop down box offers a list of options from which the user can select (see Figure 4.2 on the next page). Only one option may be selected at a time and there is always one option selected. Options cannot be deselected directly – the user needs

---

<sup>3</sup>In a mouse-controlled GUI the user does not notice this: as the user can only change the state of a checkbox by clicking the mouse button, the checkbox appears as an object that toggles its state between checked and unchecked with every click.

<sup>4</sup>At least not in a mouse-controlled GUI which was created with Java Swing using the standard radio button object.





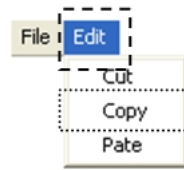
**Figure 4.2:** Arrow button (dashed line) and option (dotted line) of a drop down box.

to select a different option to deselect the currently selected option<sup>5</sup>. We represent an option of a drop-down box by a special item called the *select-item*. A select-item has the state *selected* to which it can be set by the activation *SELECT*. If the state of a select item is not selected we consider its state undefined. We model a drop down box to have as many select-items as it offers options. The drop down box furthermore has a pop-up which contains all its select-items. Additionally, the drop down box has a graphical sub-object called *arrow button* (see Figure 4.2). The arrow button can be navigated to (*NAV*) and toggles the visibility of the pop-up via its activations *OPEN* (pop-up visible) and *CLOSE* (pop-up hidden). If the option list contains more options than the pop-up is able to display in vertical direction, e.g. if the pop-up’s size in Y direction is constrained by the programmer or the screen, then a scroll bar appears allowing to scroll to currently invisible options. For the scope of this thesis we assume that the pop-up is always able to display all options, i.e., we assume that no scrolling is necessary. In the following we refer to this assumption as the *scrolling assumption*. To summarize, we model a drop down box as a set of control actions consisting of

1. one navigation that focuses the drop down box itself (*NAV*),
2. one navigation that focuses the arrow button (*NAV*),
3. two activations (*OPEN* and *CLOSE*) to toggle the visibility of the pop-up,
4. two activations (*NAV* and *SELECT*) per select-item.

**List** Like a drop down box a list offers a set of options from which the user can select. It differs from the drop down box in three ways. First, the visibility of the options cannot be toggled – they are always visible. Second, the list allows for selecting multiple options. Third, options can be deselected. We represent each option of a list by a specialized item called *select-deselect-item*. It inherits the selected state and the activation *SELECT* from the select-item. It defines the additional state *deselected* along with the activation *DESELECT* to set this new state. We model as list as being composed from as many select-deselect-item as it offers options. Similar to a drop down box, if there are more options than the list is able to display in vertical direction, the scrolling assumption applies. In summary, we model a list as a set of control actions consisting of one navigation *NAV* that focuses the list itself and three control actions (*NAV*, *SELECT*, *DESELECT*) per select-deselect-item.

<sup>5</sup>At least, this applies to Java Swing.



**Figure 4.3:** Name (dashed line) and action (dotted line) of a menu.



**Figure 4.4:** Expansion and collapsing icons (dotted line), labels and representative icons (dashed line) of a tree.

**Menu** A menu offers a list of application functions, called *actions*, which the user can invoke. We represent an action by a special item called the *action-item*. An action-item is stateless and provides an activation called *INVOKE*, which triggers the invocation of the action that it represents. A menu has a pop-up which contains all its action-items. The visibility of the pop-up is toggled by a graphical sub-object called the *name* of the menu (see Figure 4.3), for which it has the activations *OPEN* (pop-up visible) and *CLOSE* (pop-up hidden). A menu furthermore has as many action-items as it offers application functions, and, like with other graphical objects, the scrolling assumption applies. We model a menu as a set of control actions consisting of

1. one navigation that focuses the menu itself (*NAV*),
2. one navigation that focuses the name of the menu (*NAV*),
3. two activations (*OPEN* and *CLOSE*) to toggle the visibility of the pop-up,
4. two activations (*NAV* and *INVOKE*) per action-item.

**Tree** A tree is a graphical object which presents hierarchical data in the form of a tree. Each node in the tree represents a hierarchy level or a data instance. Typically, this is done by a label which names the level or data instance, and a representative icon<sup>6</sup>. Each tree node can be selected and deselected. Independently from being selected or deselected tree nodes can be expanded and collapsed via the expansion and collapsing icons (see Figure 4.4). We assume that the expansion/collapsing icons are in horizontal alignment with the label of a tree node. If a tree node is collapsed then all of its descendants are

<sup>6</sup>In Figure 4.4 a hierarchy level is represented by a folder icon whereas a data instance is represented by a bullet.



**Figure 4.5:** Decrease button (dotted line) and increase button (dashed line) of a spinner.

hidden. If a tree node is expanded then its children are visible – the visibility of further descendants depends on them being collapsed or expanded.

We represent a tree node as a special select-deselect-item which we call *tree-item*. It inherits the selected and deselected state and corresponding activations. Additionally, it defines two new states called *expanded* and *collapsed* which are independent from the inherited states. It has two new control actions: an activation called *EXPAND* which sets expanded state and an activation called *COLLAPSE* which sets the collapsed state. We consider a tree as being composed from tree-items – one tree-item per level or data instance in the hierarchy. This allows us to model a tree as a set of control actions consisting of one navigation *NAV* that focuses the tree itself and five control actions (*NAV*, *EXPAND*, *COLLAPSE*, *SELECT*, and *DESELECT*) per tree-item.

**Input Field** An input field is a rectangular area on the screen where the user can enter alphanumeric characters. We assume that newly added characters are appended to the characters which already exist in the input field, thus, we neglect the ability of real Swing input fields, to insert characters between other characters. We model the appending of a specific character  $x$  as the activation  $APPEND_x$ . Without restricting the generality of input fields we assume that input fields accept the characters a-z and 0-9. The activation *DELETE* deletes the last character that was appended. This allows us to model an input field as a set of 38 control actions  $\{NAV, APPEND_a, \dots, APPEND_z, APPEND_0, \dots, APPEND_9, DELETE\}$ .

**Spinner** A spinner is a special input field which differs from an input field in two ways. First, only numeric characters can be entered – with the assumptions made for the input field it accepts the digits 0-9. Second, the spinner allows for increasing or decreasing the value which is currently entered by two respective buttons. We call these buttons the *increase button* and the *decrease button* as illustrated in Figure 4.5. We model the functionality of the increase and decrease button by two activations called *INCREASE* and *DECREASE*, allowing us to model a spinner as a set of 14 control actions  $\{NAV, APPEND_0, APPEND_1, \dots, APPEND_9, INCREASE, DECREASE, DELETE\}$ .

**Slider** Sliders, depicted in Figure 4.6 on the next page, let the user move a mark between two fixed positions on the so called *slider axis*. The positions are called the slider's *boundaries*. If there are a few specific predefined positions for the mark on the slider axis then we call the slider a *discrete slider* (Figure 4.6(a)). If there are no such positions then we call the slider a *continuous slider* (Figure 4.6(b)). However, we consider a continuous slider a special case of a discrete slider where each pixel of the graphical representation of the slider axis denotes a predefined position – positions in between

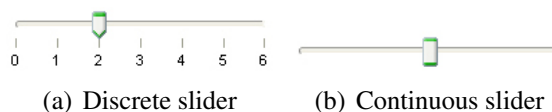


Figure 4.6: Different types of sliders.

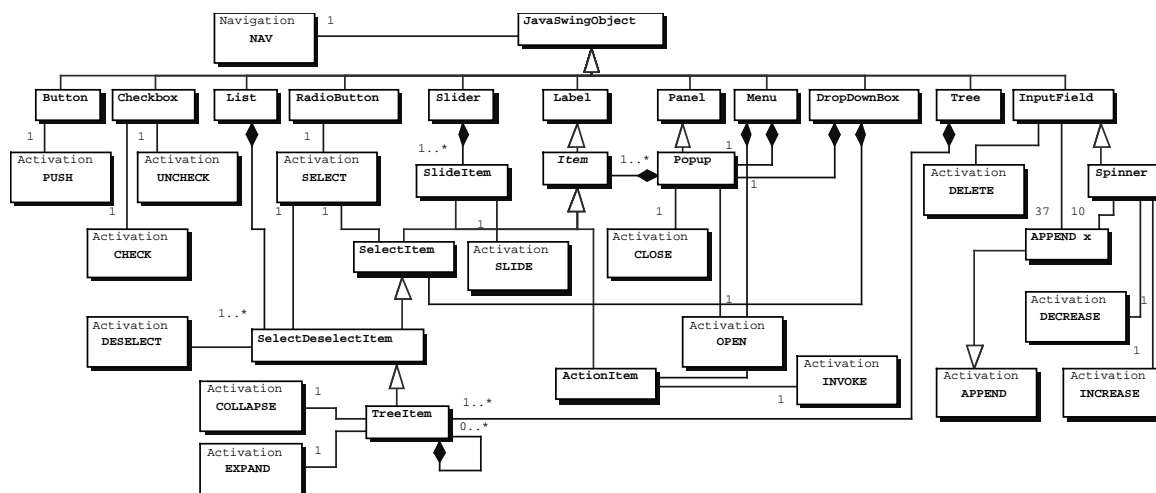


Figure 4.7: Graphical objects in the Swing catalog expressed as control actions.

these pixels cannot be represented graphically therefore the mark cannot be positioned there. In Java Swing the mark can be manipulated by using one of two methods. First, by clicking on the new position. Second, by pressing the mouse button while the mouse cursor is over the mark, moving the mouse cursor to the new position (the mark will move together with the mouse cursor), and then releasing the mouse button. The first method selects a specific position directly. It corresponds to selecting an option from a menu and can therefore be modeled by control actions. The second method is an application of drag-and-drop and can therefore not be modeled by control actions (as discussed in section 2.8.2). With this consideration we represent each position on the slider axis by a special item which we call *slide-item*. Each slide-item provides an activation called  $SLIDE_x$  which causes the mark to be positioned on the position  $x$ . Thus, we model a slider as a set of control actions consisting of one navigation *NAV* that focuses the slider itself and two control actions (*NAV*, *SLIDE*) per slide item.

As a summary of the section, we present the graphical objects just discussed as a UML class diagram in Figure 4.7 . In the following we explain the concept of executions of graphical objects.

### 4.1.2 Executions

We now use the control actions just defined to derive representative sessions for our calculations, as required by the interaction delay model.

The *purpose* of a graphical object denotes a specific effect which it accomplishes. The purpose of a button, for instance, is the activation of a specific function of the application. The purpose of a drop down box is to select an option. Graphical objects can have multiple purposes; for instance, one purpose of a list is the selection of an option, and another purpose of a list is the deselection of an option. Consequently, the *task-directed interaction* with a GUI, that is, the performing of a domain-specific task, can be abstracted as a sequence of purposes. Consider an email application as example. First, the user selects the recipient from a drop down box (purpose of a drop down box). Then, the user composes the message content by entering characters into a text field, text area, respectively (purpose of a text field). Finally, the user triggers a function that sends the email by clicking the corresponding button (purpose of a button).

An *execution* is a session of minimal length, which, when performed, accomplishes a purpose. Executions are formal representations of purposes and therefore executions are the basic building blocks for task-directed interaction with GUIs. It is therefore sufficient for our calculations to identify the executions of each graphical object from the Swing catalog, and then measure the interaction delay of each execution for different speech-controlled GUI approaches. The interaction delay of a concrete task-directed interaction can then be determined by identifying the involved executions and by summing up the corresponding interaction delays. In the following we give a more detailed definition of executions.

We define executions for graphical objects which are currently not focused. As such, executions consist of navigations and activations from one specific graphical object. We divide the navigations in an execution into two categories. The first category includes navigations which focus the graphical object as a whole. We call such navigations *inter-object navigations*. The second category includes navigations within a graphical object, i.e., between graphical sub-objects (as discussed in section 4.1.1). We call these navigations *intra-object navigations*. Now, with the terminology of inter- and intra-object navigation, we consider the navigation *NAV* as the superclass of two specialized navigations  $NAV_{inter}$  and  $NAV_{intra}$ . Hence, executions begin with an instance of  $NAV_{inter}$ , as executions are only defined on currently not focused graphical objects. Executions have additional instances of  $NAV_{intra}$  depending on whether the graphical object is logically composed from graphical sub-objects.

We require that in an execution an intra-object navigation of a specific graphical object must be preceded by an inter-object navigation to that respective graphical object, and call this requirement the *cascading navigation* requirement. This seems unrealistic at first sight: consider for instance a list which is currently not focused. Cascading navigation could be interpreted as such that, given speech-based mouse emulation, the user has to click somewhere onto the list before being able to click on the desired option. In reality, the mouse cursor can be positioned over the desired option directly, followed by a click that selects the option. This is, however, not a violation of cascading navigation: inter- and intra object navigations are

control actions, thus, they describe navigation independently of the utilized input modality. Control actions are mapped to speech functions and there is a many-to-many relationship between control actions and speech functions (refer to section 2.2). Thus, the required inter- and intra-object navigations do not have to be mapped to speech functions one by one. Instead, depending on the functionality of the input modality, specific speech functions might be able to cover multiple control actions. In the above example, the speech functions which directly place the mouse cursor over the option cover both the required inter- and intra-object navigation.

Having provided a definition of executions we now present the executions of graphical objects from the Swing catalog. A summary of these executions is provided in Table 4.2 on page 80.

**Button** The purpose of a button is the invocation of a specific application function. We call the execution of a button  $E_{button}$  and it consists of  $NAV_{inter}$  to focus the button, followed by the control action *PUSH*, which triggers the function that is associated with the button.

**Checkbox** A checkbox has two executions called  $E_{checkbox-check}$  and  $E_{checkbox-uncheck}$  which check/uncheck the checkbox.  $E_{checkbox-check}$  consists of  $NAV_{inter}$  followed by the activation *CHECK*.  $E_{checkbox-uncheck}$  consists of  $NAV_{inter}$  followed by the activation *UNCHECK*.

**Radio Button** The purpose of a radio button is to get selected – which implies the deselection of other radio buttons in the same group. We call its execution  $E_{radio}$  and it consists of  $NAV_{inter}$  followed by the activation *SELECT*.

**Drop Down Box** The purpose of a drop down box is the selection an option. The options are displayed by a pop-up, which is initially invisible. Thus, the user needs to open the pop-up to review the options, an option needs to be selected, and then the pop-up needs to be closed. As options cannot be deselected explicitly, a drop down box has a single execution called  $E_{dropdown}$ . It consists of an instance of  $NAV_{inter}$  to navigate to the drop down box, followed by the control actions  $NAV_{intra}$  and *OPEN* to reveal the pop-up (navigation and activation of the arrow button), followed by the control actions  $NAV_{intra}$  and *SELECT* to navigate and select the desired option, followed by the control actions  $NAV_{intra}$  and *CLOSE* to hide the pop-up (navigation and activation of arrow button)<sup>7</sup>.

**List** The list allows for selecting multiple options including an explicit deselecting of options. Options can be selected/deselected directly, i.e., without having to open a pop-up. Consequently, a list has the executions  $E_{list-select}$  and  $E_{list-deselect}$ . They both consist of an instance of  $NAV_{inter}$  to navigate to the list, an instance of  $NAV_{intra}$  to navigate to the option, and an instance of *SELECT/DESELECT* to select/deselect the option.

---

<sup>7</sup>A drop down box in Java Swing automatically closes the pop-up when an option is selected. In the context of our model this denotes that the speech functions with trigger the selection of the option also trigger the closing of the pop-up.

**Menu** The purpose of a menu is the invocation of one of its actions. We call the menu's execution  $E_{menu}$ . It consists of an instance of  $NAV_{inter}$  to navigate to the menu, followed by the control actions  $NAV_{intra}$  and  $OPEN$  to navigate to the menu's name and to reveal the pop-up, followed by the control actions  $NAV_{intra}$  and  $INVOKE$  to navigate and invoke the desired action item, followed by the control actions  $NAV_{intra}$  and  $CLOSE$  to hide the pop-up (navigation and activation of the menu's name)<sup>8</sup>.

**Tree** For the tree we make four assumptions. First, we assume that the root node of the tree – if it is visible at all – is always expanded, i.e., it cannot be collapsed. Second, we assume that all descendants of the root node are initially collapsed. Third, we assume that the user intends to select (or deselect) a node which is represented by a tree-item in depth  $n_{depth}$ . Hereby,  $n_{depth} = 0$  denotes the child nodes of the root node (as the root node cannot be expanded nor collapsed)<sup>9</sup>. Fourth, we assume that the path to the intended item can be inferred by the labels of the respective parent nodes<sup>10</sup>. This allows us to set up two executions for the tree, called  $E_{tree-select}$  and  $E_{tree-deselect}$ . The execution  $E_{tree-select}$  consists of an instance of  $NAV_{inter}$  to navigate to the tree, followed by  $n_{depth}$  pairs of the control actions  $NAV_{intra}$  and  $EXPAND$  to drill down into the tree until the desired tree item becomes visible (navigation and activation of  $n_{depth}$  tree items), followed by the control actions  $NAV_{intra}$  and  $SELECT$  to navigate and select the desired tree item. For  $E_{tree-deselect}$  we assume that the desired tree item is visible. As such,  $E_{tree-deselect}$  consists of an instance of  $NAV_{inter}$  to navigate to the tree, an instance of  $NAV_{intra}$  to navigate to the tree-item and an instance of  $DESELECT$  to deselect it.

**Input Field** The purpose of an input field is to input alphanumerical characters and we assume that the user intends to enter  $n_{text}$  characters. The execution of an input field, called  $E_{input}$ , consists of an instance of  $NAV_{inter}$  to navigate to the input field, followed by  $n_{text}$  instances of  $APPEND_x$  to enter  $n_{text}$  characters into the input field.

**Spinner** The spinner is a special case of an input field of which the purpose is to enter a numeric value consisting of  $n_{digit}$  digits<sup>11</sup>. The spinner has three executions. The first execution, called  $E_{spinner}$ , consists of an instance of  $NAV_{inter}$  to navigate to the spinner, followed by  $n_{digit}$  instances of  $APPEND_x$ . The second execution is called  $E_{spinner-inc}$  and represents the spinner's purpose to increase the current value of the spinner. It

<sup>8</sup>A menu in Java Swing automatically closes the pop-up when an action has been invoked. In the context of our model this denotes that the speech functions which trigger the invocation of the action also trigger the closing of the pop-up.

<sup>9</sup>Refer to Figure 4.1 on page 71: the root node is labeled "JTree" and the tree nodes in depth 0 are the nodes labeled "colors", "sports" and "food"

<sup>10</sup>The first and the second assumption denote a worst case scenario where the user must drill down through a maximum of tree levels to reach the desired node. The fourth assumption represents a typical tree search algorithm where the path to a specific tree node is determined by information about the tree item to be searched an information in the respective tree node. As such we rule out that the user performs a linear search through potentially all tree nodes.

<sup>11</sup>The value is a positive integer, as the input field, from which a spinner inherits, accepts the digits 0-9.

| Graphical Object | Execution              | Control Action(s)  |
|------------------|------------------------|--|
| Button           | $E_{button}$           | $(NAV_{inter}, PUSH)$  |
| Checkbox         | $E_{checkbox-check}$   | $(NAV_{inter}, CHECK)$   |
|                  | $E_{checkbox-uncheck}$ | $(NAV_{inter}, UNCHECK)$   |
| Radio Button     | $E_{radio}$            | $(NAV_{inter}, SELECT)$  |
| Drop Down Box    | $E_{dropdown}$         | $(NAV_{inter}, NAV_{intra}, OPEN,$<br>$NAV_{intra}, SELECT, NAV_{intra}, CLOSE)$   |
| List             | $E_{list-select}$      | $(NAV_{inter}, NAV_{intra}, SELECT)$   |
|                  | $E_{list-deselect}$    | $(NAV_{inter}, NAV_{intra}, DESELECT)$   |
| Menu             | $E_{menu}$             | $(NAV_{inter}, NAV_{intra}, OPEN,$<br>$NAV_{intra}, INVOKE, NAV_{intra}, CLOSE)$   |
| Tree             | $E_{tree-select}$      | $(NAV_{inter},$<br>$\underbrace{NAV_{intra}, EXPAND, \dots, NAV_{intra}, EXPAND}_{n_{depth}},$<br>$NAV_{intra}, SELECT)$ |
|                  | $E_{tree-deselect}$    | $(NAV_{inter}, NAV_{intra}, DESELECT)$   |
| Input Field      | $E_{input}$            | $(NAV_{inter}, \underbrace{APPEND, \dots, APPEND}_{n_{text}})$   |
| Spinner          | $E_{spinner}$          | $(NAV_{inter}, \underbrace{APPEND, \dots, APPEND}_{n_{digit}})$  |
|                  | $E_{spinner-dec}$      | $(NAV_{inter}, NAV_{intra}, INCREASE)$   |
|                  | $E_{spinner-inc}$      | $(NAV_{inter}, NAV_{intra}, DECREASE)$   |
| Slider           | $E_{slide}$            | $(NAV_{inter}, NAV_{intra}, SLIDE)$  |

**Table 4.2:** Basic executions of graphical objects in the Swing catalog.

consists of an instance of  $NAV_{inter}$  to navigate to the spinner, followed by the control actions  $NAV_{intra}$  and  $INCREASE$  to navigate and activate the increase button. The third execution is called  $E_{spinner-dec}$  and represents the spinner's purpose to decrease the current value of the spinner. It is defined analogously to  $E_{spinner-inc}$ .

**Slider** The purpose of a slider is the selection of a specific value  $x$  represented as a slide position. Its execution, which is called  $E_{slide}$  consists of an instance of  $NAV_{inter}$  to navigate to the slider, followed by the control actions  $NAV_{intra}$  and  $SLIDE_x$  to navigate and activate the desired slide position.

In Table 4.2 we provide a summary of the executions just discussed. For the remainder of this document we will refer to the executions in Table 4.2 as the *basic executions*. The following section describes the general procedure by which we calculate the interaction delay of basic executions.



### 4.1.3 Calculation Procedure

The calculation of the interaction delay is currently not supported by tools. Therefore, we now explain, how we performed the interaction delay calculations "manually". This is in contrast to the GOMS model, for which such tools exist: John et al. present *Apex* [87], which is a tool that generates CPM-GOMS models from a hierarchical task decomposition. Using *Apex*, the model is presented as a *PERT chart(s)* which visualizes parallel behavior of the tasks. For further reading about GOMS modeling tools refer to Baumeister et al. [18].

The interaction delay of a session  $\mathcal{S}$ , according to definition 4 on page 59, is defined as follows:

$$\Lambda(\mathcal{S}) = \sum_{x=1}^n (d_{\mathcal{C}_x} + r + n_{corr}(\mathcal{C}_x)(d_{\mathcal{C}_x} + r))$$

Hereby,  $n \in \mathbb{N}, n > 0$  denotes the number of commands needed to perform  $\mathcal{S}$ ,  $(\mathcal{C}_x)_{x=1, \dots, n}$  denotes the commands in the interaction used to perform  $\mathcal{S}$ ,  $r \in \mathbb{R}_0^+$  denotes the recognition delay, and  $d_{\mathcal{C}_x}$  denotes the duration of the command  $\mathcal{C}_x$ . In the following we will expand the formula so that all further parameters which are needed are visible.

The recognition delay and the duration of a word are defined by the speech-GUI model: the recognition delay is 1500ms and the duration of a word is 571ms. We will adopt these values for our calculations. Let  $l_{\mathcal{C}}$  denote the length of the command  $\mathcal{C}$ , then  $d_{\mathcal{C}} = l_{\mathcal{C}} \cdot 571$ , and we arrive at

$$\Lambda(\mathcal{S}) = \sum_{x=1}^n (l_{\mathcal{C}_x} \cdot 571 + 1500 + n_{corr}(\mathcal{C}_x)(l_{\mathcal{C}_x} \cdot 571 + 1500))$$

Now we expand  $n_{corr}$  which was defined as

$$n_{corr}(\mathcal{C}) := \frac{1}{1 - E(\mathcal{N}_{\mathcal{C},1})} - 1$$

for  $E(\mathcal{N}_{\mathcal{C},1}) < 1$  (refer to definition 3 on page 58). The case  $E(\mathcal{N}_{\mathcal{C},1}) \geq 1$  will not be considered, because then  $n_{corr} \rightarrow \infty$ . We arrive at

$$\Lambda(\mathcal{S}) = \sum_{x=1}^n \left( l_{\mathcal{C}_x} \cdot 571 + 1500 + \left( \frac{1}{1 - E(\mathcal{N}_{\mathcal{C}_x,1})} - 1 \right) (l_{\mathcal{C}_x} \cdot 571 + 1500) \right) \quad (4.1)$$

According to equation (3.8) on page 56 we may expand  $E(\mathcal{N}_{\mathcal{C},1})$  as follows ( $w \in [0; 1]$  is the word error rate):

$$E(\mathcal{N}_{\mathcal{C},1}) = (1 - (1 - w)^{l_{\mathcal{C}}}) \cdot \left( 2 - \frac{|\mathcal{CMD}_{2n_{max}}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}_{2n_{max}}|} \right)$$

With equation (3.2) on page 52 we write <sup>12</sup> for  $|\mathcal{V}| > 1$ :

$$E(\mathcal{N}_{c,1}) = \left(2 - \frac{|\mathcal{V}|^{2 \cdot n_{max} + 1} - 1}{|\mathcal{V}| - 1} - 1 - |\mathcal{CMD}_{vld}|\right) (1 - (1 - w)^{lc})$$

Hence, if we insert the expanded value for  $E(\mathcal{N}_{c,1})$  in formula (4.1), which we omit for reasons of clarity, the interaction delay calculates as a function against the recognition delay, the duration of a word, the number of necessary commands for a specific a session, the length of each command, the size of the vocabulary, the total number of valid commands, the length of the longest valid command ( $n_{max}$ ), and the word error rate. The recognition delay and duration of words have already been adopted from the speech-GUI model. In the following we explain how we intend to obtain values for the remaining parameters to facilitate concrete calculations.

For each approach that is to be calculated we will specify the available speech functions and their corresponding commands. The union of these commands denotes the set of valid commands for that approach. Thus, we can determine the number of valid commands and the length of the longest valid command. The size of the vocabulary can be determined by counting all distinct words occurring in any valid command. The basic executions serve as sessions of which the interaction delay is calculated. Using the specified speech functions we can set up intentions and corresponding interactions for each basic execution. <sup>13</sup> As such, the lengths of the occurring commands can be determined by counting the words in each command of each interaction.

Regarding the word error rate we need to consider that the environmental conditions, which a speech recognizer is exposed to, may vary during its runtime. For instance, the intensity of environmental noise may change, as well as the quality of the articulation of the user. Thus, the word error rate, as it depends on these environmental conditions, cannot assumed to be constant. We therefore calculate the interaction delay for the following word error rates: 0% (nominal interaction delay, refer to Theorem 1 on page 59), 1%, 5%, 10%, 20%, 30%, 40%, and 50%. Hereby, we assume that, at the latest, with a word error rate of over 50%, i.e., with a word error happening with every second word, users would be too frustrated to continue using the system. The value of 50% is, however, arbitrary and can be adapted. As already stated in section 3.2.1, the size of the vocabulary and the number of valid commands can (dynamically) be minimized specifically to the current screen, which improves the word error rate and which reduces interpretation complexity (Smailagic and Siewiorek [175]).

From the obtained calculation results we then determine the minimum, the average, and the maximum interaction delay per basic execution per word error rate. With calculating the average we assume that every basic execution is equally likely to occur. If this assumption does not hold, then we suggest calculating a weighted average based on the relative frequency of each basic execution, which is certainly specific to a particular GUI and/or task. In our

---

<sup>12</sup>Cases where  $|\mathcal{V}| = 1$  will not occur in our calculations, however, for  $|\mathcal{V}| = 1$  the formula becomes simpler, as show in equation (3.2) on page 52.

<sup>13</sup>The respective intentions and interactions are presented in detail in appendix B.2 for reasons of clarity.

situation, however, the relative frequency of each basic execution is a priori unknown. For calculating these cumulated figures we omit the interaction delay values for  $E_{tree-select}$ ,  $E_{input}$ , and  $E_{spinner}$ , as they contain the a priori unknown values  $n_{depth}$ ,  $n_{text}$  and  $n_{digit}$ .

In the following sections we calculate the interaction delay of command-and-control and of speech-based mouse emulation. We do this, however, without further calibrating the interaction delay model, i.e., we regard the calibration by the word duration and the recognition delay obtained from literature research as sufficient. To achieve a better calibration we suggest to use empirically measured values for the word duration and the recognition delay if the users as well as the speech recognizer is known. Furthermore, a program could be created, that takes the parameters of a specific interaction delay calculation as input (i.e., the expected word error rate, a sequence of commands, etc.), and which determines the interaction delay by simulation – possibly for a large number of iterations of the same interaction. This could, eventually, lead to the definition of specific regression coefficients for the model, by which characteristics of the environment, the task, or the user could be integrated. Thus, we regard a more accurate calibration of the interaction delay model as future work (see also section 7.3).

## 4.2 Command-and-Control

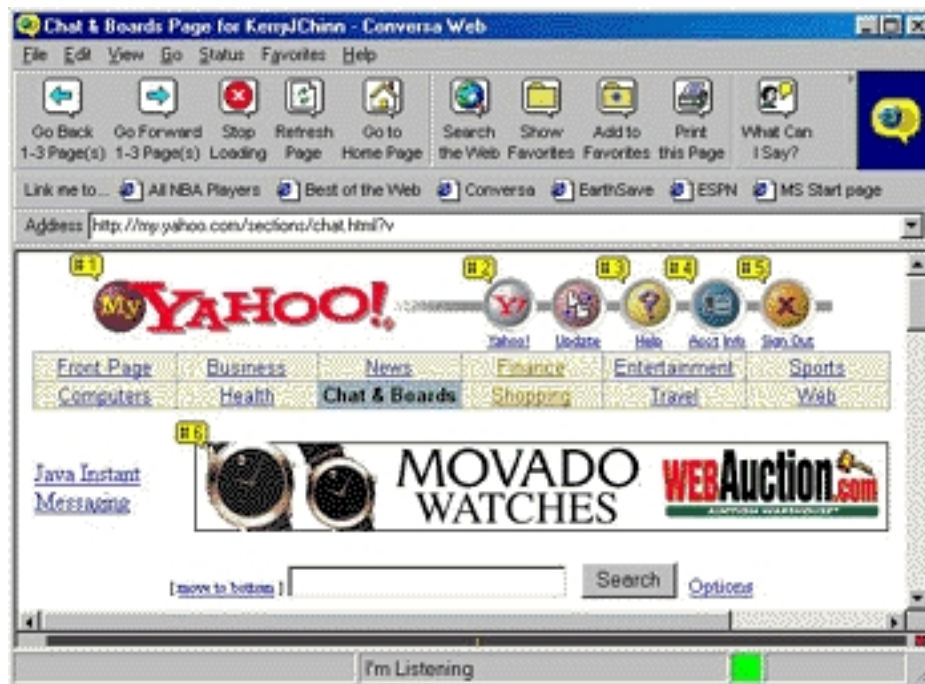
We now calculate the interaction delay of command-and-control in order to obtain reference values based on which we will show the reduced interaction delay of conversation-and-control in chapter 6. We will first discuss general conditions for the calculations of command-and-control in section 4.2.1. Then we calculate conventional command-and-control (random navigation and indirect activation) in section 4.2.2, followed by command-and-control with random navigation and direct activation in section 4.2.3. In section 4.2.4 we summarize and interpret the results.

### 4.2.1 General Conditions

We introduce a generally applicable schema for speakable identifiers, define common speech functions and commands for command-and-control, and explain the *spelling mode* for entering alphanumeric characters into input fields and spinners.

#### Speakable Identifiers

With command-and-control all graphical objects are assigned a speakable identifier. Some variants use textual properties of graphical objects to obtain the identifiers (e.g., Sipek [174]), while other variants define a specific schema to generate the identifiers. An example for the latter method is Conversay Voice Surfer [43] which uses subsequent integer numbers, starting with 1 (Figure 4.8 on the next page). Regardless of how the identifiers are obtained or generated, the vocabulary of the utilized speech recognizer must contain the words from which the identifiers are composed. In order to be able to specify the size of the vocabulary, which



**Figure 4.8:** Conversa Voice Surfer displays numeric identifiers as bubbles attached to graphical objects.

is a parameter of the interaction delay formula, we derive and justify a finite set of speakable identifiers in the following.

We require that a well-designed GUI may only consist of a specific maximum number of graphical objects<sup>14</sup>, called  $go_{max}$ . Otherwise, if the number of graphical objects exceeds  $go_{max}$ , we consider the GUI as ill-designed. The feasibility of this requirement is supported by Constantine's *Visibility Principle* and *Structure Principle* of good user interface design [42], and for the remainder of this dissertation we will neglect ill-designed GUIs.

Our calculations are not based on a specific GUI instance and therefore, even by just considering well-designed GUIs, we do not a priori know of which particular graphical objects the GUI consists. Thus, we cannot use textual properties of graphical objects for generating speakable identifiers, as we do not know which textual properties exist. Using numeric identifiers, however, allows us to specify the vocabulary in advance: a well-designed GUI is composed from maximal  $go_{max}$  graphical objects, and hence, a finite set of integer numbers as identifiers, ranging from 1 to  $go_{max}$ , is required. The verbal representations for integer numbers between 1 and  $go_{max}$  can be described by a grammar that regards distinct words as terminal symbols (e.g. "one", "two", "fourteen", etc.). The set of required terminal symbols is finite and represents the subset of the vocabulary needed for speakable identifiers. For being

<sup>14</sup>In this regard we consider graphical objects consisting of multiple sub-objects as single graphical objects. An example is a table (e.g., `JTable` from Java Swing) which potentially consists of several 100 interactive table cells.

able to perform concrete calculations we set <sup>15</sup>  $go_{max} = 100$ . A grammar that describes verbal representations of integer numbers between 1 and 100 consists of 28 terminal symbols (refer to appendix B.1) – thus, the vocabulary of the speech recognizer has a subset of 28 words for recognizing numbers between 1 and 100.

Assuming numeric identifiers does not affect the correctness of our calculations. Indeed, textual identifiers might require far more than 28 words than numeric identifiers, as every graphical object must have a unique identifier which might in turn consist of multiple words. This might result in different results for different methods to generate speakable identifiers. However, we will adhere to numeric identifiers as defined here wherever speakable identifiers are needed. Thus, all speech-controlled GUI approaches which involve speakable identifiers are treated equally, and as such speakable identifiers are a constant throughout our calculations.

### Common Speech Functions and Commands

Command-and-control allows navigating to graphical objects by uttering their speakable identifiers. With the above considerations, command-and-control provides 100 speech functions which cause a navigation to a graphical object, referred to as  $SFNAV_{id}$ ,  $id \in \{1, 2, \dots, 100\}$ . It furthermore provides 100 corresponding commands consisting of the respective speakable identifiers, referred to as  $CMDNAV_{id}$ ,  $id \in \{1, 2, \dots, 100\}$ . For facilitating concrete calculations we will assume that each command  $CMDNAV_{id}$  consists of two words – which, according to the grammar in appendix B.1, corresponds to the worst maximum length of a command that triggers a navigation.

We define that, whenever a graphical object has been navigated to, the graphical sub-objects, of which it is possibly composed, are assigned a speakable identifier and, as such, become navigable as well. This defined behavior is in accordance with observations of currently available technology and we call it the *navigation precedence*. An example is the drop down box. If it is navigated to it opens its pop-up and reveals the available options due to indirect activation, as described later in Table 4.4 on page 87. Each select-item of the drop down box is then assigned a speakable identifier. For our calculations we assume that the identifiers for graphical sub-objects are taken from the "pool" of the available 100 identifiers. In order to practically avoid a situation where the identifier pool would be exhausted – all 100 identifiers being assigned to graphical objects – we argue that the vocabulary for the 100 identifiers is also sufficient for describing identifiers of up to 999 (see grammar in appendix B.1). As such,  $SFNAV_{id}$  can be used to perform inter- and intra-object navigation.

We further define  $SFACT_x$  as a speech function of command-and-control which triggers the activation  $x$  of a currently focused graphical object. For example, the speech function  $SFACT_{OPEN}$  would open the pop-up of a drop down box (or a menu) if the drop down box (or the menu) is currently focused. We define  $CMDACT_x$  as a command of command-and-control which invokes the speech function  $SFACT_x$  and Table 4.3 on the next page presents

<sup>15</sup>The value of  $go_{max}$  can easily be adapted. It can be seen as a parameter to formal interaction delay calculations of command-and-control.

| Command             | Words  |
|---------------------|--|
| $CMDACT_{PUSH}$     | "push"   |
| $CMDACT_{CHECK}$    | "check"  |
| $CMDACT_{UNCHECK}$  | "uncheck"  |
| $CMDACT_{SELECT}$   | "select"   |
| $CMDACT_{DESELECT}$ | "deselect"   |
| $CMDACT_{OPEN}$     | "open"   |
| $CMDACT_{CLOSE}$    | "close"  |
| $CMDACT_{INVOKE}$   | "invoke"   |
| $CMDACT_{EXPAND}$   | "expand"   |
| $CMDACT_{COLLAPSE}$ | "collapse"   |
| $CMDACT_{APPEND}$   | "append <c>", <c> $\in \{ "a", "b", \dots, "z", "zero", \dots, "nine" \}$<br>respectively<br><c> $\in \{ "alpha", "beta", \dots, "zulu", "zero", \dots, "nine" \}$ |
| $CMDACT_{DELETE}$   | "delete"   |
| $CMDACT_{INCREASE}$ | "increase"   |
| $CMDACT_{DECREASE}$ | "decrease"   |
| $CMDACT_{SLIDE}$    | "slide"  |

**Table 4.3:** Common commands with command-and-control.

representative commands <sup>16</sup> .

## Spelling Mode

According to the above terminology, the speech functions  $SFACT_{APPEND_x}$  invoke the activations  $APPEND_x$ , which append the character  $x$  to an input field or a spinner ( $x \in \{a, b, \dots, z, 0, 1, \dots, 9\}$ ). The speech function  $SFACT_{DELETE}$  removes the most recently appended character from an input field or spinner. In the following we refer to these speech functions as *spelling functions*. Each of the commands  $CMDACT_{APPEND_x}$  and  $CMDACT_{DELETE}$  triggers a specific spelling function. We therefore refer to these commands as the *spelling commands*. For the remainder of this document we will refer to the entirety of spelling functions and spelling commands as *spelling mode*, as the entering of alphanumeric characters by spelling commands has a flavor of spelling the words or numbers that should be entered. Spelling mode defines 37 commands - one command per appending a character or digit and one command for the deletion. It requires a vocabulary subset of 38 words - the word "delete" and "append" and one word per character or digit.

In the following two sections we calculate the interaction delay of conventional command-and-control and command-and-control with random navigation and direct activation.

<sup>16</sup>The commands are either derived from literature or, in cases where the respective graphical object has not been used with command-and-control, invented by ourselves.

| <b>Graphical Object</b> | <b>Indirectly Performed Activations</b>  |
|-------------------------|--|
| Button                  | <i>PUSH</i> (inter-object navigation)  |
| Checkbox                | 1. <i>CHECK</i> (inter-object navigation if unchecked)<br>2. <i>UNCHECK</i> (inter-object navigation if )                                    |
| Radio Button            | <i>SELECT</i> (inter-object navigation)  |
| List                    | 1. <i>SELECT</i> (intra-object navigation to deselected select-item)<br>2. <i>DESELECT</i> (intra-object navigation to selected select-item) |
| Drop Down Box           | 1. <i>OPEN</i> (inter-object navigation)<br>2. ( <i>SELECT, CLOSE</i> ) (intra-object navigation to select-item)                             |
| Menu                    | 1. <i>OPEN</i> (inter-object navigation)<br>2. ( <i>INVOKE, CLOSE</i> ) (intra-object navigation to action-item)                             |
| Tree                    | <i>EXPAND</i> (intra-object navigation to collapsed tree-item)   |

**Table 4.4:** Indirectly triggered activations with conventional command-and-control.

## 4.2.2 Conventional Command-and-Control

From the discussion in the previous section we know that conventional command-and-control uses 100 valid commands for inter- and intra-object navigation and 37 valid commands for the spelling mode. Navigation requires a vocabulary subset of 28 distinct words and spelling mode requires a vocabulary subset of 38 distinct words, however, these two subsets have a non-empty intersection – the words "one" - "nine" are contained in both subsets. Thus, for navigation and spelling mode, a vocabulary subset of only  $28 + 38 - 9 = 57$  words is required.

Regarding valid commands and corresponding vocabulary subsets for the activation of the remaining graphical objects, i.e., other than input field and spinner, we need to consider that indirect activation is used. This means that specific activations are performed automatically whenever the respective graphical object is navigated to. Thus, we may neglect the commands which would usually trigger the speech functions for invoking these activations. We present the indirectly performed activations in Table 4.4, and we conclude that the activations *SELECT*, *DESELECT* and *COLLAPSE* of the tree, and the activations *INCREASE* and *DECREASE* of the spinner are not invoked indirectly. Correspondingly, conventional command-and-control defines 5 additional valid commands for activation, which require 5 additional words in the vocabulary.

In total, we have  $100 + 37 + 5 = 142$  valid commands, requiring a vocabulary of  $57 + 5 = 62$  words. The length of the longest valid command is 2, such as uttering an identifier or uttering a spelling command. With the considerations taken in this section we have derived intentions and interactions for the basic executions, and have calculated their interaction delays for different word error rates (refer to appendix B.2, Tables B.1 and B.2, for a detailed documentation). We summarize the results in Table 4.5 which depicts the minimum, the average, and the maximum interaction delay per basic execution of conventional command-and-control.

| Interaction Delay in ms | Word Error Rate |      |      |      |       |       |       |       |
|-------------------------|-----------------|------|------|------|-------|-------|-------|-------|
|                         | 0%              | 1%   | 5%   | 10%  | 20%   | 30%   | 40%   | 50%   |
| Minimum                 | 2642            | 2696 | 2927 | 3262 | 4128  | 5392  | 7339  | 10568 |
| Average                 | 4481            | 4567 | 4936 | 5468 | 6839  | 8828  | 11871 | 16888 |
| Maximum                 | 7355            | 7483 | 8035 | 8825 | 10845 | 13742 | 18130 | 25279 |

**Table 4.5:** Calculated minimum, average and maximum interaction delay per basic execution of conventional command-and-control.

| Interaction Delay in ms | Word Error Rate |      |       |       |       |       |       |       |
|-------------------------|-----------------|------|-------|-------|-------|-------|-------|-------|
|                         | 0%              | 1%   | 5%    | 10%   | 20%   | 30%   | 40%   | 50%   |
| Minimum                 | 4713            | 4788 | 5107  | 5563  | 6717  | 8350  | 10791 | 14710 |
| Average                 | 6379            | 6484 | 6934  | 7578  | 9213  | 11540 | 15036 | 20684 |
| Maximum                 | 9426            | 9575 | 10215 | 11126 | 13434 | 16701 | 21581 | 29420 |

**Table 4.6:** Calculated minimum, average and maximum interaction delay per basic execution of command-and-control with random navigation and direct activation.

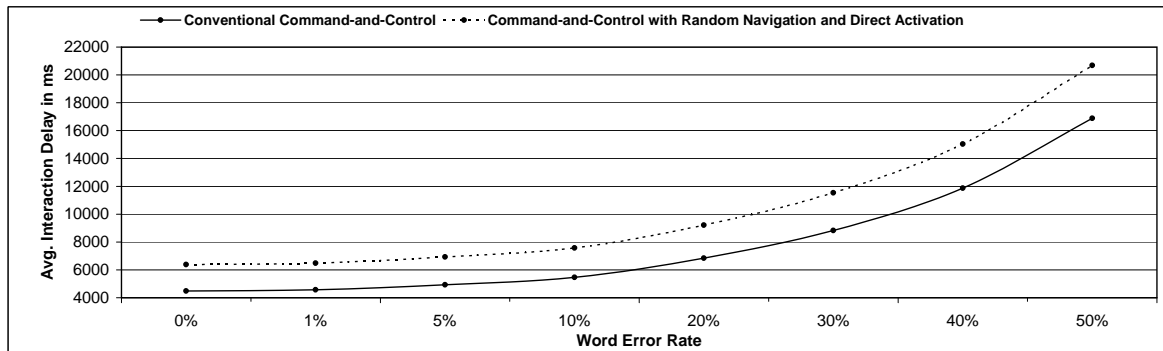
### 4.2.3 Command-and-Control with Random Navigation and Direct Activation

The only difference of command-and-control with random navigation and direct activation compared to conventional command-and-control is, that for each activation the corresponding speech function(s) must be triggered explicitly, i.e., any activation must be performed directly. This may happen as soon as the desired graphical object has been navigated to. In other words, any graphical object must be focused before it can be activated.

Reconsidering Table 4.4 we conclude that command-and-control with random navigation and direct activation, compared to conventional command-and-control, defines 8 additional valid commands for being able to directly perform any activation – all other commands can be reused. To be able to recognize the additional commands the vocabulary must be augmented by 8 corresponding words.

In total, we have  $142 + 8 = 150$  valid commands, requiring a vocabulary of  $62 + 8 = 70$  words. Similar to conventional command-and-control, the length of the longest valid command is 2. With the considerations taken in this section we have derived intentions and interactions for the basic executions, and have calculated their interaction delays for different word error rates (refer to appendix B.2, Tables B.3 and B.4, for a detailed documentation). We summarize the results in Table 4.6 which depicts the minimum, the average, and the maximum interaction delay per basic execution of command-and-control with random navigation and direct activation.





**Figure 4.9:** Calculated interaction delay graphs of command-and-control approaches.

## 4.2.4 Summary

We now discuss the results that we obtained from calculating the interaction delay of conventional command-and-control and command-and-control with random navigation and direct activation. Considering the vocabulary and the set of valid commands for both approaches we find that these are not equal but, indeed, very similar. In fact, the vocabulary and the valid commands for conventional command-and-control are subsets of the vocabulary and the valid commands of command-and-control with random navigation and direct activation. Considering this coherence and the command length-monotony (Theorem 4) it is a tempting prognosis that both approaches might have similar interaction delays. However, as the diagram in Figure 4.9 illustrates, the average interaction delay per execution of conventional command-and-control is, for every calculated word error rate, considerably lower than of command-and-control with random navigation and direct activation. The same applies for the respective minimum and maximum interaction delay values (refer to Tables 4.5 and 4.6). For word error rates which can nowadays be accomplished by speech recognizers (i.e., 5% - 20%) the difference in the average interaction delay per execution is between 42% and 34%.

Looking at the detailed interactions in Tables B.1 on page 196 and B.3 on page 198 the reason for this becomes obvious. If we neglect the order of the commands than we find that the interactions of conventional command-and-control are sub-interactions of the interactions used for command-and-control with random navigation and direct activation. Several interactions of command-and-control with random navigation and direct activation have additional commands. In other words, with command-and-control with random navigation and direct activation the user utters the same commands as with conventional command-and-control, but, in several cases, needs to utter additional commands. Consequently, according to the command count-monotony (Theorem 5), the interaction delay of command-and-control with random navigation and direct activation must be higher – which is reflected by our results. In the following we examine the actual difference.

Let us first examine the interaction delay for a word error rate of 0%. The minimum and the maximum interaction delay per basic execution (Tables 4.5 and 4.6) differ by 2071ms – which

is exactly the sum of the recognition delay and the duration of a command of length 1. If we compare the detailed interaction delays for each execution (Tables B.1 and B.3) we find that the difference is  $x \cdot 2071\text{ms}$  ( $x \in \{0, 1, 2\}$ ). This conforms to the different characteristics of the two approaches: as conventional command-and-control uses indirect activation it requires up to two commands less than command-and-control with random navigation and direct activation. Consequently, the interaction delay of the former is up to two times the interaction delay of an activating command lower than the interaction delay of the latter. The difference of the average values for the two approaches for 0%, however, is with 1900ms lower than 2071ms. This is due to the fact that there are, indeed, some interactions in both approaches which are equal, for instance, the interactions for the execution  $E_{\text{spinner-inc}}$ , or where the interactions differ by two commands for direct activation, e.g., the execution  $E_{\text{dropdown}}$ .

Comparing the minimum, maximum, and the absolute interaction delay values per execution for word error rates other than 0%, we find that the difference is always zero, one, or two times the interaction delay of a direct activation command under consideration of corrections. The difference of the average values is, for the reasons discussed above, always by a specific percentage lower than the interaction delay of one command for direct activation.

In summary, conventional command-and-control has, in average, an interaction delay that is by the interaction delay of one command for direct activation lower than command-and-control with random navigation and direct activation. For some basic executions, both approaches have an equal interaction delay. For other basic executions the interaction delay differs by two times the interaction delay of a command that triggers a direct activation.

## 4.3 Speech-based Mouse Emulation

In this section we will calculate the interaction delay of speech-based mouse emulation approaches. We will first discuss generally applicable conditions for speech-based mouse emulation approaches in section 4.3.1. Then we calculate direction-based mouse emulation with continuous movement (section 4.3.2), direction-based mouse emulation with discrete movement (section 4.3.3), target-based mouse emulation (section 4.3.4) and grid-based mouse emulation (section 4.3.5). In section 4.3.6 we summarize and interpret the results.

### 4.3.1 General Conditions

We now introduce a model of the mouse device, which allows us to explain how mouse functionality is emulated by speech functions. We furthermore define common speech functions and commands for speech-based mouse emulation approaches, and explain how we deal with input fields and spinners, which cannot be operated by just emulating the mouse device.

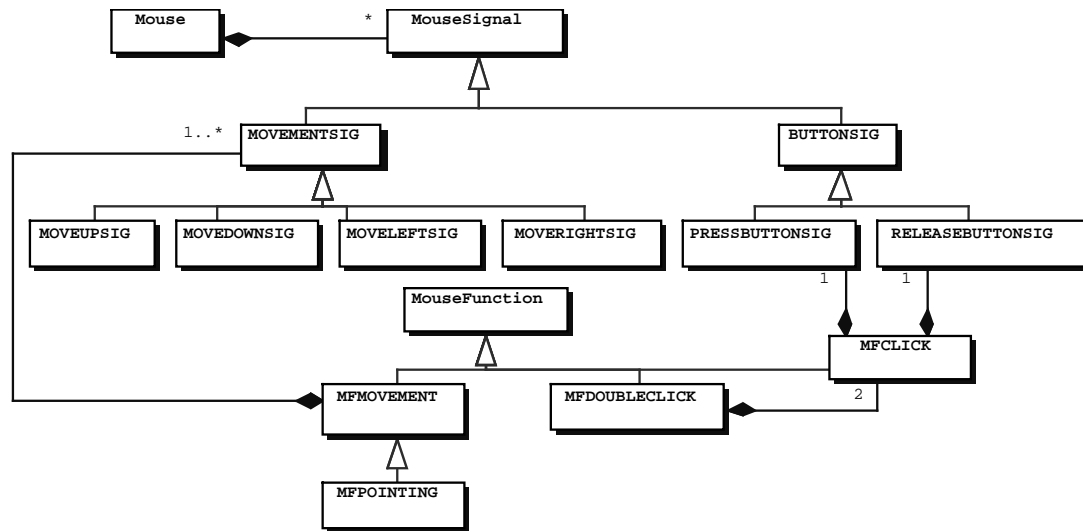


Figure 4.10: Mouse signals and mouse functions.

### Mouse Device Model

A mouse, as discussed in section 1.1, is an electro-mechanical device that sends electrical signals to a computing system, based on how the user physically operates the mouse. By *mouse signals* we understand abstract representations of these electrical signals in a computing system. Mouse signals are divided into *movement signals* and *button signals*. The movement signals *MOVEUP*, *MOVEDOWN*, *MOVELEFT* and *MOVERIGHT* indicate that the mouse device has been moved to the left or right, upward or downward. The button signals *PRESSBUTTON* and *RELEASEBUTTON* indicate the pressing and the releasing of the mouse button<sup>17</sup>. Mouse signals can be composed into *mouse functions* which represent the functionality of a mouse device on a high abstraction level. A *movement* (*MFMOVEMENT*) is a mouse function which represents an arbitrary movement of the mouse. It is composed from a sequence of movement signals. A *pointing* (*MFPOINTING*) is a movement with a minimal number of movement signals which moves the mouse cursor from one position to another. A *click* (*MFCLICK*) is a pair of a *PRESSBUTTON* signal and a *RELEASEBUTTON* signal. A *double click* (*MFDOUBLECLICK*) is a pair of two clicks. Figure 4.10 depicts mouse signals and mouse functions and their relationships as a UML class diagram.

### Common Speech Functions and Commands

Speech-based mouse emulation approaches provide speech functions which generate mouse signals (sequences of mouse signals, respectively) without a physical mouse device being

<sup>17</sup>Modern mouse devices have multiple buttons, correspondingly, each button would send specific instances of *PRESSBUTTON* and *RELEASEBUTTON* signals.

available. A study of literature regarding speech-based mouse emulation reveals that the same speech function for emulating clicks is used throughout different approaches. This speech function generates a *PRESSBUTTON* signal followed by a *RELEASEBUTTON* signal – which makes a click. Therefore, for the remainder of this work, we assume that every speech-based mouse emulation approach provides a speech function called *SFCLICK* which emulates a click. We define that *SFCLICK* is invoked by the command "click", as this command is used in the majority of respective approaches<sup>18</sup>. Thus, speech-based mouse emulation approaches differ in how they emulate movement signals.

### Operating Input Fields and Spinners

Input fields are essential constituents of modern GUIs, however, speech-based mouse emulation, at least in its pure forms (e.g. Karimullah and Sears [95] and Gori et al. [46]), does not provide for inputting alphanumeric characters. Researches have recognized this drawback and combined speech-based mouse emulation with *speech-based keyboard emulation*, e.g., *SUITEKeys* (Manaris et al. [118]). Such combined approaches allow the entering of alphanumeric characters in a manner which is similar to the spelling mode of command-and-control (as defined on page 86). Since the characteristics of speech-based keyboard-emulation are beyond the scope of this dissertation (refer to section 1.2) we assume the spelling mode to be available for input fields and spinners. As such, the speech functions *SFACT<sub>APPEND<sub>x</sub></sub>* and *SFACT<sub>DELETE</sub>* and their corresponding commands *CMDACT<sub>APPEND<sub>x</sub></sub>* and *CMDACT<sub>DELETE</sub>*, are available for every speech-based mouse emulation approach.<sup>19</sup> Speech-based mouse emulation consequently defines 38 common valid commands (1 for the click and 37 for the spelling mode), which are composed from a common vocabulary subset consisting of 39 words (1 for the click and 38 for the spelling mode).

In the following sections we calculate the interaction delay of representative speech-based mouse emulation approaches.

### 4.3.2 Direction-based Mouse Emulation with Continuous Movement

Direction-based mouse emulation with continuous movement (and discrete commands) emulates mouse movements by letting the user utter the direction into which the mouse cursor should continuously be moved. The movement will stop if the user utters a respective stop command. As already described during the introduction of speech functions in section 2.3, the following speech functions are available (e.g., Karimullah and Sears [95]):

1. Start continuous mouse cursor movement to the left (*SFML*).

---

<sup>18</sup>The emulation of a double click or other mouse buttons is done analogously, however, none of the graphical objects that we are considering for our calculations requires a double click or other mouse buttons

<sup>19</sup>In fact, speech-based keyboard emulation is not significantly different from the spelling mode.

2. Start continuous mouse cursor movement to the right (*SFMR*).
3. Start continuous mouse cursor movement upward (*SFMU*).
4. Start continuous mouse cursor movement downward (*SFMD*).
5. Stop mouse cursor movement (*SFSTOP*).

A straight movement of the mouse cursor along one of the screen's axes can correspondingly be emulated by invoking a speech function which sets the cursor in motion, followed by the speech function *SFSTOP* which stops the movement. For the scope of this section we define the abstract speech function *SFMOVEAX* as

$$SFMOVEAX := (SFML|SFMR|SFMU|SFMD), SFSTOP$$

which is invoked by the abstract command  $SFMOVEAX_{cmd}$ :

$$SFMOVEAX_{cmd} := ("left"|"right"|"up"|"down"), "stop"$$

Diagonal movements of the mouse cursor are not supported. An arbitrary pointing, i.e., an arbitrary repositioning of the mouse cursor with a minimum of movements, can therefore be achieved by at most two straight movements which are orthogonal to each other (refer to section 2.3). If the current position and the new position of the mouse cursor lie on a straight line in parallel to the screen's X or Y axis, one straight movement is sufficient. However, in general, the coordinates of current and desired position are a priori unknown. Therefore, for our calculations, we assume the worst case by defining that a pointing requires two straight movements orthogonal to each other. For the scope of this section we define the abstract speech function *SFPOINT* as

$$SFPOINT := (SFMOVEAX, SFMOVEAX)$$

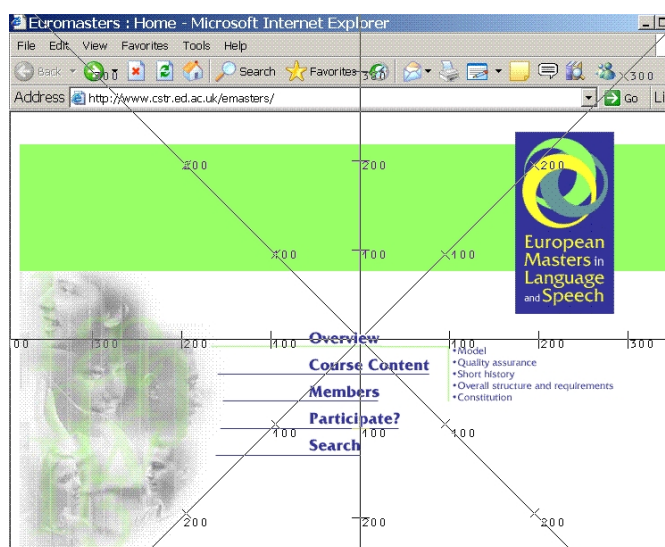
which is invoked by the abstract command  $SFPOINT_{cmd}$ , defined as

$$SFPOINT_{cmd} := (SFMOVEAX_{cmd}, SFMOVEAX_{cmd})$$

With the above considerations, any movement, especially pointings, can be realized with a total of 5 valid commands, which require a vocabulary subset consisting of the 5 words "left", "right", "up", "down" and "stop". Together with the commonly available valid commands, direction-based mouse emulation with continuous movement defines  $38 + 5 = 43$  valid commands of which the longest one has a length of 2 (spelling mode). The required vocabulary contains  $39 + 5 = 44$  words. With the considerations taken in this section we have derived intentions and interactions for the basic executions, and have calculated their interaction delays for different word error rates (refer to appendix B.2, Tables B.5 and B.6, for a detailed documentation). We summarize the results in Table 4.7 which depicts the minimum, the average,

| Interaction Delay in ms | Word Error Rate |       |       |        |       |       |       |       |
|-------------------------|-----------------|-------|-------|--------|-------|-------|-------|-------|
|                         | 0%              | 1%    | 5%    | 10%    | 20%   | 30%   | 40%   | 50%   |
| Minimum                 | 10355           | 10460 | 10900 | 11506  | 12944 | 14793 | 17258 | 20710 |
| Average                 | 11391           | 11506 | 11990 | 40270  | 14238 | 16272 | 18984 | 22781 |
| Maximum                 | 16568           | 16735 | 17440 | 184089 | 20710 | 23669 | 27613 | 33136 |

**Table 4.7:** Calculated minimum, average and maximum interaction delay per basic execution of direction-based mouse emulation with continuous movement.



**Figure 4.11:** The compass mouse.

and the maximum interaction delay per basic execution of direction-based mouse emulation with continuous movement.

### 4.3.3 Direction-based Mouse Emulation with Discrete Movement

Direction-based mouse emulation with discrete movement emulates mouse movements by letting the user utter both the direction into which the mouse cursor should be moved and the relative distance that the movement should bridge. Several variants of this approach exist. They differ in the directions into which the cursor can be moved and the manner by which the relative distance can be specified (refer to the introductory discussion in section 1.2.2).

We base our calculations on a variant called the *compass mouse* (Brøndsted and Aaskoven [27]). We consider it the most generic variant in the sense of a superclass - other variants discussed in section 1.2.2 could be derived from it. The compass mouse considers the current position of the mouse cursor as the intersection point of eight rulers which are oriented like a compass, i.e., one ruler to the north, one to the north-east, etc. (refer to Figure 4.11). Each

ruler marks a point every 100 pixels from the center and the user repositions the mouse cursor by uttering commands of the form

```
<repos> := <direction> <distance>;
<direction> := ("north"|"west"|"east"|"south"|"northeast"|"southwest"|"northwest"|"southeast");
<distance> := ("one" | "two" | ...);
```

An example is the command "northeast two hundred ten", which would position the mouse cursor to the 210th pixel northeast from the center. We represent one repositioning of the mouse cursor using the compass mouse by the abstract speech function *SFMOVECOMP* which is invoked by the abstract command *SFMOVECOMP<sub>cmd</sub>*, defined like *<repos>*. Brøndsted and Aaskoven show that with the compass mouse the mouse cursor can be positioned to every pixel on the screen – which corresponds to a pointing – by uttering two commands *SFMOVECOMP<sub>cmd</sub>* in the worst case<sup>20</sup>. Correspondingly, for the scope of this section, we define the abstract speech function

$$SFPOINT := (SFMOVECOMP, SFMOVECOMP)$$

which is invoked by the abstract command *SFPOINT<sub>cmd</sub>*, defined as

$$SFPOINT_{cmd} := (SFMOVECOMP_{cmd}, SFMOVECOMP_{cmd})$$

For our calculations we assume that *<distance>* represents verbal forms of the numbers 1 through 999, allowing us to specify the distance with a vocabulary of 28 words (see grammar in appendix B.1). As such we cover a maximum screen resolution of at least 999 x 999 pixels and in the following we examine the valid commands and the required vocabulary for the compass mouse.

The mouse cursor can be moved into eight directions. Thus,  $8 \cdot 999 = 7992$  *SFMOVECOMP<sub>cmd</sub>* commands with an average length of 4.7 words exist. Together with 38 commonly available valid commands for the spelling mode and the click, direction-based mouse emulation with discrete movement defines  $7992 + 38 = 8030$  valid commands. The longest valid command has a length of 5, e.g., "north nine hundred ninety nine".

The *SFMOVECOMP<sub>cmd</sub>* commands require a vocabulary subset of 36 distinct words – 8 words for the directions and 28 words for the distance (numbers from 1 - 999). Spelling mode requires a vocabulary subset of 38 words, however, as with command-and-control, these two subsets have a non-empty intersection. The words "one" - "nine" are contained in both subsets. Thus, a common vocabulary subset of only  $36 + 38 - 9 = 65$  words is required. The command for the mouse click adds one additional word, so that direction-based mouse emulation with discrete movement requires a total vocabulary of 66 words.

With the considerations taken in this section we have derived intentions and interactions for the basic executions, and have calculated their interaction delays for different word error rates

<sup>20</sup>Depending on the relative position of current mouse cursor location and desired mouse cursor location one *SFMOVECOMP<sub>cmd</sub>* is sufficient.

| Interaction Delay in ms | Word Error Rate |       |       |       |       |       |        |        |
|-------------------------|-----------------|-------|-------|-------|-------|-------|--------|--------|
|                         | 0%              | 1%    | 5%    | 10%   | 20%   | 30%   | 40%    | 50%    |
| Minimum                 | 10438           | 10864 | 12829 | 16030 | 26471 | 47692 | 95768  | 221628 |
| Average                 | 11481           | 11944 | 14080 | 17558 | 28893 | 51912 | 104037 | 240442 |
| Maximum                 | 16693           | 17342 | 20333 | 25196 | 41000 | 73017 | 145378 | 334513 |

**Table 4.8:** Calculated minimum, average and maximum interaction delay per basic execution of direction-based mouse emulation with discrete movement.

(refer to appendix B.2, Tables B.7 and B.8, for a detailed documentation). We summarize the results in Table 4.8 which depicts the minimum, the average, and the maximum interaction delay per basic execution of direction-based mouse emulation with discrete movement.

#### 4.3.4 Target-based Mouse Emulation

With target-based mouse emulation (e.g. QPointer VoiceMouse [117]), specific areas on the screen, the so called targets, have been assigned speakable identifiers. Targets are for instance graphical objects or specific graphical sub-objects. Whenever the user speaks an identifier, the mouse cursor is placed to a specific position within the target. Thus, target-based mouse emulation allows performing pointings to targets by speaking the identifier of the desired target.

Like with command-and-control approaches we assume well-designed GUIs (refer to section 4.2.1); this means that a maximum of 100 speakable identifiers is needed. This allows us, for the scope of this section, to define the abstract speech function  $SFPOINT$  which emulates a pointing and which is triggered by the abstract command  $SFPOINT_{cmd}$ . The  $SFPOINT_{cmd}$  command is the verbal form of a numeric identifier and, like with command-and-control, we assume a worst case length of  $SFPOINT_{cmd}$  of 2 words.

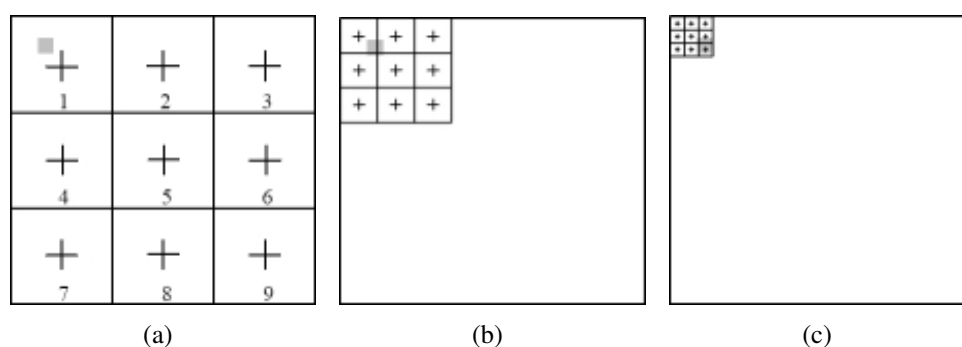
As we have 100 possible speakable identifiers we have 100 valid commands for repositioning the mouse cursor. Together with the commonly available speech functions we have  $100 + 38 = 138$  valid commands for target-based mouse emulation. The speakable identifiers require a vocabulary subset of 28 distinct words and spelling mode requires a vocabulary subset of 38 distinct words. Again, these two subsets have a non-empty intersection. The words "one" - "nine" are contained in both subsets, as such, a vocabulary subset of only  $28 + 38 - 9 = 57$  words is required. The command for the mouse click adds one additional word, so that target-based mouse emulation requires a total vocabulary of 58 words.

With the considerations taken in this section we have derived intentions and interactions for the basic executions, and have calculated their interaction delays for different word error rates (refer to appendix B.2, Tables B.9 and B.10, for a detailed documentation). We summarize the results in Table 4.9 on the facing page which depicts the minimum, the average, and the maximum interaction delay per basic execution of target-based mouse emulation.



| Interaction Delay in ms | Word Error Rate |      |       |       |       |       |       |       |
|-------------------------|-----------------|------|-------|-------|-------|-------|-------|-------|
|                         | 0%              | 1%   | 5%    | 10%   | 20%   | 30%   | 40%   | 50%   |
| Minimum                 | 4713            | 4788 | 5107  | 5563  | 6717  | 8350  | 10791 | 14710 |
| Average                 | 5499            | 5586 | 5958  | 6490  | 7837  | 9742  | 12590 | 17162 |
| Maximum                 | 9426            | 9575 | 10215 | 11126 | 13434 | 16701 | 21581 | 29421 |

**Table 4.9:** Calculated minimum, average and maximum interaction delay per basic execution of target-based mouse emulation.



**Figure 4.12:** Example for grid-based mouse-emulation.

### 4.3.5 Grid-based Mouse Emulation

Dai et al. [44] use a grid to divide the screen into specific regions which are assigned a number. It is assumed that at least one region overlaps with the graphical object under consideration. The user utters the number of a region that overlaps with the graphical object, causing the mouse cursor to be placed to a dedicated position within the respective region, such as the region's center. If the mouse cursor is not yet at the desired position within the graphical object the enclosing region can recursively be divided into smaller regions. The smaller regions are again associated with numbers and the cursor can again be positioned to a dedicated position inside them. Additionally to recursing into regions the user can shift the entire grid along the X and Y axis, i.e., to the left or right, upward or downward. We depict an example of grid-based mouse emulation in Figure 4.12. In Figure 4.12(a) the initial grid is shown. The desired object is marked by the little box and the center of each region is marked with a '+'. Figure 4.12(b) shows the grid after region 1 has been recursed in. Figure 4.12(c) shows the grid after region 1 of the recursive grid has been recursed in - the center of region 9 is now within the gray box.

Correspondingly to the functionality of this approach we define the following speech functions, called *grid functions*, along with corresponding commands, called *grid commands*, to emulate mouse movements.

1. *SFONE*, *SFTWO*, ..., *SFNINE* select one of the nine grid targets. The speech function *SFONE* is triggered by the command "target one", *SFTWO* by the command "target

two", and so on.

2. The speech functions *SFGRIDLEFT*, *SFGRIDRIGHT*, *SFGRIDUP* and *SFGRIDDOWN* shift the grid to the left, right, upward or downward. The corresponding commands are "left", "right", "up" and "down".

For the context of this section we represent a pointing by the abstract speech function *SFPOINT* triggered by the abstract command  $SFPOINT_{cmd}$ . *SFPOINT* is a sequence grid functions and  $SFPOINT_{cmd}$  is a sequence of grid commands. Dai et al. show that every graphical object can be reached by triggering  $N$  grid functions, where

$$N := \log_n \left( \frac{D}{A} \right)$$

and  $A$  denotes the size of the rectangular outline of the graphical object,  $D$  denotes the size of the screen, and  $n$  denotes the number of columns/rows in the utilized grid. They conducted an experiment by which they found out that, using a 3x3 grid, an average number of 3.18 grid commands are necessary to point at targets of sizes between 1.6 and 13.3mm in distances between 18.9 and 57.0mm. Thus, we set  $N = 3.18$ . Dai et al. further found out that 14% of the grid commands were grid shifts. Consequently,  $SFPOINT_{cmd}$  is, in average, composed of 0.45 shift commands and 2.43 target selection commands. Let  $i_{shift}$  be the interaction delay of a shift command and let  $i_{select}$  be the interaction delay of a target selection command, then the interaction delay of a pointing is

$$0.45 \cdot i_{shift} + 2.73 \cdot i_{select}$$

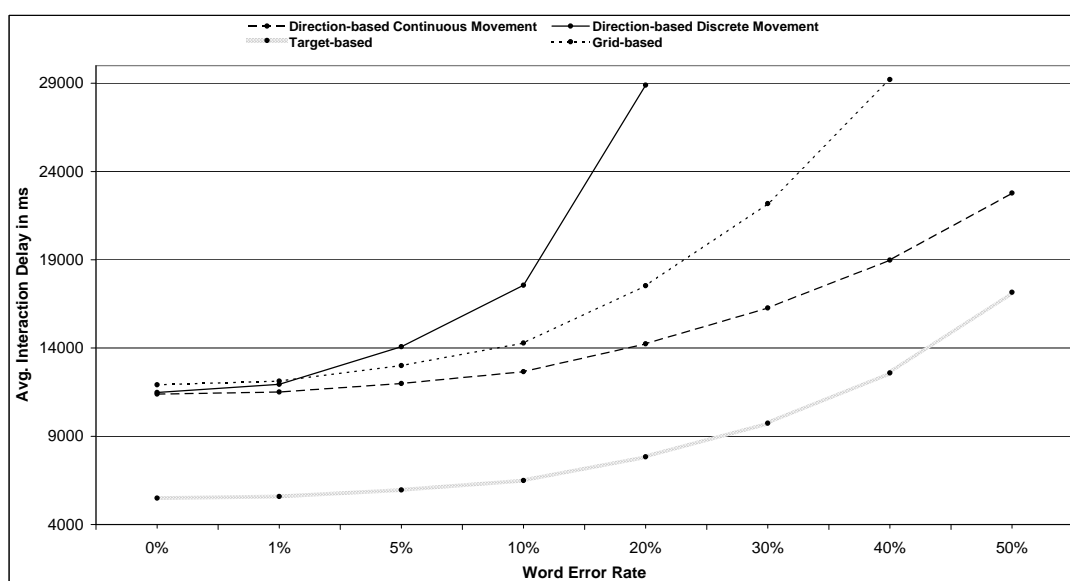
For calculating concrete values for  $i_{shift}$  and  $i_{select}$  we first need to specify the number of valid commands, the longest valid command and the size of the vocabulary. There are 13 grid commands and 38 commonly available valid commands, which makes 51 valid commands in total. The longest valid command has a length of 2, e.g., spelling mode. The grid commands require a vocabulary subset of 14 distinct words, of which 9, i.e., "one" - "nine", are already contained in the vocabulary subset for the spelling mode. This leaves 5 new words which the grid commands add to the vocabulary, so that we arrive at a vocabulary for grid-based mouse emulation that consists of 44 words in total.

Now, a concrete value for  $i_{shift}$  can be calculated by defining a dummy session  $\mathcal{S}_{dummy}$  that is mapped to an intention  $\mathcal{I}_{dummy}$  which consists of a specific grid shift function. The intention  $\mathcal{I}_{dummy}$  is mapped to a interaction with consists of exactly one command, i.e., the command that is required to trigger the grid shift function. The interaction delay of the dummy session  $\mathcal{S}_{dummy}$  corresponds to the value for  $i_{shift}$ . A value for  $i_{select}$  can be calculated analogously. For a detailed documentation of respective values for  $i_{shift}$ ,  $i_{select}$ , and corresponding interaction delay values for pointings consult Table B.12 on page 206 in appendix B.2.

With the considerations taken in this section we have derived intentions and interactions for the basic executions, and have calculated their interaction delays for different word error rates (refer to appendix B.2, Tables B.11 and B.13, for a detailed documentation). We sum-

| Interaction Delay in ms | Word Error Rate |       |       |       |       |       |       |       |
|-------------------------|-----------------|-------|-------|-------|-------|-------|-------|-------|
|                         | 0%              | 1%    | 5%    | 10%   | 20%   | 30%   | 40%   | 50%   |
| Minimum                 | 9423            | 9585  | 10273 | 11263 | 13784 | 17023 | 22839 | 31686 |
| Average                 | 10592           | 10774 | 11533 | 12591 | 15202 | 18545 | 24301 | 33123 |
| Maximum                 | 16436           | 16716 | 17830 | 19232 | 22294 | 26152 | 31612 | 40310 |

**Table 4.10:** Calculated minimum, average and maximum interaction delay per basic execution of grid-based mouse emulation.



**Figure 4.13:** Calculated interaction delay graphs of speech-based mouse emulation.

marize the results in Table 4.10 which depicts the minimum, the average, and the maximum interaction delay per basic execution of grid-based mouse-emulation.

### 4.3.6 Summary

We now discuss the calculation results of direction-based mouse emulation with continuous movement (Cont-ME), direction based mouse emulation with discrete movement (Discr-ME), target-based mouse emulation (T-ME) and grid-based mouse emulation (G-ME). In the following we refer to the graphs in Figure 4.13.

What first strikes the eye is that the average nominal interaction delays (0% word error rate) per execution of Discr-ME, Cont-ME and G-ME are all in a range between approximately 10600ms and 11500ms. In contrast, the nominal interaction delay of T-ME is about 50% lower. As the nominal interaction delay does not consider corrections, the differences in the vocabularies and the valid commands are not significant for this observation. Instead, the rea-

son is the quality of the respective interactions (for a detailed documentation refer to appendix B.2), which we explain in the following. The average number of commands per (basic) execution with T-ME is 2.3 and the average command length with T-ME is 1.5. Cont-ME only uses commands of length 1, but requires, in average, 5.5 commands per execution. Discr-ME uses less commands in average than Cont-ME, i.e., 3.3, however, the average length of the commands is 3.4. With 1.7 the average command length of G-ME is similar to T-ME, however, an average number of 4.9 commands per execution is required to work with the grid. The nominal interaction delay of T-ME is consequently the lowest, because it allows to perform the basic executions with a few short commands. The other approaches require more commands, use longer commands, or both, which, according to the command count- and the command length monotony (Theorems 5 and 4), results in the nominal interaction delay distribution as calculated.

The behavior of the graphs in Figure 4.13 for growing word error rates is particularly interesting. We observe two significant characteristics which we discuss in the following.

First, although the nominal interaction delays of Discr-ME, Cont-ME and G-ME are in the same magnitude, their expected interaction delays grows considerably different of each other for increasing word error rates. Discr-ME increases faster than G-ME, which, in turn, increases faster than Cont-ME.<sup>21</sup> The interaction delay of T-ME can be regarded as increasing similar to the interaction delay of Cont-ME. The reason for these differently growing interaction delays has to be searched in the sets of valid commands. As discussed above, Discr-ME is the approach with the highest average command length per execution. With 3.4 it is significantly higher than for the other approaches. This means that for a specific word error rate the commands of Discr-ME are more likely to be recognized incorrectly than with other approaches, because the command recognition rate decreases with increasing command length. Thus, with Discr-ME more corrections than with other approaches are expected, and consequently more recognition delays and word durations, have to be accounted for. The average command length of G-ME is lower than the one of Discr-ME, therefore its interaction delay grows slower than Discr-ME. It is, however, higher than the average command length of Cont-ME and T-ME, therefore the interaction delays of Cont-ME and T-ME grow slower than the others.

The second significant growing characteristic is that the graphs for Discr-ME and G-ME intersect between a 1% and 5% word error rate. For low word error rates the interaction delay of Discr-ME is lower than the interaction delay of G-ME. For high word error rates this relationship changes the other way round, which can be explained as follows. Discr-ME uses, in average, 3.3 commands per execution with an average length of 3.4. G-ME requires, in average, 4.9 commands of a length of 1.7. The recognition delay is about 2.6 times higher than the duration of a command, therefore, as a matter of the interaction delay model, the nominal interaction delay of G-ME is higher than the nominal interaction delay of Discr-ME. However, as Discr-ME has the higher average command length its commands are more likely to be recognized incorrectly. Thus, with growing word error rate, more corrections are

---

<sup>21</sup>We have omitted drawing Discr-ME and G-ME for high word error rates to keep the scale of the Y axis reasonably small.

expected with Discr-ME than with G-ME, and consequently G-ME has a higher interaction delay than Discr-ME.

In summary, the lowest interaction delay can be achieved with T-ME. The other three approaches have a nominal interaction delay that is about 50% higher than with T-ME. The expected interaction delay of Cont-ME, Discr-ME and G-ME is approximately equal if the speech recognizer has a low word error rate. If the word error rate increases, the interaction delay of Discr-ME increases significantly faster than the interaction delay of G-ME, which increases significantly faster than the interaction delay of Cont-ME.

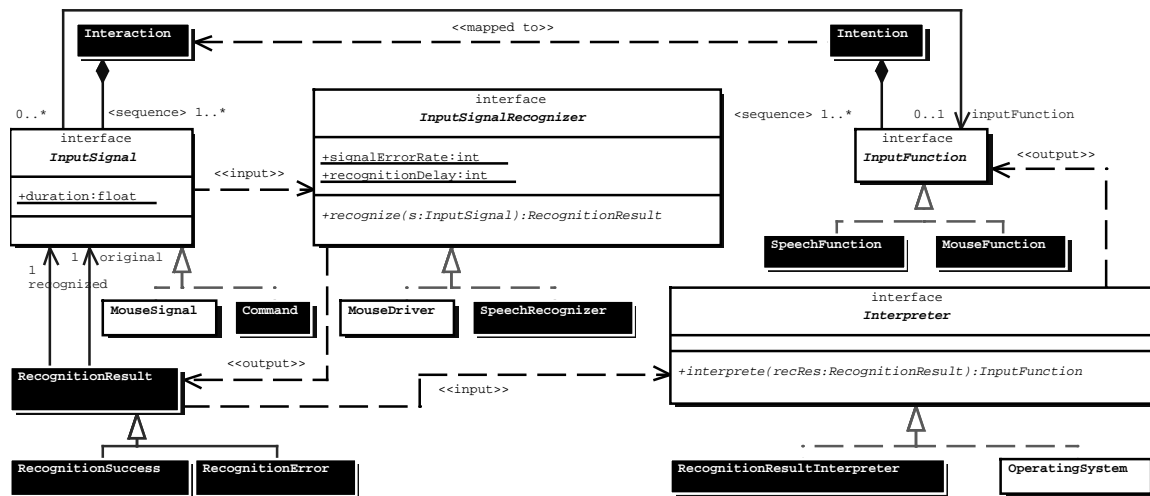
## 4.4 Mouse-controlled GUIs

In this section we estimate the interaction delay of mouse-controlled GUIs. For this to accomplish we present enhancements to the speech-GUI model, which allow for the description of mouse-controlled GUIs and speech-controlled GUIs with a common model. The enhancements are consistent with the interaction delay model, thus, we are able to reuse the formula for the expected interaction delay (definition 4 on page 59) to calculate the interaction delay of mouse-controlled GUIs (section 4.4.2). The enhancements of both models prove their extensibility as required in sections 2.1 and 3.1. In fact, the enhancements do not denote a specialization — they are merely a generalization, since the enhanced models are then applicable to a broader range of interfaces. We summarize our findings in section 4.4.3.

### 4.4.1 Enhancing the Speech-GUI Model

The enhancements of the speech-GUI model were done in four steps. First, we identified which components of mouse-controlled GUIs correspond to speech functions, commands, speech recognizers and recognition result interpreters. Second, we defined appropriate generalizations for corresponding components in speech- and mouse-controlled GUIs. Third, we refactored attributes and methods into the generalizations where appropriate. Fourth, we adapted recognition results, intentions and interactions to reflect the relationships to the new generalizations. The result after performing these four steps is shown in Figure 4.14 on the next page to which we refer in the following. In the Figure, new entities have a white background and already known entities have a black background. Control actions and sessions have been omitted in Figure 4.14 – they remain unchanged as they are already independent of any utilized input modality.

Mouse functions, introduced in section 4.3.1, correspond to speech functions. They are both specializations of so called *input functions* – high level abstractions of input functionality that a specific input modality allows to trigger. For instance, with speech the user is able to trigger a function that starts a continuous movement of the mouse cursor. With the mouse device the user is able to trigger a function that moves the mouse cursor to the right for a specific distance. Correspondingly, intentions now consist of a sequence of input functions instead of a sequence of speech functions (refer to speech-GUI model diagram on page 40). Intentions



```

context MouseDriver
inv: signalErrorRate = 0
inv: recognitionDelay = 0
context SpeechRecognizer
inv: (wordErrorRate >= 0) and (wordErrorRate <= 1)
inv: recognitionDelay = 1500
context Signal
inv: duration >= 0
context Command
inv: duration = spokenWords.iterate(elt; acc : T = 0 | acc + elt)
context RecognitionSuccess
inv: original = recognized
context RecognitionError
inv: original <> recognized
    
```

Figure 4.14: Speech-GUI model enhancements for mouse-controlled GUIs.

do, however, in the special context of speech-controlled GUIs, consist of speech functions.

Commands correspond to mouse signals. They are both specializations of so called *input signals* – high level abstractions of input modality specific signals which are transmitted to a computing system. With speech the user (the microphone or sound card, respectively) transmits audio signals, whereas the mouse device transmits characteristic electrical signals for each operation that the user performs with it. Correspondingly, interactions, to which intentions are mapped, now consist of a sequence of input signals instead of a sequence of commands. Interactions do, however, in the special context of speech-controlled GUIs, consist of commands.

*Input signal recognizers* receive the input signals from the input modality. The input sig-

nal recognizer assumes that the input signals are transmitted via a potentially noisy channel, therefore, it aims at recognizing the original signal that was transmitted in the first place. The speech recognizer is a special input signal recognizer which transforms the vocalizations of commands back into the text that was originally uttered. It has been introduced in section 2.5. The *mouse driver* is the speech recognizers equivalent in mouse-controlled GUIs. It is an input signal recognizer which recognizes mouse signals. Mouse signals are de-facto transmitted via a channel that is not noisy (refer to the RS232 protocol, the PS/2 protocol or the USB protocol for mouse devices), therefore we do not expect a mouse signal to be recognized incorrectly. Furthermore, the recognition of mouse signals does not involve complex models so that the recognition delay of mouse signals tends to be 0 – in fact it is in the magnitude of nanoseconds which is not noticeable by users.

Input signal recognizers generate recognition results, which now have two associations to input signals, called the *original signal* and the *recognized signal*. The original signal is of theoretical nature for being able to specify OCL constraints. It refers to the signal that led to the recognition result, i.e., that was transmitted in the first place. The recognized signal denotes the signal as it was recognized by the input signal recognizer, and as such corresponds to the former association between recognition results and recognized words. Recognition errors and recognition successes keep their semantics in the enhanced model, however, they are now being defined based on the original signal and recognized signal being equal or different.

Recognition results are received by *interpreters*. Depending on the semantic information of the received recognition results, the interpreter determines corresponding input functions and invokes them. A special interpreter is the recognition result interpreter which processes recognition results from a speech recognizer. It has been introduced in section 2.6. The *operating system* of a computing system can be seen as a interpreter for mouse signals, as, depending on which mouse signals it receives, it generates mouse events which manipulate the state of the mouse cursor.

To summarize, we enhanced the speech-GUI model by adding a higher level of abstraction to generalize the core components. The dynamic and static properties of speech-controlled GUIs have been refactored into this new abstraction layer consistently, thus, the interaction delay model can be reused. This allows us to reuse the formula for the interaction delay for calculating the interaction delay of mouse-controlled GUIs. In the following section we calculate the interaction delay of the basic executions under the assumption that a mouse-controlled GUI is available.

#### 4.4.2 Interaction Delay

Before we are able to calculate the interaction delay of mouse-controlled GUIs, we need to determine the duration of mouse signals. Endo et al. [54] performed an experiment with the goal to determine the time which is spent on clicking the mouse button. They found that users needed approximately 500ms to press and release the mouse button. Expressed with the terminology of our model this means that the sum of the duration of the *PRESSBUTTONSIG* signal and the *RELEASEBUTTONSIG* signal is 500ms. Since, in our model, these two signals

occur in the context of a click, where they are in a strict sequence, we model their duration as follows:

```
context PRESSBUTTONSIG
inv:   duration = 250
context RELEASEBUTTONSIG
inv:   duration = 250
```

The duration of a single *MOVEMENTSIG* signal has not yet been determined, however, by using *Fitts' Law* (Fitts [58]), which we explain in the following, we are able to determine the duration of an entire sequence of *MOVEMENTSIG* signals. Fitts' Law is a model of human psychomotor behavior. It models the act of pointing with a hand or finger and can be applied to pointing in the area of computers, for example with a mouse. Fitts' Law predicts how long a human needs to point at a target in a specific distance using specific device. It is based on the distance  $D$  between a specific starting point the center of a target to be pointed at. Fitts' Law, in its original form, considers one-dimensional movements, therefore the target is represented by its width  $W$ . The characteristics of the respective device and the characteristics of the pointing are represented by two regression coefficients  $A$  and  $B$ . They are determined empirically, e.g., MacKenzie et al. [113] have analyzed pointing with a mouse, a tablet and a trackball, which led to respective regression coefficients. With  $T$  denoting the (mean) time for the pointing, the original form of Fitts' Law it depicted below.

$$T := A + B \log_2 \left( \frac{2D}{W} \right)$$

Variations of Fitts' Law, which provide a better fit with empirical observations, have been proposed by Welford [196]

$$T_{Welford} := A + B \log_2 \left( \frac{D}{W} + 0.5 \right)$$

and MacKenzie [114]<sup>22</sup>

$$T_{MacKenzie} := A + B \log_2 \left( \frac{D}{W} + 1 \right)$$

The original form and the variants of Fitts' Law model only one-dimensional pointings, respectively targets. Pointings in our context, however, are two-dimensional, as the user can point upward, downward, to the left or right. Also the targets, i.e., the graphical objects, are two-dimensional. This problem has been recognized by MacKenzie et al. and in [112] they enhance their own variant of Fitts' Law to be able to cope with two-dimensional situations.

---

<sup>22</sup> $T_{MacKenzie}$  is also known as the *Shannon Formulation* as it exactly mimics the information theorem underlying Fitts' Law.



They include a representation of the height  $H$  of a target, so that graphical objects can now be represented by their height and width. The enhanced form, which we call  $T_{2D}$ , is given below.

$$T_{2D} := 230 + 166 \log_2 \left( \frac{D}{\min(W, H)} + 1 \right), \quad [T_{2D}] = ms$$

Experiments performed by MacKenzie et al. in [112] show that the times spans which are predicted using  $T_{2D}$  are about equal to time that passes between the start of the physical movement of the mouse and the arrival of the mouse cursor over the target. From this we conclude for our model that  $T_{2D}$  predicts the cumulated duration of all *MOVEMENTSIG* signals which are involved in a pointing. MacKenzie et al. performed an experiment where they let users point to differently sized targets ( $0.48cm \leq \min(W, H) \leq 3.83cm$ ) in different distances ( $0.96cm \leq D \leq 15.3cm$ ). They obtained a grand mean of 743ms for pointing. For our model we consequently define that, in average, all *MOVEMENTSIG* signals from which a *POINTING* is composed have a cumulated duration of 743ms.

With the considerations taken in this section we have derived intentions and interactions for the basic executions using mouse-controlled GUIs, and have calculated their interaction delays for different word error rates (refer to appendix B.2, Table B.14, for a detailed documentation). With mouse-controlled GUIs the average interaction delay per basic execution is 1450ms. The minimum achievable interaction delay per basic execution is 1234ms, whereas the maximum interaction delay per basic execution is 2486ms.

### 4.4.3 Summary

We described enhancements of the speech-GUI model which allow us to describe speech-controlled GUIs and mouse-controlled GUIs with the same model. To enhance the speech-GUI model we identified corresponding components in both speech- and mouse-controlled GUIs and defined suitable super classes. After having refactored attributes and methods into the super classes where appropriate, we adapted any affected component to reflect relationships to the new super classes. The changes integrated seamlessly into the interaction delay model. A design goal of the speech-GUI model was the extensibility to other speech-controlled GUI approaches (section 2.1). While the extension to mouse-controlled GUIs is not a direct support of this design goal, it does show that the speech-GUI model is even extensible to completely different GUI types. The extensibility to other speech-controlled GUI approaches will, however, be shown in chapter 5.

Users of mouse-controlled GUIs do not have to deal with corrections, at least not if the mouse hardware and the underlying mouse driver is functioning correctly. Therefore, the nominal interaction delay formula can be applied to calculate the interaction delay of mouse-controlled GUIs. To facilitate these calculations we derived the duration of mouse signals from empirical experiments documented in relevant literature. Our calculations resulted in an average interaction delay per execution of 1450ms. The minimum (maximum) interaction delay per execution we found to be 1234ms (2486ms).

## 4.5 Discussion

The minimum, average and maximum interaction delays per execution of all speech-controlled GUI approaches are significantly higher than the corresponding values of mouse-controlled GUIs. Our calculations show that for specific executions the interaction delay with speech-controlled GUIs is about 50% higher than with mouse-controlled GUIs. An example is the clicking of a button ( $E_{button}$ ): with conventional command-and-control an interaction delay of 2642ms could be calculated, whereas the interaction delay with mouse-controlled GUIs is 1234ms. This coherence might be interpreted as such that for specific application domains and GUI instances the 50% increased task completion time with speech-controlled GUIs compared to mouse-controlled GUIs (refer to section 1.3) is due to a 50% increase of the interaction delay. Generally speaking, however, we recognize that the average interaction delay per basic execution is far more than 50% higher than with mouse-controlled GUIs. Provided that there is a (somewhat linear) dependency between task completion time and interaction delay, these results denote that the 50% increase in task completion time for speech-controlled GUI is the best case that can currently be achieved with any speech-controlled GUI approach.

Amongst the approaches that have been calculated, conventional command-and-control has the lowest minimum, average and maximum interaction delay per basic execution. The interaction delay of target-based mouse emulation is close to the interaction delay of command-and-control with random navigation and direct activation. The reason is that both approaches use verbal forms of speakable identifiers as commands for navigations. Both approaches use exactly one command of length 1 to perform a specific activation. A majority of their interactions is equal, considering the number and length of commands, but for some specific executions, e.g.,  $E_{spinner-inc}$ , target-based mouse emulation uses less commands than command-and-control with random navigation and direct activation. This leads to the observation that both approaches have identical minimum and maximum interaction delays per basic execution, however, the average interaction delay of target-based mouse emulation is lower than of command-and-control with random navigation and direct activation.

The minimum, average, and maximum interaction delay of all remaining speech-based mouse emulation approaches (other than target-based mouse emulation) are significantly higher than of command-and-control.

A tempting hypothesis, motivated by these results, is that the relative distribution of the average interaction delays per basic execution is equal to the relative distribution of task completion times that can be achieved with the respective approaches. A direction of research that we consequently motivate is the empirical examination of the impact that the interaction delay has on the task completion time in general. We propose, as future work, to conduct a user study in several different application domains with mouse-controlled GUIs and different speech-controlled GUI approaches. The goal of this study is to support or reject the hypothesis that there is a general formal relationship between the interaction delay and the task completion time.

In the following chapter we will use the calculation results obtained for command-and-control to motivate our proposed improvements.

# 5

## Conversation-and-Control

*"Initiative in a conversation occurs at an instant of time when a person seizes control of the conversation by making an utterance that presents a domain goal for the participants to achieve"*

Robin Cohen [39]

### Overview

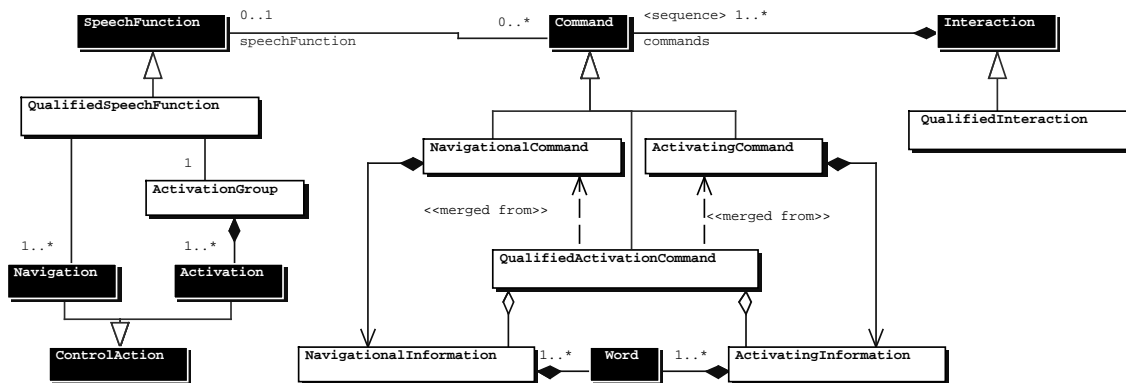
In this chapter we introduce a new speech-controlled GUI approach, which we call *conversation-and-control*. It is based on an extension of conventional command-and-control by three techniques, called *qualified activation*, *automatic information error recovery* and *qualified feedback*, which we motivate from the interaction delay calculation results from the previous chapter. At first we present qualified activation, which can be seen as the core principle for conversation-and-control. By qualified activation we reduce the average number of commands which are necessary for performing a basic execution. Then we present automatic information error recovery and qualified feedback, which emerge from qualified activation and which avoid, if possible, the rejection of recognition errors. As such, both techniques contribute to reducing the average number and length of corrections – which in turn decreases the interaction delay. By the application of qualified activation, automatic information error recovery and qualified feedback, the interaction with a command-and-control interface obtains the flavor of a conversation, i.e., a dialog, of which we present a model in this chapter. We conclude the chapter which discussing the characteristics of the conversation-and-control dialog model and by comparing it to relevant pieces of related work.

## 5.1 Qualified Activation

In section 4.2 (appendix B.2, respectively) we have derived interactions, i.e., command sequences, for basic executions using command-and-control. On close examination of these interactions we recognize two types of commands. The types can be distinguished by the information which the words of the command represent and by the speech functions which the commands trigger. With the first command type the words represent *navigational information*, such as the speakable identifier of the graphical object of which an activation should be performed. Such commands exclusively trigger the performing of navigations and we call them *navigational commands*. With the second command type the words represent *activating information*, such as an identifier of the activation class that is referred to (e.g., "select" or "push"). Such commands exclusively trigger the performing of activations and therefore we call them *activating commands*. There is no overlap between these two command types, i.e., a command is either a navigational or an activating command. We further observe that activating commands are preceded by a set of navigational commands, which identify the graphical object to which the activation, that the activating command refers to, belongs (i.e., which set the focus).

In order to reduce the interaction delay of command-and-control we merge activating commands and their preceding navigational command(s) into a single command. This reduces the number of commands in the interaction, and thus, according to the command count-monotony (Theorem 5), the interaction delay decreases. Consider a sequence of  $n, n \in \mathbb{N}^+$  corresponding navigational and activating commands. Then, assuming that the merged command would contain all the words from the originating commands, the interaction delay would decrease by  $n - 1$  times the recognition delay – since a single command instead of  $n$  commands has to be recognized. As a result, explicit navigational commands and explicit activating commands become obsolete. Every merged command now de facto refers to an activation and contains respective navigational and activating information, such that a specific activation instance can unambiguously be inferred from the merged command. We call such a merged command a *qualified activation command* and the principle of *qualified activation* denotes, that an interaction may only consist of qualified activation commands. We consequently refer to such an interaction as *qualified interaction*.

The speech functions which qualified activation commands trigger are called *qualified speech functions*. A specific qualified speech function can be seen as the merger of the speech functions, which the separate navigational and activating commands, from which the qualified activation command has been merged, would have triggered. As such, a qualified speech function performs exactly those control actions which the separate speech functions would have performed. In particular, qualified speech functions perform multiple navigations and one activation, since a qualified activation command has been merged from multiple navigational commands and one activating command. However, the particular activation which a qualified speech function performs, is not predetermined, because we define that qualified speech functions may logically group activations of the same type into *activation groups*. The particular activation from the activation group of a qualified speech function is then selected



**Figure 5.1:** Enhancements of the speech-GUI model induced by qualified activation.

by specific information contained in the qualified activation command and we will go into the details of this selection process later in this section. As such, there is a 1:N relationship between qualified speech functions and activations. An example is the spinner. It provides 10 activations of type *APPEND* of which each appends a specific digit between 0 and 9 to the spinner. These activations form a specific *activation group* and the specific digit that should be appended is expected to be specified as part of the activation information of the qualified activation command. Activation groups can also just contain a single activation, i.e., if there is only one activation of a particular type, such as the *PUSH* activation of a button. The concepts which have been introduced so far are described in UML in Figure 5.1. In the Figure, entities which are already known from the speech-GUI model (Figure 2.21), are drawn with a black background.

Per definition, a qualified activation command contains the navigational and activating information which was formerly distributed over multiple commands. Thus, we expect a qualified activation command to be longer than the lengths of the original commands, which might compensate the interaction delay reduction achieved by the merger: the increased command length increases the duration of the command, and consequently, according to the command length-monotony (Theorem 4), the interaction delay increases. We go against the compensation effect by hypothesizing that it is possible to construct qualified activation commands in a way so that the interaction delay achieved by the command merger is practically not compensated. This hypothesis, which we call *construction hypothesis*, is motivated by the recognition delay being about 2.6 times higher than the duration of a word. Thus, for example, given 2 original commands, a qualified activation command can have 2 more words than the sum of the lengths of the original commands – the interaction delay would still be reduced by 0.6 times the duration of a word (analogously for more than 2 original commands). We prove the construction hypothesis in section 6.1.3, where we present a framework, which defines qualified activation commands for graphical objects from the Swing catalog.

Qualified activation is related to indirect activation, which conventional command-and-control is based on. Both concepts allow navigating and activating a graphical object with

one command, however, indirect activation persistently couples a specific navigation and a specific activation. In contrast, qualified activation allows combining any navigation with any activation. An example is the tree node: with indirect activation only the expansion and the collapsing of the tree node can be done by a single command (refer to Table 4.4). For selecting and deselecting, two separate commands are required. With qualified activation, however, the navigational information for the specific tree node can be combined with the activating information that either expands, collapses, selects, or deselects the tree node. In particular, with qualified activation the user could utter "expand <nodeid>", "collapse <nodeid>", "select <nodeid>" or "deslect <nodeid>", where <nodeid> is the speakable identifier of the tree node. Thus, we consider qualified activation to be a generalization of indirect activation.

Qualified activation breaks with the navigation precedence principle of command-and-control (refer to section 4.2.1), which denotes, that graphical sub-objects cannot be focused before their enclosing graphical object has been focused. Qualified activation further requires that any graphical object has a speakable identifier at any time. An example is the tree node just discussed: once the user knows its speakable identifier, it can be expanded, collapsed, selected or deselected without having to focus the actual tree. In contrast, with command-and-control, the user first would have had to focus the tree with one command and then expand the tree node with another command. It is important to note that qualified activation still fulfills the cascading navigation requirement (refer to section 4.1.2), i.e., the requirement that intra-object navigations must be preceded by inter-object navigations – it is just that now a single command is required to perform inter- and intra-object navigation.

Qualified activation commands can, in principle, be interpreted in the same way, in which the commands for other speech-controlled GUI approaches are interpreted. For instance, the recognition result interpreter could derive the set of valid commands and match each newly available recognition result against it. Upon a match, the qualified speech function, that the matching valid command is associated with, could be triggered. Otherwise, the qualified activation command could be rejected. Rejections can occur due to *information errors*, that is, if the user does not specify the entire navigational and activating information that is required, if the user incorrectly combines navigational information with activating information, or if a recognition error leads to one of the two former conditions. We find that such a rigid rejection behavior – rejection without informing the user why the command was rejected – results in bad usability. We will therefore show in the following sections how the system can detect and handle information errors gracefully, i.e., how the system can automatically or in cooperation with the user resolve the information error. As a side effect, the user no longer needs to repeat commands for correcting information errors (at least for specific conditions). Instead, the user can utter corrections – if corrections need to be uttered at all – which are shorter than the originally uttered command, thus, the interaction delay is further reduced.

## 5.2 Interpretation of Qualified Activation Commands

We now derive the theoretical framework for the interpretation of qualified activation commands, based on which we will later in this chapter introduce automatic information error recovery and qualified feedback. We will first present a general model for the interpretation in section 5.2.1. Based on a sample GUI, which we present in section 5.2.2 and which we will use for our explanations throughout the remainder of this chapter, we define two concrete principles for the interpretation of qualified activation commands in sections 5.2.3 and 5.2.4.

### 5.2.1 Interpretation Model

We will first define a data structure by which we represent navigational and activating information. Then we will present an abstract process by which navigational and activating information is extracted from new recognition results and evaluated.

#### Interpretation Data Structure

Navigational information consists of words which represent a speakable identifier. The words of which activation information consists can be separated into two categories. The first category is called the *activation name* and encompasses words which represent an identifier for the type of the activation, e.g., "select" for the *SELECT* activation. The second category is called *parameter information*. It consists of words which further specify a specific activation from an activation group, e.g., the digit to identify a specific *APPEND* activation

For a specific qualified speech function to be determined, exactly the above introduced information is necessary. As such, a qualified speech function instance can be determined by a speakable identifier, an activation name and parameter information – if it has parameters at all. We refer to these three types of information as the *qualifier* of the qualified speech function and model the qualifier as a set of name-value pairs, which we call *slots*. The qualifier consists of a *speakable identifier slot*, an *activation name slot* and multiple *parameter slots*. The speakable identifier slot and the activation name slot are called the *identification slots*. If there is currently no value associated with a slot then we say that the slot is *empty*. Otherwise, we say that the slot is *populated*. Populating a slot means assigning a value to a slot that was previously empty.

Depending on which slots of a qualifier are populated, we define different *qualifier states*. The *empty* state denotes that no slots of the qualifier are populated, whereas the *populated* state denotes that all slots of the qualifier are populated. If the state of a qualifier is not empty and not populated, we call its state *partially populated*, which is an abstraction of two sub-states called *parameters incomplete* and *parameters complete*. *Parameters incomplete* denotes, additionally to the qualifier being partially populated, that at least one parameter slot is empty. *Parameters complete* denotes that, indeed, the qualifier is partially populated, but all parameter slots are populated. In other words, *parameters complete* denotes that the parameter slots are all populated but at least one identification slot is empty.

The concepts which we have introduced so far are summarized as a UML class diagram in

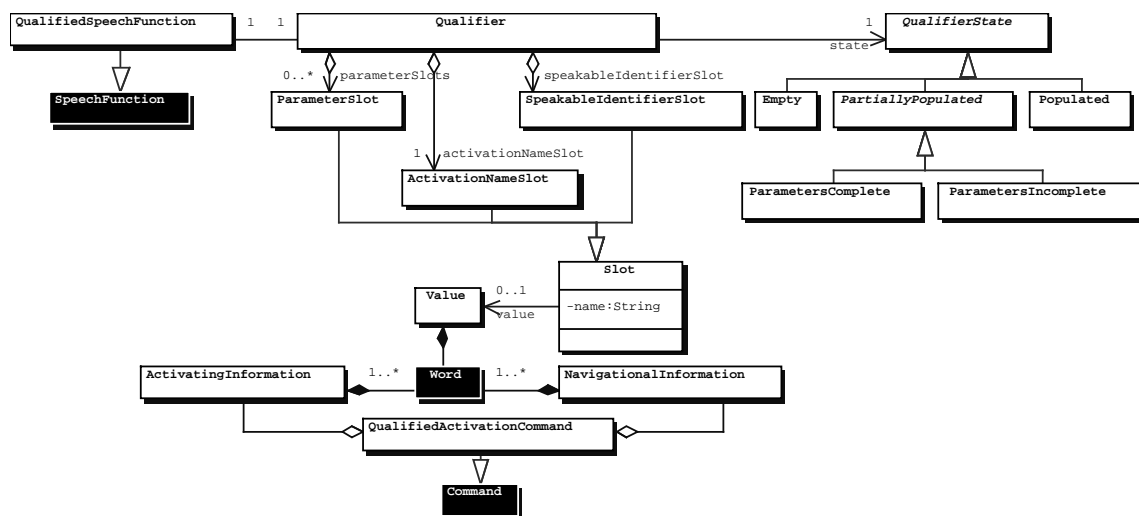


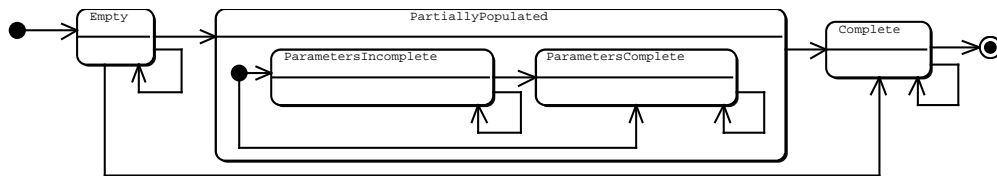
Figure 5.2: Qualified activation commands and qualified speech functions.

Figure 5.2. The Figure specifies a general data structure that we will use to specify an abstract interpretation process of recognition results, which we describe in the following.

### Abstract Interpretation Process

We define that each qualified speech function has a *semantic analyzer*, which provides an operation called *semantic update*. This operation takes a recognition result as input and performs the following two actions. First, semantic update extracts exactly this navigational and activating information from the recognition result, which the associated qualified speech function requires. Therefore, knowledge about the syntactic structure of the information must be encapsulated by the semantic analyzer. We do, however, not specify how this knowledge is represented. A specific semantic analyzer could use string matching or keyword spotting, while other semantic analyzers could use more advanced technologies like robust parsing or probabilistic grammars (Huang et al. [78]). By the second action of semantic update the semantic analyzer populates the slots of the qualifier of the associated qualified speech function with the previously extracted information. Therefore, the semantic analyzer also requires the encapsulation of semantic knowledge, since it must know which syntactic constructs may be assigned to which qualifier slot. In other words, the semantic analyzer requires to know which syntactic constructs represent a speakable identifier or an activation name. Again, we do not specify how this semantic knowledge should be represented. While both syntactic and semantic knowledge can be kept separate on the representational level it is possible – even more feasible due to performance reasons – to use one representation for both types of knowledge (McTear [124]). We argue that the solutions for representing syntax mentioned above are also suitable to represent the semantic knowledge, e.g., grammars can be annotated by rules which populate slots upon productions which recognize specific syntactic constructs.





**Figure 5.3:** Qualifier state transitions.

Based on the specific slots which get populated by a semantic update, the qualifier undergoes specific state transitions, which are depicted in Figure 5.3. Initially the qualifier is in the empty state. The transition from empty to populated denotes that a semantic update has populated all slots at once. If the semantic update could not populate any slot, the qualifier remains in the empty state. Otherwise, i.e., if the semantic update could populate some slots, the qualifier moves on to the partially populated state. In the partially populated state, depending on whether all parameter slots could be populated or not, the qualifier transitions into the parameters complete or the parameters incomplete state. Upon subsequent semantic updates, the qualifier might remain in its current state or transition towards the populated state as depicted in Figure 5.3. It is important to note that there are no backward transitions – once a slot has been populated, it cannot be unpopulated.

With the considerations from above, instances of qualified speech functions, qualifiers and semantic analyzers always occur in triplets, and are, within a triplet, associated amongst each other. We call such a triplet a *conversation topic* and there is one conversation topic per qualified speech function that is available. The conversation topic, to which we shortly refer to as *topic* in the following, is de facto a composition of one qualified speech function, one qualifier and one semantic analyzer. Therefore, for the remainder of this document, we make the following terminology definitions: the *state of a topic* denotes the state of its qualifier. The *semantic update of a topic* denotes executing semantic update of the semantic analyzer of the topic. *Triggering a topic* denotes triggering the qualified speech function of the topic, and since there is a 1:1 relationship between topics and qualified speech functions, we say that a speech-controlled GUI provides a specific number of topics. The *population of a topic* denotes the population of its associated qualifier. The conversation topic and its relationship to the other discussed entities is depicted in Figure 5.4 on the following page as a UML class diagram.

We call the specific qualified speech function, which the user intends to trigger by a qualified activation command, the *intended speech function*. Thus, the intended speech function is represented by a specific topic, which we call the *intended topic*. Once the intended topic has been determined, it can be triggered. Initially, e.g., after system boot-up, no information about the intended topic is available – it could be any topic that is provided by the speech-controlled GUI. Once the user has uttered a qualified activation command, the information from the corresponding recognition result can be used to determine the intended topic as follows. We define that a newly available recognition result is used to perform semantic updates of any available topic, which possibly leads to a population of some of them. A specific topic only

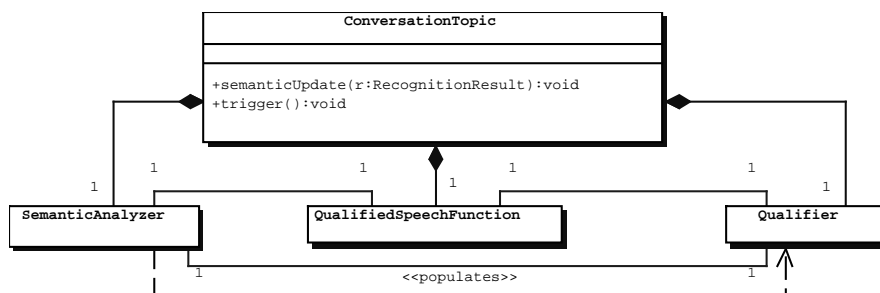


Figure 5.4: Conversation topic.

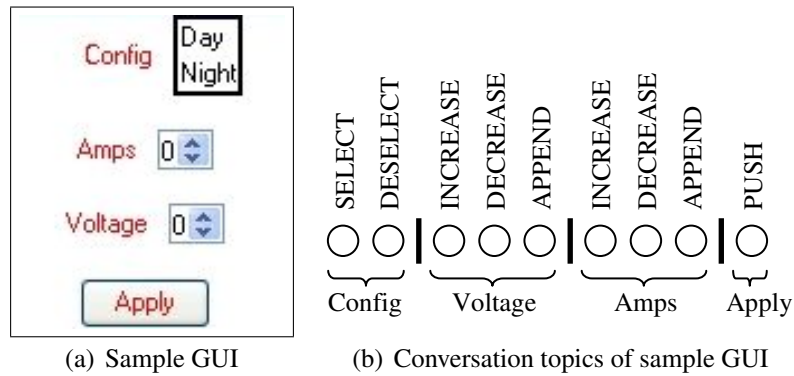
gets populated if the recognition result contains information, which the topic "understands". Thus, the "degree" of population of a topic – which is reflected by its state – is an indication regarding the intended topic. A topic which is empty after a semantic update indicates that the recognition result did not contain any meaningful information for it. Thus, this topic is most likely not the intended topic. If a topic is partially populated it is a candidate for the intended topic, since the recognition result apparently contained some meaningful information for it – however, not the entire required information. A topic being in the populated after a semantic update is most likely the intended topic.

Depending on which specific topics are available and depending on which specific information is contained in the recognition result, various distributions of topic states can occur after a semantic update. In the following we will discuss specific topic state distributions, which we illustrate based on the example of a sample GUI introduced in section 5.2.2. In section 5.2.3 and 5.2.4 we will define two heuristics based on specific topic state distributions, called *isolated-topic-distribution heuristic* and *peering-topic-distribution heuristic*, which the intended speech function can immediately inferred. Any other topic state distribution will be considered as a manifestation of an informational error and we will present mechanisms by which these error conditions can be resolved automatically (section 5.3) or be resolved with subsequent interaction with the user (section 5.4).

## 5.2.2 Sample GUI

Figure 5.5(a) on the next page contains the GUI based on which we will illustrate our explanations in the following. The GUI consists of a button, labeled "Apply" (apply button), two spinners, which are labeled with "Voltage" (voltage spinner) and "Amps" (amps spinner), and a list labeled "Config" (config list) with two options "Day" and "Night" (day option and night option). We assume, without restricting the generality, that the labels of the graphical objects are their speakable identifiers.

The spinners can each be represented by three topics, which we name after the activations which their qualified speech functions define (activation group types, respectively). As such, each spinner provides the *INCREASE*-topic, the *DECREASE*-topic and the *APPEND*-



**Figure 5.5:** Sample GUI for illustrating conversation topic distributions.

topic. We omit the *DELETE*-topic it will not provide further insights into our explanations. The *APPEND*-topic defines one parameter, named *digit*, which it refers to the actual digit that should be appended to the spinner – thus, it identifies the specific  $APPEND_x$  activation ( $x \in \{0, 1, 2, \dots, 9\}$ ) in the activation group of the *APPEND*-topic. The apply button defines a single topic, namely the *PUSH*-topic. The list provides the *SELECT*-topic and the *DESELECT*-topic, each of which has a parameter named "option". This parameter identifies the specific select-items, i.e., option, that should be selected or deselected. For simplifying further discussions we will denote topics and their respective states as depicted below.

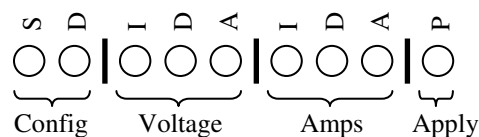
Empty:  ParametersIncomplete:  ParametersComplete:  Populated:

Consequently, the sample GUI can be described by the notion in Figure 5.5(b). In the following two sections we will describe two topic state distributions, from which the intended topic can immediately be inferred.

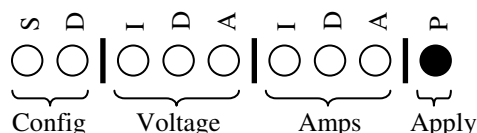
### 5.2.3 Isolated-Topic-Distribution Heuristic

We call a topic state distribution where a single topic is in the populated state and all other topics are in the empty state the *isolated-topic distribution*. We define that if after a semantic update the isolated-topic distribution can be detected, the intended topic is the populated topic. Although this definition is straight-forward, it is still a heuristic, because upon a misunderstanding, the populated topic is in fact not the topic intended by the user. Therefore, we call this approach to infer the intended topic the *isolated-topic-distribution heuristic*.

Let us assume that the sample GUI from Figure 5.5 has just booted up. The respective topics in their initial empty state are depicted below:



Let us now assume that the recognition result "push apply" becomes available. It contains the word "push", which the semantic update of the *PUSH*-topic will recognize as its activation name. The recognition result further contains the word "apply" which the semantic update of the *PUSH*-topic will recognize as its speakable identifier. Thus, both identification slots of the *PUSH*-topic are populated. As the *PUSH*-topic has no parameter slots it transitions to the populated state immediately. Any other topic does not "understand" the recognition result, thus, all other topics remain empty, as depicted below.



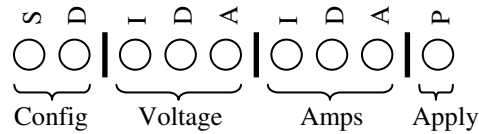
This topic state distribution matches the definition of the isolated-topic-distribution heuristic as only the *PUSH*-topic is populated and all other topics are empty. Consequently, the *PUSH*-topic is inferred per definition as the intended topic. Assuming that the user really has uttered "push apply" the user's intention has been correctly inferred. Otherwise, a misunderstanding has happened, which we deal with in section 5.6.2.

Since now the *PUSH*-topic is the intended topic, it gets triggered. This may, in general, cause changes in the state of the GUI. Consequently, the available topics must be regenerated as new graphical objects might become visible, or currently visible graphical object might become invisible or change their state. However, for our explanations, we assume that the sample GUI does not change its state, i.e., that it always consists of the very same graphical objects as in Figure 5.5(a). Therefore, the same topics as in Figure 5.5(b) are available after the triggering of the *PUSH*-topic. In the following section we introduce another topic state distribution upon which a heuristic must be applied to infer the intended topic.

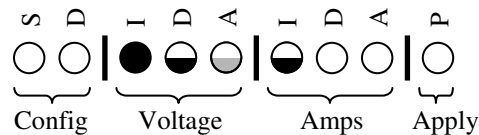
## 5.2.4 Peering-Topic-Distribution Heuristic

We call a topic state distribution where a single topic is in the populated state and all other topics are in any but the populated state the *peering-topic distribution*, as virtually the populated topic peers out of the other topics. We define that if after a semantic update the peering-topic distribution can be detected, the intended topic is the populated topic. This is a heuristic definition, since obviously the information that matches the populated topic completely was also "understood" by other topics, however, not entirely. Therefore we call this approach to infer the intended topic the *peering-topic-distribution heuristic*. As we will see with the following example, the peering-topic-distribution heuristic is practical.

Let us again consider our sample GUI from Figure 5.5(a). Since in the previous example the intended topic was triggered, all topics have now been re-derived (reset, respectively) as depicted below:



We now assume that the recognition result "increase voltage" becomes available. It contains the word "increase" which represents the activation name of all *INCREASE*-topic, thus, the activation name slots of any *INCREASE*-topic gets populated with the value "increase". The word "voltage" corresponds to the speakable identifier of the voltage spinner, and consequently, the speakable identifier slots of any topic that refers to the voltage spinner gets populated with the value "voltage". The topic state distribution after the semantic update is depicted below:



The *INCREASE*-topic of the voltage spinner is populated because both its activation name slot and the speakable identifier slot are populated (there are no parameter slots for this topic). The *DECREASE*-topic of the voltage spinner is partially populated (indicating parameters complete as there are no parameter slots) because its speakable identifier slot contains "voltage". The *APPEND*-topic of the voltage spinner is also partially populated, because its speakable identifier slot contains "voltage". Since it has parameter slots, but none of which are populated, it indicates parameters incomplete. The *INCREASE*-topic of the amps spinner is partially populated (indicating parameters complete as there are no parameter slots) because its activation name slot contains "increase". There is consequently a single topic in the populated state and the others are in different states, which matches the criterion for the peering-state distribution. Consequently, the *INCREASE*-topic of the voltage spinner is selected as the intended topic and gets triggered. Assuming that the user really has uttered "increase voltage" the user's intention has been correctly inferred. Otherwise, a misunderstanding has happened, which we deal with in section 5.6.2.

### 5.2.5 Summary

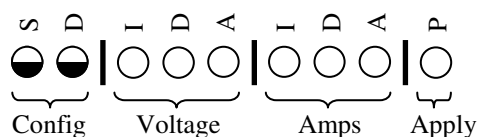
With the abstraction of the conversation topic and the conversation topic state distribution we have presented a simple, yet effective, model for the interpretation of qualified activation commands. On the example of a sample GUI we have introduced the isolated-topic-distribution heuristic and the peering-topic-distribution heuristic, which, if they apply, allow the direct inferring of the intended topic. In the following section we will show how the system can detect information errors and aim at automatically resolving them.

## 5.3 Automatic Information Error Recovery

We consider any topic state distribution other than the isolated-topic distribution and the peering-topic distribution to be an information error. In general, two cases can happen: either there is more than one topic in the populated state or there is no topic in the populated state. We call the former case – two or more topics populated – the *illegal distribution*, as it denotes a situation where the information from the recognition result completely matched the information required for the triggering of two different qualified speech functions. It is illegal, because of the recognition result were a recognition success, two or more populated topics are a strong indication for a GUI design error. However, the illegal distribution could also have been caused by a recognition error and we will see in section 5.6.2 how the system deals with it. The case that no topic is in the populated state remains, and in the following we will present the *relative-maximum-identification heuristic* (section 5.3.1) and the procedure *resolution-by-historical-topic* (section 5.3.2) for automatically recovering from the information error. We summarize the heuristic and the procedure by the term *automatic information error recovery*.

### 5.3.1 Relative-Maximum-Identification Heuristic

We derive the relative-maximum-identification heuristic considering our sample GUI from Figure 5.5 as example, and similar to the beginning of the previous two examples, we begin with all topics being in the empty state. Now we assume that the recognition result "select night" becomes available. The semantic update with this recognition result causes the activation name slot of the *SELECT*-topic of the config list to be populated. It furthermore causes the option parameter slot of both the *SELECT*- and the *DESELECT*-topic of the config list to be populated, which leads to the following topic state distribution:

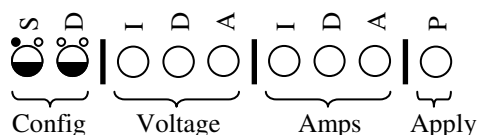


We recognize that no topic is in the populated state and multiple topics are in the partially populated state. Consequently, the state distribution does neither meet the criteria for the isolated-topic-distribution heuristic, nor the peering-topic-distribution heuristic. Since the state distribution is not illegal we are facing an information error. From a human perspective, what should happen is obvious – the night option from the config list should be selected for two reasons: first, there is not other thing that can be selected ("select"), and second, there is no other thing that has the night option ("night").

In order to formalize this human inferring we define the relative-maximum-identification heuristic. The precondition for the application of this heuristic is the presence of an information error. In the following we call all partially populated topics of an information error the *ambiguous topics*. The key idea of the relative-maximum-identification heuristic is to deter-

mine the topic which has the most identification slots populated<sup>1</sup>. If exactly one such topic exists it is considered to be the intended topic. As such, the heuristic basically counts the number of populated identification slots for each ambiguous topic and determines the maximum, called  $id_{max}$ . If there is exactly one topic which has  $id_{max}$  identification slots populated, then it is considered to be the intended topic. Otherwise, if multiple topics with  $id_{max}$  identification slots populated exist, the relative-maximum-identification heuristic has failed and other mechanism, as described later in this chapter, will be applied.

Going back to our example, the *SELECT*- and the *DESELECT*-topic of the config list are ambiguous. The *DESELECT*-topic has no identification slots populated – only the parameter slot is populated. The *SELECT*-topic has one identification slot populated, i.e., the activation name slot. To visualize the population state of the identification slots we will augment our notation of topics as follows. A small circle left of the topic represents the activation name slot, a small circle right of the topic represents the speakable identifier slot. If a small circle is filled, the respective slot is populated, otherwise empty. Thus, our current topic state distribution look as follows:



According to the relative-maximum-identification heuristic, the *SELECT*-topic of the config list is selected as the intended topic. It is not populated, however, it can be triggered right away since all parameter slots are populated. Otherwise, the parameter slot value would still be unclear, and it would have to be determined by a mechanism that we describe in section 5.4.3.

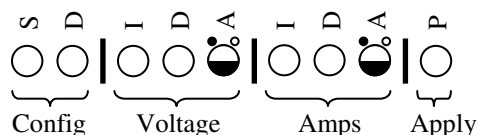
In the following section we examine an information error where the relative-maximum-identification heuristic fails and show how the information error can automatically be resolved by considering the previously triggered topic.

### 5.3.2 Resolution-by-Historical-Topic

Similar to the relative-maximum-identification heuristic we derive resolution-by-historical-topic by considering our sample GUI from Figure 5.5 as example. Again, we begin with all topics being in the empty state. We assume, as precondition, that the recognition result "decrease amps" has become available previously, which, according to the mechanisms discussed above, has caused the triggering of the *DECREASE*-topic of the amps spinner. We now assume that the user intends to append the digit '5' to the amps spinner. For this, the user utters the command "append five" and we assume that the recognition result "append five" becomes available. The semantic update with this recognition result has two consequences. First, the *APPEND*-topics of both the amps and the voltage spinner get their activation name

<sup>1</sup>Identification slots are, according to section 5.2.1, the activation name slot and the speakable identifier slot.

slot populated with "append". Second, both these topics get their parameter slot named "digit" populated with "five"<sup>2</sup>. Thus, the following topic state distribution has occurred:



Both *APPEND*-topics are partially populated and neither the isolated-topic-distribution heuristic nor the peering-topic-distribution heuristic applies. The state distribution is, however, not illegal. Thus, we are facing an information error – it is a priori not clear to which spinner the digit '5' should be appended. As both ambiguous topics have an equal amount of identification slots populated, i.e., their activation name slot, the relative-maximum-identification heuristic fails as well. From the user's perspective, however, the situation seems clear: the user has previously decreased the amps spinner and now, by uttering "append five", it's the user's intention to append '5' to the very same spinner.

The system, however, cannot infer this intention by just considering the current topic state distribution. Instead, it must consider the previously triggered topic – just like the user implicitly does. We call the topic that was triggered right before the current topic state distribution occurred, the *historical topic* – in our case it is the *DECREASE*-topic of the amps spinner. In the following we specify a procedure, called resolution-by-historical-topic, to resolve the information error under consideration of the historical topic:

1. Identify the topic set  $M$  as the specific subset amongst the ambiguous topics, which have a maximum number of identification slots populated<sup>3</sup>.
2. Of every topic within  $M$  determine the names of the empty slots.
3. For every distinct (empty) slot name  $n$ 
  - a) determine the value  $v$  of the slot  $n$  from the historical topic; if the historical topic does not exist or if the slot name does not exist in the historical topic  $v = \perp$ ;
  - b) perform a semantic update of all topics in  $M$  using  $v$  as pseudo recognition result (we define that semantic update will do nothing if  $\perp$  is passed the recognition result).
4. Possible outcomes: if the obtained topic state distribution after the semantic update is the peering-topic distribution, apply the peering-topic-distribution heuristic to identify intended topic; otherwise

<sup>2</sup>We assume that the respective semantic analyzers recognize "five" as a value for the parameter slot digit

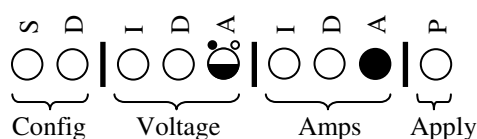
<sup>3</sup>This is similar to the first step of the relative-maximum-identification heuristic. In other words, the relative-maximum-identification heuristic determines the candidate topics for resolution-by-historical-topic.



- a) if the topic state distribution has not changed as compared to before the semantic update with the pseudo recognition result, resolution-by-historical-topic has failed; otherwise, i.e., if the topic state distribution has changed:
  - i. if the obtained topic state distribution is illegal, resolution-by-historical-topic has failed;
  - ii. else apply the relative-maximum-identification heuristic to identify the intended topic; if this fails, resolution-by-historical-topic has failed;
5. if resolution-by-historical-topic has failed discard the topic state distribution obtained by the semantic update with the pseudo recognition result

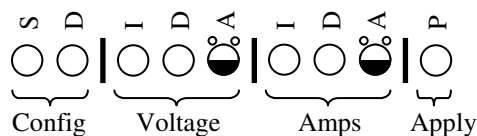
It is important to note that in the step "Possible outcomes" the isolated-topic-distribution heuristic does not have to be considered, since we are facing an information error, where at least two topics are not empty. Resolution-by-historical-topic basically examines the changing of the topic state distribution of the ambiguous topics under the assumption, that the current recognition result would have contained exactly the information from the historical topic, which is now simultaneously missing in any slot of the ambiguous topics. If resolution-by-historical-topic fails, other mechanisms, which we describe in the remainder of this chapter, will be invoked.

Let us go back to our example. The first step of resolution-by-historical-topic reveals, that the speakable identifier slots of all ambiguous topics are missing (illustrated by respective empty small circles on the right of each partially populated topic). Then, resolution-by-historical-topic would determine the value "amps" as the value of the speakable identifier slot from the historical topic. It would then (temporarily) perform a semantic update using the pseudo recognition result "amps" on all ambiguous topics. This causes only the speakable identifier slot of the *APPEND*-topic of the amps spinner to get populated – the other ambiguous topic does not accept "amps" as speakable identifier. Consequently, one topic is on the populated state as depicted below:

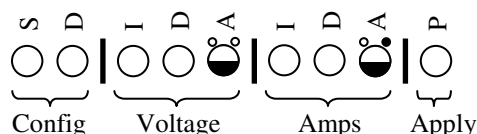


The topic state distribution now corresponds to the peering-topic distribution and resolution-by-historical-topic has determined the *APPEND*-topic of the amps spinner as the intended topic. This is in correspondence with our previous assumption that the user intends to append '5' to the amps spinner.

It is important to note that resolution-by-historical-topic would also have led to a success if the recognition result "five", instance of "append five", had become available, e.g., due to a recognition error. In this case, only the parameter slots of the *APPEND*-topics get populated and prior to applying resolution-by-historical-topic the following topic state distribution is given:



Resolution-by-historical-topic then determines, that both the speakable identifier slot and the activation name slot of each *APPEND*-topic are missing. The corresponding slot values from the historical topic are "amps" and "decrease". The slot value "decrease" does not lead to any further population of a slot of the two ambiguous topic, however, the "amps" value populates the speakable identifier slot of the *APPEND*-topic from the amps spinner, as depicted below:



Thus, the application of the relative-maximum-identification heuristic on the above topic state distribution, as defined by resolution-by-historical-topic, determines the *APPEND*-topic of the amps spinner as intended topic.

### 5.3.3 Summary

We showed that information errors can be detected by the absence of any populated topic and by the presence of multiple (including 1) partially populated topics. The situation can attempted to be resolved by applying the relative-maximum-identification heuristic, which selects the topic with the maximum number of populated identification slots, if a unique such topic exists. If the relative-maximum-identification heuristic fails, the procedure resolution-by-historical-topic can attempted to be applied next, which aims at resolving the information error by considering the historical topic. From a practical perspective, resolution-by-historical-topic facilitates repeated activations with a low average command length. For example, if the number '789' should be entered in the initially empty amps spinner, then the commands "amps append seven" followed by "eight" followed by "nine" would accomplish this. If resolution-by-historical-topic would not be available, then the user would have to utter "amps append seven" followed by "amps append eight" followed by "amps append nine", which is four words longer. In the following section we explain how information errors can be resolved if even resolution-by-historical-topic fails.

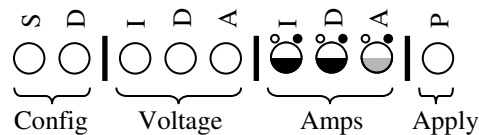
## 5.4 Qualified Feedback

In this section we will show that there are information errors which cannot automatically be resolved by the mechanisms explained in the preceding sections. We present mechanisms, referred to as *qualified feedback*, by which such information errors can be resolved. We will first explain, in section 5.4.1, how the system can generate a prompt for the user in which

it asks the user to clarify the information error with the subsequent utterance. We call this procedure *resolution-by-clarification-question*. In section 5.4.2 we show how the system can detect and handle situations, where the user does not respond to the prompt as expected by a procedure which we call *topic-state-backtracking*. Section 5.4.3 presents a procedure called *clarify-parameter*, which is designed for the system to be able to prompt the user for missing parameter information. Finally, in section 5.4.4 we summarize qualified feedback.

### 5.4.1 Resolution-by-Clarification-Question

We assume again that all topics from our sample GUI (Figure 5.5) are in the empty state. The recognition result "amps" becomes available. The semantic update with this recognition result causes the speakable identifier slot of any topic from the amps spinner to be populated with "amps". All other topics remain empty. Thus, both the *INCREASE*- and the *DECREASE*-topic of the amps spinner are partially populated indicating parameters complete. The amps spinner's *APPEND*-topic is also partially populated. It indicates, however, parameters incomplete. This situation is depicted below, including the population state of the identification slots as small circles:



No topic is in the populated state, some topics are partially populated and all topics have one identification slot populated. Thus, we are facing an information error which cannot be resolved by the relative-maximum-identification heuristic – it is not clear which activation of the amps spinner should be performed and possibly with which parameters. Resolution-by-historical-topic is currently the last known option to resolve the information error. The commonly missing slot amongst the ambiguous topics is the activation name slot and the value of the activation name slot from the historical topic is "append" (see section 5.3.2). The semantic update of the ambiguous topics using the pseudo recognition result "append" causes the activation name slot of the *APPEND*-topic to be populated. The state of this topic, however, remains unchanged as still the parameter slots are empty. Since resolution-by-historical-topic only takes further actions if the state of at least one topic changes, resolution-by-historical-topic fails as well.

To resolve the situation we define a procedure which we call resolution-by-clarification-question. The idea of this procedure is, as the name suggests, to clarify the ambiguous topics actively with the user, i.e., to generate and present a clarification question of which the expected response provides enough information to resolve the information error. If resolution-by-clarification-question would, hypothetically, be performed by a human, then the question that the human would most likely ask would be "Do you want to increase or decrease Amps, or append a digit?".<sup>4</sup> The semantic information in the human clarification question is twofold.

<sup>4</sup>The deletion of a digit from a spinner is, as mentioned in the introduction of this discussion, not considered.

First, it reflects that there are three possible actions that can be performed with the amps spinner. Second, it contains names for each of the three actions that can be performed.

Exactly the same information can be inferred from the ambiguous topics: there are three ambiguous topics which expect a specific activation name. Consequently, by iterating over the ambiguous topics, a question could be generated that looks like "Do you mean 'increase', 'decrease', or 'append'?". Given a different situation where the ambiguous topics were originating from different graphical objects, the speakable identifier of the graphical objects would have to be considered as well: assume, for instance, that instead of "amps" the recognition result "increase" had become available. The *INCREASE*-topic of the amps and the voltage spinner would be partially populated, and the question could then look like "Do you mean 'increase amps' or 'increase voltage'?". In any case, the clarification question would be presented to the user and if the user answers as expected, the information would lead to a resolution of the ambiguity as we describe further down in this section. The steps which make up resolution-by-clarification-question are as follows:

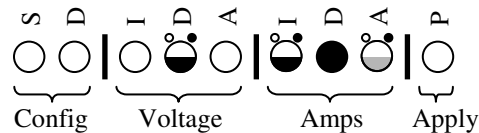
1. Identify the topic set  $M$  as the subset amongst the ambiguous topics, which have a maximum number of identification slots populated<sup>5</sup>.
2. Determine the number  $d_a$  of distinct activation names represented by topics in  $M$ .
3. Determine the number  $d_g$  of distinct speakable identifiers represented by topics in  $M$ .
4. Let  $q$  represent the clarification question and append the string "Do you mean " to  $q$ .
5. For every topic  $n$  in  $M$  do
  - a) If  $d_a > 1$  append to  $q$  the activation name expected by  $n$ .
  - b) If  $d_g > 1$  append to  $q$  the speakable identifier expected by  $n$ .
  - c) If there are more topics in  $M$  append " or " to  $q$ .
6. Append "?" to  $q$ .
7. Present  $p$  to the user.
8. Perform semantic update with the subsequent recognition result on any available topic.
  - a) If the topic state distribution changes, apply known heuristics and procedures;
  - b) otherwise fall back to a different mechanism, which we describe in the following section.

---

<sup>5</sup>This is similar to the first step of the relative-maximum-identification-heuristic. It restricts the clarification question to those ambiguous topics, which have the most identification information

It is important to recognize that the semantic update with the recognition result that follows the clarification question is performed with all currently available topics – and not on just the ambiguous ones. This facilitates the detection of the user changing the intention, which we explain in section 5.4.2.

Going back to our example, we consequently assume the generation of the question "Do you mean 'increase', 'decrease', or 'append'?" and the system expects "increase", "decrease" or "append" as subsequent recognition result. We will, for now, assume that the user answers as expected, i.e., uttering "increase", "decrease" or "append". Such assumptions have also been made by others (Kacioglu and Ward [92]), as it is mandated by the Cooperative Principle (Grice [71]) (introduced in section 2.5) in order to avoid conversation break-down. Thus, we will, for now, assume that the user answers "decrease" and that a corresponding recognition result becomes available. The semantic update with the "decrease" recognition result causes the activation name slot of any *DECREASE*-topic to be populated. Thus, the topic state distribution is now as follows:

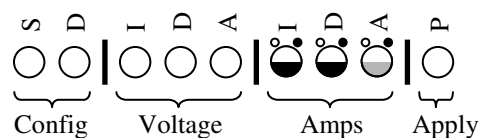


This topic state distribution matches the criteria of the peering-topic distribution, as one topic is populated and some others are partially populated. Thus, the *DECREASE*-topic of the amps spinner is selected as the intended topic according to the peering-topic-distribution heuristic.

To summarize, we have shown that information errors might occur which cannot be automatically resolved. Such information errors can be resolved by a clarification question generated from the properties of those ambiguous topics, which have the most identification slots populated. We believe that there is a specific maximum number of topics, from which it is reasonable to generate a clarification question. We have not investigated towards this maximum in our work and consider it as a direction of future work. In the following section we will discuss the situation that arises, if the user does not respond as expected by resolution-by-clarification-question.

### 5.4.2 Topic-State-Backtracking

For explaining topic-state-backtracking let us consider the topic state distribution from the previous scenario after the "amps" recognition result had become available. The topic state distribution, after semantic update with this recognition result, as depicted below

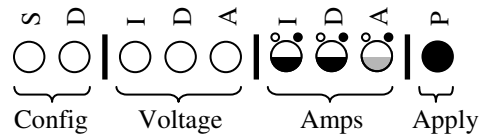


Let us further assume that the system has successfully generated and presented the clarification question "Do you mean 'increase', 'decrease', or 'append'?". Consequently, the system now

expects "increase", "decrease" or "append" as subsequent recognition result.

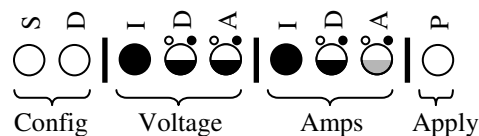
Let us now assume that the user answers "push apply" – which is not the expected answer – and that the corresponding recognition result becomes available. The user would expect that now the system pushes the apply button, i.e., that it triggers the *PUSH*-topic of the apply button, just like it did in section 5.2.3. Such an unexpected answer could be motivated by the user changing his mind, as to not anymore intending to perform something with the amps spinner. To cope with the situation, three capabilities are required. First of all, the system must be able to detect such a situation. Second, it must get rid of the belief that "user intends to perform something with the amps spinner". Third, the system must infer the new intention and react accordingly. In the following we examine several unexpected user answers, including the above one.

At first, let us consider the situation that arises if the recognition result "push apply" becomes available. The semantic update with it causes both speakable identifier slot and activation name slot of the *PUSH*-topic to be populated. The topic state distribution looks as follows:



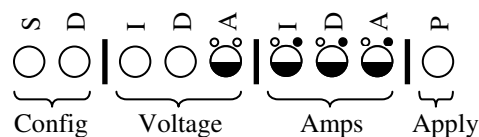
The system can consequently detect that the user has answered unexpected by the topic state changes occurring with topics other than the original ambiguous topics. We further recognize that, although the user has answered unexpected, the new intention of the user manifests as the peering-topic distribution. Thus, the peering-topic-distribution heuristic correctly identifies the *PUSH*-topic of the apply button as the intended topic.

Let us now examine what had happened after the semantic update with "increase voltage", i.e., if the user changed mind differently. The word "increase" will populate any corresponding activation name slot, unless not already populated, and "voltage" will populate any speakable identifier slot of the voltage spinner. Thus, the topic state distribution would look as follows:



What first strikes the eye is that two topics are in the populated state, i.e., we are facing the illegal distribution. If the user had responded to the clarification question as expected, there would not have been an illegal distribution. Thus, the user answering unexpected can be detected by an illegal state after the user's response to a clarification question.

If the user had answered "five" then the parameter slots of both *APPEND*-topics get populated, leading to the following topic state distribution:



The states have, indeed, changed, but there are even more ambiguous topics than before, and neither previously known heuristic or procedure is successful. Thus, the user answering unexpected, can be detected by an increasing number of ambiguous topics.

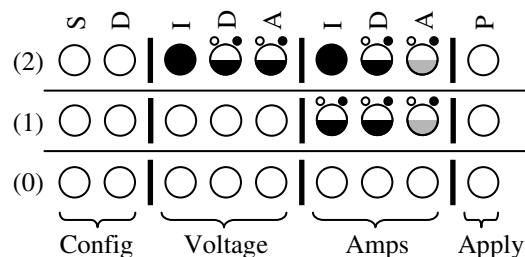
If the user had answered "amps" again the topics states would not have changed at all, as "amps" has led to the information error in the first place. Thus, another indicator for the user not having responded as expected is, if the topic states remain unchanged.

For summarizing this discussion we refer to the isolated-topic-distribution heuristic, the peering-topic-distribution heuristic and the relative-maximum-identification heuristic as *heuristic resolving*. If the intended topic can be inferred by heuristic resolving after a semantic update with a recognition result that becomes available as response to a clarification question, we have the situation as discussed in section 5.4.1 – the question of whether the intended topic could be inferred from an expected or unexpected response is not significant. Otherwise, if the illegal distribution occurs or if the intended topic could still not be inferred, topic-state-backtracking will be invoked. It is described in the following.

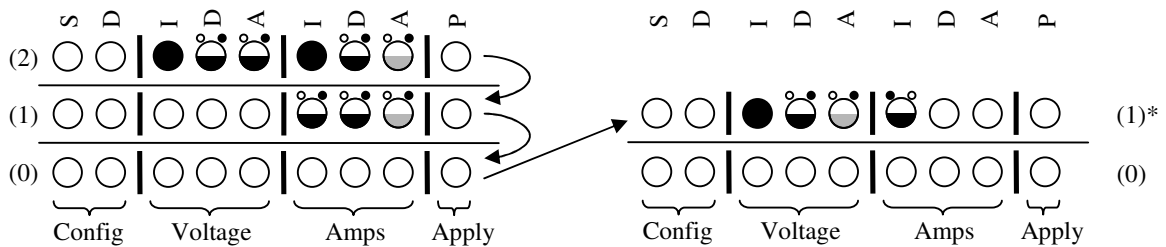
1. Discard the two latest topic state distributions.
2. If now all topics are empty, start the interpretation of the command/response from scratch;
3. otherwise:
  - a) Perform semantic update with user's response.
  - b) Perform heuristic resolving.
  - c) If illegal distributions was accomplished or if heuristic resolving failed, continue with step 1.

Topic-state-backtracking is a recursive algorithm which terminates on two conditions. Either, if during its execution the intended topic could be inferred by heuristic resolving. Or if all previous topic state distributions have been discarded, i.e., backtracked, until the initial situation, where all topics are empty, has been reached. Before topic-state-backtracking begins, there have at least two topic state distributions been considered – the distribution that led to the information error and the distributions after applying the users response to the clarification question. Thus, there are at least two latest topic state distributions (see step 1 of topic-state-backtracking).

Let us go back to our example and we consider the unexpected response "increase voltage". The situation before the application of topic-state-backtracking looks as follows:



The topic state distribution marked with (0) is the initial distribution. Distribution (1) is after the semantic update with the recognition result "amps" from which the clarification question "Do you mean 'increase', 'decrease', or 'append'?" has been generated. Finally, distribution (2) is the illegal state after the recognition result "increase voltage" became available. Since the illegal distribution (2) occurred while a response to a clarification question was expected, topic-state-backtracking is invoked. Distribution (2) is discarded, followed by the discarding of distribution (1). As now all topic states are empty, we start the entire recognition result interpretation from scratch, i.e., "increase voltage" is applied to state (0). The semantic update with "increase voltage" leads to a partial population of the *INCREASE*-topic of the amps spinner. Additionally, the *INCREASE*-topic of the voltage spinner is populated. As such, as illustrated below, the illegal distribution (2) has been transitioned into the legal distribution (1)\* by topic-state-backtracking. Furthermore, to (1)\* the peering-topic-distribution heuristic can successfully be applied:

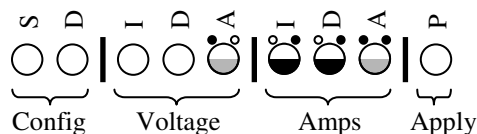


In the next section we show how we deal with empty parameter slots.

### 5.4.3 Clarify-Parameter

In the previous examples we have, for reasons of reducing complexity, omitted to discuss how empty parameter slots are dealt with. We will catch up with this in this section.

In any situation where the relative-maximum-identification heuristic has inferred a single topic which is in the parameters incomplete state, we must deal with empty parameter slots – independently of whether this situation was induced by the direct application of this heuristic or by indirect application from within resolution-by-historical-topic, resolution-by-clarification-question or topic-state-backtracking. As a representative example let us assume that all topics are in the empty state and that the recognition result "amps append" becomes available. The subsequent semantic update causes all activation name slots of *APPEND*-topics and all speakable identifier slots of topics from the amps spinner to be populated. The resulting information error is depicted below:



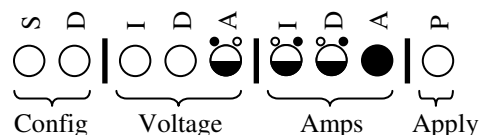
There is no topic in the populated state, therefore the relative-maximum-identification-heuristic will at some point in time be invoked. It succeeds in extracting the *APPEND*-topic



of the amps spinner, as this is the topic with the most identification information. However, the situation has, compared to the preceding examples, a different quality: according to the definition of the relative-maximum-identification-heuristic, the resulting *APPEND*-topic represents the intended topic – this is, indeed, the case in reality, but as the topic is in the parameters incomplete state, it cannot be triggered. It can obviously not be triggered, as the digit, that should be appended to the amps spinner has not been specified. We do not consider it a good idea to infer the missing parameter information automatically, e.g., from the historical topic. Parameters might be of equal type, but their semantics in the context of different graphical objects might vary significantly. It might, however, be worthwhile to give this idea more attention as future work (refer to section 7.3).

To still resolve the situation we introduce a procedure which we call *clarify-parameter* and which queries the missing parameter value(s) from the user. Let us, again, temporarily assume a human. Then, obviously, the human would ask the user a question like "Which digit do you want to append to the amps spinner?". The semantic information in this question is the name of the graphical object ("amps"), the name of the activation ("append"), and the name of the missing parameter ("digit"). Exactly this information can be inferred from the topic that is in the parameters incomplete state. It expects a specific activation name and a specific speakable identifier. Furthermore, the name of the missing parameter must be the name of the parameter slot which is empty. From this information a prompt like "Append amps, specify digit!" can be generated. Then, with the next recognition result becoming available, the system behaves as described in section 5.4.2 – there is one slight difference, however: the possibility that the states of the topics do not change after a semantic update with the response. does not necessarily mean that the recognition result was unexpected: if multiple parameter slots of the respective topic are empty, it will not change its state until all parameters have been populated. In particular, unless the recognition result is unexpected, it will not change the state of any topic, as such *clarify-parameter* can be applied iteratively to clarify all missing parameter slots. Thus, changes in the number of populated parameter slots must be considered as well as a state change. Other than that, the very same mechanism as described in section 5.4.2, including dialog state backtracking, can be used to interpret a response to *clarify-parameter*. However, the Swing catalog (section 4.1.1) does not involve topics with more than one parameter.

Going back to our example let us assume that the system generated "Append amps, specify digit!" and the recognition result "seven" becomes available. This causes the parameter slots of the *APPEND*-topic of both voltage and amps spinner to be populated and the topic state distribution changes as follows:



According to the peering-topic-distribution heuristic, the *APPEND*-topic of the amps spinner is correctly identified as the intended topic.

In summary, *clarify-parameters* is the procedure to actively inquire missing parameter infor-

mation. It is used whenever the relative-maximum-identification-heuristic determines a single topic with missing parameters, and as such can be seen as an extension to this heuristic.

#### 5.4.4 Summary

We have discussed information errors which cannot be resolved automatically. Such information errors can be detected by a failure of heuristic resolving and by resolution-by-historical goal. In order to resolve such information errors we have defined a collection of procedures which we call qualified feedback. These procedures generate clarification questions based on the characteristics of the information error, or process the recognition results which become available after a clarification question has been presented to the user.

### 5.5 Dialog Model for Conversation and Control

The result of the explanations in the previous sections is a set of techniques for the interpretation of qualified activation commands and for the detection and resolving of information errors. The techniques are, indeed, defined generically, however, their interplay when dealing with concrete cases has just been discussed on the basis of several examples. These examples are, indeed, representative, but there is still the lack of a generic core algorithm that can be used to create a working user interface. As the interaction with a speech-controlled GUI that is augmented by the introduced techniques has a flavor of a conversation, we call it a *conversation-and-control* interface. In particular, since there are two parties involved, the user and the system, such a core algorithm is generally referred to as a *dialog model*. In this section we will derive a dialog model for conversation-and-control, called the *conversation-and-control dialog model*. It manages the spoken interaction between those two parties, and it is therefore, in literature, also referred to as the *dialog management strategy*.

Dialog models are typically used in the research area of *spoken dialog systems*. According to Fraser [65] a spoken dialog system is a computer system which interacts with humans on a turn-by-turn basis and in which natural language plays an important role in facilitating this interaction. Spoken dialog systems have long emerged from a mainly research oriented focus, e.g., CMU Communicator (Rudnicky et al. [160]), TRAINS (Allen et al. [4]), or the Philips automatic train timetable information system (Aust et al. [15]), to productively used applications, e.g., the automatic train time table information system of Deutsche Bahn <sup>6</sup>, the voice banking portal of Postbank <sup>7</sup>, or the voice-based cinema information service of Cinecitta <sup>8</sup>. However, spoken dialog systems with cognitive and verbal skills as sophisticated as the HAL 9000 computer in the movie "2001: A Space Odyssey" [38] are still science fiction.

According to Fraser's definition a speech-controlled GUI is not quite a spoken dialog system: indeed, the process of uttering a command and awaiting the system's reaction can be

---

<sup>6</sup>Available at +49 800 1507090.

<sup>7</sup>Available at 0180 3040 700 (Germany) or +49 69 47867684 (international)

<sup>8</sup>Available at +49 911 20 666 7

regarded as a turn, however, the user does not use a natural language – at least, the commands "feel artificial". Artificial spoken languages for interaction with a computing system improves performance, since the languages, due to their design, improve the command recognition rate. Examples include Computer Pidgin Language (Hinde and Belrose [178]) and Universal Speech Interface (Rosenfeld et al. [157]). Users adapt well to these artificial languages (Tomko and Rosenfeld [185]). Some experiments show that they even prefer them over natural language (Tomko and Rosenfeld [186]). We therefore conclude that natural language is no prerequisite for a spoken dialog system. Instead, we interpret the concept of a spoken dialog system more broadly. We include any system that uses speech recognition technology as input and which has an internal dialog model, according to which it generates responses, prompts the user or triggers other actions.

Going back to the initial objective of this section, the definition of the conversation-and-control dialog model, we give an overview of dialog models as they are currently used with spoken dialog systems in section 5.5.1. We continue in section 5.5.2 with the definition of a data structure upon which we base the specification of the core algorithm of the dialog model for conversation-and-control (section 5.5.3) and summarize the dialog model in section 5.5.4.

### 5.5.1 Dialog Model Categories

According to McTear [124] three high-level categories of dialog models can be distinguished, called the *graph-based* dialog model, the *template-based* dialog model and the *agent-based* dialog model. Common to all categories is that the models have an understanding of the *dialog state*, that is, a representation of the systems view regarding the semantic information that has been exchanged with and received from the user<sup>9</sup>. The dialog model categories differ in how they represent the dialog state, in how they perform semantic updates, and in how they decide about subsequent actions. Another significant differentiation criterion is the manner in which they allow the user to take the *initiative* in the dialog, i.e., if and when they allow the user to seize control of the dialog by making an utterance, that introduces a (new) conversation topic for both user and system (Cohen et al. [39]). We discuss the three categories in the following.

#### Graph-based Dialog Model

A graph-based dialog model represents the dialog flow as a graph. Edges denote alternatives in the dialog flow in respect to the user's input. Nodes represent the system's reaction to the user's input, e.g., the acceptance or rejection of recognition results, the generation of respective prompts, or the calls of application functions. As such, each node in the graph represents a specific dialog state and thus, graph-based dialog models represent their dialog state implicitly. Therefore, they are commonly not referred to as being dialog-state-driven, although clearly, it is the dialog state that determines subsequent actions in respect to the next recognition result.

---

<sup>9</sup>In literature, the dialog state is also referred to as *discourse state*. This terminology is more generic as there might possibly be more than two parties involved in the conversation. For the scope of this thesis, however, we assume a dialog and stick to the term dialog state.

Graph-based dialog models typically restrict the user's input to responses of carefully designed system prompts. Thus, the mode of the resulting interaction is called *system-initiative*, as the system actively leads the user through the dialog. System-initiative mode is not limited to graph-based dialog models. While graph-based dialog models tend to use system-initiative mode throughout the conversation, other dialog models, as we show later in the section, use system-initiative if they detect that the user needs guidance in the dialog or when the system intends to recover from an error. An exemplary system-initiative dialog from a graph-based air travel system is given below. We discuss this example in the following.

- (1) System: "What is your destination?"
- (2) User: "London."
- (3) System: "London. What day do you want to leave?"
- (4) User: "Friday, at 10 AM."
- (5) System: "London on Friday. What time?"
- (6) User: "10 AM."

The advantage of graph-based models is that the required configuration data for the speech recognizer, e.g., the vocabulary, is simple and can precisely be specified in advance. This decreases the likelihood of recognition errors (Smailagic and Siewiorek [175]). In the above example, the vocabulary required to recognize utterance (2) can for instance be restricted to the destination airports known to the system. However, graph-based dialog models inhibit the user's opportunities for taking the initiative in the dialog or for asking questions. For instance, after the system prompt in (1) the user has no other choice than specifying the destination airport. Another example is utterance (4) where the user, by including "10AM", specifies more information than the system is able process in the current state of the dialog. Thus, the user has to specify this information again when the system is prepared to process it in (5) and (6). Corrections of recognition errors have to be described explicitly by a graph as well, which increases the complexity of the model.

An example for a spoken dialog system based on a graph-based dialog model is the telephone banking application for Lloyds TSB Telephone Banking <sup>10</sup> of which an overview is given in McTear [124]. Today, the creation of graph-based dialog system is supported by (commercially available) architectures and frameworks, such as *VoiceXML* [170] – an XML based language to describe and manage spoken dialogs between a computer and a user. *VoiceXML* provides an abstract interface to speech recognition and speech synthesis technology. *VoiceXML* provides further support in declaratively specifying the dialog flow based on a finite state model. As *VoiceXML* was originally designed for spoken dialog systems which are accessed by phone, it provides built-in support for DTMF-based <sup>11</sup> input. *SpeechObjects* [191, 82] is a commercially available framework for creating graph-based spoken dialog systems using object-oriented techniques. Developers create atomic dialogs, e.g., for specifying

---

<sup>10</sup><http://www.prnewswire.co.uk/cgi/news/release?id=20233>

<sup>11</sup>Dual Tone Multi-Frequency - the sounds that telephones use for dialing.

numbers or dates, and encapsulate the necessary dialog control logic into so called *speech objects*. Speech objects can be programmed directly by using a built-in Java library, or by wrapping Voice-XML. Using object-oriented techniques, speech objects can then be assembled to entire working spoken dialog systems, possibly using Voice-XML for the overall dialog control. Hartl [74] introduces *audio widgets* which also aims at introducing fine grained, reusable dialog units. However, their approach is more abstract, clearly separating the implementation of the dialog unit from its underlying model. As such, audio widgets are independent of VoiceXML. Furthermore, Hartl aims to address the issues of localization (i.e., reusing dialog units across different languages) and the dealing with speech recognizer-specific (i.e., proprietary) features. The CSLU Toolkit [123] is a tool that supports rapid creation of spoken dialog systems by letting developers graphically specify the graph that underlies the dialog model. The tool further supports to compile the graphical representation into a working spoken dialog system. The need for supporting the design phase of spoken dialog systems has been recognized by Anoop et al. [173]. They have created SUEDE, a prototyping tool for designing and testing dialog models without having to implement the actual spoken dialog system before testing can take place. Using SUEDE, a dialog model can be specified graphically and tested by the system emulating the functions of the resulting spoken dialog system, including the simulation of recognition errors. Graph-based dialog models can also be used as the basis for multi-modal interactions which involve spoken input. An example is described by MacWilliams et al. [116] who use a petri-network for modeling a point-and-speak user interface for an augmented reality application.

### Template-based Dialog Model

A *template* is an object which defines multiple placeholders for specific types of information instances. As such, a specific template can be considered as the specification of the information which a system requires to perform a specific application function. With a template-based dialog model the system populates the placeholders of a specific template with the information that it extracts from newly available recognition results. Thus, the dialog state is essentially the union of all currently available, possibly populated, templates. We will illustrate this technique considering the following example:

- (1) System: "What is your destination?"
- (2) User: "London."
- (3) System: "London. What day do you want to leave?"
- (4) User: "Friday, at 10 AM."
- (5) System: "London on Friday, 10AM. Window or Aisle?"

On first sight the dialog seems similar to the example that was used for graph-based dialog model. However, on closer examination, some interesting lessons can be learned. Let us examine the system's reaction on utterance (4): the user gives the expected response ("Friday"), but additionally gives the information that 10AM is the desired departure time. Apparently,

the system is able to recognize and process this additional information, as it is included in the correct context in utterance (5). With the graph-based example, the system just recognized "Friday" and went on with prompting for the time although the user had already specified it. In contrast, the template-based system goes on with asking about the desired seat location (5). This behavior can easily be understood, if we consider that the template-based dialog model for this particular situation might have the following template: a placeholder for the destination, for the departure day, for the departure time and for the desired seat location. This is analog to conversation topics, which have placeholders for the speakable identifier, the activation name and parameters. From the information in utterance (2) the system can populate the destination placeholder. It then chooses the next empty placeholder, i.e., the departure day placeholder. Based on meta information, like the placeholders name or a pre-defined prompt, the system asks the question in utterance (3). From the information given in (4) the system can populate the departure day placeholder, but as well the departure time placeholder.

With template-based dialog models the actual dialog flow is not pre-determined. It merely depends on the information that is contained in the respective recognition result. Both user and system can, to a certain extent, take over the initiative in the dialog. For instance, the system has the initiative when it asks the user for the departure day. The user responds as expected but at the same time takes over the initiative by specifying the additional information about the departure time. Thus, the mode of communication with a template-based dialog model is called *mixed-initiative*. However, for mixed-initiative mode, a specific level of natural language understanding is required, as not just the expected information must be expected by the system, but also additional information that is likely to occur. Similar to graph-based dialog models there are architectures and frameworks which support the creation of spoken dialog systems based on template-based dialog model, such as work from Papineni et al. [145] and Denecke [48], which we discuss in section 5.6.4.

### **Agent-based Dialog Model**

Like template-based dialog models, agent-based dialog models do not pre-define the dialog flow. Any partner in the conversation can take over the initiative in the dialog at any time, however, in contrast to template-based dialog models, in more elaborate ways. For instance, both user and system can introduce new topics, or can make informational contributions which are not otherwise constrained by the preceding prompt or utterances. Agent-based dialog models are better prepared for unanticipated input from the user. Thus, like template-based dialog models, agent-based dialog models operate in mixed-initiative mode. Agent-based dialog models are not constrained in the way they represent the dialog state. It is usually a complex data structure upon complex operations are defined, as for instance the *information state* established in the TRINDI project (Traum and Larsson [187]). It could, however, also be a simpler, template-like structure. In agent-based dialog models it is common, that more than one internal component modifies the dialog state, following the metaphor of a *blackboard*. The blackboard, i.e., the dialog state, contains the cumulated knowledge which all components have contributed. Whenever the dialog state changes, e.g., due to a component

having contributed information, other components contribute more information based on the current information on the blackboard. Agent-based dialog models are further characterized by their view of the dialog as an information exchange between entities, which are able to reason about their own actions and their beliefs – possibly, also about the action and beliefs of the respective partner in the communication. As such, an agent-based dialog model typically includes a model of the user and the user’s beliefs as the following extract from a sample communication from Sadek and de Mori [161] shows:

- (1) User: I’m looking for a job in the Calais area. Are there any servers?
- (2) System: No, there aren’t any employment servers for Calais. However, there is an employment server for Pasde-Calais and an employment server for Lille. Are you interested in one of these?

In (1) the user asks for job offers in a specific area which the system answers negatively. However, not just by uttering "no" – instead, the system is more cooperative and infers, that the user could be interested on other job offers which are in the local area. For this to be realized the system, i.e., the respective agent, must have a model about the users possible interests and, as such, is able to anticipate possible next questions from the user. The details of agent-based dialog models, however, are beyond the scope of this dissertation and we refer to McTear [124] for continuative reading. In the following we give examples for existing architectures, frameworks and systems which employ agent-based dialog models.

An early piece of work in the domain of agent-based dialog models is GALAXY (Goddeau et al. [67]), first introduced in 1994, which is a client-server architecture for *spoken dialog information systems* – spoken dialog system by which users can access information, e.g., from databases. Its core characteristic is that it employs a central hub, via which all components of the system communicate. In 1996, a significant architectural redesign was performed to permit access to GALAXY-based systems via a web browser (Lau et al. [104]). As of 1998, GALAXY-II (Seneff et al. [164]), which emerged from GALAXY, was used as the reference architecture for the DARPA Communicator program<sup>12</sup> with the challenge task to create a spoken dialog information system for the air travel planning domain. Within this program several air travel planning systems have been created and compared with specific metrics (Walker et al. [193]). Example systems include the CMU Communicator (Rudnicky et al. [160]) and Mercury (Seneff and Polifroni [165]). GALAXY was also used to create spoken dialog systems in different application domains, such as the weather information system JUPITER (Zue et al. [199]). RavenClaw (Bohus and Rudnicky [23]) is rooted in the GALAXY architecture and aims at rapid-prototyping of complex spoken dialog information systems, providing transparent support for the CMU Sphinx speech recognizer (Huang et al. [77]) and the Festival speech synthesizer (Black and Lenzo [21]). It further includes a rich set of domain-independent conversational behaviors. As such, in contrast to former GALAXY-based applications, the system development effort is more focused on the specification of the dialog task. Several systems

---

<sup>12</sup>The DARPA Communicator Program is now closed and has turned into the GalaxyCommunicator Open Source Software Infrastructure <http://communicator.sourceforge.net/>

based on RavenClaw exist, including Let's Go (Raux et al. [1]) or RoomLine<sup>13</sup>.

Agent-based dialog models are typically used in application domains where both user and system have partial knowledge about a task to be completed – neither user nor system can perform the task alone. An example is *problem solving* and an early representative in this domain is the Circuit-Fix-It Shop system (Biermann et al. [19]), which helps users to fix an electronic circuit by engaging them in a spoken dialog. Another example in this domain is TRAINS (Allen et al. [4]), where both the system and the user decide about routing trains through a network of railways. *Tutoring* is another application domain for agent-based dialog models. Examples include the DIALOG project (Buckley and Benz Müller [30]), which aims at creating a spoken dialog system that is capable of tutoring naive set theory, and the BEETLE system (Zinn et al. [198]), which tutors basic electronics.

## Verification

By *verification* we understand the actions which a spoken dialog system performs in order to make sure that it has correctly understood what the user has said. Verification can be characterized by two features which are orthogonal to each other.

The first feature denotes the point within the dialog flow where the system performs verification. Two variants, which we call *just-in-time* verification and *delayed* verification are commonly used: with just-in-time verification the system verifies the input before it moves on to the next dialog state or before the dialog state gets updated. The advantage is that the input is verified right when the user provides it. With delayed verification the system verifies the input right before an application function is invoked. The advantage is that this type of verification is more natural, however, the user might have to verify input that has been specified several dialog turns in the past.

The second feature is whether verification is performed *explicitly* or *implicitly*. With explicit verification the system formulates an explicit prompt in which it asks the user whether a specific input is correct or not. The user is expected to answer with "yes" or "no" and an example is given below:

- (1) System: "What is your destination?"
- (2) User: "London."
- (3) System: "Is London correct?"

Implicit verification means that the system reiterates the input but in the same utterance does something else, e.g., asks for another bit of information. An example is given below

- (1) System: "What is your destination?"
- (2) User: "London."
- (3) System: "London. What day do you want to leave?"

Implicit verification is more natural than explicit verification, however, it requires more

---

<sup>13</sup>A description of RoomLine, including sample dialogs, are available at <http://www.cs.cmu.edu/~dbohus/RoomLine/>



natural language understanding and a flexible dialog model, as the user, in cases where the input was misrecognized, answers like "No, I meant Paris". Thus, explicit verification is more frequently used by graph-based dialog models and implicit verification more with template-based or agent-based dialog models. The above examples denote just-in-time verification. The same example, using delayed verification, could look like:

- (1) System: "What is your destination?"
- (2) User: "London."
- (3) System: "What day do you want to leave?"
- (4) User: "Friday."
- (5) User: "Book a flight to London on Friday?."

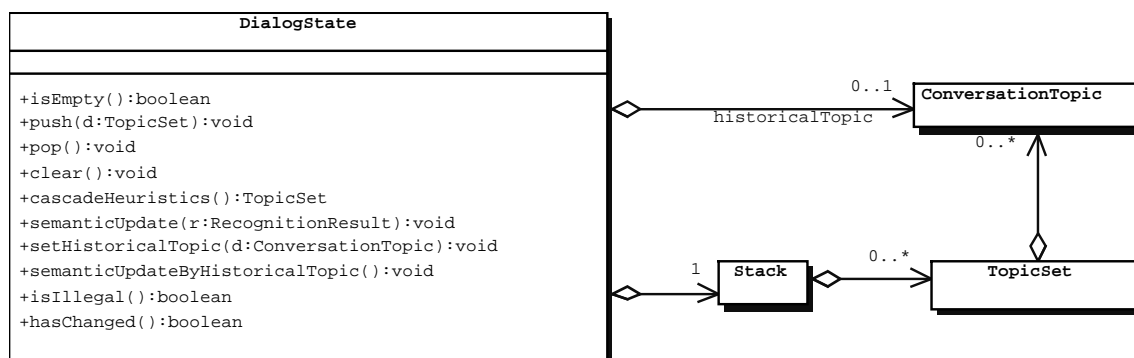
To summarize, we have introduced three dialog model categories which are currently used for spoken dialog systems. Common to the categories is that they have an internal model of the dialog state. They differ in how they represent the dialog state, in how they determine the system's reaction upon a new recognition result, and how and when they allow the user to take the initiative in the dialog. Based on this background knowledge we now specify a dialog model for conversation-and-control in two steps. In the first step (section 5.5.2) we introduce a data structure which we call the *dialog state*. It represents, as the name suggests, the current state of the dialog between the user and the conversation-and-control interface. It provides several operations, e.g., an encapsulation of the semantic update process. In the second step (section 5.5.3) we specify a generic algorithm, i.e., the actual dialog model, based on the dialog state. As it will become clear to the reader, the dialog model we are suggesting is template-based.

## 5.5.2 Dialog State

The design of the dialog state is based on four observations made by examining the examples from sections 5.2 through 5.4 . First, the systems reaction is always derived from (a subset of) the current topic state distribution. Thus, the dialog state is based on *topic sets*. Second, as in specific situations not only the current topic state distribution must be considered, but also previously obtained topic state distributions (e.g., topic-state-backtracking), we use a *stack* of topic sets.<sup>14</sup> Third, in specific situations, the most recently triggered topic must be considered. Therefore, the dialog state has an association to the respective historical topic. Fourth, we observe a specific reoccurring cascade of heuristics and procedures upon the current topic state distributions. We will specify this cascade as an operation on the dialog state, along with other operations which will prove useful for specifying

---

<sup>14</sup>We imply that the reader is familiar with the concept of a stack as this is one of computer science's most fundamental data structures. Otherwise, Knuth [102] provides an introduction into the characteristics of a stack.



**Figure 5.6:** Dialog state for conversation-and-control.

the dialog model in section 5.5.3. As such, we define the following operations on the dialog state: `IsEmpty`, `Push`, `Pop`, `Clear`, `CascadeHeuristics`, `SemanticUpdate`, `SetHistoricalTopic`, `SemanticUpdateByHistoricalTopic`, `IsIllegal` and `HasChanged`. Figure 5.6 shows a diagram of the dialog state in UML. We explain the effects of the methods, the operations, respectively, in the following.

**IsEmpty** The `IsEmpty` operation takes no argument and returns the boolean value `false` if the stack contains elements, and `true` otherwise.

**Push** The operation `Push` has no return value and takes a topic set as argument which it pushes onto the stack.

**Pop** The `Pop` operation takes no argument and does not return a value. If the stack is not empty it removes the top element, i.e., the top most topic set, from the stack; otherwise the method has no effect.

**Clear** The `Clear` operation takes not argument and has no return value. It removes all elements from the stack.

**CascadeHeuristics** This operation takes no argument and returns a topic set according to the following rules:

1. If the stack is empty then the return value is an empty topic set.
2. If the stack is not empty the operation checks the top most element of the stack for the isolated-topic-distribution heuristic. If it applies, the operation returns a topic set that contains the isolated topic as only element.
3. Otherwise:
  - a) The operation applies the peering-topic-distribution heuristic to the top most element of the stack. If it succeeds, the operation returns a topic set that contains the peering topic as only element.

- b) Otherwise, if the peering-topic-distribution heuristic fails, the return value of the operation is defined as the topic set determined by the relative-maximum-identification heuristic, i.e., the operation returns the topics with a maximum number of identification slots populated (this can be a single topic or multiple topics).

**SemanticUpdate** This operation takes a recognition result as argument and returns no value. The operation performs the following steps:

1. Create a clone of all topics in the top most stack element.
2. Perform a semantic update on each topic clone with the given recognition result.
3. Put the clones into a new topic set and push the topic set onto the stack.

**SetHistoricalTopic** This operation takes a topic as argument and returns no value. The argument is set to be the historical topic of the dialog state.

**SemanticUpdateByHistoricalTopic** This operation takes no argument and returns no value. If the historical topic is defined, the operation performs resolution-by-historical-topic on the top most element of the stack, and pushes the result as a new topic set onto the stack. Otherwise, if the historical topic is not defined, the operation performs the operation `SemanticUpdate` with an empty pseudo recognition result.

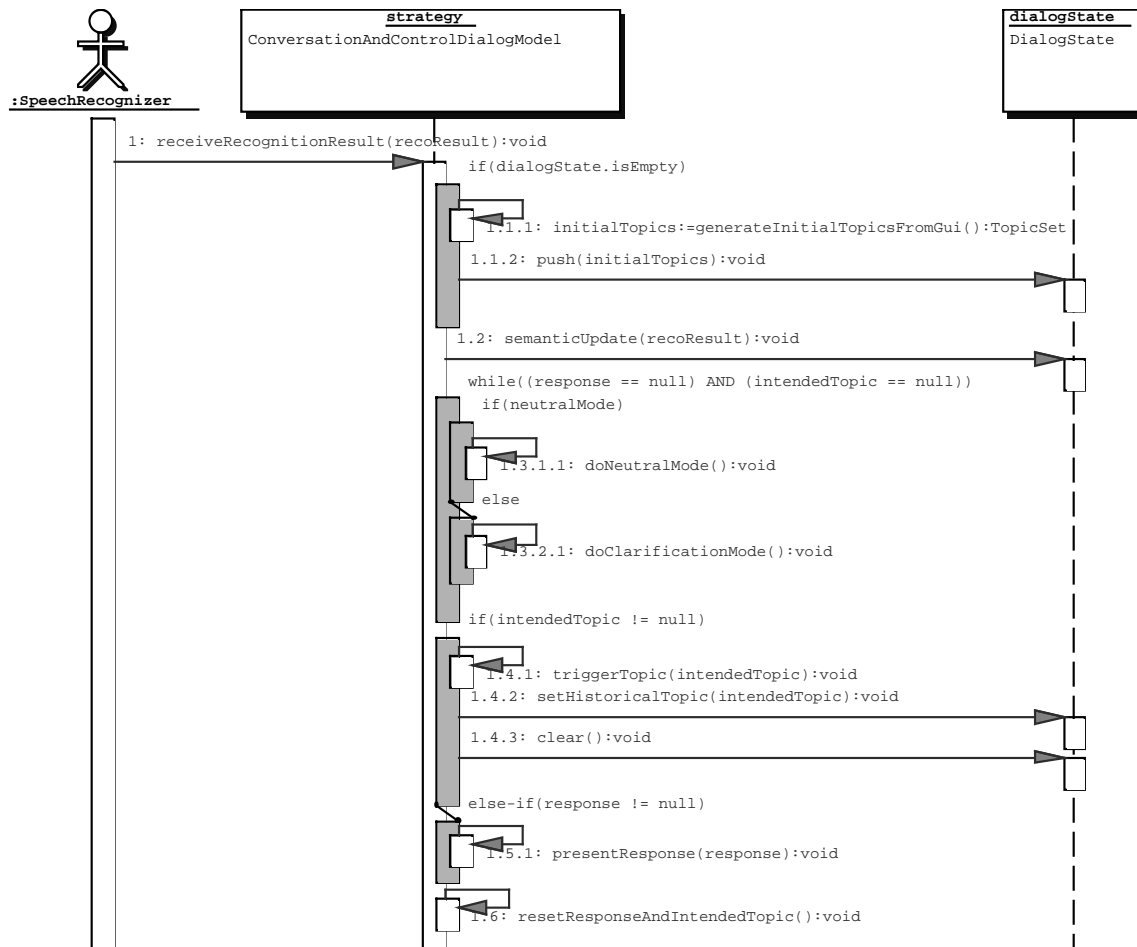
**IsIllegal** Takes no argument and returns a boolean value. Returns `true` if the top most stack element is in the illegal distribution. Otherwise, the operation returns `false`.

**HasChanged** Takes no argument and returns a boolean value. If the stack has no or one element, the operation returns `false`. Otherwise, the operation returns `true` if the topic state distribution of the top most stack element is different from topic state distribution the second top most element. If those topic state distributions are equal, the operation returns `false`.

### 5.5.3 Core Algorithm

We now specify the core algorithm for conversation-and-control dialog model as a UML sequence diagram. We have, for reasons of clarity, divided the actual diagram into three separate Figures. Figure 5.7 on the next page shows the main processing loop and defines the *neutral mode* and the *clarification mode*, depicted in Figures 5.9 on page 142 and 5.8 on page 141. We begin with explaining the high level processing loop.

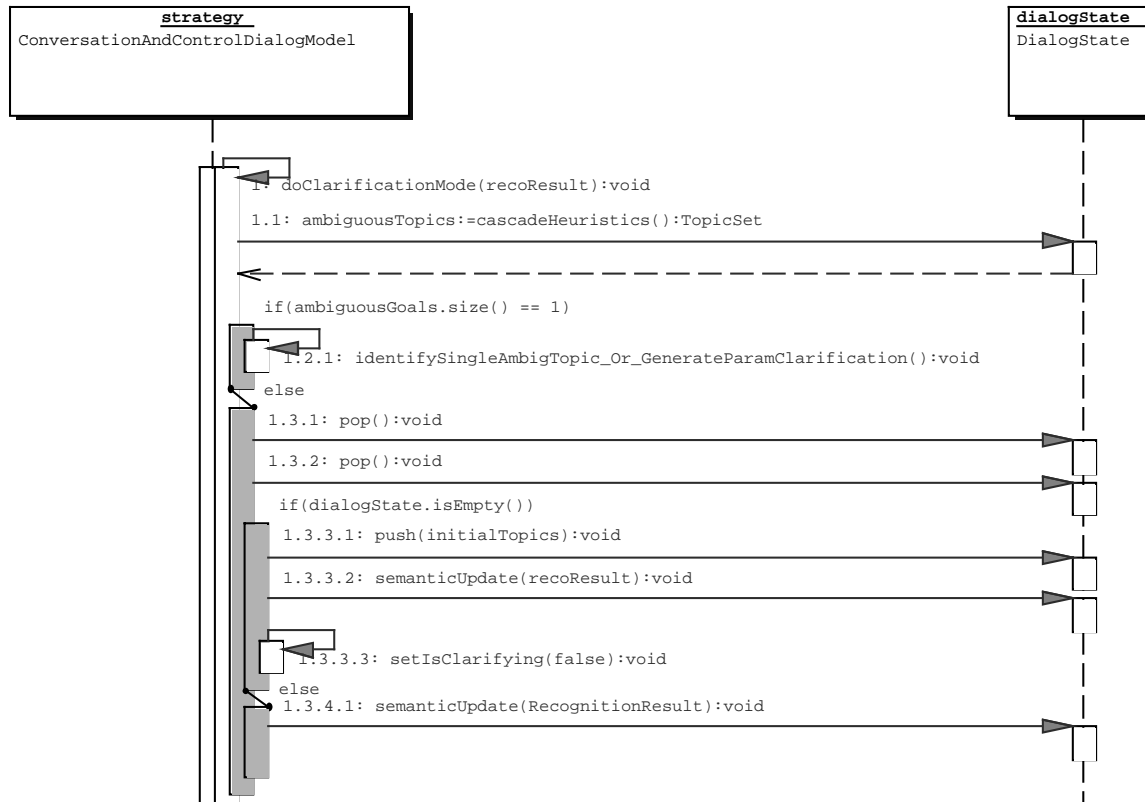
The conversation-and-control dialog model is represented by an object which receives a new recognition result from the speech recognizer (1). If the dialog state is empty, e.g., if the system just has booted-up or if previously an activation was successfully performed, the dialog model generates new topics from the current state of the GUI (1.1.1) and pushes them, as initial topics, onto the dialog state (1.1.2). Then the dialog model triggers a semantic update



**Figure 5.7:** Conversation-and-control dialog model, main processing loop.

of the dialog state with the just obtained recognition result (2). After the semantic update has been performed, the dialog model repeats a sequence of steps (`while`-loop) in which it either determines the intended topic (`intendedTopic`) or generates a system response (`response`).<sup>15</sup> Within the loop, the dialog model distinguishes between the neutral mode and the clarification mode. Clarification mode is entered (1.3.2.1), if the dialog model, by the just received recognition result, expects the user's response to a previously presented clarification question. Otherwise (1.3.1.1), the neutral mode is entered. We will explain both modes in the following paragraphs. If either a response has been generated or the intended topic has been found, the loop terminates. Upon the identification of the intended topic, the dialog model triggers it (1.4.1), stores it as the historical topic (1.4.2) and finally clears the dialog state (1.4.3). Otherwise, if a response has been generated, the dialog model presents it to the user (1.5.1) and leaves the dialog state it is, so that it can be merged with the semantics

<sup>15</sup>We will later in this section see that the system does not only generate clarification questions.



**Figure 5.8:** Conversation-and-control dialog model, clarification mode.

of the subsequent recognition result. Finally, the dialog model resets, i.e., discards the just identified topic or response (1.6).

We now go into the details of the neutral mode (Figure 5.9 on the following page), i.e., the processing of a recognition result, if the dialog model does not expect an answer to a clarification question. A sequence diagram showing the neutral mode is depicted in Figure 5.9, to which we refer in the following. The dialog model will only take further actions, if the semantics of the recognition result changed the dialog state or of the dialog state then is not illegal. Otherwise, the dialog model causes the presentation of a "Sorry, I did not understand"-response (1.2.1) and clears the dialog state (1.2.2). The clearing is performed, so that if possibly the state of the GUI has changed, e.g., due to an external event, the dialog model generates up-to-date topics (see (1.1.1) of Figure 5.7). If, however, the dialog state in fact changed, the dialog model performs heuristic resolving (1.1.1). If this led to the identification of a single topic, the dialog model still has to check, whether this topic can be triggered as is – if it is in the parameters complete or the populated state – or if the dialog model still has to clarify parameters with the user. These are the only possibilities, as, if the topic would have been in the empty state, it would not have been identified by the

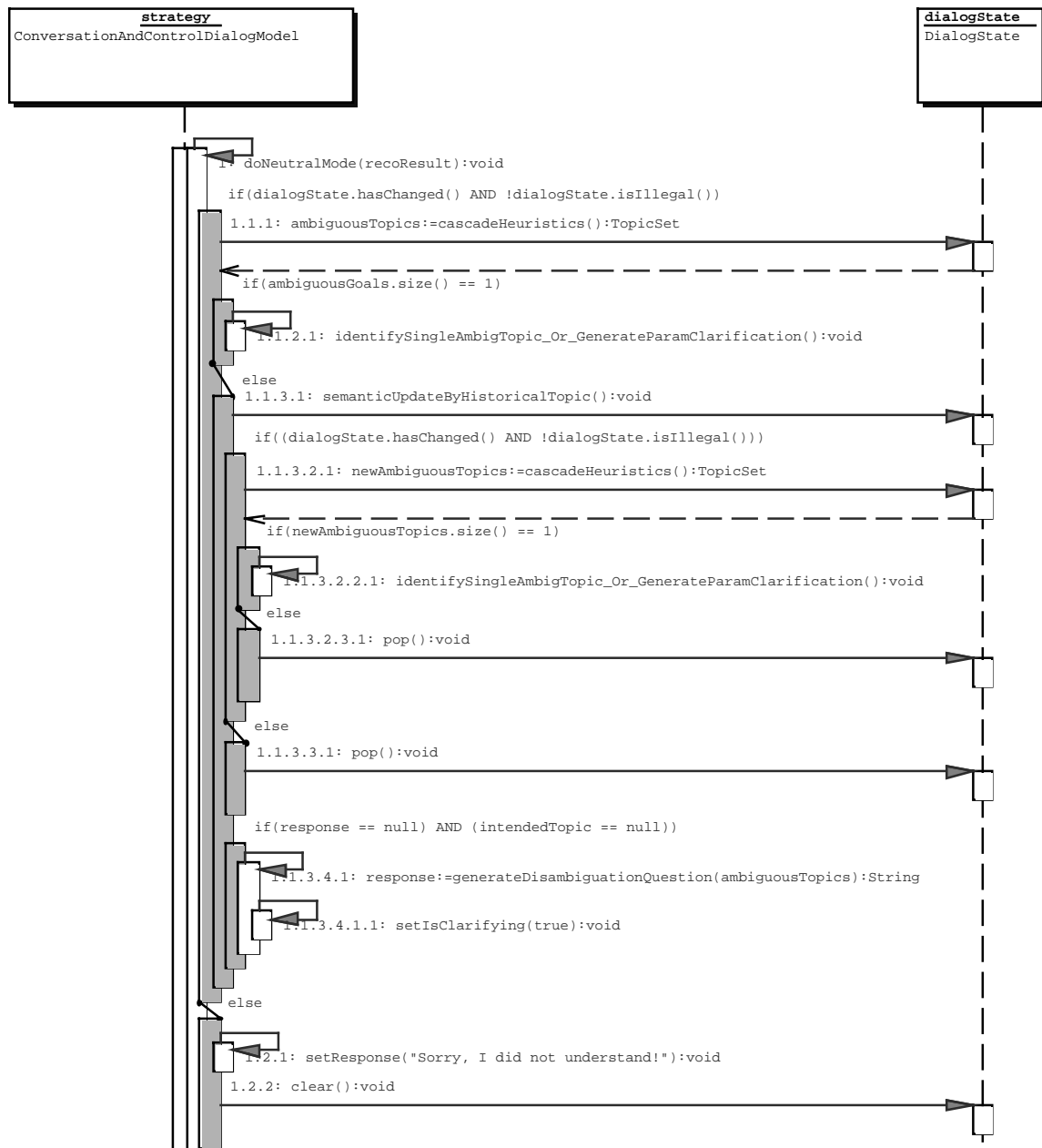


Figure 5.9: Conversation-and-control dialog model, neutral mode.

heuristics. For clarity reasons we encapsulate the evaluation of this single topic by the method `identifySingleAmbigTopic_Or_GenerateParamClarification` (1.1.2.1) which is, in pseudo code, defined as follows:

```
Topic topic = getTheOneAmbiguousTopic();
```

```
if (topic.isParamsComplete() OR topic.isPopulated()) {
    intendedTopic = topic;
} else {
    response = generateParameterClarification(topic);
    setIsClarifying(true);
}
```

The `identifySingleAmbigTopic_Or_GenerateParamClarification` method will occur in two further places, thus, it can be regarded as a macro. As such, by the step (1.1.2.1) the dialog model has either identified a topic or generated a system response – as such, the termination criterion for the main loop has been achieved.

We now discuss the remaining case – the heuristics have determined multiple ambiguous topics. In this case, according to the explanations in section 5.3, the dialog model first attempts resolution-by-historical-topic (1.1.3.1). This might have two outcomes: either, resolution-by-historical-topic leads to an illegal state or does not change the dialog state at all, in which case the dialog model discards the state obtained by resolution-by-historical-topic (1.1.3.3.1). In the opposite case, the dialog model applies heuristic resolution again, this time, however, on the topics as they have been changed by resolution-by-historical-topic (1.1.3.2.1). If the heuristics could now identify a single topic, the dialog model will check, whether the topic can be triggered as is, or if parameters must be inquired (1.1.3.2.2.1) – in any case, the termination criterion for the main loop has been satisfied. Otherwise, the dialog state obtained by resolution-by-historical-topic is discarded (1.1.3.2.3.1).

If resolution-by-historical-topic has succeeded, the dialog model does nothing, so that the main loop will be iterated through from the beginning. Otherwise, if neither a response has been generated nor a topic has been identified by resolution-by-historical-topic, the dialog model now invokes resolution-by-clarification-question. This is done by two steps. First, the dialog model generates a clarification question from the originally ambiguous topics (1.1.3.4.1). This satisfies the termination criterion of the main loop so that the response will be presented to the user. As the dialog model now expects a response from the user, it sets a flag (1.1.3.4.1.1) by which the main loop will enter the clarification mode if the next recognition result becomes available.

We will now discuss the characteristics of clarification mode, of which a sequence diagram is depicted in Figure 5.8. In clarification mode, differently to the neutral mode, the dialog model will take further actions regardless of whether the latest semantic update changed the dialog state. The reason is, as discussed in section 5.4.3, that parameter clarification might not necessarily change the state of a topic, i.e., if more than one parameter is missing. Thus, the dialog model, in clarification mode, immediately begins with heuristic resolving (1.1.1). If this succeeds, i.e., if a single topic is obtained, the topic's parameter state needs to be considered (1.2.1). Otherwise, i.e., the dialog model may assume that the user has answered unexpectedly (see discussion in section 5.4.2), and consequently the dialog model enters dialog state backtracking (1.3.1) and (1.3.2). If the dialog state is empty after discarding the last two topic state distributions, the recursion of dialog state backtracking must terminate. This is

achieved by pushing the initial topics onto the dialog state (1.3.3.1), by performing a semantic update in the just initialized dialog state (1.3.3.2) and by resetting the mode flag to neutral mode (1.3.3.3). As the termination criterion of the main loop has not yet been accomplished, the main loop will be entered again, but now the neutral mode will be considered. This is feasible, since apparently the (unexpected) recognition result did not lead to the successful inferring of the intended activation or a suitable response using dialog state backtracking. As such, the dialog model will start from scratch (dialog state is just initialized and neutral mode will be entered), which immediately results in a "Sorry, I did not understand"-response, the identification of a topic or a subsequent clarification.

If the dialog state is not empty after the discarding or already backtracked topic sets, the dialog model just performs a semantic update (1.3.4.1). The termination criterion is not accomplished so that the loop will be reentered in clarification mode upon another historical dialog state.

### 5.5.4 Summary

We have introduced a generic model for managing the dialog between the user and the conversation-and-control interface. The model is based on several types of topic state distributions, and involves different heuristics and procedures based on topic state distributions, from which the intended topic can be inferred in a cascading manner. That is, topic state distributions, heuristics and procedures are subsequently applied if the respective successor failed. This process is partially encapsulated by the dialog state data structure, which models the current state of the dialog as the top most element of a stack of sets of topics. The dialog history is represented by the dialog state as well: one the one hand by the maintaining a reference to the historical topic, and on the other hand by the elements of the stack which are not the top most one.

The dialog model for conversation-and-control dialog model is clearly dialog-state driven, as any decision that the generic algorithm makes, is based on it. Furthermore, the dialog state is represented explicitly, as a stack of sets of topics. The conversation-and-control dialog model is not graph-based as it does not involve a graph to represent the dialog flow. Although the conversation-and-control dialog model allows for mixed-initiative interaction (as discussed below) it is clearly not agent-based, since the model does not reason about its own beliefs and the beliefs of the user. The conversation-and-control dialog model is therefore template-based and the topic(s) represent(s) the template(s).

The user primarily has the initiative in the conversation, because the dialog model, unless an information error occurs, just reacts on the user's commands. If, however, an information error occurs the model has two characteristics which emphasize its mixed-initiative character. First, the dialog model claims the initiative through disambiguation questions or parameter clarification questions. Second, by being able to detect unexpected responses to the above questions and by adjusting the dialog state accordingly, the user is able to re-claim the initiative without causing a conversation break-down (refer to sections 5.4.1, 5.4.2 and 5.4.3).



## 5.6 Discussion

Conversation-and-control is a new speech-controlled GUI approach which is based on enhancements of the speech-GUI model (section 5.2.1). This shows, that the speech-GUI model is extensible to new speech-controlled GUI approach, as required by the discussion in section 2.1. We now summarize the characteristics of the conversation-and-control dialog model (section 5.6.1) and discuss inherent limitations (section 5.6.1). After that, in section 5.6.3, we clarify the role of speech synthesis technology – a key technology for spoken dialog systems – in the scope of conversation-and-control. Finally, in section 5.6.4 we discuss other related work.

### 5.6.1 Characteristics of the Dialog Model

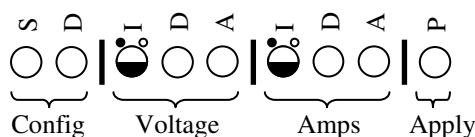
We call the characteristics of the conversation-and-control dialog model *decentralized encapsulation*, *legacy emulation* and *simple pronoun resolution*, and discuss them in the following.

**Decentralized Encapsulation** This characteristic denotes that there is no central semantic analyzer component which has the entire knowledge for analyzing recognition results. We employ multiple, decentralized components, i.e., the semantic analyzer components of a conversation topic, of which each encapsulates exactly the syntactic and semantic knowledge required for a specific qualified speech function. This is in contrast to existing architectures for spoken dialog systems, such as GALAXY-II (Seneff et al. [164]) or RavenClaw (Bohus and Rudnicky [23]), which employ a central semantic analyzer component that holds the entire syntactic and semantic knowledge for the application domain. We argue that decentralized encapsulation our approach is more flexible regarding changes or enhancements of the application domain. For instance, if due to usability studies, the syntactic structure of a specific qualified activation command should be changed, only one (at least only a few) semantic analyzer objects must be altered. Also, if new graphical objects should be supported, the necessary knowledge can simply be "plugged-in" into the existing knowledge by providing respective conversation topics.

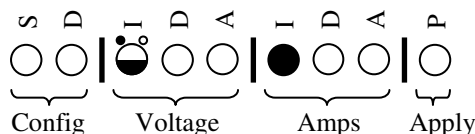
**Legacy Emulation** It has to be pointed out that the conversation-and-control dialog model can emulate conventional command-and-control, i.e., command-and-control with indirect activation and random navigation. This can be achieved by defining that a topic is not associated with a qualified speech function, but with an ordinary speech function as it is used with conventional command-and-control. As such, a navigation can be performed by uttering a speakable identifier, similar to conventional command-and-control. In the conversation-and-control dialog model this would lead to the population of the speakable identifier slot of one specific topic, and in turn, the relative-maximum-identification-heuristic would identify this topic as the intended topic. An indirect activation, i.e., an activation that is coupled with a navigation, is transparent for

the conversation-and-control dialog model, as the speech function, which is triggered by just a navigational command, performs it. Regular activations are performed by uttering their activation name following by a parameter, e.g., "append five". Assuming that the user has previously navigated to a specific graphical object, as it is required by conventional command-and-control, the procedure resolution-by-historical-topic would resolve the missing speakable identifier in the above command. While this characteristic might not necessarily have consequences for the actual usage of conversation-and-control, it facilitated the creation of a testing framework with which we could easily measure and compare task completion times with conversation-and-control and conventional command-and-control using the very same GUI.

**Simple Pronoun Resolution** The conversation-and-control dialog model is, to a certain extent, able to deal with anaphoric relationships in the form of pronouns. Referring to our exemplary GUI in Figure 5.5 consider that the recognition result "append amps five" becomes available, which would have caused the digit '5' to be appended to the current value of the amps spinner. Let us now consider that the user utters "increase it", which contains the activation name of the *INCREASE* activation and the pronoun "it". After the initial semantic analysis, any topic that represents an *INCREASE* activation would consequently get its activation name slot get populated, as depicted below:



As there is no topic in the populated state, the peering-topic-distribution heuristic fails. The relative-maximum-identification heuristic fails as well, as there are multiple *INCREASE*-topics with one identification slot populated. Resolution-by-historical-topic determines that the identification slot is missing on the maximal identified topics, and the corresponding value from the historical topic, "amps", changes the states of the topics as follows:



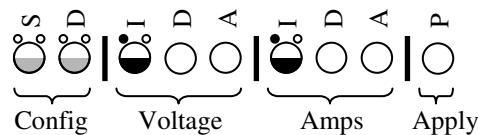
Thus, the peering-topic-distribution heuristic will correctly identify the *INCREASE*-topic of the amps spinner, even though the user used a pronoun. However, the capability to deal with anaphoric expression is not due to resolving the anaphoric expression explicitly – it is merely achieved by ignoring it, however, for the speech-GUI domain, this way of dealing with anaphoric expressions is sufficient.

Nevertheless, the conversation-and-control dialog model also has inherent limitations, which we discuss in the following section.

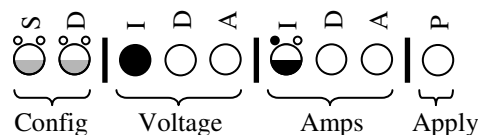
## 5.6.2 Limitations

We now point out and discuss inherent limitations of the conversation-and-control dialog model.

**Identification Precedence** The conversation-and-control dialog model weights identification information higher than parameter information, which we call *identification precedence*. This can lead to unexpected, i.e., unintuitive side effects. Given our sample GUI from Figure 5.5 let us a situation after system-boot up, i.e., all dialog topics are in the empty state and there is no historical topic. The user utters "increase day" which includes the name of an option of the config list and the name of the *INCREASE* activation, so that the topics get populated as follows.



The dialog model will apply resolution-by-clarification-question, as neither the peering-topic-distribution heuristic nor the relative-maximum-identification heuristic is successful, and a historical topic is not available. From a human perspective we would then expect the system to clarify between the config list, the amps spinner and voltage spinner, as all three graphical objects are "affected" by the utterance. However, as the base topics for resolution-by-clarification-question are delivered by the relative-maximum-identification heuristic, as such, the dialog model just clarifies between the amps spinner and the voltage spinner ("Do you mean 'Amps' or 'Voltage'?"). Another such situation may arise if we assume that the user had initially uttered "increase Voltage", which causes the voltage spinner to be increased. If now the user utters "increase day", we have the same topic states as above, however, now a historical topic is available. As the commonly missing slot amongst the ambiguous topics is the speakable identifier slot, the corresponding value of the historical topic with by used by resolution-by-historical-topic, and we face the dialog states as depicted below:



Consequently, according to the peering-topic-distribution heuristic, the *INCREASE*-topic of the voltage spinner will be identified, and the *INCREASE* activation of the voltage spinner will be invoked. From a human perspective we would have expected a clarification between the three graphical objects as above.

**Verification Absence** Input from the user is not verified as long as the dialog model does not detect an error condition. In other words, if the conversation-and-control dialog model identifies a topic, it is triggered without re-presenting to the user what the speech recognizer has delivered. In the case of errors, e.g., an ambiguous situation or a situation of missing information, the dialog model implicitly verifies the input by the way it asks clarification question, however, the clarification question might already be generated from corrupt data. While we argue that explicit clarification in general would unnecessarily lengthen the interaction, we regard it as valuable from a developers perspective, if the dialog model could be configured to explicitly verify the user's input for specific critical activations. An example is a button that triggers the deletion of data from a database. We further believe that the conversation-and-control dialog model would benefit from an implicit verification mechanism, which kicks in if the confidence of the recognition result drops below a specific threshold. The incorporation of verification capabilities is considered as future work 7.3.

**Dealing with Misunderstandings** It is the nature of a misunderstanding that both parties in the dialog are convinced of their beliefs about the respective other party, and in the further flow of the dialog it becomes obvious that at least one of these beliefs was wrong. Even humans do not have the ability to detect the misunderstanding at the time it occurs, so it is not surprising that there is currently no spoken dialog system which has this ability. The very same applies, of course, for conversation-and-control – once a misunderstanding has happened, i.e., if the intended topic has not been correctly inferred, the associated qualified speech function still gets triggered.

We argue that this limitation of conversation-and-control can be mitigated. The speech-controlled GUI could be equipped with a speech function *SFUNDO*, which the user could trigger by uttering "undo" (this solution idea was already in section 3.2.2 when we presented the interaction delay model) and which would undo the most recently performed speech function. However, this solution does not work for application functions which cannot be undone. Thus, the effects of misunderstandings could further be mitigated by enhancing the mechanism as described in the context of verification absence: not only if the confidence of the recognition result drops below a specific threshold an explicit verification is required, but in general for any application function that cannot be undone.

Misunderstandings can be detected indirectly after they have happened. For instance, a communication break down can very likely occur due to a misunderstanding. One line of research in detecting communication breakdowns aims at using specific prosodic features of an audio signal to detect the user's emotional state, e.g., frustration or annoyance (Ang et al. [6]), which can, amongst other things, indicate the occurrence of (subsequent) misunderstandings. Thus, the evaluation of the user's emotional state can influence the way a spoken dialog system reacts (Holzapfel et al. [76]). For instance, if a user is frustrated, an automated call routing system could decide to transfer the call

to a human operator versus risking the user to hang up. The system could also try to calm down the user even if the actual problem cannot be solved (Klein [101]) Application domains, where there is currently a strong emphasis in the detection of the user's emotional state for design of spoken dialog, include *tutoring* (Litman and Forbes-Riley [109]) and *social robotics* (Oudeyer [143]), i.e., robots, which interact with humans. Another feature that can be extracted from audio signals and which indirectly indicates misunderstandings is *hyperarticulated speech*, where users try to speak "extremely clear and slow" (Soltau and Waibel [176]).

**Concatenated Activations** With conversation-and-control, as with all other speech-controlled GUI approaches, activations have to be performed sequentially with sequential commands. This complies with the usual way of working with a GUI, and we argue that the sequential performance of activations is a relict of the mouse-based operation. We believe that, since with speech complex semantic information can be conveyed, it should be possible to perform sequential activations with a single command. In other words, we suggest that it should be feasible to concat the commands for two different activations into a single command and call this technique *concatenated activations*. This technique is not supported by conversation-and-control: the presence of the identification and parameter information of two different activations, i.e., two different topics, in a single command would most certainly lead to an illegal state, as two topics would be populated. It is currently unclear how conversation-and-control can be augmented to support concatenated activations and if this would help to decrease the interaction delay.

### 5.6.3 The Role of Speech Synthesis Technology

The term *speech synthesis* stands for technology that allows for generating, i.e., synthesizing human speech. A *speech synthesizer* is generally any device which implements speech synthesis technology. The details of speech synthesis are beyond the scope of this thesis. Speech synthesis is, however, a key technology for creating spoken dialog systems, since it gives a spoken dialog system the ability to respond verbally to the user's verbal input. A conversation-and-control interface is inherently a graphical user interface, as such, there is no immediate necessity to use a speech synthesizer to present system responses – the responses are available as text and can be presented by the graphical output channel directly. We consider the usage of a speech synthesizer for conversation-and-control as an option. We believe that it is an interesting direction of future work to investigate if system responses, in the speech-controlled GUI domain, should be rendered on the screen, played back by the sound card, or both (see section 7.3). Since consequently speech synthesis is closely related to our work, we give a brief overview.

Today, speech synthesizers exist in the form of software components, but their history reaches (at least) back to 1779, where the Russian Professor Christian Kratzenstein created a mechanical apparatus, which could produce vowel sounds. The first fully electrical sound synthesis device was introduced by Stewart in 1922, and it was also limited to vowel sounds.

The basics for software-based speech synthesizers components were laid by Allen, Hunnicutt, and Klatt, who demonstrated the MITalk system [3] and the Klattalk system [100]. Their systems established the theoretical foundation for speech synthesizers of today, such as Festival [21], Rethorical [156], or FreeTTS [181]. Refer to Lemmetty [106] for a review of speech recognition technology. The input for a speech synthesizer is a specific representation of the speech to be synthesized. They transform the text into digital audio signals, which can then be played back using sound cards. Possibly, the text can be augmented with pronunciation information, so that the speed, the pitch, the loudness, or other characteristics of speech can be influenced (e.g., Java Speech Grammar Format [128] or Speech Synthesis Markup Language [192]).

Speech synthesis technology can roughly be divided into two classes. The first class is based on pre-recorded audio samples of words, e.g., one sample for each word in the vocabulary of a specific application domain. The synthesis process consists of concatenating the audio samples corresponding to the sequence of words in the textual input, and the resulting speech is called *pre-recorded speech*. The second class is based on a set of mathematical functions against the time, which model the wave forms of the available phonemes of a specific world language. The synthesis process consists of concatenating the mathematical functions over the time, corresponding to the phonemes that the textual input represents. The resulting speech is called *artificial speech*. Experiments show that users prefer pre-recorded speech over artificial speech (Lines et al. [108] and Atkinson et al. [14]), as it sounds more natural, i.e., not robotic, and that it is better understood than synthesized speech (Tsimhoni et al. [188]). Pre-recorded speech, however, is not suited for the presentation of highly dynamic data, as the speech that can be produced is limited to sequences of permutations of the pre-recorded samples. For this task, artificial speech is better suited – as a real world language can be modeled satisfactory by a limited amount of phonemes, artificial speech covers a greater range of words. Hybrid speech synthesis approaches aim at solving this problem by primarily using pre-recorded speech, and by falling back to artificial speech if pre-recorded samples for words do not exist. Experiments, however, show that users understand hybrid speech even worse than pre-recorded speech and synthesized speech (Gong et al. [68]). Speech synthesis is closely related to the research area of *natural language generation* which deals with the generation of natural language in the form of text, which can then be used as the input for a speech synthesizer. An overview into the technological details of natural language generation and further references is given by Reiter and Dale [155].

#### 5.6.4 Related Work

In this section we present work that relates to conversation-and-control. In particular, we discuss work from Papineni et al. [145], extensions of this work by Denecke [48], *Speech Application Language Tags* (SALT) [162, 125], *Mercator* (Edwards and Mynatt [53]), *ICIE-Voice* Olsen et al. [91] and *LARRI* (Bohus and Rudnicky [22]).

**Dialog Goals** Papineni et al. [145] present a generic spoken dialog system, that is, a spoken

dialog system which can be configured for different application domains. Their system abstracts the application as a set of functions and models each function a set of name-value pairs, called a *dialog goal*. The semantic information of recognition results leads to the population of specific dialog goals. Application specific dependencies between particular application functions, such as the execution of one function as a precondition of another (e.g., first login then accessing data) are modeled by the concept of *admissibility*. The configurator of the spoken dialog system specifies, which dialog goals are initially admissible (e.g., the dialog goal that represents a login function). The configurator further specifies which dialog goals become newly admissible after a currently admissible dialog goal could be executed.

Specific characteristics of the system of Papineni et al. correspond to the conversation-and-control dialog model. Most obviously, dialog goals and conversation topics correspond to each other – we just introduced the name *slot* for a name-value pair. Furthermore, both dialog models represent the currently available functionality as a set of dialog goals, conversation topics, respectively. Another communality is that semantic information in a recognition result directly populates any matching name-value pair or slot. Both dialog models allow for a specific degree of freedom in the dialog flow. While the conversation-and-control model potentially allows for every available conversation topic to be invoked at any time, Papineni et al. restrict the available dialog goals to the currently admissible ones. Within a specific dialog goal, the dialog flow is unconstrained and both models apply heuristics to the respective population state of the conversation topics/dialog goals to determine the next action. However, a conversation topic is specialized to the domain of controlling of a GUI, which manifests in the fact that specific slots have specific semantic meaning to the dialog model. An example is the speakable identifier slot which indicates that the recognition result contained the identifier of a specific graphical object. With our dialog model, the dialog flow is therefore influenced by the *qualitative population state*, that is, the dialog model employs heuristics which are defined on specific types of slots (such as the relative-maximum-identification heuristic defined in section 5.3.1, which only operates on identification slots). Thus, our dialog model has a deep understanding of the application domain that it is used with, which allows for an application specific handling of recognition errors (e.g., resolution-by-historical-topic, section 5.3.2). In contrast, the dialog model of Papineni et al. is application-blind. They do not assign any special semantic meaning to the name-value pairs of dialog goals, and thus, the dialog flow is only determined by the *quantitative population state*. That is, the dialog model includes heuristics which are defined on quantitative measures, such as the current number of populated name-value pairs per dialog goal or changes of the populated name-value pairs per dialog goal between subsequent semantic updates.

**Ariadne** Ariadne (Denecke [48]) is a rapid prototyping environment for spoken dialog systems. Like the system of Papineni et al. it is generic, i.e., it can be configured for specific application domains. It also employs the concept of a dialog goal, however, it

uses typed features structures (Carpenter [35]), which are enhanced with object-oriented concepts (Denecke [47]), to model the dialog goals. This allows describing complex application entities, which in turn allows for the system to be configured for a wider range of application domains than the system of Papineni et al. However, the configuration process is complex. Not only the application domain must be configured, but also specific elements of the dialog model, including system prompts, clarification questions for each name-value pair, recognition grammars and recognition result parsing grammars. Our dialog model and domain entity representation (i.e., the conversation topic) is simpler since the application domain that it is targeted to (the controlling of GUIs) does not involve complex domain entities. Instead, the assigning of a semantic meaning to each slot, which is known to the dialog model, allows for the automatic generation of clarification questions, instead of having to specify them for each particular usage (see section 5.4). Common to Ariadne and our dialog model is the abstraction of the quantitative population state. As such, both dialog models are aware of an *empty* population state (no information present), a *complete* population state (all information present) and a *partial* population state (some information present). Our model, however, further decomposes the partial population state under consideration of the quantitative population state (*parameters complete* and *parameters incomplete*), which allows for a deeper understanding of the information content in the conversation topic.

**SALT** *Speech Application Language Tags* (SALT) [162, 125] is an extension to XML-based user interface markup languages (like HTML) providing an abstract interface to speech recognition and speech synthesis technology. This interface consists of a set of specific speech events (e.g., an event that represents a new recognition result) and a set of corresponding event handlers (e.g., `onreceive`, which is invoked if a new recognition result is available). The developer has access to the interface via specific XML elements, the *SALT tags*. A *SALT-enabled* browser detects and interprets them in the context of the current XML document. Given a web browser, the developer could for instance implement the `onreceive`-event handler as a JavaScript function which analyzes the recognition result and which populates a web form accordingly. The advantage of SALT is that it provides transparent support for speech-related services from within web pages and potentially across specific web browsers. The disadvantage is that it only supports the input and output technologies for spoken dialog systems – it is the developers tasks to design and implement the "glue", e.g., the dialog model and the semantic analysis algorithms. However, using SALT as interface to speech services the conversation-and-control dialog model could be implemented according to our framework specification (see section 6.1).

**Mercator** Mercator (Edwards and Mynatt [53]) is an early system that aims at supporting visually impaired or blind users in the usage of GUIs, by mapping the GUI to an auditory interface. For this, Mercator defines *audio interface components* (AIC) for commonly used graphical objects. AICs, speaking in terms of the MVC pattern (see section 2.8.3),



can be considered as the auditory views of graphical objects. Thus, a graphical object has both a graphical view, which it has inherently, and an auditory view, which is created by Mercator. The system generates and manages a tree of AICs corresponding to the tree structure of the graphical objects visible on the screen. Mercator keeps this tree in synchronization with the screen throughout the usage of the system. Users physically perform navigation and activation on the original graphical objects via keyboard commands, e.g., via the cursor keys. Whenever the focus changes, Mercator detects this and presents the AIC corresponding to the newly focused graphical object. Thus, the user gets an auditory cue as to which graphical object is currently focused or in which state the object currently is. Since Mercator relies on conventional input via mouse and keyboard, conversation-and-control is an orthogonal approach that could be combined seamlessly with Mercator to create a 1:1 mapping of a GUIs to spoken dialog systems. Such a combined system could provide access to GUIs for users, who suffer from both visual and motor impairments.

**ICIE-Voice** The ICIE environment (Olsen et al. [141]) is an architecture and a framework for providing pervasive access to applications. *ICIE-Voice* (Olsen et al. [91]) is part of the ICIE environment and facilitates pervasive access to GUIs via a speech-only channel, such as a phone line. With ICIE-Voice, graphical objects are augmented with semantic information, such as their name, their type, the spatial relationship to other graphical objects and information about the containment in other (groups of) components. From this semantic information, a spoken representation of a specific graphical object is generated and presented via synthesized speech. In order to make the spoken presentation configurable, each graphical object has an optional speech descriptor which defines further rules for generating the synthesized speech. Similar to Mercator, and in contrast to conversation-and-control, ICIE-Voice does not allow for spoken activation of graphical objects. Only navigation of graphical objects is possible, and the currently focused graphical object is presented to the user as described above. Using spoken commands for navigation, ICIE-Voice offers two different navigation types: *type-based navigation* and *geometry-based navigation*. With type-based navigation the user navigates between graphical objects of the same type using "next <type>", "previous <type>", "first <type>" or "last <type>" commands (this was already described in section 1.2.3). Hereby ICIE-Voice silently assumes that programmers include an ordering into the screen objects, e.g., graphical objects representing days in a calendar application. Geometry-based navigation allows for moving upward or downward in the graphical object hierarchy with respect to the current position, or based on the spatial relationship to each other. This is in contrast to conversation-and-control which employs random navigation – any graphical object can be navigated to at any time.

**LARRI** The *Language-based Agent for Retrieval of Repair Information* (LARRI) described in Bohus and Rudnicky [22], is a multi-modal application for supporting activities in the maintenance and repair domain. Due to the preponderance of hands- and eye-busy

situations in this domain, the system has a GUI which is designed for being run on a wearable computer with a head-mounted display, and which can be controlled by a rotary mouse (Siegel et al. [172]) or by spoken dialog. LARRI and applications built with our conversation-and-control framework (described later in section 6.1) share the commonality that the user can seamlessly switch between spoken input and conventional input (mouse, rotary mouse, respectively) to control the GUI – depending on which input modality better suits the current situation. However, our approach is application-blind as the conversation-and-control dialog model is designed to control the functionality of graphical objects. It is therefore applicable to different application domains, whereas the dialog model in LARRY is specific to the repair and maintenance domain. It is, however, based on a generic dialog model, described in Wei and Rudnicky [194].

# 6

## Validation

*"The most exciting phrase to hear in science, the one that heralds the most discoveries, is not 'Eureka!' (I found it!) but 'That's funny...'"*

Isaac Asimov (1920 - 1992)

### Overview

In this chapter we validate the conversation-and-control approach in three steps. First, we introduce the design and implementation of a conversation-and-control framework based on which we have created prototype systems from the technical maintenance domain. This shows the practical feasibility of conversation-and-control interfaces. Second, we calculate the interaction delay of conversation-and-control and compare it to the interaction delay of conventional command-and-control. The results of these calculations show, that the interaction delay of conversation-and-control is, as expected, significantly lower than the interaction delay of conventional command-and-control. Third, we have conducted an experiment with 16 test subjects to empirically validate the reduction of the task completion time. The results of the experiment, which we discuss in detail, demonstrate a significant reduction of the average task completion time.

## 6.1 Conversation-and-Control Framework

In this section we describe a conversation-and-control framework. We begin with the core requirements in section 6.1.1 which influenced the design, described in section 6.1.2. The implementation of the framework is discussed section 6.1.3 and section 6.1.4 explains a new heuristic which emerged from testing experience during the implementation phase. Finally, section 6.1.5 describes prototype systems from the technical maintenance domain which have been created with the framework.

### 6.1.1 Requirements

The following requirements have led the design of the conversation-and-control framework.

**Core Component** We require that a reusable core component for conversation-and-control interfaces exists, which is independent of the characteristics of the graphical objects that it controls.

**Swing Catalog** The framework must support the Swing catalog (section 4.1.1), since we used the Swing catalog as the conceptual basis for developing conversation-and-control.

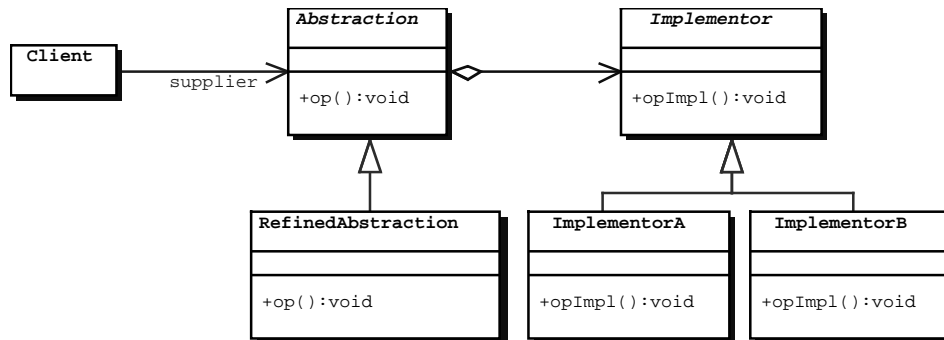
**Extensibility** The framework must be extensible beyond the Swing catalog, beyond Java Swing, respectively, so that it can also be implemented for other GUI toolkits (e.g., Microsoft Foundation Classes [151] or Standard Widget Toolkit (SWT) [139, 115]).

**Mode Switching** The framework must support the *command mode*, allowing controlling graphical objects by conventional command-and-control. Furthermore, the *conversational mode* must allow for controlling graphical objects by conversation-and-control. This will reduce the implementation effort for the experiment that measures the task completion time of command-and-control versus conversation-and-control.

**Markup Language-based GUIs and Native GUIs** The framework must support both markup language-based GUIs and native GUIs. This requirement was driven by the *mobile maintenance project* (see section 6.1.5).

### 6.1.2 Design

The framework design is based on state-of-the-art *software design patterns*. Detailed explanations of software design patterns, including the ones we used, can be found in Brügge and Dutoit [26], Shalloway and Trott [168] and Gamma et al. [62]. Crucial to our framework design is the combination of the *adapter pattern* and the *bridge pattern*. The adapter pattern adapts different interfaces of components or objects. Its characteristics will become clear in the remainder of our discussion. The bridge pattern helps to "decouple an abstraction from its implementation so that the two can vary independently" (Gamma et al. [62]). Since its



**Figure 6.1:** Bridge pattern.

characteristics are complex, we explain them in the following, before we go into the details of its application to the conversation-and-control framework design.

A *client* (refer to Figure 6.1) accesses the functionality of a *supplier* via a standardized interface. Internally, the supplier is separated into the so called *abstraction* and the *implementor*. The abstraction represents the standardized interface for the client, therefore, any (internal) refinements of the abstraction (e.g., subclassing) are transparent to the client. The abstraction has an association to a specific implementor which provides the actual functionality of the supplier. The implementor has a standardized interface as well, which is, however, only exposed to the abstraction. Therefore, the implementor can change independently from the abstraction, and therefore independently from client. The actual association between abstraction and implementation could even be changed dynamically, e.g., by using the *factory pattern*. An example for an application of the bridge pattern is networking. Consider the abstraction (the supplier, respectively) to be an object which represents a network link. It could provide a generic interface to send and receive data, however, it could use different implementors to send and receive data via the TCP or the UDP protocol (Tanenbaum [184]). The abstraction itself could vary in that a secure refinements exists, which could apply an encryption algorithm before sending the data – either via TCP or UDP, depending on the current implementor.

Having explained the bridge pattern we now look into the essentials of the conversation-and-control framework design, referring to Figure 6.2 on the following page. The *conversation-and-control manager* encapsulates both the conversation-and-control dialog model and the dialog state<sup>1</sup>. It exposes a simplified interface to the logic of the conversation-and-control dialog model, and can therefore be considered as a *facade* (*facade pattern*). This interface includes the methods `setConversationMode()` and `setCommandMode()`, which toggle the operational mode of the conversation-and-control dialog model between conversation mode and command mode (refer to the legacy emulation characteristic, section 5.6.1). This fulfills the mode switching requirement.

<sup>1</sup>These objects were introduced in section 5.5.

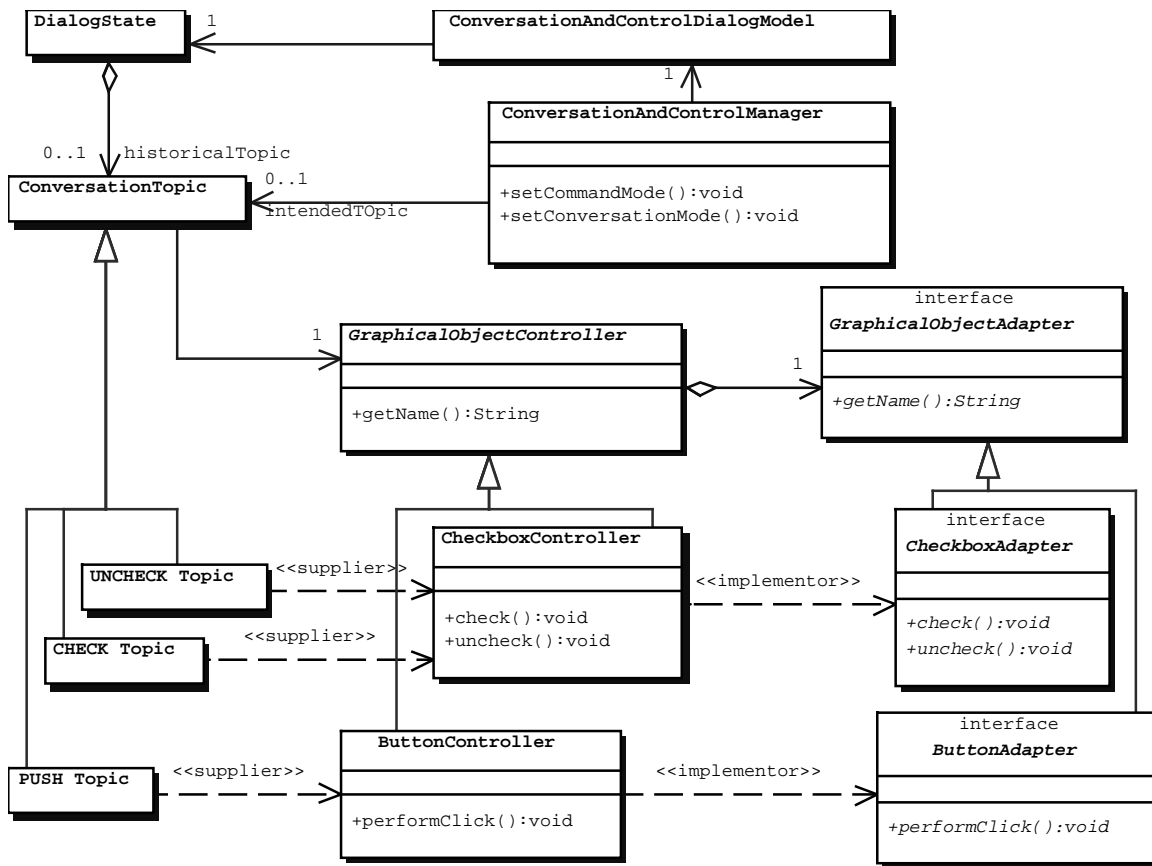


Figure 6.2: Conversation-and-control framework design.

The clients from the bridge pattern map to the conversation topics, which are also depicted in Figure 6.2. Each specific conversation topic represents a specific control action of a specific graphical object, thus, each conversation topic intends to call specific functions of a graphical object. We decouple conversation topics from concrete graphical objects by introducing an abstraction layer, called *graphical object controllers*. Each such controller abstracts functions and properties of a specific type of graphical object, and thus, provides a standardized interface for respective conversation topics. Graphical object controllers map to the abstractions from the bridge pattern. However, as there might be common functions and properties of graphical objects, there is an entire hierarchy of graphical object controllers. We indicate this coherence in the Figure using the example of the *button controller*, the *checkbox controller* and the respective conversation topics *PUSH*, *CHECK* and *UNCHECK*. In order to support different GUI toolkits as required, we introduce an additional abstraction layer, called the *graphical object adapters*. A specific graphical object adapter adapts the interface of a concrete graphical object to the interface which the corresponding graphical object controller expects. Thus, the same graphical object controller can, transparent to the conversation topic, control differ-

ent implementations of graphical objects. Consequently, the (hierarchy of) graphical object adapters map to the implementors from the bridge pattern.

The application of the bridge pattern inherently suggests that the conversation-and-control framework design is extensible. This covers the Swing catalog requirement, the extensibility requirement and the requirement for markup language-based and native GUIs.

### 6.1.3 Implementation

The applications from the mobile maintenance domain, which we dealt with in the scope of the mobile maintenance project (section 6.1.5), are based on *Java 2 Enterprise Edition* (J2EE) [2], of which we give a brief overview in section 6.1.3. As a logical consequence, we created an implementation of the conversation-and-control framework in J2EE and describe its core characteristics in sections 6.1.3 and 6.1.4.

#### J2EE

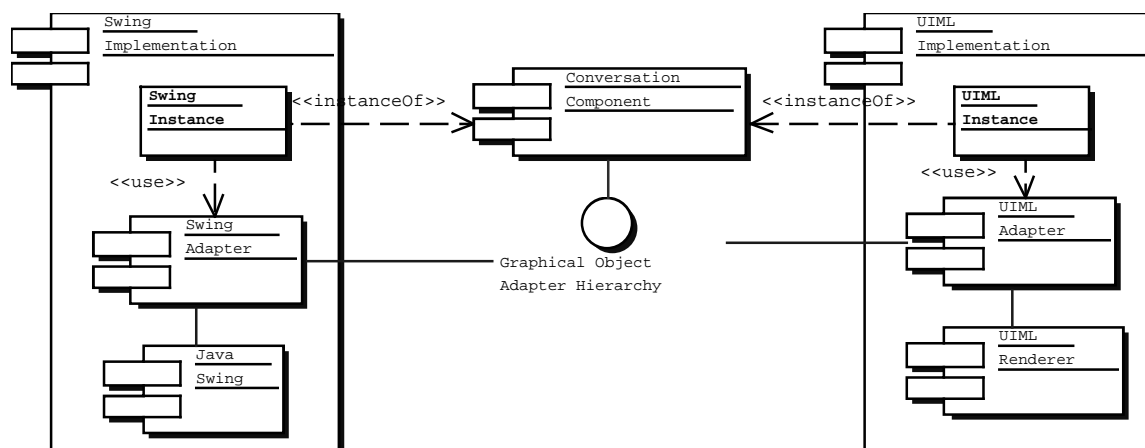
J2EE is a specification for web-based client-server applications using Java. It merges together many of the Java API extensions and defines interaction rules between them. J2EE enhances the classical high level 3-tier application architecture (presentation layer, application layer and persistent storage layer) by splitting the presentation layer across the client and the server. Thus, J2EE specifies four layers: the *client tier*, the *web tier*, the *Enterprise Java Bean* (EJB) tier and the *Enterprise Information System* (EIS) tier. Client tier and web tier emerge from the split of the presentation layer: the client tier models the part of the presentation logic which runs on the client, and the web tier models the part of the presentation logic which runs on the server. A typical example is a J2EE application which involves a web browser-based GUI: the web tier provides or generates the HTML pages <sup>2</sup> of which the GUI consists, and the client tier presents the HTML pages (e.g., the web browser). The EJB tier of J2EE represents the application layer and is initially based on the *EJB* specification. The EIS tier represents the persistent storage layer and involves Java technologies like *JDBC* or *Hibernate* <sup>3</sup>.

J2EE provides a set of tools and technologies which is more than sufficient for the creation of web-based client-server applications, however, the actual creation process is often unstructured and ad-hoc. This problem has been recognized and different solutions have been proposed. For instance, in Löhr et al. [110], we concentrate on the J2EE client and web tier and describe a framework for creating web browser-based GUIs in J2EE. It encapsulates solutions to common problems occurring with this task, e.g., stale web browser requests and missing support for reusing HTML components. In Bass et al. [17] we describe a case study with the *Luther Architecture*, a high level architecture for J2EE applications, which suggests to assemble J2EE applications from generic components that live in any of the four J2EE layers. The

---

<sup>2</sup>Various technologies for creating HTML can be used in the scope of J2EE, e.g., JSP [129], Java Servlets [130], or Java Server Faces.

<sup>3</sup>Hibernate is not part of the J2EE specification but is commonly used as a light-weight substitute of the EJB technology.



**Figure 6.3:** Conversation-and-control framework implementation.

Luther architecture also employs multiple J2EE-specific software design patterns, as described in Alur et al. [5]. Such patterns also have been used by other architecture proposals, such as JAFAR (Guelfi and Sterges [72]). However, since the applications from the technical maintenance domain, for which we provided prototype conversation-and-control interfaces, are also based on the Luther architecture, we describe the integration of the conversation-and-control framework with the Luther architecture in the following section.

### Conversation-and-Control and the Luther Architecture

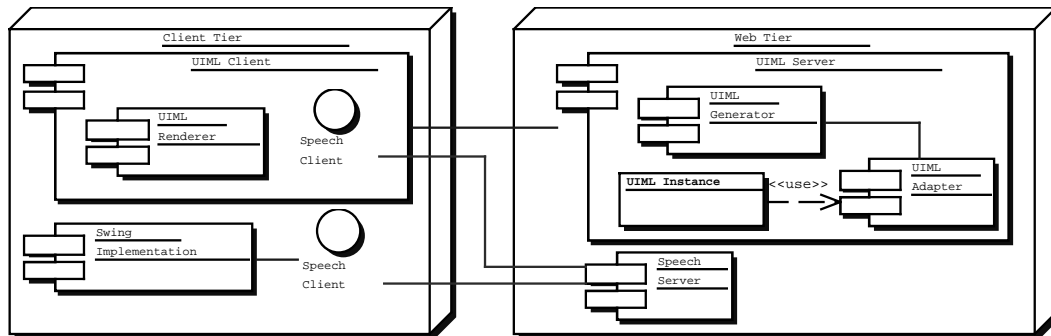
We created a Java implementation of the generic core components of the conversation-and-control framework, called the *conversation component*. We furthermore created two implementations of the graphical object adapter hierarchy (using Java as well): the *Swing adapter* was implemented against Java Swing to support the Swing catalog requirement and the native GUI requirement, and the *UIML adapter* was implemented against the *User Interface Markup Language* (UIML)<sup>4</sup> in order to support the markup language-based GUI requirement. For the latter we reused a UIML rendering component and a corresponding UIML vocabulary provided by Funck [61]. As a result, we obtained a Java Swing-based implementation of the conversation-and-control framework, which we call *Swing implementation* and a UIML-based implementation which we call *UIML implementation*, as depicted in Figure 6.3.

In order to integrate the conversation-and-control framework implementations with the J2EE environment, with the Luther architecture, respectively, we performed three further steps. First, by extending a Java framework for server-based speech services<sup>5</sup> with the Java Speech API [131], provided by Sipek [174], we created a middleware component for distributed speech recognition that can be deployed both in the J2EE client tier and the J2EE web

<sup>4</sup><http://www.uiml.org>

<sup>5</sup>We consider speech recognition and speech synthesis to be speech services.





**Figure 6.4:** Conversation-and-control framework in J2EE.

tier. We call this middleware component the *speech framework* and it consists of a *speech client* and a *speech server*. The speech client lives in the J2EE client tier and provides transparent access to the speech server, which lives in the web tier. Second, we distributed the particular components of our conversation-and-control framework implementations over the client and the web tier as follows: the entire Swing implementation was embedded into the J2EE client tier, accessing the therein living speech client. The UIML renderer of the UIML implementation was embedded into the client tier and accessed the therein living speech client as well, forming together the so called *UIML client*. The UIML instance of the conversation component and the UIML adapter were embedded into the web tier. Then, as a third step, reusing conceptual ideas from our framework for web browser-based J2EE clients (Löhr et al. [110]), we created a J2EE web tier component for generating UIML-based GUIs, called the *UIML generator*. Similar to Java Servlets or JSPs it generates UIML which is delivered to and rendered by the UIML renderer. The UIML generator, the UIML instance of the conversation-and-control component and the UIML adapter together form the *UIML Server*. We depict this setup in Figure 6.4.

#### 6.1.4 Self-Elimination Heuristic

During the implementation of the framework we created several conversation-and-control interfaces as test cases. We observed, that the conversation-and-control dialog model asked clarification questions although the user's intention was clear from the current internal state of the affected graphical object. For instance, if a specific selection item of a list was currently deselected and the user uttered its speakable identifier, the system asked question like "Do you mean select or deselect?". This is in accordance with the model definition: by uttering just the selection items identifier, eventually, the relative-maximum-identification heuristic would return the respective *SELECT*- and *DESELECT*-topic for being clarified by resolution-by-clarification-question. From a usability perspective, it should however be obvious, that the user intends to invoke the respective *SELECT*-topic, as the selection item is currently not

selected.

To solve the above described problem, we took advantage of the fact that the conversation topic, as being designed for a specific graphical object, knows about the internal state of a graphical object (the interface of the respective graphical object controllers just needs to be enhanced). We defined a procedure for the implementation of the `SemanticUpdate` operation of conversation topics, called *self-elimination*, which is described below:

1. Perform the semantic update just as required for the conversation topic.
2. Before returning the call, validate the contents of the populated slots against the state of the graphical object.
3. If the slot content is inconsistent with the graphical object state, remove the values of all slots.
4. Return call.

Thus, whenever a conversation topic detects, that the current state of the graphical object is not consistent with the semantic information from the recognition result, it clears out all values from its slots. Self-elimination would, however, break with the principle that the state of a conversation topic cannot move towards the empty state (see section 5.2.1). To avoid compromising the logic of the dialog model, we only apply self-elimination on conversation topics which are currently empty. As a result, a topic that eliminates itself remains empty after a semantic update and the dialog model is not aware of the existence of self-elimination.

Going back to the introductory example, the *DESELECT*-topic would get the parameter slot populated with the identifier that the user intends to select. By determining the selection state of item identified by the value in the parameter slot, the *DESELECT*-topic can infer, that "the user can actually not mean to invoke it" – the item is not selected, therefore it cannot be deselected. Thus, it eliminates itself. For the dialog model, it would remain empty and the relative-maximum-identification heuristic would just identify the *SELECT*-topic.

Self-elimination supports the legacy emulation characteristic of conversation-and-control, since, as described with the example, it allows to omit activation names – at least for specific activations.

### 6.1.5 Prototype Systems

We are particularly interested in the question as to whether conversation-and-control is applicable to real application domains and, if so, how it then will perform. This leaves, however, the question as to which application domain should be chosen. We recognized that there is currently little experience with speech-controlled applications in industrial domains (Bürky et al. [31]), such as *technical maintenance*. Since the technical maintenance domain typically involves graphical data, it seems to be a promising fit for speech-controlled GUIs <sup>6</sup>

---

<sup>6</sup>This has also been recognized by others, as for instance the creators of the LARRI system (Bohus and Rudnicky [22]).

. Therefore, within the *mobile maintenance project* in the scope of the *High-Tech-Offensive Zukunft Bayern* program of Bayerische Staatskanzlei, we created two conversation-and-control interfaces for the maintenance applications *Mentor* and *Publisher* [84]. Partial results of the mobile maintenance project are documented in Atabey and Kayali [12]. The conversation-and-control interfaces prove the practical feasibility of conversation-and-control from a developers perspective, i.e., they show that using the very same conversation-and-control framework, multiple, different applications can be created. At the same time it prepares and motivates future work emerging from this dissertation towards validating the conversation-and-control approach one the one hand, and, on the other hand, towards gathering more experience with speech-controlled industrial applications in the field.

Describing the created conversation-and-control interfaces in detail goes beyond the scope of this dissertation, therefore, we only give a brief overview. Both *Mentor* and *Publisher* are J2EE applications based on the Luther architecture (refer to sections 6.1.3 and 6.1.3) employing web browser-based user interfaces. The Luther architecture clearly defines the actual application logic as a facade (see facade design pattern in Gamma et a. [62]) to a therein encapsulated assembly of generic Luther components, such as a generic component for user management, for document management or for workflow management. The facade, internally, defines interaction rules between the generic components and applies domain specific logic, which it exposes via a well-defined interface, to which we refer as the *application interface*. Any user interface, such as the respective web browser-based user interfaces of *Mentor* and *Publisher*, access the application interface. Following this paradigm we have created conversation-and-control interfaces for both *Mentor* and *Publisher*, which accessed their respective application interfaces. Although, up to now, we only implemented a limited set of the available functionality, the design of the conversation-and-control framework integrated well with the Luther architecture. To conclude this chapter, we present and briefly discuss selected screen shots.

*Mentor* (Figure 6.5 on the next page) is a configurable application platform which allows for processing of encapsulated data packages, so called *documents*, by multiple users. Documents are sent through roles of a workflow and are manipulated by users which are assigned to participate in respective roles. In Figure 6.5(a) appears the login screen of the web based *Mentor* user interface. It consists of two input fields for the user name and the password and an image button to submit this information to the server. Figure 6.5(b) shows the corresponding screen from the conversation-and-control interface. Note, that we substituted the input field for the user name by a drop down box containing registered users, which speeds up logging in. To further improve this, it might be reasonable to personalize the respective client installation, so that the name is pre-populated. We further omitted, i.e., neglected, the password field. It contradicts security to spell out loud a password, therefore, we propose to use other authentication mechanisms, such as *speaker recognition*<sup>7</sup>. The image button has been replaced by an ordinary button, so that the user knows what to speak. Figure 6.5(d) shows the

---

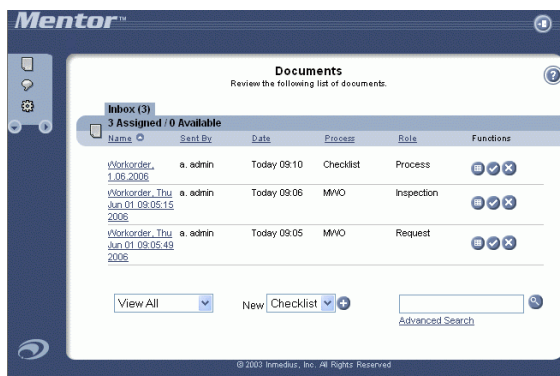
<sup>7</sup>Speaker recognition, as opposed to speech recognition, is the task of recognizing users from their voice patterns.



(a) Login screen, web



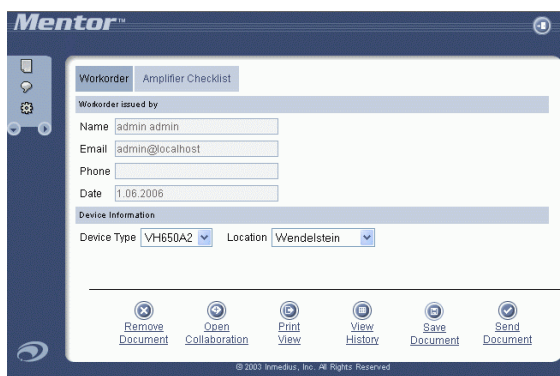
(b) Login screen, conversation



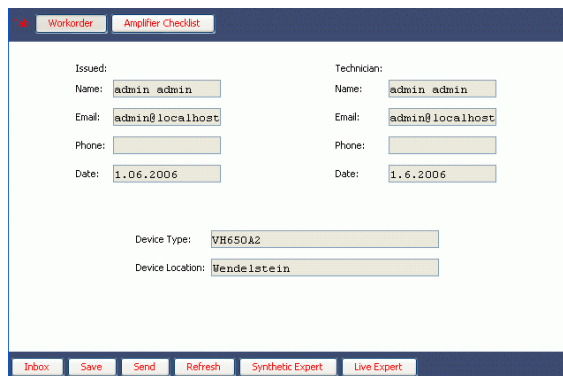
(c) Inbox screen, web



(d) Inbox screen, conversation



(e) Document screen, web



(f) Document screen, conversation

**Figure 6.5:** Screenshots of Mentor conversation-and-control interface and web interface.

so called *inbox* of Mentor, which gives access to the documents which the currently logged in user may manipulate. Each table row represents a document and gives status information. The buttons at the end of the row represent specific document functions, such as propagation in the workflow or deletion. The link on the left of each row opens the respective document.

In the conversation-and-control interface (Figure 6.5(c)) we have modeled the inbox as a radio button list and factored out the buttons for the document functions as a button bar on the bottom of the screen. They become enabled as soon as a document has been selected. Figure 6.5(e) depicts a document view in the web browser-based interface and Figure 6.5(f) contains its conversation-and-control counterpart. Image links/buttons have been replaced by ordinary buttons.

Publisher (Figure 6.6 on the following page) is a display system for class 4 and 5 *interactive electronic technical manuals* (IETM) [41, 57]. Figure 6.6(a) contains an exemplary screen from a German television broadcasting amplifier maintenance IETM (refer to Atabey and Kayali [12]). The transition to the conversation-and-control interface (Figure 6.6(b)) is straight forward<sup>8</sup>, although the conversation-and-control framework is currently limited in the available screen elements (e.g., graphics are not yet supported). Figures 6.6(c) and 6.6(d) contain further screen shots from Publisher's conversation-and-control interface, depicting standard IETM content. Our approach in creating a conversation-and-control interface corresponding to the existing web browser-based interface of Publisher exploits the fact that users are already familiar with it. We argue that this results in a smaller learning curve, because users "only" have to learn the new input modality speech, instead of additionally facing the burden of familiarizing with a new GUI (cp. LARRI [22]). The validation of this argument is, however, considered future work. Also, for future research, we propose to create a user experiment with the presented conversation-and-control interfaces, to evaluate their performance in a domain-specific, i.e., hands-free, environment.

## 6.2 Interaction Delay of Conversation-and-Control

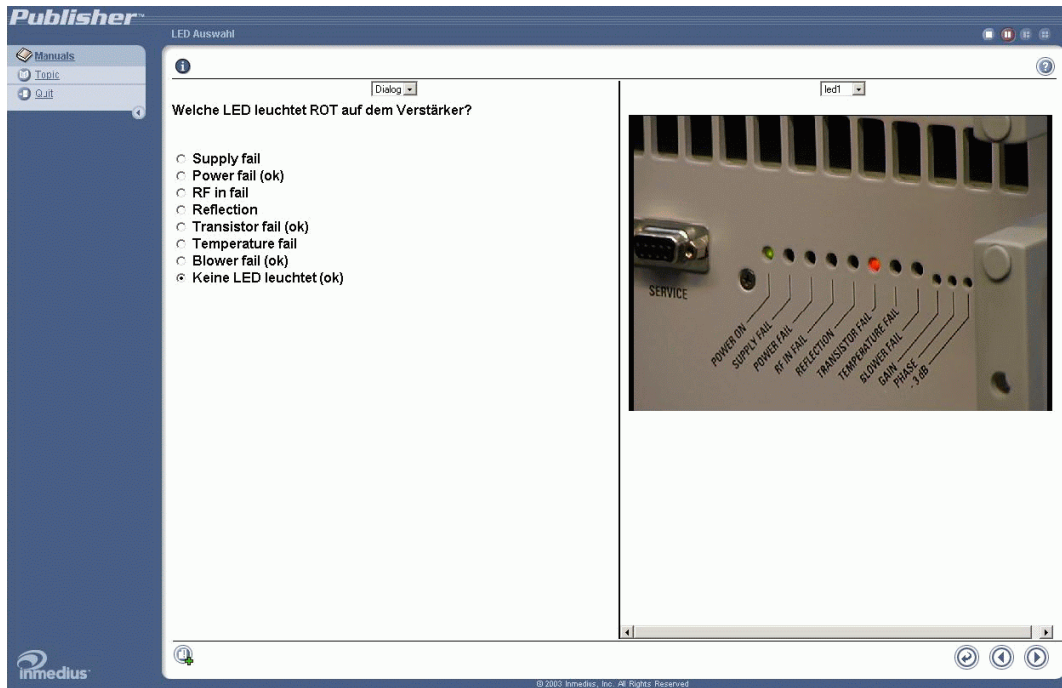
We now deal with the interaction delay of conversation-and-control. In section 6.2.1 we calculate the interaction delay according to the interaction delay model (presented in chapter 3) and in section 6.2.2 we discuss and compare the results against the interaction delay of conventional command-and-control.

### 6.2.1 Calculations

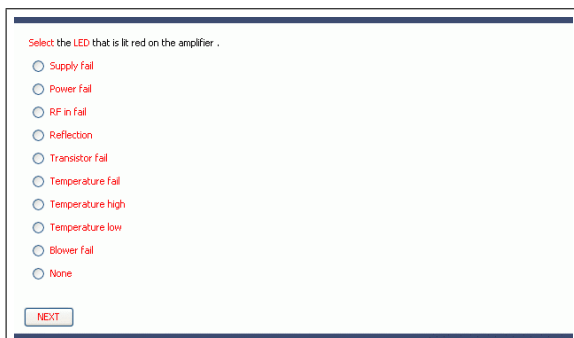
Calculating the interaction delay requires us, like with the analyses of other speech-controlled GUI approaches in chapter 4, to derive the vocabulary and the set of valid commands. In section 4.2.1 we have identified a vocabulary superset for command-and-control approaches of 70 words. This superset represents the entire vocabulary of conversation-and-control: the contained words make up the constituents of speakable identifiers, define the names of all occurring activations and include the constituents of textual representations of all possible parameters for these activations – at least, in the scope of the Swing catalog. Regarding the number of valid commands we refer to Table 6.1 on page 167, which defines qualified activa-

---

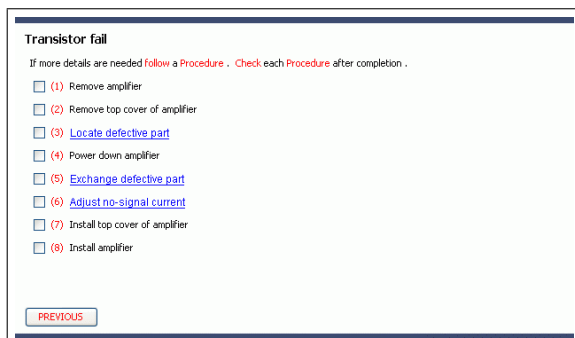
<sup>8</sup>To save space we have cut out only the content area of the manual.



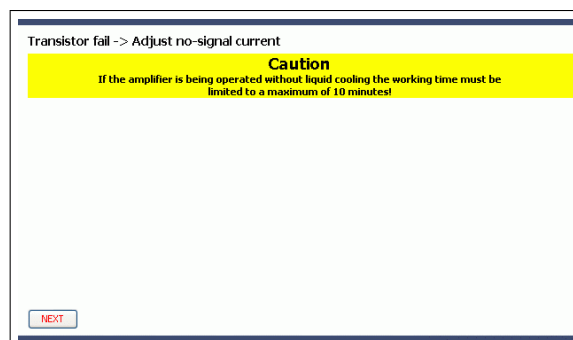
(a) Publisher, manual view, web



(b) Publisher, manual view, conversation



(c) Publisher, manual view 2, conversation



(d) Publisher, manual view 3, conversation

**Figure 6.6:** Screenshots of Publisher web interface and conversation-and-control interface.

## 6.2. INTERACTION DELAY OF CONVERSATION-AND-CONTROL

| Execution              | Interaction with Conv.-and-Ctrl.<br>Notion $x \oplus y$ denotes concatenation of command $x$ and $y$ . | Interaction with Conventional Cmd.-and-Ctrl.   |
|------------------------|--|--|
| $E_{button}$           | $(CMDNAV_{id})$  | $(CMDNAV_{id})$  |
| $E_{checkbox-check}$   | $(CMDNAV_{id})$  | $(CMDNAV_{id})$  |
| $E_{checkbox-uncheck}$ | $(CMDNAV_{id})$  | $(CMDNAV_{id})$  |
| $E_{radio}$            | $(CMDNAV_{id})$  | $(CMDNAV_{id})$  |
| $E_{dropdown}$         | $(CMDNAV_{id} \oplus CMDNAV_{id})$   | $(CMDNAV_{id}, CMDNAV_{id})$   |
| $E_{list-select}$      | $(CMDNAV_{id} \oplus CMDNAV_{id})$   | $(CMDNAV_{id}, CMDNAV_{id})$   |
| $E_{list-deselect}$    | $(CMDNAV_{id} \oplus CMDNAV_{id})$   | $(CMDNAV_{id}, CMDNAV_{id})$   |
| $E_{menu}$             | $(CMDNAV_{id} \oplus CMDNAV_{id})$   | $(CMDNAV_{id}, CMDNAV_{id})$   |
| $E_{tree-select}$      | $(\underbrace{CMDNAV_{id}, \dots, CMDNAV_{id}}_{n_{depth}}, CMDACT_{SELECT} \oplus CMDNAV_{id})$       | $(CMDNAV_{id}, \underbrace{CMDNAV_{id}, \dots, CMDNAV_{id}}_{n_{depth}}, CMDACT_{SELECT})$ |
| $E_{tree-deselect}$    | $(CMDNAV_{id} \oplus CMDACT_{DESELECT} \oplus CMDNAV_{id})$  | $(CMDNAV_{id}, CMDNAV_{id}, CMDACT_{DESELECT})$  |
| $E_{input}$            | $(CMDNAV_{id} \oplus \underbrace{CMDACT_{APPEND_x}, \dots, CMDACT_{APPEND_x}}_{n_{text}-1})$           | $(CMDNAV_{id}, \underbrace{CMDACT_{APPEND_x}, \dots, CMDACT_{APPEND_x}}_{n_{text}})$       |
| $E_{spinner}$          | $(CMDNAV_{id} \oplus \underbrace{CMDACT_{APPEND_x}, \dots, CMDACT_{APPEND_x}}_{n_{digit}-1})$          | $(CMDNAV_{id}, \underbrace{CMDACT_{APPEND_x}, \dots, CMDACT_{APPEND_x}}_{n_{digit}})$      |
| $E_{spinner-dec}$      | $(CMDACT_{DECREASE} \oplus CMDNAV_{id})$   | $(CMDNAV_{id}, CMDACT_{DECREASE})$   |
| $E_{spinner-inc}$      | $(CMDACT_{INCREASE} \oplus CMDNAV_{id})$   | $(CMDNAV_{id}, CMDACT_{INCREASE})$   |
| $E_{slide}$            | $(CMDNAV_{id} \oplus CMDNAV_{id})$   | $(CMDNAV_{id}, CMDNAV_{id})$   |

**Table 6.1:** Qualified activation commands for basic executions, specified using terminology that has been established in Table 4.3 on page 86.

tion commands for basic execution from the Swing catalog. Three characteristics of Table 6.1, which we discuss in the following, strike the eye.

First, we recognize that the qualified activation commands contain very few activation names, although qualified activation requires the commands to contain activating information. Instead, many commands just consist of two concatenated speakable identifiers. This is, however, no violation of the qualified activation principle: missing activation information, e.g., activation names, just denotes an information error which the system must aim to resolve. As we showed previously in section 6.1.4, self-elimination is suited for dealing with missing activations names. Thus, the commands in Table 6.1 are optimized regarding their length, under consideration of the system's capabilities to automatically resolve information errors.

The second characteristic of Table 6.1 is, that the length of some qualified activation com-

mands is equal to the sum of the words of the commands that they are merged from – however, this only applies to interactions which already consisted of one command only, e.g., the interactions for  $E_{button}$  or  $E_{checkbox-check}$ . Other than that, Table 6.1 proves the construction hypothesis from section 5.1: qualified activation commands can be constructed from  $N$  commands, so that their increased length does not compensate the interaction delay reduction, which was gained from saving  $N - 1$  recognition delays.

Third, some qualified activation commands contain information which might not (yet) be visible on the screen at the time the user starts uttering the command. For instance, the interaction for  $E_{dropdown}$  contains the identifier of the option to be selected, although the drop down box might not be open. We argue that the necessity to open a drop down box (or for instance a menu) is a relict of its mouse based operation – as long as the options are not visible they cannot be navigated to with the mouse. The situation is different if speech is used as input modality, and if we assume that the user is skilled in using the particular GUI instance. In other words, we assume that the user recalls, e.g., from previous working sessions, which options the drop down box contains. Then, with speech, there is no need to open the drop down box first – the desired option can be selected right away, and we show the practical feasibility of this assumption later on when we discuss the outcome of the experiment (section 6.3.6). There are, however, preconditions for this assumption to be feasible. The drop down box must not be populated with dynamically changing information, and the speakable identifiers for the respective options must be constant. If the user does not remember the option's identifier or the preconditions do not hold, the user can still "fall back" to open the drop down box prior to selecting an option. This would, however, not reduce the interaction delay compared to conventional command-and-control for this particular graphical object.

We now derive the number of valid commands for conversation-and-control. We argue that each word in the vocabulary represents a valid command, namely, a correction – any speakable identifier of length 1, any character and digit, and any activation name is subject to be spoken by the user as an answer to a clarification question. Thus, we initially have 70 valid commands (corrections) of length 1. Conversation-and-control inherits 100 possible speakable identifiers, of which each is a valid command according to Table 6.1. The verbal forms of the digits 1 – 9, which are speakable identifiers as well, have already been accounted for – they occur as distinct words in the vocabulary. Thus, we have 91 additional valid commands representing a single speakable identifier, e.g., the pushing of a button. Looking at Table 6.1 we further recognize that some executions can be performed by uttering a concatenation of a pair of speakable identifiers. Since we do not restrict the way according to which speakable identifiers are assigned to graphical objects, there are  $100 \cdot 99$  possible combinations of speakable identifier pairs, which makes another 9900 valid commands. In addition to that, the formerly separate commands  $CMDACT_{SELECT}$ ,  $CMDACT_{Deselect}$ ,  $CMDACT_{DECREASE}$  and  $CMDACT_{INCREASE}$  can be concatenated with any speakable identifier, which makes an additional 400 valid commands<sup>9</sup>. Additionally, the command  $CMDACT_{Deselect}$  can also occur in combination with two

<sup>9</sup>The conversation-and-control dialog model "understands" more than the given concatenations of formerly separate commands with speakable identifiers, e.g., it understands  $(CMDACT_{PUSH} \oplus CMDNAV_{id})$ ,



| Interaction Delay in ms | Word Error Rate |      |      |      |       |       |       |        |
|-------------------------|-----------------|------|------|------|-------|-------|-------|--------|
|                         | 0%              | 1%   | 5%   | 10%  | 20%   | 30%   | 40%   | 50%    |
| Minimum                 | 2642            | 2696 | 2927 | 3262 | 4128  | 5392  | 7339  | 10568  |
| Average                 | 3356            | 3473 | 4005 | 4839 | 7379  | 12085 | 21759 | 44647  |
| Maximum                 | 4355            | 4579 | 5628 | 7375 | 13290 | 25912 | 56006 | 139360 |

**Table 6.2:** Calculated minimum, average and maximum interaction delay per basic execution of conversation-and-control.

speakable identifiers, which makes another 9900 valid commands. Finally, the spelling mode, which conversation-and-control inherits as well, defines 37 commands which can either occur separately or concatenated with a speakable identifier – which adds another 3737 valid commands. In total, conversation-and-control defines  $70 + 91 + 9900 + 9900 + 400 + 3737 = 24098$  valid commands out of a vocabulary of 70 words. The length of the longest valid command is 5 ( $E_{tree-deselect}$ ).

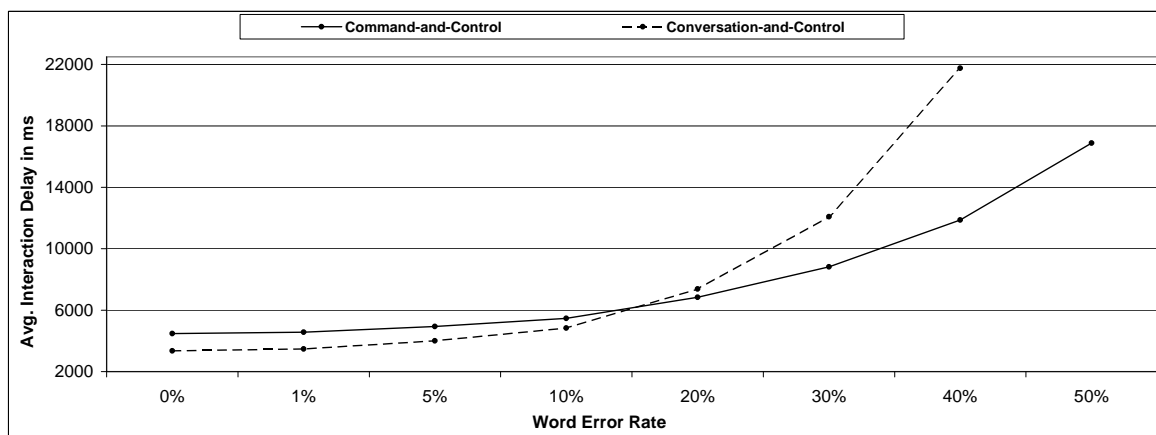
With these considerations we have calculated the interaction delays for different word error rates of basic executions using conversation-and-control (refer to appendix B.2, Table B.15, for a detailed documentation). The intentions, which the interactions are based on, are the same as with conventional command-and-control. We summarize the results in Table 6.2 which depicts the minimum, the average, and the maximum interaction delay per basic execution of conversation-and-control. Like with the cumulative interaction delay tables in chapter 4 we exclude the basic executions  $E_{tree-select}$ ,  $E_{input}$  and  $E_{spinner}$ , as they contain the a priori unknown values  $n_{depth}$ ,  $n_{text}$  and  $n_{digit}$  (also refer to section 4.1.3). In the following section we compare the results for conversation-and-control to conventional command-and-control.

## 6.2.2 Comparison to Conventional Command-and-Control

From the discussion in sections 4.2.4 and 4.4.3 we know that conventional command-and-control has the lowest average interaction delay amongst the speech-controlled GUI approaches which have been considered by this work. We now compare the average interaction delay of conventional command-and-control to conversation-and-control, referring to Figure 6.7 on the next page. The Figure graphically depicts the development of the average interaction delay of both approaches against growing word error rates (derived from Tables 4.5 and 6.2). We will first discuss the graphs' behavior for the average nominal interaction delay (0% word error rate) and for low word error rates ( $< 15\%$ ), and then examine their behavior for higher word error rates ( $\geq 15\%$ )<sup>10</sup>.

( $CMDACT_{UNCHECK} \oplus CMDNAV_{id}$ ) or ( $CMDACT_{OPEN} \oplus CMDNAV_{id}$ ). While this is merely to increase its usability, for our calculations we only examine the smallest set of valid commands by which the basic executions can be successfully performed.

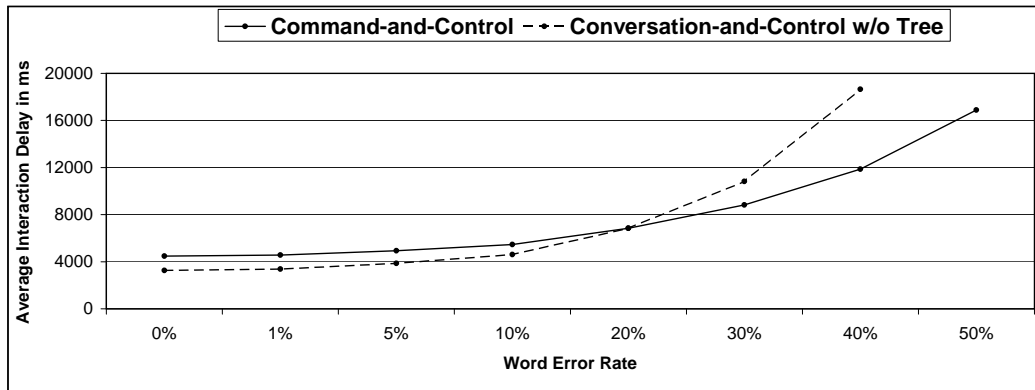
<sup>10</sup>We have omitted drawing conversation-and-control for a 50% word error rate to keep the scale of the Y axis reasonable small.



**Figure 6.7:** Calculated interaction delay graphs of conventional command-and-control and conversation-and-control.

The average nominal interaction delay of conversation-and-control is significantly, i.e., by 25% lower than the average nominal interaction delay of conventional command-and-control. The nominal interaction delay does not consider corrections, which rules out differences in the vocabularies or the valid commands to be responsible for this observation. Instead, the reason is the quality of the interactions by which the basic executions can be performed, which we explain as follows. From Table B.1 (appendix B) we derive that the average number of commands per basic execution with conventional command-and-control is 1.75 and the average command length of this approach is 1.86. From Table 6.1 we derive that conversation-and-control only requires 1 command per basic execution, which leads to a reduction of the interaction delay according to the command count-monotony (Theorem 5). In fact, the average number of commands per basic execution could thus be reduced by 43%. The average command length of conversation-and-control increased, compared to conventional command-and-control, by 75%, namely to 3.25. This has, according to the command length-monotony (Theorem 4), a tendency of compensating the interaction delay reduction from the reduced number of commands. However, this situation was anticipated: qualified activation commands result from merging multiple conventional command-and-control commands, therefore it is not surprising that their lengths grow. The fact that, in total, the nominal interaction delay of conversation-and-control is lower than conventional command-and-control is due to the recognition delay being about 2.6 times higher than the duration of a word. Thus, the interaction delay reduction from minimizing the average number of commands is not compensated by the increase in the average command length. If the recognition delay were 0, the nominal interaction delay of both approaches would be equal, as the products of average command number and command lengths of both approaches are alike <sup>11</sup>. For this case, which is of hypothetical nature from the perspective of the current state-of-the-art in speech recognition, we argue, that users prefer conversation-and-control over conventional command-and-control, as

<sup>11</sup> $1.75 \cdot 1.86 = 3.225 \approx 1 \cdot 3.25$



**Figure 6.8:** Calculated interaction delay graphs without considering the tree for conversation-and-control.

the qualitative feedback from our experiment suggests (discussed later in section 6.3.6).

We now discuss Figure 6.7 for growing word error rates. The Figure shows, that for word error rates  $<15\%$  the average interaction delay of conversation-and-control remains below the average interaction delay of conventional command-and-control. Their absolute difference, however, decreases. In other words, with growing word error rate, the interaction delay of conversation-and-control increases faster than with conventional command-and-control. The reason for this observation will become clear in the next paragraph.

Between the word error rate of  $10\%$  and  $20\%$  we observe an intersection of the two graphs at roughly  $15\%$ . For word error rates above this intersection we observe that the interaction delay of conversation-and-control is significantly higher than conventional command-and-control. Furthermore, we observe that it grows significantly faster than conventional command-and-control. We explain this behavior as follows<sup>12</sup>. Conventional command-and-control, in average, requires 1.75 commands of an average length of 1.86, whereas conversation-and-control requires fewer commands in average (1) with a higher average length (3.25). Due to their average length, the conversation-and-control commands are more likely to be mis-recognized if the word error rate increases, because the command recognition rate decreases with increasing command length. Thus, with conversation-and-control we expect more corrections than with command-and-control for high word error rates. Therefore, for high word error rates, conversation-and-control has the higher interaction delay. For low word error rates, the effect, which conversation-and-control achieves by sparing 1 or 2 times the recognition delay, prevails: in average, it does not matter if a long command is sometimes mis-recognized, as the majority of the long commands is correctly recognized. Thus, for low word error rates, the interaction delay of conversation-and-control is lower than conventional command-and-control.

We regard the execution  $E_{tree-deselect}$  as being mainly responsible for the early intersection

<sup>12</sup>In fact, this behavior is due to analog reasons as observed with grid-based mouse emulation and direction-based mouse emulation with discrete movement, as discussed in section 4.3.6

point between the two graphs of Figure 6.7. The reason is that the interaction for  $E_{tree-deselect}$  inherently has an interaction delay, which is nominally by 33% higher than the average of the others. This affects the total average interaction delay of conversation-and-control. Hypothetically, if we would not consider the tree for conversation-and-control, the graphs would look like as depicted in Figure 6.8 – the intersection point moved to the right (now at 20% instead of 15%) and the increase of conversation-and-control’s interaction delay is not as drastic as before. We conclude, that the tree might be a graphical object which is not suitable to be controlled by conversation-and-control. We do, however, not provide further evidence for this and regard it as future work.

It is important to note that the fast growing of conversation-and-control’s interaction delay for high word error rates is, most likely, not as drastic as it is depicted in the Figure 6.7. This reason is, that the interaction delay model assumes, that the user’s reaction on a rejection is a repetition. Conversation-and-control tries to avoid rejecting and repeating commands by qualified feedback. Thus, it is likely, that the command that follows a mis-recognized command is in fact shorter than the actual mis-recognized command. This effect is, however, not accounted for by the interaction delay model, as this effect is difficult to quantify – the quality of a recognition error is a priori unknown.

The executions  $E_{tree-select}$  and  $E_{input}$  have not been considered in the discussion, as they contain a priori unknown variables (as discussed in section 4.1.3  $E_{spinner}$  is a special case of  $E_{input}$ , as a spinner is an input field limited to numeric characters). We will, however, show in the following, that conversation-and-control also suggests a lower interaction delay for these executions on a theoretical basis. Representatively, we calculate the reduction  $\Delta$  of the average nominal interaction delay of  $E_{tree-select}$  and  $E_{input}$  (the respective values are taken from Tables B.15 and B.2).

$$\Delta_{E_{input}} = n_{text} \cdot 2642 + 2642 - ((n_{text} - 1) \cdot 2642 + 3784) = 1500$$

$$\Delta_{E_{tree-select}} = (n_{depth} + 1) \cdot 2642 + 2071 - (n_{depth} \cdot 2642 + 3213) = 1500$$

Thus, conversation-and-control reduces the average nominal interaction delay of  $E_{tree-select}$  and  $E_{input}$  by 1500ms ( $E_{spinner}$ , respectively) .

## 6.3 User Experiment

We now describe an experiment with a conversation-and-control interface by which we measured the task completion time against conventional command-and-control. We begin with formally setting up the research hypothesis for the experiment (section 6.3.1) and discussing the utilized test subjects (section 6.3.2). Then, in sections 6.3.3 and 6.3.4, we describe the details of the actual experiment, of which the results are presented in section 6.3.5 and discussed in section 6.3.6.

### 6.3.1 Research Hypothesis

We hypothesize that the interaction delay reduction involved with conversation-and-control significantly reduces the task completion time compared to conventional command-and-control. Let  $\mu_{cmd}$  and  $\mu_{conv}$  denote the measured average task completion time using conventional command-and-control and conversation-and-control. Thus, we formulated the null hypothesis ( $H_0$ ) and the research hypothesis ( $H_1$ ) as follows.

$$H_0 : \mu_{cmd} = \mu_{conv}$$

$$H_1 : \mu_{cmd} > \mu_{conv}$$

We designed an experiment, where test subjects complete a specific set of tasks with both conventional command-and-control and conversation-and-control. The acceptance or rejection of  $H_0$  can be then be inferred using statistical inference methods on the measured average task completion times as follows.

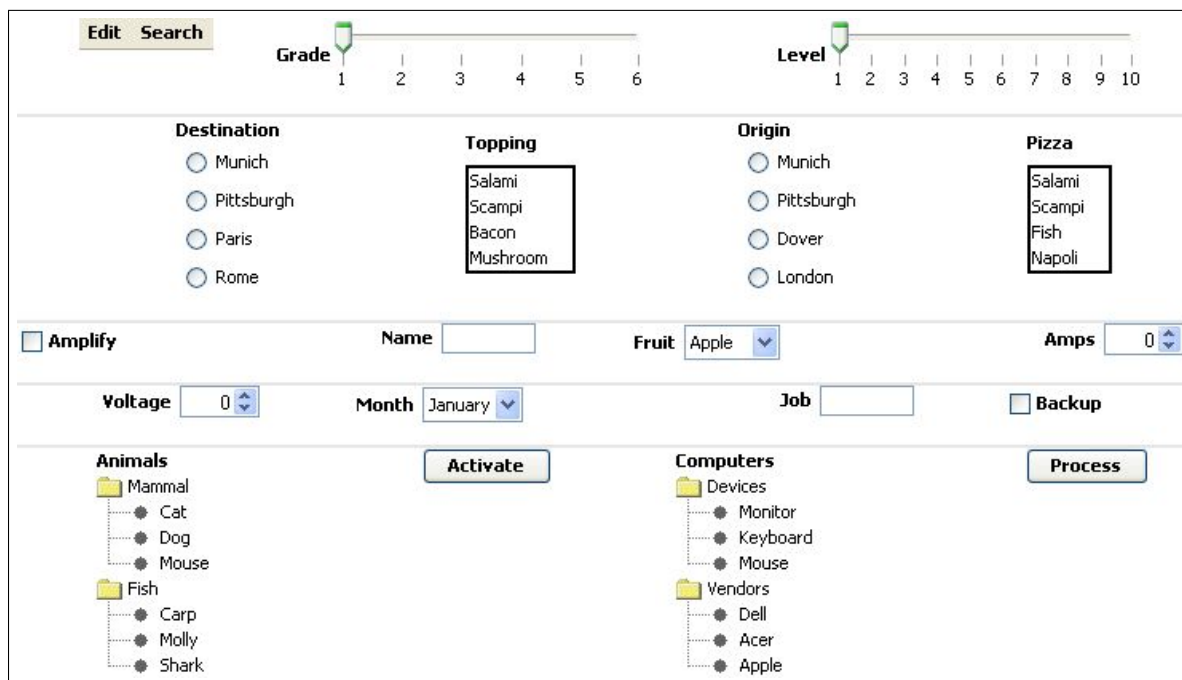
The *t-Test* (Bortz [24]) is a statistical significance test which allows to infer, if the average values of two normally distributed groups of measurement values differ significantly, respectively, if the average value of one group is significantly greater or lower than the average value of the other group. In our case, the two groups are determined by the two speech-controlled GUI approaches. A t-Test is therefore suitable to determine the acceptance or rejection of  $H_0$ , provided that we can assume normal distribution of the task completion times within each group. In fact, we may assume a normal distribution, since the experiment tasks are fixed; we will, however, perform both a *Kolmogorov-Smirnov-Test with Lilliefors correction* and a *Shapiro-Wilk-Test* (Shapiro and Wilk [169]) on our data – statistical tests which allow to infer if empirical results are sufficiently normally distributed.

T-Tests usually assume that the variance of the groups are equal, however, due to individual capabilities of test subjects we expect differences in the variances of the groups. We will therefore use the *paired t-Test*, a t-Test variant, which is robust to these differences by considering dependent pairs of measurement values across the groups<sup>13</sup>. In our case, the dependency between the two groups is given by the fact that each test subject "generates" one value in each group, since each test subject performs the task with both speech-controlled GUI approaches. In other words, the paired t-Test determines, if on the average the difference per test subject is significantly large enough. If it is,  $H_0$  can be rejected.

### 6.3.2 Test Subjects

We used a total of 16 test subjects for our experiment; 5 of these were female and 11 were male. All of the test subjects were between the age of 20 and 30 and had different professions: physician (6), physics student (2), insurance agent (1), software engineer (1), computer scientist (2), architect (1), industrial engineer (1), electrical engineer (1) and business economist

<sup>13</sup>The paired t-Test is therefore also called *dependent t-Test*.



**Figure 6.9:** Conversation-and-control interface from the experiment.

(1). All of them had significant experience with controlling GUIs by mouse and keyboard. All test subjects were German native speakers, however, all test subjects had significant experience with speaking English due to their profession or education. This was important for the experiment, as we used an English language speech recognizer (as described in section 6.3.4). The computer scientists and two physicians had experience in using applications that involved speech recognition technology (e.g., a dictation program for radiologic reports), but none of the test subjects had experience with speech-controlled GUIs.

### 6.3.3 Experiment GUI

Using the Swing implementation of the conversation-and-control framework (section 6.1) we created the *experiment GUI*, depicted in Figure 6.9. Derived from the experiment GUI we defined a sequence of 32 tasks and each task involved a single basic execution. The experiment GUI provided two instances of each graphical object<sup>14</sup>, and thus, the task sequence contained at least two instances of each basic execution. For  $E_{dropdown}$  four tasks existed: for two tasks we specified the option that should be selected, and for the other two tasks, we just told the test subjects to select an option other than the current one. By this setup we wanted to test, if the test subjects could exploit recalling the options in the drop down box, so that, with conversation-and-control, there would be no need to open the drop down box. Ta-

<sup>14</sup>Since a standalone radio button does not provide any useful possibility for interaction – it can never be deselected – we have used a radio button list object.

ble B.16 on page 210, appendix B.1, documents the task sequence in detail. Due to the mode switching feature of the conversation-and-control framework (described in section 6.1.1), the tasks could be performed using either conventional command-and-control (command mode) or conversation-and-control (conversation mode).

The experiment GUI, the underlying application, respectively, did not provide any domain specific functions. Thus, a misunderstanding did not have a consequence on the domain level, which is why we omitted the explicit implementation of an undo function (cp. *SFUNDO* as introduced in 3.2.2). Instead, for undoing activations, we relied on the fact that most activations essentially have a complementary activation, which can be considered to be the undo function if no application logic is present, e.g., *SELECT* versus *DESELECT*, or *APPEND<sub>x</sub>* versus *DELETE*. However, this does not imply that misunderstandings have no negative effect on the task completion time: just like if application logic were present, the state that a misunderstanding could bring the experiment GUI into, affected the completion of tasks. For instance, consider a situation where a misunderstanding causes the appending of a character to an input field, although a button should have to be activated. The appending might not have an impact on the current task, just like a misunderstanding might not immediately affect the work with a real application. But if a subsequent task involves this input field, the test subject must take care of the mis-appended character to be removed – otherwise, the completion criteria of the input field task, which might require that the input field contains a specific value, would not be fulfilled. Additionally, we designed several sources of potential information errors, so that the testing of automated error recovery and qualified feedback would not depend on the occurrence of recognition errors alone. For instance, two of the four selection items from the "Origin" and the "Destination" radio button list overlap.

The experiment GUI does not use numbers as speakable identifiers, as the conversation-and-control framework lacks consistent support for these identifiers. It does provide numbered speakable identifiers only for specific graphical objects, e.g., for the options of a drop down box (see later in section 6.1.5). Instead, it uses speakable names as identifiers<sup>15</sup>. Therefore, for reasons of consistency, we completely relied on speakable names for our experiment. This does not contradict the assumption of numeric identifiers used for the interaction delay calculations, for the following two reasons. First, the utilized names do not exceed a length of 2, which corresponds to the assumed maximum length of a numeric speakable identifier. Second, the conversation-and-control framework configures the speech recognizer with the maximally available vocabulary, regardless of the mode. In other words it does not exploit the fact that vocabulary and valid commands can be minimized specific to the current screen (state) in order to increase speech recognition performance. While this might in general be a drawback for a productive application, it was harmless for our experiment, as the increased size of the vocabulary (now 107 versus 70 assumed) affected the word error rate for both command mode and conversation mode. Thus, the therefore increased interaction delay equally affects both modes. This behavior of the framework also simplified the statistical evaluation

---

<sup>15</sup>In Figure 6.9 the names of graphical objects are printed in bold, whereas the names of graphical sub-objects, such as the names of selection items or tree nodes, are printed in normal font.

of the collected data: the speech recognizer was equally configured for both modes, therefore, our results were independent of it. Finally, there is empirical evidence that test subjects prefer named identifiers over numeric ones (Christian et al. [37]). We do, however, encourage as future work to augment the conversation-and-control framework with screen-specific speech recognizer configuration capabilities.

### 6.3.4 Procedures

The order in which the sequence of tasks should be completed was predefined and constant throughout the experiment; by a *test run* we consequently understand the performing (completion) of the 32 tasks in their predefined order. Each task consisted of a short textual instruction, which the experiment GUI displayed to the test subject at the beginning of each task. We developed a model for the completion criteria of a specific task, so that our experiment system could automatically detect the completion of a task and switch to the next task in the sequence. This facilitated the automatic recording of the task completion time of each test run.

We divided the test subjects randomly into 2 groups of 8 members each, called group *A* and *B*. Members of group *A* first performed test runs in command mode and then in conversation mode, whereas members of group *B* first performed test runs in conversation mode and then in command mode. As such, for taking the overall average, we compensated for order effects: the test subjects become more familiar with speech-controlling the experiment GUI with increasing experiment duration independently of the respective mode. With our setup we do not favor a specific mode by putting it second. Additionally, this setup allows us to detect order effects and interpret them accordingly.

The actual experiment session was divided into three parts. In the first part the test subject was given a thorough introduction into speech-controlled user interfaces. In the second part the test subject trained an individual speech profile of the Microsoft Speech SDK (MS SDK), version 5.1, which the available Swing implementation of the conversation-and-control framework was based on. In particular, each test subject used the "Microsoft English Recognizer v5.1, SAPI5, Microsoft" recognition engine of MS SDK, which is a speech recognizer for the English language. Finally, in the third part of the experiment each test subject performed two test runs with each mode, i.e., a total of four test runs per test subject was performed. The first run was considered training in order to make the subject familiar with the experiment GUI and the respective first mode as determined by the group membership. Prior to starting the training test run, the test subjects were, again, given a thorough introduction into the characteristics of the respective mode. The second test run was performed in the same mode as the first test run, however, test subjects were told that now the system would record a protocol. The third and fourth test run were conducted in the same fashion using the remaining mode.

All test subjects used the Logitech Internet Chat Headset for both training the speech recognizer and performing the experiment. The microphone of the headset had a specified frequency range of 100 – 10,000Hz and a specified sensitivity of -59dBV/uber, -39dBV/Pa +/-4dB. Both the MS SDK and the experiment GUI were installed on a Dell Dimension 8400 workstation with 1GB of RAM and with a Pentium IV processor running at a speed of 3.19 GHz. The



|              | <b>Command Mode</b> | <b>Conversation Mode</b> | <b>Reduction</b> |
|--------------|---------------------|--------------------------|------------------|
| Avg. group A | 257 (41.0)          | 203 (45.0)               | 21.0%            |
| Avg. group B | 255 (48.9)          | 223 (52.3)               | 13.0%            |
| Avg. Total   | 256 (44.0)          | 213 (48.2)               | 16.8%            |

**Table 6.3:** Average task completion times in seconds with standard deviation in parenthesis.

operating system was Windows XP Professional, Service Pack 2.

### 6.3.5 Results

Prior to performing the paired t-Test on the experimental results we need to test if our data can assumed to be normally distributed. Using SPSS 14.0 we applied the Kolmogorov-Smirnov-Test with Lilliefors correction and the Shapiro-Wilk-Test to our data. The Kolmogorov-Smirnov-Test produced a low boundary for the significance of 0.2; the Shapiro-Wilk-Test resulted in a significance of 0.295 for the command mode data and in 0.124 for the conversation mode data. This allowed us to assume that our measurement values are significantly normally distributed<sup>16</sup>. The command mode test series had a variance of 1902 and the conversation mode test series of 2331. Thus, as expected, the variances are not equal which supports the usage of the paired t-Test.

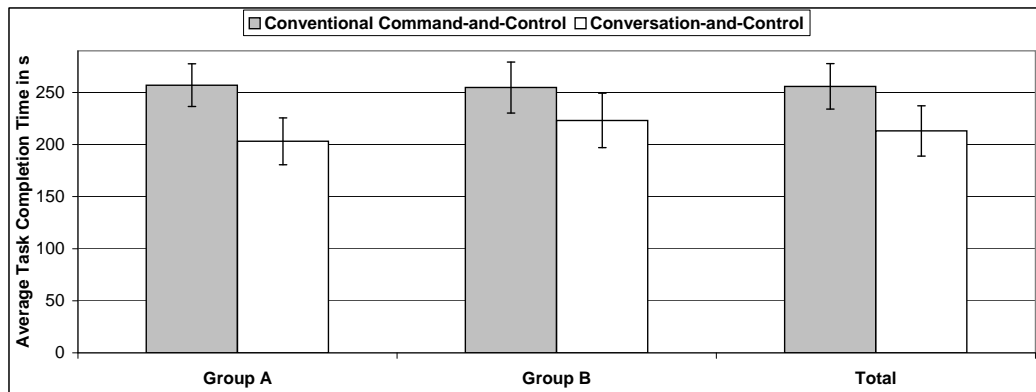
The *type 1 error probability* is the probability of rejecting  $H_0$  when it is actually true. It can be determined from the respective test data. The  $\alpha$  value is a parameter for the t-Test and denotes the highest allowed type 1 error probability for rejecting  $H_0$ . Thus, the  $\alpha$  value is also referred to as the *confidence level* for the t-Test. Per global statistical convention we set  $\alpha = 0.05$ . The *df* value is another parameter for the t-Test and denotes the *degrees of freedom* which, in our case, is the number of test subjects minus 1, i.e.,  $df = 15$ . Both  $\alpha$  and *df* determine the so called *cut-off value*. If the result of the t-Test is above the cut-off value, we can reject  $H_0$  with the respective confidence level. With our values for  $\alpha$  and *df* we obtain<sup>17</sup> a cut-off value for the paired t-Test of 1.75. We defined the following decision rule for the acceptance or rejection of  $H_0$  based on the t-Test result  $t$ :

$$rule(t) := \begin{cases} \text{reject } H_0, \text{ thus, accept } H_1, & \text{if } t \geq 1.75 \\ \text{accept } H_0, \text{ thus, reject } H_1, & \text{if } t < 1.75 \end{cases}$$

Using Microsoft Excel, Version 11.8012.6568 (SP2), we performed a paired t-Test on our data. The result was 7.74, thus, according to the above rule we reject  $H_0$  and accept  $H_1$ . In other words, our results indicate that conversation-and-control reduces the task completion time as compared to conventional command-and-control. This result is statistically significant at a confidence level of  $\alpha = 0.05, p = 6.5 \cdot 10^{-7} < \alpha$ .

<sup>16</sup>The default configuration of SPSS denotes that test results  $> 0.05$  indicate a sufficient adherence to a normal distribution.

<sup>17</sup>Statistics applications deliver cut-off values automatically.



**Figure 6.10:** Average task completion times, graphically.

We summarize the outcome of the experiment by depicting the mean task completion times for each mode within each group and in total in Table 6.3 on the preceding page. The numbers in parenthesis denote the respective standard deviation. Figure 6.10 illustrates the content of Table 6.3 graphically. Conversation-and-control achieves an overall task completion time reduction of 16.8% compared to conventional command-and-control. Group A achieved a reduction of 21% in average, whereas group B achieved a reduction of only 13% in average.

### 6.3.6 Discussion

The outcome of the experiment supports the claim of this dissertation: conversation-and-control reduces the achievable task completion time with speech-controlled GUIs compared to existing art. In the following we will discuss the results of our experiment in more detail.

All of the test subjects could successfully complete all tasks of each test run. However, as the standard deviation figures in Table 6.3 show, the individual results do indeed vary. We discuss three likely reasons for this. First, we observed that during training the MS SDK's speech recognition engine, some test subjects were "understood better" than others. This manifested in some test subjects having to speak more slowly and possibly with slight pauses between the words, than others. This experience of whether the test subject could speak fast or slow was carried over to working with the sample GUI: test subjects also spoke slowly or fast during the experiment, which could be responsible for the task completion time variations. Second, without having empirical evidence, though, we believe that there is a difference between the cognitive processes amongst the test subjects. For instance, we observed that even within the third and fourth test run, some test subjects were sometimes searching for the graphical object that was involved in the task, whereas other test subjects could remember the position of the graphical objects instantaneously. Thus, some test subjects made a significant pause between reading the instructions, while others did not. This certainly contributed to some variance in the individual results. Third, recognition errors and the therein involved corrections and

answers to clarification questions lengthened individual test runs. Obviously, a test run with no or a low amount of recognition lasts shorter than a test run with a high number of recognition errors, and the concrete amount of recognition errors depends, amongst other factors, on the voice of the respective test subject.

Clarification questions were helpful if they were short, i.e., if they provided only two alternatives to select from, e.g., a clarification between two graphical object names, or specifying a missing parameter value. Whenever the clarification question was long, i.e., more than two alternatives, it confused the test subject instead of providing further guidance, and it took a considerably long amount of time to understand their meaning. There are several explanations. First, we observed that clarification questions frequently did not provide an alternative which corresponded with the test subject's intention. In these cases the test subject was confused, since the clarification seemed to have "nothing to do" with what was uttered, and the test subject fell back repeating the original command. This phenomenon is probably due to the fact that the conversation-and-control dialog model does not consider confidence values<sup>18</sup> of recognition results. As such, it could happen that the system asked a clarification question based on a recognition result of which even the speech recognizer "thinks that it is wrong". Consequently, the clarification question did not make sense in the context of the test subject's intention. Second, we believe that test subjects had difficulties with long clarification questions because there was no visual connection between the alternatives provided and the graphical objects which were ambiguous or which were requiring more information. Test subjects spent time on thinking about the meaning of the question and mapping it to the state of the GUI. They were not always clear about why the system asked this particular question. We believe that a visual indication of the graphical objects that the question affects and a visual model of the current state of the dialog model could be beneficial (see section 7.3).

Several test subjects found it "cumbersome" to interact in command mode, in particular, they complained about the necessity to explicitly navigate to a graphical object. In contrast, they found it more natural to interact in conversation mode, especially, if they had already knowledge about the characteristics of the GUI. The latter effect can also be derived from our result data, because there is an order effect between the test groups. The test subjects in group A began with command mode. Their task completion time was reduced by 21.0%. The test subjects in group B began with command mode, achieving a task completion time reduction of only 13.0%. We interpret this order effect as follows. We may assume, that after having completed the respective first half of the experiment, the test subjects had reached a certain level of familiarity with the experiment GUI. Thus, group A could exploit this familiarity and transition it into more task completion time reduction, whereas group B was forced to perform every thing "step by step", as several of our test subjects mentioned. As such, the test subjects in group A could benefit from recalling details about the properties of graphical objects when they were interacting in command mode. Consider the drop down box as example. As stated in section 6.3.3 we have included a total of 4 instances of the  $E_{dropdown}$  execution. As depicted in

---

<sup>18</sup>Confidence values indicate "how confident" a speech recognizer is about the correctness of a recognition result; they were discussed in section 2.8.2.

|                          |              | Interaction delay for experiment<br>in s for word error rates of |     |     |     |     |
|--------------------------|--------------|--|-----|-----|-----|-----|
|                          |              | 0%   | 1%  | 5%  | 10% | 20% |
| Conventional<br>Control  | Command-and- | 175  | 178 | 193 | 214 | 268 |
| Conversation-and-Control |              | 133  | 137 | 158 | 190 | 288 |

**Table 6.4:** Predicted interaction delay for completing experiment.

Table B.16, 2 of these 4 tasks explicitly stated the selection item to select (specified selection), whereas the other 2 instructed the test subject to select an arbitrary item other than the currently selected item (unspecified selection). Obviously, with a specified selection, the test subjects could utter both the speakable identifier of the drop down box and the selection item, as these were merely given in the instruction. However, test subjects also did this in the unspecified case – which means that they recall the options from prior test runs or tasks. Conversation-and-control allows them to exploit this knowledge, whereas with conventional command-and-control test subjects have to open the drop down box, regardless of whether they know the options.

However, the order effect can also be interpreted as such that conversation-and-control requires more training in order to be more effective than conventional command-and-control. From the fact that group *B* required more time for the conversation mode test runs than group *A* (see Table 6.3), we conclude that conversation-and-control might initially impose more cognitive load in mentally constructing the commands, whereas conventional command-and-control, due to being more explicit, imposes less initial cognitive load. However, as our results show, the increase in cognitive load is marginal, as otherwise group *B* would not have had a 13.0% increase in task completion time or the difference would have been negative. Nevertheless, the observation of the order effect suggests a hybrid dialog model distinguishing between novice and expert mode. In novice mode, users interact with command-and-control, in order to spent less cognitive effort on learning the characteristics of the GUI than on controlling it. Once users become familiar with the GUI, they can switch to expert mode and control the GUI using conversation-and-control.

We found it particularly interesting to compare the measured average task completion times with the expected interaction delays predicted by the interaction delay model. For this we first calculated the predicted interaction delay for our 32 tasks for different word error rates up to 20% – above 20% conversation-and-control has a higher interaction delay than conventional command-and-control. The calculations are based on the previously obtained basic execution-specific interaction delays for conventional command-and-control (Table B.2) and for conversation-and-control (Table B.15). The result of these calculations are given in Table 6.4. Then, we estimated the average word error rate of the speech recognizer from the data gathered by our experiment; it was approximately 11% and the predictions for a 10% word error rate are consequently the closest ones. Thus, our model predicts an interaction delay of

214 seconds for conventional command-and-control. The measured average task completion time with this approach was 256 seconds, which makes a difference of 42 seconds (16% from measured value). The difference between measurement and prediction is expected: the interaction delay model only describes the time spent on the plain interaction. The time difference emerges from the remaining temporal constituents of task completion time (see decomposition in section 1.3.1), such as the time required to read and understand the task's instructions and to mentally construct the command. Looking at conversation-and-control our model predicts an interaction delay of 190 seconds for a word error rate of 10%. The measured average task completion time with this approach was 213 seconds, which makes a difference of 23 seconds (11% from measured value). Again, the difference is spent on other temporal components of task completion time. We interpret the results of this comparison in two ways. First, the interaction delay model is well-defined – the difference between measurement and prediction must be a positive value, since the interaction delay model only describes the time spent on the plain interaction. Second, the interaction delay model correctly predicts the reduction of the task completion time: if we have a significantly lower interaction delay for a specific approach compared to another approach, we expect tasks to be completed faster with the former approach. We argue, however, that it is not feasible to draw any further conclusions. For such conclusions to be feasible the interaction delay model must be calibrated according to the experiment situation. In particular, the actual average word duration of test subjects and the actual average recognition delay must be measured and used as corresponding parameters in the interaction delay model – our current parameters are derived from literature research and do not necessarily match our experiment setup. We consider further experiments in this direction as future work in the context of improving the calibration of our models (refer to section 7.3). Thus, once the interaction delay model is calibrated to a specific experiment setup, the difference between measured task completion time and predicted interaction delay can be used to draw further conclusions, e.g., towards a model for the cognitive load imposed by a specific speech-controlled GUI approach.

To conclude our discussion, we informally provide further interesting qualitative results. Two test subjects noted, that it would be helpful for an input field or spinner to have a *CLEAR*-topic by which the entire content could be cleared out. Especially upon misunderstandings which wrongly append more than one character, this could help minimizing the impact to the task completion time. We found that although none of the test subjects was a native English speaker, the speech recognizer performed well during the experiment. We argue that this was due to the test subjects having significant experience in English conversations due to their professions and education. The good performance of the speech recognizer during the experiment was, however, not anticipated by the experience during its training, where the amount of recognition errors was significantly higher than during the experiment. We explain this by the training session involving a much larger vocabulary than our experiment. Furthermore, the MS SDK speech recognizer was fairly robust to environmental noise, which could not entirely be ruled out. For instance, during one test run we had church bells in the background which did not noticeable affect the speech recognizer's performance.

## 6.4 Summary

We have specified a framework for conversation-and-control interfaces and implemented the framework in J2EE. Using the framework we have created prototype conversation-and-control interfaces for applications from the technical maintenance domain and thus show, that conversation-and-control interfaces are feasible. The claim of this thesis is supported by both a formal and an empirical validation.

Formally, we have calculated the interaction delay of conversation-and-control by deriving a generic vocabulary and valid commands under the conditions established in chapter 4. The results of these calculations showed, that the interaction delay of conversation-and-control is significantly lower than the interaction delay of conventional command-and-control – at least for word error rates below 15% (20% without considering the tree object). The reason for the decrease is the quality of the qualified activation commands: they allow to reduce the average number of required commands for a basic execution by 43%, which spares multiple times the recognition delay. This effect is not compensated by qualified activation commands being longer than conventional command-and-control commands. In the following section we provide empirical evidence for the core claim of this dissertation, stating that conversation-and-control reduces the task completion time as compared to conventional command-and-control.

Empirically, we have performed a user experiment to support the hypothesis that conversation-and-control significantly reduces the task completion time compared to conventional command-and-control. Using a specific GUI, created with the Swing implementation of the conversation-and-control framework, we collected data from which we could statistically infer the acceptance of this hypothesis. In particular, we used a total of 16 test subjects and showed that conversation-and-control is capable of reducing the task completion time by 21%. This result is statistically backed by a dependent t-Test which is significant at  $\alpha = 0.05$ . From qualitative user feedback during the experiment we conclude that the test subjects preferred conversation-and-control over command-and-control, which supports our statistically inferred result. However, the experiment also indicated, that conversation-and-control has limitations, which need to be dealt with in future research projects, such as improving the generation of clarification questions.

# 7

## Conclusion

### Overview

This chapter concludes the dissertation by reflecting on our work on conversation-and-control. We further summarize the contributions which this thesis offers to the research community, and point out directions for future research.

## 7.1 Thesis Summary

Speech-controlled GUIs allow manually impaired users to control GUIs by spoken commands versus the utilization of mouse and keyboard. The approach *command-and-control* currently promises the lowest achievable task completion times, however, experimental results show (Dai et al. [44], Van Buskirk and LaLomia [32], Christian et al. [37], Arnold et al. [7]), that the achievable task completion times with speech-controlled GUIs, including command-and-control, are still at least 50% higher than with mouse and keyboard.

In order to decrease the achievable task completion times we proposed the approach *conversation-and-control*, which reduces the *interaction delay*, that is, the time spent on uttering and recognizing spoken commands. Conversation-and-control is an extension of command-and-control by the techniques *qualified activation*, *automatic information error recovery* and *qualified feedback*. Qualified activation achieves an interaction delay reduction by identifying sets of  $n$  formerly separate commands, which it merges into single commands. Now, the user needs to utter a single command instead of  $n$  commands; thus, qualified activation reduces the interaction delay by  $n - 1$  times the time that a speech recognizer requires for recognizing a command. Automatic information error recovery defines a set of heuristics and procedures, by which the system can automatically recover from specific recognition errors, which lead to missing or ambiguous information. These recognition errors are called *information errors* and by attempting to automatically resolve them, we avoid the rejection of mis-recognized commands. This technique is motivated by the observation that, although mis-recognized, the recognition error might contain information from which the user's intention can be inferred by consulting other sources of knowledge, e.g., the command history. Thus, automatic information error recovery avoids the repetition of commands, which in turn, reduces the interaction delay. Qualified feedback is a procedure by which we actively involve the user in clarifying information errors, with the objective to induce a user's response, which is shorter than the original command. Thus, qualified feedback reduces the interaction delay as well – instead of repeating the entire command, qualified feedback just requires uttering a fraction of the original command.

For verifying and quantifying the interaction delay reduction of conversation-and-control we first developed the *speech-GUI* model, a new generic model for speech-controlled GUIs. Then, from quantitative and qualitative properties of the speech-GUI model we derived the *interaction delay model*, a generic formalism for calculating the interaction delay of a specific speech-controlled GUI approach. The formalism calculates the interaction delay as a function against the average word error rate of the utilized speech recognizer, the size of the speech recognizer's vocabulary, the number of valid commands in the respective speech-controlled GUI approach and specific characteristics of the task that is to be performed. It distinguishes between the *nominal interaction delay* and the *expected interaction delay*. The nominal interaction delay does not consider the effects of recognition errors, i.e., it assumes a word error rate of 0%. The expected interaction delay accounts for the effects of recognition errors and subsequent corrections for a specific average word error rate.

We calculated a reduction of the nominal interaction delay of conversation-and-control



compared to command-and-control of 25%, which is due to a reduction of the number of necessary commands by 43%. For word error rates  $< 20\%$ , the expected interaction delay of conversation-and-control is lower than of command-and-control, whereas for word error rates  $\geq 20\%$  the expected interaction delay of conversation-and-control is higher. In order to validate the claim that conversation-and-control also reduces the task completion time we conducted an empirical experiment, in which users performed a series of tasks using both command-and-control and conversation-and-control. By capturing the respective task completion times we found, that conversation-and-control, involving a real error-prone speech recognizer, is capable of reducing the task completion time by 21%.

## 7.2 Contributions

With this thesis we offer contributions to the area of speech-controlled GUI research, which we summarize in the following.

The thesis introduces *conversation-and-control*, which is a novel speech-controlled GUI approach, emerging from an extension of prior art. The novelty of conversation-and-control is threefold. First, the user needs fewer commands to control the GUI than the current state-of-the-art, which results in lower task completion times. Second, our approach allows the user to speak more naturally. Third, by asking clarification questions to resolve error conditions, our approach engages the user into a natural dialog. For this, we developed a specialized *dialog model for conversation-and-control*, which is based on heuristic evaluation of the semantic information provided by a recognition result. The dialog model is application-blind; it can be transitioned to different application domains and to different GUI toolkits.

Conversation-and-control is based on two new models in the domain of speech-controlled GUIs, which build upon each other: the *speech-GUI model* and the *interaction delay model*. The speech-GUI model is a common model for describing different speech-controlled GUI approaches on a common basis. It is extensible to new speech-controlled GUI approaches, such as conversation-and-control. The interaction delay model is derived from quantitative and qualitative characteristics of the speech-GUI model and predicts the time which a user spends on the plain interaction with a speech-controlled GUI; that is, the time which is needed for speaking commands and for the utilized speech recognizer to recognize these commands. The interaction delay model considers the word error rate of the speech recognizer and estimates the additional time delay that is introduced to correct the effects of incorrectly recognized commands. Both speech-GUI model and interaction delay model *extend the GOMS model* for analyzing interaction towards speech recognition in two regards. First, by introducing an abstraction layer between GOMS operators and goals, the speech-GUI model covers alternative spoken commands for the same functionality. Second, the interaction delay model predicts the duration for successfully performing a GOMS low-level goal under consideration of the inherent limitations of speech recognition technology.

We have implemented the underlying theory of conversation-and-control as a *framework for conversation-and-control interfaces*, using state-of-the-art object-oriented software engineer-

ing techniques. The framework hides the details of speech recognition and language processing from the developer. This allows for the creation of conversation-and-control interfaces in the same manner as developers create conventional GUIs today, i.e., by assembling graphical components without dealing with low level input details. Using the framework we have created *prototype systems from the technical maintenance domain*, which show the practical feasibility of conversation-and-control.

We further validated conversation-and-control by the means of a *user experiment*, which was designed to compare task completion times with conversation-and-control against existing art. The experiment confirmed, with statistical significance, that conversation-and-control allows for a task completion time reduction of 21%. The outcome of our experiment suggests the usage of a hybrid dialog model for conversation-and-control interfaces, which distinguishes between a novice and an expert mode. In novice mode the user interacts according to the current state-of-the-art, i.e., the user utters multiple simple spoken commands to control the GUI. This allows to memorize the characteristics of the GUI and to understand the characteristics of speech-based GUI control in general. In expert mode the user interacts according to conversation-and-control. This approach allows exploiting the previously gained knowledge in a way so that the necessary information for controlling the GUI can be conveyed with less commands than in novice mode – which reduces the task completion time.

### 7.3 Future Work

This thesis is a first step into the domain of conversation-and-control. We show that this approach is feasible, however, its current limitations and the results from the user experiment point out further directions of research in this area.

Our user experiment showed that clarification questions were not always helpful. It happened, that the clarification question confused the test subject instead of providing further guidance – especially, if the clarification question provided more than two alternatives to select from. We motivate more research regarding a mechanism allowing for reducing the set of alternatives offered by clarification questions. An interesting piece of work towards this goal is Mankoff et al. [119], who developed a model for ambiguity in recognition-based systems. Their work could be adapted to the speech-controlled GUI domain and used to filter ambiguous conversation topics.

We developed a basic set of heuristics and procedures for resolving information errors, however, more research towards finding new such heuristics or towards improving the provided ones, is promising. For instance, the dialog model could "try out" different paths in identifying the user's intention, by considering the N-best recognition results which a speech recognizer possibly provides for the users utterance.

Currently, the user's input is not verified before a specific control action of a graphical object is triggered. Our experimental results indicate that this is not problematic as long as the speech recognizer has a low word error rate. However, for situations where the word error rate increases (e.g., due to environmental noise), research towards a suitable verification concept

is necessary. It would improve robustness by avoiding misunderstandings. Verification could for instance be configured to be necessary for critical and unrecoverable operations, such as the deletion of data. Furthermore, a confidence threshold could be established, below which a recognition result is rather rejected than being evaluated.

Test subjects did not always recognize that the system was asking a clarification question although it was printed in large font, bold, and red. We propose future research regarding the utilization of speech synthesis technology to present clarification questions aurally, i.e., to give them a more ubiquitous presence.

Qualitative feedback from our experiment indicates that at the time the system poses a clarification question, the test subjects felt to be "out of control". The reason was, that the test subjects first had to read the question and then infer as to why the system asked the question. We propose to more intensively make use of the graphical output channel, to make the meaning of clarification questions more clear. For example, the system could mark the graphical objects which are currently under clarification, so that users get a visual feedback as to which graphical objects are "affected" by the information error.

As stated in section 4.1.3 we have performed interaction delay calculations without further calibrations, i.e., with regarding the calibration values obtained from literature research as sufficient. We made suggestions regarding a more accurate calibration, but we also motivate research towards integrating further factors that would improve its preciseness. For instance, currently, the reduced length of corrections being answers to clarification questions is not considered.





# Appendix: Mathematics

## Overview

This appendix explains mathematical calculations of which the results have been used in this dissertation.

## A.1 Theoretical Limit of $P(Rej)$

In section 3.2.3 we have initially approximated the probability of a recognition error,  $P(Rej)$ , by the term

$$\frac{|\mathcal{CMD}| - |\mathcal{CMD}_{vld}|}{|\mathcal{CMD}|} \quad (\text{A.1})$$

where  $\mathcal{CMD}$  denotes the number of commands which can be built using a specific speech recognizer's lexicon, and  $\mathcal{CMD}_{vld}$  denotes the valid commands of a speech-controlled GUI approach that is built using this specific speech recognizer. The number of elements in  $\mathcal{CMD}$  is infinite, because, given an arbitrary finite set (such as the speech recognizer's lexicon) an infinite number of sequences of elements from that set can be constructed. The number of elements in  $\mathcal{CMD}_{vld}$  is finite (see section 2.4).

Per definition  $|\mathcal{CMD}| \geq 1$ ,  $|\mathcal{CMD}_{vld}| \geq 1$ , and  $|\mathcal{CMD}| \geq |\mathcal{CMD}_{vld}|$ . We have used the application Maple [120] to calculate the limit of (A.1) for  $|\mathcal{CMD}| \rightarrow \infty$  using the commands which are depicted in Figure A.1. As such,

```

> assume(CMD >= 1);
> assume(CMDVLD >= 1);
> assume(CMD >= CMDVLD);
> Limit((CMD - CMDVLD) / (CMD), CMD=infinity);
                                      $\lim_{\text{CMD} \rightarrow \infty} \left( \frac{\text{CMD} - \text{CMDVLD}}{\text{CMD}} \right)$ 
> limit((CMD - CMDVLD) / (CMD), CMD=infinity);
                                     1
    
```

**Figure A.1:** Maple commands to calculate the theoretical limit of  $P(\text{Rej})$ .

```

> assume(CNMAX > 0);
> assume(CVLD > 0);
> assume(R >= 0);
> assume(R <= 1);
> solve((1-R) * (2 - (CNMAX - CVLD) / (CNMAX)) >= 1, R);
                                      $\left\{ R \leq \frac{CVLD}{CNMAX + CVLD} \right\}$ 
>
    
```

**Figure A.2:** Maple commands to calculate the critical command recognition rate.

$$\lim_{|CMD| \rightarrow \infty} \frac{|CMD| - |CMD_{vld}|}{|CMD|} = 1$$

## A.2 Calculating the Critical Word Error Rate

In section 3.2.3 we have derived the inequality

$$(1 - \mathcal{R}_c) \cdot \left( 2 - \frac{|CMD_{2n_{max}}| - |CMD_{vld}|}{|CMD_{2n_{max}}|} \right) \geq 1 \quad (\text{A.2})$$

With the preconditions that  $|CMD_{vld}| > 0$ ,  $|CMD_{2n_{max}}| > 0$  and  $0 \leq \mathcal{R}_c \leq 1$  we have used the application Maple [120] to solve (A.2) using the commands which are depicted in Figure A.2. As such (A.2) has solutions if the following condition is satisfied:

$$\mathcal{R}_c \leq \frac{|CMD_{vld}|}{|CMD_{2n_{max}}| + |CMD_{vld}|}$$

```

> assume(CNMAX > 0);
> assume(CVLD > 0);
> assume(nmax >= 1);
> assume(w >= 0);
> assume(w <= 1);
> solve((1-w)^nmax > (CVLD)/(CNMAX + CVLD), w);

```

$$\left\{ w \sim < -e^{\left( \frac{\ln\left( \frac{CVLD\sim}{CNMAX\sim + CVLD\sim} \right)}{nmax\sim} \right) + 1} \right\}$$

**Figure A.3:** Maple commands to calculate a formula for the critical word error rate.

We have furthermore derived the inequality

$$w^{n_{max}} > \frac{|CM\mathcal{D}_{vld}|}{|CM\mathcal{D}_{2n_{max}}| + |CM\mathcal{D}_{vld}|} \tag{A.3}$$

With the preconditions that  $n_{max} \geq 1$ ,  $|CM\mathcal{D}_{vld}| > 0$ ,  $|CM\mathcal{D}_{2n_{max}}| > 0$  and  $0 \leq w \leq 1$  we have used the application Maple [120] to solve (A.3) using the commands which are depicted in Figure A.3. As such (A.3) has solutions if the following condition is satisfied:

$$\begin{aligned}
 w < 1 - e^{\frac{\ln\left(\frac{|CM\mathcal{D}_{vld}|}{|CM\mathcal{D}_{2n_{max}}| + |CM\mathcal{D}_{vld}|}\right)}{n_{max}}} &\Leftrightarrow \\
 w < 1 - \left( e^{\ln\left(\frac{|CM\mathcal{D}_{vld}|}{|CM\mathcal{D}_{2n_{max}}| + |CM\mathcal{D}_{vld}|}\right)} \right)^{\frac{1}{n_{max}}} &\Leftrightarrow \\
 w < 1 - \left( \frac{|CM\mathcal{D}_{vld}|}{|CM\mathcal{D}_{2n_{max}}| + |CM\mathcal{D}_{vld}|} \right)^{\frac{1}{n_{max}}}
 \end{aligned}$$

## A.3 Geometric Series starting at 1

A geometric series has the general form

$$\sum_{k=0}^n x^k \quad (x \in \mathbb{R}, n \in \mathbb{N})$$

For  $|x| < 1$  the following claim is satisfied (Bronstein et al. [28])

$$x \neq 1 \Rightarrow \sum_{k=0}^n x^k = \frac{x^{n+1} - 1}{x - 1}$$

$$x = 1 \Rightarrow \sum_{k=0}^n x^k = n + 1$$

We apply conversions to arrive at a convenient formula for  $\sum_{k=1}^n x^k$ .

$$\sum_{k=0}^n x^k = 1 + \sum_{k=1}^n x^k \Leftrightarrow$$

$$\sum_{k=1}^n x^k = \sum_{k=0}^n x^k - 1$$

As such:

$$x \neq 1 \Rightarrow \sum_{k=1}^n x^k = \frac{x^{n+1} - 1}{x - 1} - 1$$

$$x = 1 \Rightarrow \sum_{k=1}^n x^k = n$$



# B

## **Appendix: Calculations and Studies**

## B.1 Time to Interpret a Recognition Result

We performed an empirical experiment to measure the time for interpreting a recognition result, i.e., the time for determining, if a recognition result matches a valid command or not. We based our study on three conclusions which we have drawn from examining the various research papers which have been published for the speech-controlled GUI approaches (see discussion in section 1.2). First, we conclude that a speech recognizer used for a specific speech-controlled GUI approach needs a per-screen vocabulary with a maximum size of maximal 70 words. Second, we conclude that a specific speech-controlled GUI approach defines a maximal number of roughly 8000 valid commands per screen over the speech recognizers vocabulary. The size of the vocabulary and the number of valid commands can (dynamically) be minimized specifically to the current screen, which improves the word error rate and which reduces interpretation complexity (Smailagic and Siewiorek [175]). However, the figures of 70 words and 8000 commands are maximum values as the current screen context is a priori unknown in our situation. As such, we examine the worst case. Third, we conclude that the valid commands of a specific speech-controlled GUI approach have a length  $\leq 5$ . The speech-GUI model defines a maximum word error rate of 100% (see section 2.5), therefore a recognition result, induced by a command of length 5, has a maximum length of 10 if all occurring word errors are insertions.

For the study we created a Java program which generated 8000 random word sequences with a length of 5 from a vocabulary of 70 words. To obtain the vocabulary of 70 words we used a grammar for representing integer numbers between 1 and 999 depicted in Figure B.2 on the next page – which provided 28 words. For the remaining 42 words we chose different distinct words as they occur in commands for speech-controlled GUIs and other arbitrary words with the same average number of characters. These commands represented the valid commands, as such we examined the worst case where all valid commands have a length of 5. The valid commands were stored in a hash table using the respective command itself as the key. After that we generated 2,000,000 random commands from the same vocabulary with a maximum length of 10 words to simulate recognition results, and looked them up in the hash table. If the hash table contained the newly generated command we interpreted this as a match, otherwise the newly generated command was rejected. We compiled the Java program using Sun's Java compiler for Windows platforms in the version 1.5.0\_01 and ran the program using Sun's Java Virtual Machine (same version) on a Dell Dimension 8400 workstation with 1GB of RAM and with a Pentium IV processor running at a speed of 3.19 GHz. The operating system was Windows XP Professional, Service Pack 2. The study revealed that the interpretation of a single command can be done in an average time of  $3 \cdot 10^{-4}$ ms (refer to exemplary output of the test program in Figure B.1 on the facing page).

```

C:\WINDOWS\system32\cmd.exe
D:\PhD Project\Codebase\Dissertation Stuff>java -cp . CmdInterpretationTest
Command Interpretation Test
Initializing Lexicon ..... done
Initializing 8000 valid commands ..... done
Generating 2000000 commands and interpreting them ..... done

Number of distinct commands in hashtable: 8000
Total Time: 609
Total Cms interpreted: 2000000
Average Interpretation Time: 3.045E-4ms
Max. Interpretation Time : 16ms
Min. Interpretation Time : 0ms

D:\PhD Project\Codebase\Dissertation Stuff>

```

**Figure B.1:** Output of recognition result interpretation test.

```

<number> := [<digit> hundred] (<digit>|<teen>|<tenner> [<digit>]);
<digit> := (one|two|three|four|five|six|seven|eight|nine);
<teen> := (ten|eleven|twelve|thirteen|fourteen|fifteen|sixteen|
          <seventeen|eighteen|nineteen);
<tenner> := (twenty|thirty|forty|fifty|sixty|seventy|eighty|ninety);

```

**Figure B.2:** Grammar for positive integer numbers between 1 and 999.

## B.2 Data for Interaction Delay Calculations

This section contains tables which model intentions and interactions for basic executions of graphical objects from the Swing catalog. Also in this sections are tables, which contain detailed interaction delay calculations for the respective basic executions.

## B.3 Tasks from the User Study

Table X depicts the tasks which we used during the user study.

| <b>Execution</b>                                | <b>Intention</b>   | <b>Interaction</b>   |
|---|--|--|
| $E_{button}$                                    | $(SFNAV_{id})$   | $(CMDNAV_{id})$  |
| $E_{checkbox-check}/$<br>$E_{checkbox-uncheck}$ | $(SFNAV_{id})$   | $(CMDNAV_{id})$  |
| $E_{radio}$                                     | $(SFNAV_{id})$   | $(CMDNAV_{id})$  |
| $E_{dropdown}$                                  | $(SFNAV_{id}, SFNAV_{id})$   | $(CMDNAV_{id}, CMDNAV_{id})$   |
| $E_{list-select}/$<br>$E_{list-deselect}$       | $(SFNAV_{id}, SFNAV_{id})$   | $(CMDNAV_{id}, CMDNAV_{id})$   |
| $E_{menu}$                                      | $(SFNAV_{id}, SFNAV_{id})$   | $(CMDNAV_{id}, CMDNAV_{id})$   |
| $E_{tree-select}$                               | $(SFNAV_{id},$<br>$\underbrace{SFNAV_{id}, \dots, SFNAV_{id}}_{n_{depth}},$<br>$SFACT_{SELECT})$ | $(CMDNAV_{id},$<br>$\underbrace{CMDNAV_{id}, \dots, CMDNAV_{id}}_{n_{depth}},$<br>$CMDACT_{SELECT})$ |
| $E_{tree-deselect}$                             | $(SFNAV_{id}, SFNAV_{id},$<br>$SFACT_{DESELECT})$  | $(CMDNAV_{id}, CMDNAV_{id},$<br>$CMDACT_{DESELECT})$   |
| $E_{input}$                                     | $(SFNAV_{id},$<br>$\underbrace{SFACT_{APPEND_x}, \dots, SFACT_{APPEND_x}}_{n_{text}})$           | $(CMDNAV_{id},$<br>$\underbrace{CMDACT_{APPEND_x}, \dots, CMDACT_{APPEND_x}}_{n_{text}})$            |
| $E_{spinner}$                                   | $(SFNAV_{id},$<br>$\underbrace{SFACT_{APPEND_x}, \dots, SFACT_{APPEND_x}}_{n_{digit}})$          | $(CMDNAV_{id},$<br>$\underbrace{CMDACT_{APPEND_x}, \dots, CMDACT_{APPEND_x}}_{n_{digit}})$           |
| $E_{spinner-inc}/$<br>$E_{spinner-dec}$         | $(SFNAV_{id}, SFACT_{INCREASE})/$<br>$(SFNAV_{id}, SFACT_{DECREASE})$                            | $(CMDNAV_{id}, CMDACT_{INCREASE})/$<br>$(CMDNAV_{id}, CMDACT_{DECREASE})$                            |
| $E_{slide}$                                     | $(SFNAV_{id}, SFNAV_{id})$   | $(CMDNAV_{id}, CMDNAV_{id})$   |

**Table B.1:** Intentions and interactions for basic executions using conventional command-and-control.

### B.3. TASKS FROM THE USER STUDY

| Execution                                       | Interaction delay in ms with a word error rate of |  |  |  |  |  |  |   |
|---|---|--|--|--|--|--|--|---|
|   | 0%  | 1%                                       | 5%                                       | 10%                                      | 20%                                      | 30%                                      | 40%                                      | 50%                                       |
| $E_{button}$                                    | 2642  | 2696                                     | 2927                                     | 3262                                     | 4128                                     | 5392                                     | 7339                                     | 10568                                     |
| $E_{checkbox-check}/$<br>$E_{checkbox-uncheck}$ | 2642  | 2696                                     | 2927                                     | 3262                                     | 4128                                     | 5392                                     | 7339                                     | 10568                                     |
| $E_{radio}$                                     | 2642  | 2696                                     | 2927                                     | 3262                                     | 4128                                     | 5392                                     | 7339                                     | 10568                                     |
| $E_{dropdown}$                                  | 5284  | 5391                                     | 5855                                     | 6523                                     | 8256                                     | 10784                                    | 14678                                    | 21137                                     |
| $E_{list-select}/$<br>$E_{list-deselect}$       | 5284  | 5391                                     | 5855                                     | 6523                                     | 8256                                     | 10784                                    | 14678                                    | 21137                                     |
| $E_{menu}$                                      | 5284  | 5391                                     | 5855                                     | 6523                                     | 8256                                     | 10784                                    | 14678                                    | 21137                                     |
| $E_{tree-select}$                               | $(n_{depth} + 1) \cdot$<br>2642+<br>2071          | $(n_{depth} + 1) \cdot$<br>2696+<br>2092 | $(n_{depth} + 1) \cdot$<br>2927+<br>2180 | $(n_{depth} + 1) \cdot$<br>3262+<br>2301 | $(n_{depth} + 1) \cdot$<br>4128+<br>2589 | $(n_{depth} + 1) \cdot$<br>5392+<br>2959 | $(n_{depth} + 1) \cdot$<br>7339+<br>3452 | $(n_{depth} + 1) \cdot$<br>10568+<br>4142 |
| $E_{tree-deselect}$                             | 7355  | 7483                                     | 8035                                     | 8825                                     | 10845                                    | 13742                                    | 18130                                    | 25279                                     |
| $E_{input}$                                     | 2642<br>+<br>$n_{text}$<br>·<br>2642              | 2696<br>+<br>$n_{text}$<br>·<br>2696     | 2927<br>+<br>$n_{text}$<br>·<br>2927     | 3262<br>+<br>$n_{text}$<br>·<br>3262     | 4128<br>+<br>$n_{text}$<br>·<br>4128     | 5392<br>+<br>$n_{text}$<br>·<br>5392     | 7339<br>+<br>$n_{text}$<br>·<br>7339     | 10568<br>+<br>$n_{text}$<br>·<br>10568    |
| $E_{spinner}$                                   | 2642<br>+<br>$n_{digit}$<br>·<br>2642             | 2696<br>+<br>$n_{digit}$<br>·<br>2696    | 2927<br>+<br>$n_{digit}$<br>·<br>2927    | 3262<br>+<br>$n_{digit}$<br>·<br>3262    | 4128<br>+<br>$n_{digit}$<br>·<br>4128    | 5392<br>+<br>$n_{digit}$<br>·<br>5392    | 7339<br>+<br>$n_{digit}$<br>·<br>7339    | 10568<br>+<br>$n_{digit}$<br>·<br>10568   |
| $E_{spinner-inc}/$<br>$E_{spinner-dec}$         | 4713  | 4788                                     | 5107                                     | 5563                                     | 6717                                     | 8350                                     | 10790                                    | 14710                                     |
| $E_{slide}$                                     | 5284  | 5391                                     | 5855                                     | 6523                                     | 8256                                     | 10784                                    | 14678                                    | 21137                                     |

**Table B.2:** Interaction delay calculation details of conventional command-and-control.

| <b>Execution</b>       | <b>Intention</b>   | <b>Interaction</b>   |
|------------------------|--|--|
| $E_{button}$           | $(SFACT_{PUSH})$   | $(CMDNAV_{id}, CMDACT_{PUSH})$   |
| $E_{checkbox-check}$   | $(SFACT_{CHECK})$  | $(CMDNAV_{id}, CMDACT_{CHECK})$  |
| $E_{checkbox-uncheck}$ | $(SFACT_{UNCHECK})$  | $(CMDNAV_{id}, CMDACT_{UNCHECK})$  |
| $E_{radio}$            | $(SFACT_{SELECT})$   | $(CMDNAV_{id}, CMDACT_{SELECT})$   |
| $E_{dropdown}$         | $(SFACT_{OPEN}, SFACT_{SELECT})$   | $(CMDNAV_{id}, CMDACT_{OPEN}, CMDNAV_{id}, CMDACT_{SELECT})$   |
| $E_{list-select}$      | $(SFACT_{SELECT})$   | $(CMDNAV_{id}, CMDNAV_{id}, CMDACT_{SELECT})$  |
| $E_{list-deselect}$    | $(SFACT_{DESELECT})$   | $(CMDNAV_{id}, CMDNAV_{id}, CMDACT_{DESELECT})$  |
| $E_{menu}$             | $(SFACT_{OPEN}, SFACT_{INVOKE})$   | $(CMDNAV_{id}, CMDACT_{OPEN}, CMDNAV_{id}, CMDACT_{INVOKE})$   |
| $E_{tree-select}$      | $EX := (SFACT_{EXPAND}),$ then<br>$(SFACT_{SELECT})$<br>$\underbrace{EX, \dots, EX}_{n_{depth}}$ | $EX := (CMDNAV_{id}, CMDACT_{EXPAND}),$ then<br>$(CMDNAV_{id},$<br>$\underbrace{EX, \dots, EX}_{n_{depth}}, CMDNAV_{id}, CMDACT_{SELECT})$ |
| $E_{tree-deselect}$    | $(SFACT_{DESELECT})$   | $(CMDNAV_{id}, CMDNAV_{id}, CMDACT_{DESELECT})$  |
| $E_{input}$            | $(SFACT_{APPEND_x}, \dots, SFACT_{APPEND_x})$<br>$\underbrace{\hspace{10em}}_{n_{text}}$         | $(CMDNAV_{id},$<br>$\underbrace{CMDACT_{APPEND_x}, \dots, CMDACT_{APPEND_x}}_{n_{text}})$  |
| $E_{spinner}$          | $(SFACT_{APPEND_x}, \dots, SFACT_{APPEND_x})$<br>$\underbrace{\hspace{10em}}_{n_{digit}}$        | $(CMDNAV_{id},$<br>$\underbrace{CMDACT_{APPEND_x}, \dots, CMDACT_{APPEND_x}}_{n_{digit}})$   |
| $E_{spinner-inc}$      | $(SFACT_{INCREASE})$   | $(CMDNAV_{id}, CMDACT_{INCREASE})$   |
| $E_{spinner-dec}$      | $(SFACT_{DECREASE})$   | $(CMDNAV_{id}, CMDACT_{DECREASE})$   |
| $E_{slide}$            | $(SFACT_{SLIDE})$  | $(CMDNAV_{id}, CMDNAV_{id}, CMDACT_{SLIDE})$   |

**Table B.3:** Intentions and interactions for basic executions using command-and-control with random navigation and direct activation.

| Execution              | Interaction delay in ms with a word error rate of |             |             |             |             |             |             |             |
|------------------------|---|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
|                        | 0%  | 1%          | 5%          | 10%         | 20%         | 30%         | 40%         | 50%         |
| $E_{button}$           | 4713  | 4788        | 5107        | 5563        | 6717        | 8350        | 10791       | 14710       |
| $E_{checkbox-check}$   | 4713  | 4788        | 5107        | 5563        | 6717        | 8350        | 10791       | 14710       |
| $E_{checkbox-uncheck}$ | 4713  | 4788        | 5107        | 5563        | 6717        | 8350        | 10791       | 14710       |
| $E_{radio}$            | 4713  | 4788        | 5107        | 5563        | 6717        | 8350        | 10791       | 14710       |
| $E_{dropdown}$         | 9426  | 9575        | 10215       | 11126       | 13434       | 16701       | 21581       | 29420       |
| $E_{list-select}$      | 7355  | 7483        | 8035        | 8825        | 10845       | 13742       | 18130       | 25278       |
| $E_{list-deselect}$    | 7355  | 7483        | 8035        | 8825        | 10845       | 13742       | 18130       | 25278       |
| $E_{menu}$             | 9426  | 9575        | 10215       | 11126       | 13434       | 16701       | 21581       | 29420       |
| $E_{tree-select}$      | 2642  | 2696        | 2927        | 3262        | 4128        | 5392        | 7339        | 10568       |
|                        | +   | +           | +           | +           | +           | +           | +           | +           |
|                        | $n_{depth}$                                       | $n_{depth}$ | $n_{depth}$ | $n_{depth}$ | $n_{depth}$ | $n_{depth}$ | $n_{depth}$ | $n_{depth}$ |
|                        | .   | .           | .           | .           | .           | .           | .           | .           |
|                        | 4713  | 4788        | 5107        | 5563        | 6717        | 8350        | 10791       | 14710       |
| $E_{tree-deselect}$    | +   | +           | +           | +           | +           | +           | +           | +           |
|                        | 2071  | 2092        | 2180        | 2301        | 2589        | 2959        | 3452        | 4142        |
|                        | 7355  | 7483        | 8035        | 8825        | 10845       | 13742       | 18130       | 25278       |
| $E_{input}$            | 2642  | 2696        | 2927        | 3262        | 4128        | 5392        | 7339        | 10568       |
|                        | +   | +           | +           | +           | +           | +           | +           | +           |
|                        | $n_{text}$  | $n_{text}$  | $n_{text}$  | $n_{text}$  | $n_{text}$  | $n_{text}$  | $n_{text}$  | $n_{text}$  |
|                        | .   | .           | .           | .           | .           | .           | .           | .           |
| $E_{spinner}$          | 2642  | 2696        | 2927        | 3262        | 4128        | 5392        | 7339        | 10568       |
|                        | +   | +           | +           | +           | +           | +           | +           | +           |
|                        | $n_{digit}$                                       | $n_{digit}$ | $n_{digit}$ | $n_{digit}$ | $n_{digit}$ | $n_{digit}$ | $n_{digit}$ | $n_{digit}$ |
|                        | .   | .           | .           | .           | .           | .           | .           | .           |
| $E_{spinner-inc}$      | 4713  | 4788        | 5107        | 5563        | 6717        | 8350        | 10791       | 14710       |
| $E_{spinner-dec}$      | 4713  | 4788        | 5107        | 5563        | 6717        | 8350        | 10791       | 14710       |
| $E_{slide}$            | 7355  | 7483        | 8035        | 8825        | 10845       | 13742       | 18130       | 25278       |

**Table B.4:** Interaction delay calculation details of command-and-control with random navigation and direct activation.

| <b>Execution</b>                                 | <b>Intention</b>   | <b>Interaction</b>  |
|--|--|---|
| $E_{button}$                                     | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{checkbox-check}$ /<br>$E_{checkbox-uncheck}$ | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{radio}$                                      | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{dropdown}$                                   | (SFPOINT, SFCLICK,<br>SFMOVEAX, SFCLICK)                                 | (SFPOINT <sub>cmd</sub> , "click", )<br>SFMOVEAX <sub>cmd</sub> , "click")                    |
| $E_{list-select}$ /<br>$E_{list-deselect}$       | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{menu}$                                       | (SFPOINT, SFCLICK<br>SFMOVEAX, SFCLICK)                                  | (SFPOINT <sub>cmd</sub> , "click", )<br>SFMOVEAX <sub>cmd</sub> , "click")                    |
| $E_{tree-select}$                                | $\underbrace{(SFPOINT, SFCLICK)}_{n_{depth}times}$<br>SFPOINT, SFCLICK)  | $\underbrace{(SFPOINT_{cmd}, "click")}_{n_{depth}times}$<br>SFPOINT <sub>cmd</sub> , "click") |
| $E_{tree-deselect}$                              | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{input}$                                      | (SFPOINT, SFCLICK,<br>$\underbrace{SFACT_{APPEND_x}}_{n_{text}times}$ )  | SFPOINT <sub>cmd</sub> , "click",<br>$\underbrace{CMDACT_{APPEND_x}}_{n_{text}times}$         |
| $E_{spinner}$                                    | (SFPOINT, SFCLICK,<br>$\underbrace{SFACT_{APPEND_x}}_{n_{digit}times}$ ) | SFPOINT <sub>cmd</sub> , "click",<br>$\underbrace{CMDACT_{APPEND_x}}_{n_{text}times}$         |
| $E_{spinner-inc}$ /<br>$E_{spinner-dec}$         | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{slide}$                                      | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |

**Table B.5:** Intentions and interactions for basic executions using direction-based mouse emulation with continuous movement.



| Execution                                       | Interaction delay in ms with a word error rate of |  |  |  |  |  |  |   |
|---|---|--|--|--|--|--|--|---|
|   | 0%  | 1%                                     | 5%                                     | 10%                                    | 20%                                    | 30%                                    | 40%                                    | 50%                                     |
| $E_{button}$                                    | 10355   | 10460                                  | 10900                                  | 11506                                  | 12944                                  | 14793                                  | 17258                                  | 20710                                   |
| $E_{checkbox-check}/$<br>$E_{checkbox-uncheck}$ | 10355   | 10460                                  | 10900                                  | 11506                                  | 12944                                  | 14793                                  | 17258                                  | 20710                                   |
| $E_{radio}$                                     | 10355   | 10460                                  | 10900                                  | 11506                                  | 12944                                  | 14793                                  | 17258                                  | 20710                                   |
| $E_{dropdown}$                                  | 16568   | 16735                                  | 17440                                  | 18408                                  | 20710                                  | 23669                                  | 27613                                  | 33136                                   |
| $E_{list-select}/$<br>$E_{list-deselect}$       | 10355   | 10460                                  | 10900                                  | 11506                                  | 12944                                  | 14793                                  | 17258                                  | 20710                                   |
| $E_{menu}$                                      | 16568   | 16735                                  | 17440                                  | 18408                                  | 20710                                  | 23669                                  | 27613                                  | 33136                                   |
| $E_{tree-select}$                               | $(n_{depth} + 1) \cdot$<br>10355                  | $(n_{depth} + 1) \cdot$<br>10460       | $(n_{depth} + 1) \cdot$<br>10900       | $(n_{depth} + 1) \cdot$<br>11506       | $(n_{depth} + 1) \cdot$<br>12944       | $(n_{depth} + 1) \cdot$<br>14793       | $(n_{depth} + 1) \cdot$<br>17258       | $(n_{depth} + 1) \cdot$<br>20710        |
| $E_{tree-deselect}$                             | 10355   | 10460                                  | 10900                                  | 11506                                  | 12944                                  | 14793                                  | 17258                                  | 20710                                   |
| $E_{input}$                                     | 10355<br>+<br>$n_{text}$<br>·<br>2642             | 10460<br>+<br>$n_{text}$<br>·<br>2696  | 10900<br>+<br>$n_{text}$<br>·<br>2927  | 11506<br>+<br>$n_{text}$<br>·<br>3261  | 12944<br>+<br>$n_{text}$<br>·<br>4128  | 14793<br>+<br>$n_{text}$<br>·<br>5392  | 17258<br>+<br>$n_{text}$<br>·<br>7339  | 20710<br>+<br>$n_{text}$<br>·<br>10568  |
| $E_{spinner}$                                   | 10355<br>+<br>$n_{digit}$<br>·<br>2642            | 10460<br>+<br>$n_{digit}$<br>·<br>2696 | 10900<br>+<br>$n_{digit}$<br>·<br>2927 | 11506<br>+<br>$n_{digit}$<br>·<br>3261 | 12944<br>+<br>$n_{digit}$<br>·<br>4128 | 14793<br>+<br>$n_{digit}$<br>·<br>5392 | 17258<br>+<br>$n_{digit}$<br>·<br>7339 | 20710<br>+<br>$n_{digit}$<br>·<br>10568 |
| $E_{spinner-inc}/$<br>$E_{spinner-dec}$         | 10355   | 10460                                  | 10900                                  | 11506                                  | 12944                                  | 14793                                  | 17258                                  | 20710                                   |
| $E_{slide}$                                     | 10355   | 10460                                  | 10900                                  | 11506                                  | 12944                                  | 14793                                  | 17258                                  | 20710                                   |

**Table B.6:** Interaction delay calculation details of direction-based mouse emulation with continuous movement.

| <b>Execution</b>                                | <b>Intention</b>   | <b>Interaction</b>   |
|---|--|--|
| $E_{button}$                                    | $(SFPOINT, SFCLICK)$   | $(SFPOINT_{cmd}, "click")$   |
| $E_{checkbox-check}/$<br>$E_{checkbox-uncheck}$ | $(SFPOINT, SFCLICK)$   | $(SFPOINT_{cmd}, "click")$   |
| $E_{radio}$                                     | $(SFPOINT, SFCLICK)$   | $(SFPOINT_{cmd}, "click")$   |
| $E_{dropdown}$                                  | $(SFPOINT, SFCLICK,$<br>$SFMOVECOMP, SFCLICK)$                             | $(SFPOINT_{cmd}, "click", )$<br>$SFMOVECOMP_{cmd}, "click")$                   |
| $E_{list-select}/$<br>$E_{list-deselect}$       | $(SFPOINT, SFCLICK)$   | $(SFPOINT_{cmd}, "click")$   |
| $E_{menu}$                                      | $(SFPOINT, SFCLICK$<br>$SFMOVECOMP, SFCLICK)$                              | $(SFPOINT_{cmd}, "click", )$<br>$SFMOVECOMP_{cmd}, "click")$                   |
| $E_{tree-select}$                               | $(\underbrace{SFPOINT, SFCLICK}_{n_{depth,times}})$                        | $(\underbrace{SFPOINT_{cmd}, "click"}_{n_{depth,times}})$                      |
| $E_{tree-deselect}$                             | $(SFPOINT, SFCLICK)$   | $(SFPOINT_{cmd}, "click")$   |
| $E_{input}$                                     | $(SFPOINT, SFCLICK,$<br>$\underbrace{SFACT_{APPEND_x}}_{n_{text,times}})$  | $SFPOINT_{cmd}, "click",$<br>$\underbrace{CMDACT_{APPEND_x}}_{n_{text,times}}$ |
| $E_{spinner}$                                   | $(SFPOINT, SFCLICK,$<br>$\underbrace{SFACT_{APPEND_x}}_{n_{digit,times}})$ | $SFPOINT_{cmd}, "click",$<br>$\underbrace{CMDACT_{APPEND_x}}_{n_{text,times}}$ |
| $E_{spinner-inc}/$<br>$E_{spinner-dec}$         | $(SFPOINT, SFCLICK)$   | $(SFPOINT_{cmd}, "click")$   |
| $E_{slide}$                                     | $(SFPOINT, SFCLICK)$   | $(SFPOINT_{cmd}, "click")$   |

**Table B.7:** Intentions and interactions for basic executions using direction-based mouse emulation with discrete movement.

| Execution                                       | Interaction delay in ms with a word error rate of |  |  |  |  |  |  |  |
|---|---|--|--|--|--|--|--|--|
|   | 0%  | 1%                                     | 5%                                     | 10%                                    | 20%                                    | 30%                                    | 40%                                    | 50%                                      |
| $E_{button}$                                    | 10438   | 10864                                  | 12829                                  | 16030                                  | 26471                                  | 47692                                  | 95768                                  | 221628                                   |
| $E_{checkbox-check}/$<br>$E_{checkbox-uncheck}$ | 10438   | 10864                                  | 12829                                  | 16030                                  | 26471                                  | 47692                                  | 95768                                  | 221628                                   |
| $E_{radio}$                                     | 10438   | 10864                                  | 12829                                  | 16030                                  | 26471                                  | 47692                                  | 95768                                  | 221628                                   |
| $E_{dropdown}$                                  | 16693   | 17342                                  | 20333                                  | 25196                                  | 41000                                  | 73017                                  | 145378                                 | 334513                                   |
| $E_{list-select}/$<br>$E_{list-deselect}$       | 10438   | 10864                                  | 12829                                  | 16030                                  | 26471                                  | 47692                                  | 95768                                  | 221628                                   |
| $E_{menu}$                                      | 16693   | 17342                                  | 20333                                  | 25196                                  | 41000                                  | 73017                                  | 145378                                 | 334513                                   |
| $E_{tree-select}$                               | $(n_{depth} + 1) \cdot$<br>10438                  | $(n_{depth} + 1) \cdot$<br>10864       | $(n_{depth} + 1) \cdot$<br>12829       | $(n_{depth} + 1) \cdot$<br>16030       | $(n_{depth} + 1) \cdot$<br>26471       | $(n_{depth} + 1) \cdot$<br>47692       | $(n_{depth} + 1) \cdot$<br>95768       | $(n_{depth} + 1) \cdot$<br>221628        |
| $E_{tree-deselect}$                             | 10438   | 10864                                  | 12829                                  | 16030                                  | 26471                                  | 47692                                  | 95768                                  | 221628                                   |
| $E_{input}$                                     | 10438<br>+<br>$n_{text}$<br>·<br>2642             | 10864<br>+<br>$n_{text}$<br>·<br>2696  | 12829<br>+<br>$n_{text}$<br>·<br>2927  | 16030<br>+<br>$n_{text}$<br>·<br>3262  | 26471<br>+<br>$n_{text}$<br>·<br>4128  | 47692<br>+<br>$n_{text}$<br>·<br>5392  | 95768<br>+<br>$n_{text}$<br>·<br>7339  | 221628<br>+<br>$n_{text}$<br>·<br>10568  |
| $E_{spinner}$                                   | 10438<br>+<br>$n_{digit}$<br>·<br>2642            | 10864<br>+<br>$n_{digit}$<br>·<br>2696 | 12829<br>+<br>$n_{digit}$<br>·<br>2927 | 16030<br>+<br>$n_{digit}$<br>·<br>3262 | 26471<br>+<br>$n_{digit}$<br>·<br>4128 | 47692<br>+<br>$n_{digit}$<br>·<br>5392 | 95768<br>+<br>$n_{digit}$<br>·<br>7339 | 221628<br>+<br>$n_{digit}$<br>·<br>10568 |
| $E_{spinner-inc}/$<br>$E_{spinner-dec}$         | 10438   | 10864                                  | 12829                                  | 16030                                  | 26471                                  | 47692                                  | 95768                                  | 221628                                   |
| $E_{slide}$                                     | 10438   | 10864                                  | 12829                                  | 16030                                  | 26471                                  | 47692                                  | 95768                                  | 221628                                   |

**Table B.8:** Interaction delay calculation details of direction-based mouse emulation with discrete movement.

| <b>Execution</b>                                 | <b>Intention</b>   | <b>Interaction</b>  |
|--|--|---|
| $E_{button}$                                     | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{checkbox-check}$ /<br>$E_{checkbox-uncheck}$ | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{radio}$                                      | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{dropdown}$                                   | (SFPOINT, SFCLICK,<br>SFPOINT, SFCLICK)                                  | (SFPOINT <sub>cmd</sub> , "click", )<br>SFPOINT <sub>cmd</sub> , "click")                     |
| $E_{list-select}$ /<br>$E_{list-deselect}$       | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{menu}$                                       | (SFPOINT, SFCLICK<br>SFPOINT, SFCLICK)                                   | (SFPOINT <sub>cmd</sub> , "click", )<br>SFPOINT <sub>cmd</sub> , "click")                     |
| $E_{tree-select}$                                | $\underbrace{(SFPOINT, SFCLICK)}_{n_{depth}times}$<br>SFPOINT, SFCLICK)  | $\underbrace{(SFPOINT_{cmd}, "click")}_{n_{depth}times}$<br>SFPOINT <sub>cmd</sub> , "click") |
| $E_{tree-deselect}$                              | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{input}$                                      | (SFPOINT, SFCLICK,<br>$\underbrace{SFACT_{APPEND_x}}_{n_{text}times}$ )  | SFPOINT <sub>cmd</sub> , "click",<br>$\underbrace{CMDACT_{APPEND_x}}_{n_{text}times}$         |
| $E_{spinner}$                                    | (SFPOINT, SFCLICK,<br>$\underbrace{SFACT_{APPEND_x}}_{n_{digit}times}$ ) | SFPOINT <sub>cmd</sub> , "click",<br>$\underbrace{CMDACT_{APPEND_x}}_{n_{text}times}$         |
| $E_{spinner-inc}$ /<br>$E_{spinner-dec}$         | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |
| $E_{slide}$                                      | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")  |

**Table B.9:** Intentions and interactions for basic executions using target-based mouse emulation.

| Execution                                       | Interaction delay in ms with a word error rate of |                                       |                                       |                                       |                                       |                                       |  |   |
|---|---|---------------------------------------|---------------------------------------|---------------------------------------|---------------------------------------|---------------------------------------|--|---|
|   | 0%  | 1%                                    | 5%                                    | 10%                                   | 20%                                   | 30%                                   | 40%                                    | 50%                                     |
| $E_{button}$                                    | 4713  | 4788                                  | 5107                                  | 5563                                  | 6717                                  | 8350                                  | 10791                                  | 14710                                   |
| $E_{checkbox-check}/$<br>$E_{checkbox-uncheck}$ | 4713  | 4788                                  | 5107                                  | 5563                                  | 6717                                  | 8350                                  | 10791                                  | 14710                                   |
| $E_{radio}$                                     | 4713  | 4788                                  | 5107                                  | 5563                                  | 6717                                  | 8350                                  | 10791                                  | 14710                                   |
| $E_{dropdown}$                                  | 9426  | 9575                                  | 10215                                 | 11126                                 | 13434                                 | 16701                                 | 21581                                  | 29421                                   |
| $E_{list-select}/$<br>$E_{list-deselect}$       | 4713  | 4788                                  | 5107                                  | 5563                                  | 6717                                  | 8350                                  | 10791                                  | 14710                                   |
| $E_{menu}$                                      | 9426  | 9575                                  | 10215                                 | 11126                                 | 13434                                 | 16701                                 | 21581                                  | 29421                                   |
| $E_{tree-select}$                               | $(n_{depth} + 1) \cdot$<br>4713                   | $(n_{depth} + 1) \cdot$<br>4788       | $(n_{depth} + 1) \cdot$<br>5107       | $(n_{depth} + 1) \cdot$<br>5563       | $(n_{depth} + 1) \cdot$<br>6717       | $(n_{depth} + 1) \cdot$<br>8350       | $(n_{depth} + 1) \cdot$<br>10791       | $(n_{depth} + 1) \cdot$<br>14710        |
| $E_{tree-deselect}$                             | 4713  | 4788                                  | 5107                                  | 5563                                  | 6717                                  | 8350                                  | 10791                                  | 14710                                   |
| $E_{input}$                                     | 4713<br>+<br>$n_{text}$<br>·<br>2642              | 4788<br>+<br>$n_{text}$<br>·<br>2696  | 5107<br>+<br>$n_{text}$<br>·<br>2927  | 5563<br>+<br>$n_{text}$<br>·<br>3262  | 6717<br>+<br>$n_{text}$<br>·<br>4128  | 8350<br>+<br>$n_{text}$<br>·<br>5392  | 10791<br>+<br>$n_{text}$<br>·<br>7339  | 14710<br>+<br>$n_{text}$<br>·<br>10568  |
| $E_{spinner}$                                   | 4713<br>+<br>$n_{digit}$<br>·<br>2642             | 4788<br>+<br>$n_{digit}$<br>·<br>2696 | 5107<br>+<br>$n_{digit}$<br>·<br>2927 | 5563<br>+<br>$n_{digit}$<br>·<br>3262 | 6717<br>+<br>$n_{digit}$<br>·<br>4128 | 8350<br>+<br>$n_{digit}$<br>·<br>5392 | 10791<br>+<br>$n_{digit}$<br>·<br>7339 | 14710<br>+<br>$n_{digit}$<br>·<br>10568 |
| $E_{spinner-inc}/$<br>$E_{spinner-dec}$         | 4713  | 4788                                  | 5107                                  | 5563                                  | 6717                                  | 8350                                  | 10791                                  | 14710                                   |
| $E_{slide}$                                     | 4713  | 4788                                  | 5107                                  | 5563                                  | 6717                                  | 8350                                  | 10791                                  | 14710                                   |

**Table B.10:** Interaction delay calculation details of target-based mouse emulation.

| Execution  | Intention  | Interaction  |
|--|--|--|
| $E_{button}$                                     | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")   |
| $E_{checkbox-check}$ /<br>$E_{checkbox-uncheck}$ | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")   |
| $E_{radio}$                                      | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")   |
| $E_{dropdown}$                                   | (SFPOINT, SFCLICK,<br>SFPOINT, SFCLICK)  | (SFPOINT <sub>cmd</sub> , "click" ,<br>SFPOINT <sub>cmd</sub> , "click")   |
| $E_{list-select}$ /<br>$E_{list-deselect}$       | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")   |
| $E_{menu}$                                       | (SFPOINT, SFCLICK<br>SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click" ,<br>SFPOINT <sub>cmd</sub> , "click")   |
| $E_{tree-select}$                                | (SFPOINT, SFCLICK<br>$\underbrace{\hspace{10em}}_{n_{depth}times}$<br>SFPOINT, SFCLICK)                      | (SFPOINT <sub>cmd</sub> , "click")<br>$\underbrace{\hspace{10em}}_{n_{depth,times}}$<br>SFPOINT <sub>cmd</sub> , "click")    |
| $E_{tree-deselect}$                              | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")   |
| $E_{input}$                                      | (SFPOINT, SFCLICK,<br>$\underbrace{\hspace{10em}}_{n_{text}times}$<br>SFACT <sub>APPEND<sub>x</sub></sub> )  | SFPOINT <sub>cmd</sub> , "click" ,<br>$\underbrace{\hspace{10em}}_{n_{text}times}$<br>CMDACT <sub>APPEND<sub>x</sub></sub> ) |
| $E_{spinner}$                                    | (SFPOINT, SFCLICK,<br>$\underbrace{\hspace{10em}}_{n_{digit}times}$<br>SFACT <sub>APPEND<sub>x</sub></sub> ) | SFPOINT <sub>cmd</sub> , "click" ,<br>$\underbrace{\hspace{10em}}_{n_{text}times}$<br>CMDACT <sub>APPEND<sub>x</sub></sub> ) |
| $E_{spinner-inc}$ /<br>$E_{spinner-dec}$         | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")   |
| $E_{slide}$                                      | (SFPOINT, SFCLICK)   | (SFPOINT <sub>cmd</sub> , "click")   |

**Table B.11:** Intentions and interactions for basic executions using grid-based mouse emulation.

|   | Interaction delay in ms with a word error rate of |      |      |      |       |       |       |       |
|---|---|------|------|------|-------|-------|-------|-------|
|   | 0%  | 1%   | 5%   | 10%  | 20%   | 30%   | 40%   | 50%   |
| $i_{shift}$   | 2071  | 2092 | 2180 | 2301 | 2589  | 2959  | 3452  | 4142  |
| $i_{select}$  | 2642  | 2696 | 2927 | 3262 | 4128  | 5392  | 7339  | 10568 |
| Int. Del. of<br>Pointing<br>$0.45i_{shift}$ +<br>$2.73i_{select}$ | 8145  | 8301 | 8971 | 9941 | 12434 | 16052 | 21589 | 30715 |

**Table B.12:** Values for  $i_{shift}$  and  $i_{select}$  for specific word error rates.

### B.3. TASKS FROM THE USER STUDY

| Execution                                       | Interaction delay in ms with a word error rate of |  |                                       |  |  |  |  |   |
|---|---|--|---------------------------------------|--|--|--|--|---|
|   | 0%  | 1%                                     | 5%                                    | 10%                                    | 20%                                    | 30%                                    | 40%                                    | 50%                                     |
| $E_{button}$                                    | 10216   | 10393                                  | 11151                                 | 12242                                  | 15023                                  | 19011                                  | 25041                                  | 34857                                   |
| $E_{checkbox-check}/$<br>$E_{checkbox-uncheck}$ | 10216   | 10393                                  | 11151                                 | 12242                                  | 15023                                  | 19011                                  | 25041                                  | 34857                                   |
| $E_{radio}$                                     | 10216   | 10393                                  | 11151                                 | 12242                                  | 15023                                  | 19011                                  | 25041                                  | 34857                                   |
| $E_{dropdown}$                                  | 20432   | 20786                                  | 22302                                 | 24484                                  | 30046                                  | 38022                                  | 50082                                  | 69714                                   |
| $E_{list-select}/$<br>$E_{list-deselect}$       | 10216   | 10393                                  | 11151                                 | 12242                                  | 15023                                  | 19011                                  | 25041                                  | 34857                                   |
| $E_{menu}$                                      | 20432   | 20786                                  | 22302                                 | 24484                                  | 30046                                  | 38022                                  | 50082                                  | 69714                                   |
| $E_{tree-select}$                               | $(n_{depth} + 1) \cdot$<br>10216                  | $(n_{depth} + 1) \cdot$<br>10393       | $(n_{depth} + 1) \cdot$<br>11151      | $(n_{depth} + 1) \cdot$<br>12242       | $(n_{depth} + 1) \cdot$<br>15023       | $(n_{depth} + 1) \cdot$<br>19011       | $(n_{depth} + 1) \cdot$<br>25041       | $(n_{depth} + 1) \cdot$<br>34857        |
| $E_{tree-deselect}$                             | 10216   | 10393                                  | 11151                                 | 12242                                  | 15023                                  | 19011                                  | 25041                                  | 34857                                   |
| $E_{input}$                                     | 10216<br>+<br>$n_{text}$<br>·<br>2642             | 10393<br>+<br>$n_{text}$<br>·<br>2696  | 11151<br>+<br>$n_{text}$<br>·<br>2927 | 12242<br>+<br>$n_{text}$<br>·<br>3262  | 15023<br>+<br>$n_{text}$<br>·<br>4128  | 19011<br>+<br>$n_{text}$<br>·<br>5392  | 25041<br>+<br>$n_{text}$<br>·<br>7339  | 34857<br>+<br>$n_{text}$<br>·<br>10568  |
| $E_{spinner}$                                   | 10216<br>+<br>$n_{digit}$<br>·<br>2642            | 10393<br>+<br>$n_{digit}$<br>·<br>2696 | 11151+<br>$n_{digit}$<br>·<br>2927    | 12242<br>+<br>$n_{digit}$<br>·<br>3262 | 15023<br>+<br>$n_{digit}$<br>·<br>4128 | 19011<br>+<br>$n_{digit}$<br>·<br>5392 | 25041<br>+<br>$n_{digit}$<br>·<br>7339 | 34857<br>+<br>$n_{digit}$<br>·<br>10568 |
| $E_{spinner-inc}/$<br>$E_{spinner-dec}$         | 10216   | 10393                                  | 11151                                 | 12242                                  | 15023                                  | 19011                                  | 25041                                  | 34857                                   |
| $E_{slide}$                                     | 10216   | 10393                                  | 11151                                 | 12242                                  | 15023                                  | 19011                                  | 25041                                  | 34857                                   |

**Table B.13:** Interaction delay calculation details of grid-based mouse emulation.

| <b>Execution</b>                                | <b>Intention</b>   | <b>Interaction Delay in ms</b> |
|---|--|--------------------------------|
| $E_{button}$                                    | (MFPOINTING, MFCLICK)  | 1243                           |
| $E_{checkbox-check}/$<br>$E_{checkbox-uncheck}$ | (MFPOINTING, MFCLICK)  | 1243                           |
| $E_{radio}$                                     | (MFPOINTING, MFCLICK)  | 1243                           |
| $E_{dropdown}$                                  | (MFPOINTING, MFCLICK,<br>MFPOINTING, MFCLICK)  | 2486                           |
| $E_{list-select}/$<br>$E_{list-deselect}$       | (MFPOINTING, MFCLICK)  | 1243                           |
| $E_{menu}$                                      | (MFPOINTING, MFCLICK<br>MFPOINTING, MFCLICK)   | 2486                           |
| $E_{tree-select}$                               | $\underbrace{(MFPOINTING, MFCLICK)}_{n_{depth} \text{ times}}$<br>MFPOINTING, MFCLICK) | $(n_{depth} + 1) \cdot 1243$   |
| $E_{tree-deselect}$                             | (MFPOINTING, MFCLICK)  | 1243                           |
| $E_{spinner-inc}/$<br>$E_{spinner-dec}$         | (MFPOINTING, MFCLICK)  | 1243                           |
| $E_{slide}$                                     | (MFPOINTING, MFCLICK)  | 1243                           |

**Table B.14:** Intentions for basic executions using mouse-controlled GUIs.



| Execution                                       | Interaction delay in ms with a word error rate of |  |  |  |  |   |   |  |
|---|---|--|--|--|--|---|---|--|
|   | 0%  | 1%   | 5%   | 10%  | 20%  | 30%   | 40%   | 50%  |
| $E_{button}$                                    | 2642  | 2696   | 2927   | 3262   | 4128   | 5392  | 7339  | 10568  |
| $E_{checkbox-check}/$<br>$E_{checkbox-uncheck}$ | 2642  | 2696   | 2927   | 3262   | 4128   | 5392  | 7339  | 10568  |
| $E_{radio}$                                     | 2642  | 2696   | 2927   | 3262   | 4128   | 5392  | 7339  | 10568  |
| $E_{dropdown}$                                  | 3784  | 3939   | 4646   | 5767   | 9238   | 15760   | 29198   | 60544  |
| $E_{list-select}/$<br>$E_{list-deselect}$       | 3784  | 3939   | 4646   | 5767   | 9238   | 15760   | 29198   | 60544  |
| $E_{menu}$                                      | 3784  | 3939   | 4646   | 5767   | 9238   | 15760   | 29198   | 60544  |
| $E_{tree-select}$                               | $n_{depth}$<br>$\cdot 2642 +$<br>3213             | $n_{depth}$<br>$\cdot 2696 +$<br>3311            | $n_{depth}$<br>$\cdot 2927 +$<br>3747            | $n_{depth}$<br>$\cdot 3262 +$<br>4407            | $n_{depth}$<br>$\cdot 4128 +$<br>6275            | $n_{depth}$<br>$\cdot 5392 +$<br>9367             | $n_{depth}$<br>$\cdot 7339 +$<br>14875            | $n_{depth}$<br>$\cdot 10568 +$<br>25704            |
| $E_{tree-deselect}$                             | 4355  | 4579   | 5628   | 7375   | 13290  | 25912   | 56006   | 139360   |
| $E_{input}$                                     | 3784<br>+<br>$n_{text}$<br>$-1) \cdot$<br>2642    | 3939<br>+<br>$(n_{text}$<br>$-1) \cdot$<br>2696  | 4646<br>+<br>$(n_{text}$<br>$-1) \cdot$<br>2927  | 5767<br>+<br>$(n_{text}$<br>$-1) \cdot$<br>3262  | 9238<br>+<br>$(n_{text}$<br>$-1) \cdot$<br>4128  | 15760<br>+<br>$(n_{text}$<br>$-1) \cdot$<br>5392  | 29198<br>+<br>$(n_{text}$<br>$-1) \cdot$<br>7339  | 60544<br>+<br>$(n_{text}$<br>$-1) \cdot$<br>10568  |
| $E_{spinner}$                                   | 3784<br>+<br>$n_{digit}$<br>$-1) \cdot$<br>2642   | 3939<br>+<br>$(n_{digit}$<br>$-1) \cdot$<br>2696 | 4646<br>+<br>$(n_{digit}$<br>$-1) \cdot$<br>2927 | 5767<br>+<br>$(n_{digit}$<br>$-1) \cdot$<br>3262 | 9238<br>+<br>$(n_{digit}$<br>$-1) \cdot$<br>4128 | 15760<br>+<br>$(n_{digit}$<br>$-1) \cdot$<br>5392 | 29198<br>+<br>$(n_{digit}$<br>$-1) \cdot$<br>7339 | 60544<br>+<br>$(n_{digit}$<br>$-1) \cdot$<br>10568 |
| $E_{spinner-inc}/$<br>$E_{spinner-dec}$         | 3213  | 3311   | 3747   | 4407   | 6275   | 9367  | 14875   | 25704  |
| $E_{slide}$                                     | 3784  | 3939   | 4646   | 5767   | 9238   | 15760   | 29198   | 60544  |

Table B.15: Interaction delay calculation details of conversation-and-control.

| Nr. | Execution              | Description   |
|-----|------------------------|---|
| 1   | $E_{button}$           | Activate button "Activate"  |
| 2   | $E_{dropdown}$         | Select the fruit "<currentlyUnselectedFruit>"                               |
| 3   | $E_{checkbox-check}$   | Check "Backup"  |
| 4   | $E_{list-select}$      | Select the pizzas "<currentlyUnselPizza>" and "<currentlyUnselPizza>"       |
| 5   | $E_{slide}$            | Select level "<currentlyUnselLevel>"  |
| 6   | $E_{dropdown}$         | Select month other than "<currentlyUnselMonth>"                             |
| 7   | $E_{checkbox-uncheck}$ | Uncheck "Backup"  |
| 8   | $E_{dropdown}$         | Select a fruit other than "<currentlySelectedFruit>"                        |
| 9   | $E_{list-select}$      | Select the toppings "<currentlyUnselTopping>" and "<currentlyUnselTopping>" |
| 10  | $E_{slide}$            | Select grade "<currentlyUnselGrade>"  |
| 11  | $E_{button}$           | Activate button "Process"   |
| 12  | $E_{tree-deselect}$    | Unselect "Fish" from "Animals"  |
| 13  | $E_{spinner}$          | Enter '12' into the amps field  |
| 14  | $E_{list-select}$      | Select origin "<currentlyUnselOrigin>"                                      |
| 15  | $E_{tree-deselect}$    | Unselect "Devices" from "Computer"  |
| 16  | $E_{input}$            | Enter 'vet' into the job field  |
| 17  | $E_{menu}$             | Invoke arbitrary menu item from "Search"                                    |
| 18  | $E_{spinner-inc}$      | Increase the voltage field  |
| 19  | $E_{list-deselect}$    | Unselect the topping "<currentlySelTopping>"                                |
| 20  | $E_{input}$            | Enter 'tom' into the name field   |
| 21  | $E_{menu}$             | Invoke arbitrary menu item from "Edit"                                      |
| 22  | $E_{list-select}$      | Select destination "Pittsburgh"   |
| 23  | $E_{spinner}$          | Enter '45' into the voltage field   |
| 24  | $E_{spinner-dec}$      | Decrease the amps field   |
| 25  | $E_{checkbox-check}$   | Check "Amplify"   |
| 26  | $E_{dropdown}$         | Select the month "<currentlyUnselectedMonth>"                               |
| 27  | $E_{tree-select}$      | Select an arbitrary mammal from animals                                     |
| 28  | $E_{spinner-inc}$      | Increase the amps field   |
| 29  | $E_{tree-select}$      | Select an arbitrary device from computers                                   |
| 30  | $E_{spinner-dec}$      | Decrease the voltage field  |
| 31  | $E_{list-deselect}$    | Unselect the pizza "<currentlySelectedPizza>"                               |
| 32  | $E_{checkbox-check}$   | Uncheck "Amplify"   |

**Table B.16:** Tasks to be completed in the experiment.

# Bibliography

- [1] Raux A., Langner B., Bohus D., Black A., and Eskenazi M. Let's Go Public! Taking a Spoken Dialog System to the Real World. In *Interspeech*, 2005. 136
- [2] Subrahmanyam Allamaraju, Karl Avedal, Richard Browett, Jason Diamond, John Griffin, Mac Holden, Andrew Hoskinson, Rod Johnson, Tracie Karsjens, Larry Kim, Andrew Longshaw, Tom Myers, Alexander Nakimovsky, Daniel O'Connor, Sameer Tyagi, Geert Van Damme, Gordan van Huizen, Mark Wilcox, and Stefan Zeiger. *Professional Java Server Programming, J2EE Edition*. Wrox Press Ltd., 2000. 159
- [3] J. Allen, M. S. Hunnicutt, and D. H. Klatt. *From Text to Speech: The MITalk System*. Cambridge University Press, 1987. 150
- [4] James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski. Robust Understanding in a Dialogue System. In *Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics (ACL'96)*, pages 62–70, 1996. 130, 136
- [5] Deepak Alur, John Crupi, and Dan Malks. *Core J2EE Patterns, Best Practices and Design Patterns*. Sun Microsystems Press, a Prentive Hall Title, 2001. 160
- [6] Jeremy Ang, Rajdip Dhillon, Ashley Krupski, Elizabeth Shriberg, and Andreas Stolcke. Prosody-based Automatic Detection of Annoyance and Frustration in Human-Computer Dialog. In *Proceedings of International Conference on Spoken Language Processing 2002 (ICSLP '02)*, 2002. 148
- [7] Stephen C. Arnold, Leo Mark, and John Goldthwaite. Programming by Voice, Vocal-Programming. In *Proceedings of the fourth international ACM conference on Assistive technologies (ASSETS '00)*, pages 149–155, 2000. 11, 14, 184
- [8] Barry Arons. Techniques, Perception, and Applications of Time-Compressed Speech. In *Proceedings of 1992 Conference, American Voice I/O Society*, pages 116–122, September 1992. 30
- [9] Mark Ashdown, Kenji Oka, and Yoichi Sato. Combining Head Tracking and Mouse Input for a GUI on Multiple Monitors. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, pages 1188–1191, 2005. 5

- [10] Sprex' ASNR. How fast is Speech Recognition?  
<http://cassandra.sprex.com/faq/realtime.ASR.php>. 32
- [11] TMA Associates. Microsoft launches Microsoft Speech Server 2004 with Pricing Details. *Telephone Strategy News, Newsletter*, April 2004.  
[http://www.tmaa.com/microsoft\\_announcement.htm](http://www.tmaa.com/microsoft_announcement.htm). 33
- [12] Fuat Atabey and Recep Kayali. Analyse und modellierung von wartungsabläufen zur migration von papierbasierten wartungsanweisungen zu interactive electronic tech manuals (ietm). Technical report, Technische Universität München, Institut für Informatik, 2004. 163, 165
- [13] MD Atif Zafar, MD J. Marc Overhage, and MD Clement J. McDonald. Continuous Speech Recognition for Clinicians. *J Am Med Inform Assoc*, 6(3):195–204, 1999.  
<http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=61360>. 30, 35
- [14] Robert K. Atkinson, Richard E. Mayer, and Mary Margaret Merrill. Fostering Social Agency in Multimedia Learning: Examining the Impact of an Animated Agent's Voice. *Contemporary Educational Psychology*, 30(1):117–139, 2005. 150
- [15] H. Aust, Oerder M., F. Seide, and V. Steinbiss. The Philips Automatic Train Timetable Information System. *Speech Communication*, 17:249–262, 1995. 130
- [16] L. Bass, R. Faneuf, R. Little, N. Mayer, B. Pellegrino, S. Reed, R. Seacord, S. Sheppard, and M. R. Szczur. A Metamodel for the Runtime Architecture of an Interactive System. *ACM SIGCHI Bulletin*, 24(1):32–37, 1992. 42, 45
- [17] Tanya Bass, James Beck, Kelly Dolan, Ciuwei Li, Andreas Löhr, Richard Martin, William Ross, Tobias Weishäupl, and Gregory Zelesnik. *Software Architecture in Practice*, chapter The Luther Architecture, a Case Study in Mobile Applications Using J2EE, pages 427–452. Addison Wesley, 2003. 159
- [18] L. Baumeister, B. E. John, and M. Byrne. A comparison of tools for building goms models. In *Proceedings of CHI*, pages 502–509, 2000. 81
- [19] A. W. Biermann, C. I. Guinn, D. R. Hipp, and R. W. Smith. Efficient Collaborative Discourse: A Theory and Its Implementation. In *Proc. Human Language Technology Workshop '93*, pages 177–182, 1994. 136
- [20] Jeff Bilmes, Jonathan Malkin, Xiao Li, Susumu Harada, Kelley Kilanski, Katrin Kirchhoff, Richard Wright, Amarnag Subramanya, James Landay, Patricia Dowden, and Howard Chizeck. The Vocal Joystick. In *(to appear) Proceedings of the IEEE International Conference on Audio, Speech and Signal Processing*, 2006. 9

- 
- [21] Alan W Black and Kevin A. Lenzo. Festvox: Festival. <http://www.festvox.org/festival/>. 135, 150
- [22] D. Bohus and A. Rudnicky. LARRI: A Language-Based Maintenance and Repair Assistant. In *ISCA Tutorial and Research Workshop on Multi-Modal Dialogue in Mobile Environments*, 2002. 150, 153, 162, 165
- [23] Dan Bohus and Alexander I. Rudnicky. RavenClaw: Dialog Management using Hierarchical Task Decomposition and an Expectation Agenda. In *Eurospeech*, 2003. 135, 145
- [24] Jürgen Bortz. *Statistik für Human- und Sozialwissenschaftler*. Springer, 2004. 173
- [25] G.E.P. Box. *Robustness in the Strategy of Scientific Model Building*. Academic Press: New York, 1979. 19
- [26] Bernd Brügge and Allen H. Dutoit. *Objektorientierte Softwaretechnik*. Pearson Studium, 2004. 156
- [27] Tom Brøndsted and Erik Aaskoven. Voice-controlled Internet Browsing for Motor-handicapped Users. Design and Implementation Issues. In *Proceedings of the 9th European Conference on Speech Communication and Technology*, 2005. 10, 94
- [28] I. N. Bronstein, K.A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 2000. 192
- [29] P. F. Brown, V. J. D. Petra, P. V. deSouza, J. C. Lai, and R. L. Mercer. Class-based n-gram Models of Natural Language. *Computational Linguistics*, 18(4):467 – 479, 1992. 7
- [30] Mark Buckley and Christoph Benzmüller. A Dialog Manager Supporting Tutorial Natural Language Dialogue on Proofs. *Electronic Notes in Theoretical Computer Science*, page to appear, 2006. 136
- [31] Christian Bürgy, Edwin Vogt, and Ilija Komljenovic. Speech-Controlled Wearbles: Are We There, Yet ? Technical report, Xybernaut Corporation, Research and Development, 2005. [http://www.xybernaut.com/case\\_studies/pdfs/Buergy\\_IFAWC\\_2005\\_final.pdf](http://www.xybernaut.com/case_studies/pdfs/Buergy_IFAWC_2005_final.pdf). 162
- [32] Ron Van Buskirk and Mary LaLomia. A Comparison of Speech and Mouse/Keyboard GUI Navigation. In *Conference companion on Human factors in computing systems*, page 96. ACM Press, 1995. 14, 184
- [33] S. Card, T. Moran, and A. Newell. *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1983. 21

- [34] Stuart Card, Thomas Moran, and Allen Newell. The keystroke-level model for user performance with interactive systems. *Communications of the ACM*, 23:396 – 210, 1980. 20
- [35] B. Carpenter. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1992. 152
- [36] Everest Chiu and Alexander Chu. Computer mouse controlled by eye. Technical report, Electrical and Computer Engineering, 1999.  
[http://courses.ece.uiuc.edu/ece445/projects/spring99/project18\\_final\\_paper.doc](http://courses.ece.uiuc.edu/ece445/projects/spring99/project18_final_paper.doc). 5
- [37] Kevin Christian, Bill Kules, Ben Shneiderman, and Adel Youssef. A Comparison of Voice Controlled and Mouse Controlled Web Browsing. In *Proceedings of the fourth international ACM conference on Assistive technologies (ASSETS '00)*, pages 13–15, November 2000. 14, 16, 176, 184
- [38] Arthur C. Clarke. *2001: A Space Odyssey*. various publishers, 1968. 130
- [39] Robin Cohen, Coralee Allaby, Christian Cumbaa, Mark Fitzgerald, Kinson Ho, Bowen Hui, Celine Latulipe, Fletcher Lu, Nancy Moussa, David Pooley, Alex Qian, and Saheem Siddiqi. What is Initiative? *User Modeling and User-Adapted Interaction*, 8(3-4):171–214, 1998. 107, 131
- [40] Ronald A. Cole, Joseph Mariani, Hans Uszkoreit, Annie Zaenen, and Victor Zue. Survey of the State of the Art in Human Language Technology. Technical report, Cambridge University, 1996.  
<http://cslu.cse.ogi.edu/HLTsurvey/>. 35
- [41] Electronic Commerce Connection. Interactive electronic technical manuals.  
<http://www.ietm.net/>. 165
- [42] Larry L. Constantine. What Do Users Want? Engineering Usability into Software. *Windows Tech Journal*, 1995. 84
- [43] Conversay. Conversay browser products.  
<http://www.conversay.com/>. 12, 27, 83
- [44] Liwei Dai, Rich Goldman, Andrew Sears, and Jeremy Lozier. Speech-Based Cursor Control: A Study of Grid-Based Solutions. *ACM SIGACCESS Accessibility and Computing*, 77-78:94–101, 2004. 10, 14, 97, 184
- [45] C. Danis and J. Karat. Technology-driven Design of Speech Information Systems. In *Proceedings of the Symposium on Designing Interactive Systems*, pages 17–24, 1995. 8

- [46] C. de Mauro, M. Gori, M. Maggini, and E. Martinelli. A Voice Device with an Application-adapted Protocol for Microsoft Windows. In *Proceedings of the International Conference on Multimedia Computing and Systems*, volume 2, pages 1015–1016, 1999. 9, 92
- [47] M. Denecke. Object-Oriented Techniques in Grammar and Ontology Specification. In *Workshop on Multilingual Speech Communication. 2000. Kyoto, Japan: pp. 59-64*, 2000. 152
- [48] Matthias Denecke. *Generische Interaktionsmuster für aufgabenorientierte Dialogsysteme*. PhD thesis, Universität Karlsruhe (Technische Hochschule), Fakultät für Informatik, 2002. 134, 150, 151
- [49] L. Deng, A. Acero, M. Plumpe, and X. Huang. Large-Vocabulary Speech Recognition under Adverse Acoustic Environments. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP*, October 2000. 7
- [50] Foraker Design. Usability First: Usability Glossary: task completion time. [http://www.usabilityfirst.com/glossary/term\\_1042.txt](http://www.usabilityfirst.com/glossary/term_1042.txt). 13
- [51] Angela Terese DiDomenico. *An Investigation on Subjective Assessments of Workload and Postural Stability under Conditions of Joint Mental and Physical Demands*. PhD thesis, Faculty of the Virginia Polytechnic Institute and State University, 2003. 16
- [52] Bryan Duggan. Creating Effective, Efficient and Desirable Voice Enabled Web Interfaces. In *Proceedings of the 8th ERCIM Workshop "User Interfaces for All"*, June 2004. 67
- [53] W. Keith Edwards and Elizabeth D. Mynatt. An Architecture for Transforming Graphical Interfaces. In *Proceedings of the Seventh Symposium on User Interface Software and Technology (UIST '94)*, 1994. 150, 152
- [54] Yasuhiro Endo, Zheng Wang, J. Bradley Chen, and Margo Seltzer. Using Latency to Evaluate Interactive System Performance. In *Proceedings of the 1996 Symposium on Operating System Design and Implementation (OSDI)*, 1996. 103
- [55] C. Engelbart and William K. English. A Research Center for Augmenting Human Intellect. In *AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference*, volume 33, pages 395 – 410, December 1968. <http://sloan.stanford.edu/mousesite/Archive/ResearchCenter1968/ResearchCenter1968.html>. 3
- [56] Daniele Falavigna, Toni Giorgino, and Roberto Gretter. A Frame Based Spoken Dialog System for Home Care. In *Proceedings of Interspeech 2005*, pages 201–204, September 2005. <http://ditelo.itc.it/people/falavi/EUROSPEECH-2005.pdf>. 8

- [57] IETM FAQ.  
<http://www.docsoft.com/ietm-faq.htm>. 165
- [58] P.M Fitts. The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology*, 47:381–391, 1954. 104
- [59] J. D. Foley and V. L. Wallace. The Art of Natural Graphic Man-Machine Conversation. *Proceedings of the IEEE*, 62(4):462 – 471, May 1974. 42, 43
- [60] Otto Forster. *Analysis I*. vieweg studium, 1996. 58
- [61] Thomas Funck. A Pluggable Display Module for a Technical Manual Display System. Master's thesis, Technische Universität München, Institut für Informatik, 2002. 160
- [62] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Pub. Co, 1995. 42, 156, 163
- [63] Xiaorong Gao, Dingfeng Xu, Ming Cheng, and Shangkai Gao. A BCI-based Environmental Controller for the Motion-Disabled. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 11(2):137–140, 2003. 4
- [64] M. Gavalda. Miso: A Parser for Real World Spontaneous Speech. In *Proceedings of the 6th International Workshop on Parsing Technologies*, 2000. 36
- [65] D. Gibbon, R. Moore, and R. Winski. *Handbook of Standards and Resources for Spoken Language Systems*, pages 564–614. Eds. Mouton de Gruyter, New York, 1997. 130
- [66] James R. Glass, Timothy J. Hazen, and I. Lee Hetherington. Real-time Telephone-based Speech Recognition in the Jupiter Domain. In *Proceedings of the IEEE International Conference On Acoustics, Speech, and Signal Processing*, March 1999. 32, 33
- [67] D. Goddeau, E. Brill, J. Glass, C. Pao, M. Phillips, J. Polifroni, S. Seneff, and V. Zue. Galaxy: A Human-language Interface to Online Travel Information. In *Proc. ICSLP '94*, pages 707–710, 1994. 135
- [68] Li Gong and Jennifer Lai. To Mix or Not To Mix Synthetic Speech and Human Speech? Contrasting Impact on Judge-Rated Task Performance versus Self-Rated Performance and Attitudinal Responses. *International Journal of Speech Technology*, 6(2):123–131, 2003. 150
- [69] Allen L. Gorin, Alicia Abella, Tirso Alonso, Giuseppe Riccardi, and Jeremy H. Wright. Automated Natural Spoken Dialog. *IEEE Computer*, 35(4):51–56, 2002. 6
- [70] Mark Green. Report on Dialogue Specification Tools. in *Günther E. Pfaff's "User Interface Management Systems"*, 1985. 42, 45



- 
- [71] H. P. Grice. Logic and Conversation. *Syntax and semantics: Speech acts*, 3:41–58, 1975. 31, 125
- [72] Nicolas Guelfi and Paul Sterges. JAFAR: Detailed Design of a Pattern-based J2EE Framework. In M. H. Hamza, editor, *Proceedings of the Sixth IASTED International Conference on Software Engineering and Applications, November 2002, Cambridge, USA*, 2002. 160
- [73] Thomas K. Harris and Roni Rosenfeld. A universal speech interface for appliances. In *ICSLP*, 2004. 6
- [74] Andreas Hartl. A Widget-based Approach for Creating Voice Applications. In *Proceedings of Physical Interaction (PI03) Workshop on Real World User Interfaces*, September 2003. 133
- [75] Aaron Hillegass. *Cocoa® Programming for Mac® OS X*. Addison-Wesley Professional, 2004. 70
- [76] Hartwig Holzapfel, Christian Fuegen, Matthias Denecke, and Alex Waibel. Integrating Emotional Cues into a Framework for Dialogue Management. In *Proceedings of the 4th IEEE International Conference on Multimodal Interfaces*, pages 141–148, 2002. 148
- [77] X. Huang, F. Alleva, H.-W. Hon, K.-F. Hwang, M.-Y. Lee, , and R. Rosenfeld. The SPHINX-II Speech Recognition System: an Overview. *Computer Speech and Language*, 7(2):137 – 148, 1992. 135
- [78] X. D. Huang, A. Acero, HW. Hon, and R. Reddy. *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Prentice Hall, 2001. 8, 34, 36, 112
- [79] X. D. Huang, Y. Ariki, and M. A. Jack. *Hidden Markov Models for Speech Recognition*. Edinburgh University Press, 1990. 7
- [80] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing*. Prentice Hall, 2001. 8, 33
- [81] T. Igarashi and J. F. Hughes. Voice as Sound: Using Non-verbal Voice Input for Interactive Control. In *Proceedings of the 14th Annual Symposium on User Interface Software and Technology*, 2001. 9
- [82] Nuance Communications Inc. Nuance - The Speech Recognition Leader. <http://www.nuance.com>. 132
- [83] since then modified by various authors initial author Otto Dornblüth, then Willibald Pschyrembel. *Pschyrembel, Klinisches Wörterbuch*. de Gruyter Verlag, 1993. 3

- [84] Inmedius. Inmedius | software to capture, create manage and deploy technical information.  
<http://www.inmedius.com>. 163
- [85] Frankie James and Jeff Roelands. Voice over Workplace (VoWP): Voice Navigation in a Complex Business UI. In *Proceedings of the fifth international ACM conference on Assistive technologies (ASSETS '02)*, pages 9–15, October 2004. 12
- [86] Hyuk Jeong, Jong-Sung Kim, and Jin-Seong Choi. A Study of an EMG-controlled HCI Method by Clenching Teeth. In *APCHI*, pages 163–170, 2004. 4
- [87] B. John, A. Vera, M. Matessa, M. Freed, and R. Remington. Automating cpm-goms. In *Proceedings of CHI*, 2002. 81
- [88] B. E. John and D. E. Kieras. The goms family of user interface analysis techniques: Comparison and contrast. *Transactions on Computer-Human Interaction*, 3(4):320 – 351, 1996. 21
- [89] Bonnie John. Why GOMS? *Interactions*, 2(4):80 – 89, 1995. 21
- [90] Bonnie E. John. Extension of goms analyses to expert performance requiring perception of dynamic visual and auditory information. In *CHI '90*, 1990. 35
- [91] Dan R. Olsen jr., Chung Man Tam, Genevieve Conaty, Matthew Phelps, and Jeremy M. Heiner. Speech Interaction with Graphical User Interfaces. In *Proceedings of the Eighth Conference on Human-Computer Interaction*, July 2001. 11, 150, 153
- [92] K. Kacioglu and W. Ward. A Figure of Merit for the Analysis of Spoken Dialog Systems. In *ICSLP-2002: Inter. Conf. on Spoken Language Processing*, volume 2, pages 877–880, 2002. 125
- [93] C.M. Karat, C. Halverson, D. Horn, and Je. Karat. Patterns of Entry and Correction in Large Vocabulary Continuous Speech Recognition Systems. In *CHI 99 Conference Proceedings*, pages 568–575, 1999. 30
- [94] A. S. Karimullah, A. Sears, M. Lin, and R. Goldmann. Understanding the Effects of Variable Cursor Speed on Target Selection. In *Proceedings of the Tenth International Conference on Human - Computer Interaction (HCII '03)*, June 2003. 9
- [95] Azfar A. Karimullah and Andrew Sears. Speech-Based Cursor Control. In *Proceedings of the fifth international ACM conference on Assistive technologies (ASSETS '02)*, July 2002. 9, 22, 29, 92
- [96] L. Karl, M. Pettey, and B. Shneiderman. Speech-Activated versus Mouse-Activated Commands for Word Processing Applications: An Empirical Evaluation. *International Journal of Man-Machine Studies*, 39(4):667–687, October 1993. 14

- 
- [97] Kenji Kita. Large-Vocabulary Speech Recognition Algorithms. *Journal of Natural Language Processing*, 3(4):103–113, October 1996. 7
- [98] K. Kitajima, Y. Sato, and H. Koike. Vision-Based Face Tracking System for Windows Interface: Prototype Application and Empirical Studies. In *CHI 2001*, pages 359–360, 2001. 5
- [99] Nils Klarlund. Editing by Voice and the Role of Sequential Symbol Systems for Improved Human-to-Computer Information Rates. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2003*, volume 5, pages 728–731, June 2003. 16
- [100] D. H. Klatt. The Klattalk Text-to-Speech Conversion System. In *Proceedings on the International Conference on Acoustic, Speech and Signal Processing '82*, pages 1589–1592, 1982. 150
- [101] J. Klein. Computer Response to User Frustration. Master's thesis, MIT, 1998. 149
- [102] Donald E. Knuth. *The Art of Computer Programming 1-3*. Addison-Wesley Professional, 1998. 137
- [103] Blade Kotelly. *The Art and Business of Speech Recognition, Generating the Noble Voice*. Addison-Wesley Professional M, 2003. 8
- [104] R. Lau, G. Flammia, C. Pao, and V. Zue. WEBGALAXY – Integrating Spoken Language and Hypertext Navigation. In *Proc. EUROSPEECH-97*, pages 883–886, 1997. 135
- [105] BW Leeming, D Porter, JD Jackson, HL Bleich, and M Simon. Computerized Radiologic Reporting with Voice Data-entry. *Radiology*, 138(3):585–588, 1981. 14
- [106] Sami Lemmetty. Review of Speech Synthesis Technology. Master's thesis, Helsinki University of Technology, Department of Electrical and Communications Engineering, 1999. available at <http://www.acoustics.hut.fi/~slemmett/dippa/thesis.pdf>. 150
- [107] J.R. Lewis. Effect of Error Correction Strategy on Speech Dictation Throughput. In *Proceedings of the Human Factors and Ergonomics Society*, pages 457–461, 1999. 30
- [108] Lorna Lines and Kate S. Hone. Older Adults' Evaluations of Speech Output. In *Proceedings of the fifth international ACM conference on Assistive technologies (ASSETS '02)*, pages 170–177, July 2002. 150
- [109] Diane J. Litman and Kate Forbes-Riley. Recognizing Student Emotions and Attitudes on the Basis of Utterances in Spoken Tutoring Dialogues with both Human and Computer Tutors. *Speech Communication*, April 2006. to appear. 149

- [110] Andreas Löhr, Jim Beck, Richard Martin, and Bernd Brügge. A Framework for Browser-based User Interfaces in J2EE. In M. H. Hamza, editor, *Proceedings of the Sixth IASTED International Conference on Software Engineering and Applications, November 2002, Cambridge, USA*, pages 323–330, 2002. 159, 161
- [111] Arnold M. Lund. Measuring Usability with the USE Questionnaire. *Usability Interface*, 8(2), Oktober 2001. 67
- [112] I. Scott MacKenzie and William Buxton. Extending Fitts' Law to Two-Dimensional Tasks. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '92)*, May 1992. 104, 105
- [113] I. Scott MacKenzie, Abigail Sellen, and William Buxton. A Comparison of Input Devices in Elemental Pointing and Dragging Tasks. In *Proceedings of the CHI '91 Conference on Human Factors in Computing Systems*, pages 161–166, 1991. 104
- [114] I.S. MacKenzie. A Note on the Information-theoretic Basics for Fitts' Law. In *Journal of Motor Behavior*, volume 21, pages 323–330, 1989. 104
- [115] Carolyn MacLeod and Steve Northover. SWT Standard Widget Toolkit, Part 2. Technical report, Object Technology International, 2001.  
<http://www.eclipse.org/articles/swt-design-2/swt-design-2.html>. 156
- [116] A. MacWilliams, C. Sandor, M. Wagner, M. Bauer, G. Klinker, and B. Bruegge. Herding Sheep: Live System Development for Distributed Augmented Reality. In *Proceedings of The Second International Symposium on Mixed and Augmented Reality (ISMAR 2003)*, pages 323–330, 2003. 133
- [117] ABILITY Magazine. Hands-free Mouse "Listens" for Instruction. *ABILITY Magazine*, November 2002.  
[http://abilitymagazine.com/news\\_voicemouse.html](http://abilitymagazine.com/news_voicemouse.html). 10, 96
- [118] Bill Manaris, Renée McCauley, and Valanne MacGyvers. An Intelligent Interface for Keyboard and Mouse Control – Providing Full Access to PC Functionality via Speech. In *Proceedings of the 14th International Florida AI Research Symposium*, pages 182–188, May 2001. 9, 92
- [119] Jeniffer Mankoff, Scott E. Hudson, and Gregory D. Abowd. Providing Integrated Toolkit-Level Support for Ambiguity in Recognition-Based Interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '00)*, pages 368–375, April 2000. 186
- [120] Maplesoft. Maplesoft - Command the Brilliance.  
<http://www.maplesoft.com>. 189, 190, 191

- 
- [121] Steven G. Mason and Gary E. Birch. A General Framework for Brain-Computer Interface Design. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 11(1):70–85, 2003. 4
- [122] Takuya Matsuzaki, Yusuke Miyao, and Jun’ichi Tsujii. Probabilistic CGF with Latent Annotations. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics*, pages 75–82, 2005. 7
- [123] Michael F. McTear. Software to Support Research and Development of Spoken Dialogue Systems. In *Proceedings of the Sixth European Conference on Speech Communication and Technology*, September 1999. 133
- [124] Michael F. McTear. Spoken Dialog Technology: Enabling the Conversational Interface. *ACM Computing Surveys*, 34(1):90–169, 2002. 5, 112, 131, 132, 135
- [125] Microsoft. Microsoft Speech - Speech Application Language Tags (SALT). <http://www.microsoft.com/speech/evaluation/spechtags/>. 150, 152
- [126] Microsoft. Microsoft Speech: Home Page. <http://www.microsoft.com/speech/default.aspx>. 10, 20, 48
- [127] Sun Microsystems. Java Speech API. <http://java.sun.com/products/java-media/speech/>. 10, 20
- [128] Sun Microsystems. Java Speech Grammar Format. <http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/>. 150
- [129] Sun Microsystems. JavaServer Pages Technology – White Paper. Technical report, Sun Microsystems. <http://java.sun.com/products/jsp/whitepaper.html>. 159
- [130] Sun Microsystems. The Java Servlet API White Paper. Technical report, Sun Microsystems. <http://java.sun.com/products/servlet/whitepaper.html>. 159
- [131] Sun Microsystems. The JSAPI White Paper. Technical report, Sun Microsystems. <http://java.sun.com/products/java-media/speech/reference/whitepapers/index.html>. 20, 48, 160
- [132] Sun Microsystems. The Swing Architecture. [http://java.sun.com/products/jfc/tsc/archive/what\\_is\\_arch/swing-arch/swing-arch.html](http://java.sun.com/products/jfc/tsc/archive/what_is_arch/swing-arch/swing-arch.html). 15, 45

- [133] Yoshiyuki Mihara, Etsuya Shibayama, and Shin Takahashi. Migratory Cursor: Accurate Speech-Based Cursor Movement by Moving Multiple Ghost Cursors using Non-Verbal Vocalization. In *. of the seventh international ACM SIGACCESS conference on Computers and Accessibility*, pages 76–83, 2005. 9
- [134] Yoshiyuki Mihara, Shin Takahashi, and Etsuya Shibayama. WATARIDORI: Multiple Ghost Cursors for Speech-based Cursor Movement. In *ACM UIST '04 Companion*, pages 19–20, October 2004. 9
- [135] Wolfgang Minker, Udo Haiber, Paul Heisterkamp, and Sven Scheible. The SENECA Spoken Language Dialog System. *Speech Communication*, 43(1-2):89–102, 2004. 6
- [136] Melody M. Moore. Real-World Applications for Brain-Computer Interface Technology. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 11(2):162–165, 2003. 4
- [137] Brad A. Myers. A New Model for Handling Input. *ACM Transactions on Information Systems*, 8(3):289–320, 1990. 42, 44
- [138] Jakob Nielsen. Success Rate: The Simplest Usability Metric. *Jakob Nielsen's Alertbox*, February 2001.  
<http://www.useit.com/alertbox/20010218.html>. 68
- [139] Steve Northover. SWT Standard Widget Toolkit, Part 1. Technical report, Object Technology International, 2001.  
<http://www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html>. 156
- [140] Jan Nouza, Dana Nejedlova, Jindrich Zdansky, and Jan Kolorenc. Very Large Vocabulary Speech Recognition for Automatic Transcription of Czech Broadcast Programs. In *Proceedings of the International Conference on Spoken Language Processing ICSLP 2004*, pages 409–412, 2004. 7
- [141] D.R. Olsen, S. E. Hudson, T. Veratti, J. M. Heiner, and M. Phelps. Implementing Interface Attachments Based on Surface Representations,. In *Proceedings of CHI '99*, 1999. 153
- [142] A. Olwal and S. Feiner. Interaction Techniques Using Prosodic Features of Speech and Audio Localization. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*, pages 284–286, 2005. 9
- [143] P-Y. Oudeyer. The Production and Recognition of Emotions in Speech: Features and Algorithms. *International Journal of Human Computer Interaction*, 59(1-2):157–183, 2003. special issue on Affective Computing. 149

- 
- [144] K.A. Papineni, S. Roukos, , and R.T. Ward. Brain Computer Interface Cursor Measures for Motion-impaired and Able-bodied Users. In *Proc. Eurospeech 99*, pages 205–210, 1999. 4
- [145] K.A. Papineni, S. Roukos, , and R.T. Ward. Free-Flow Dialog Management Using Forms. In *Proc. Eurospeech 99*, pages 205–210, 1999. 134, 150
- [146] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes, 2nd Ed.*, pages 37–38. New York: McGraw-Hill, 1984. 53
- [147] W. Penny and S. Roberts. Experiments with an EEG-based Computer Interface. In *BCI Workshop, Albany, USA., June 1999.*, 1999. 4
- [148] Inc Perkins Engineering. The e-Belt from Hands Free Mobile.  
<http://www.handsfreemobile.com/ebelt.htm>. 4
- [149] Inc Perkins Engineering. The Hands Free Mobile Mid-Riff Brain.  
<http://www.handsfreemobile.com/midriffbrain.htm>. 4
- [150] Walter Pfrommer. Tracking my head – head trackers compared: A user’s perspective. *Closing the Gap*, page June/July, 2005. <http://www.closingthegap.com>. 5
- [151] Jeff Prosis. *Programming Windows with MFC*. Microsoft Press, 1999. 15, 45, 70, 156
- [152] Reed R. Voice Recognition for the Radiology Market. *Top Health Records Manag.*, 12(3):58–63, 1992. 30
- [153] Mohan R. Ramaswamy, Gregory Chaljub, Oliver Esch, Donald D. Fanning, and Eric vanSonnenberg. Continuous Speech Recognition in MR Imaging Rep. *American Journal of Roentgenology*, 174(3):617–622, 2000.  
<http://www.ajronline.org/cgi/content/full/174/3/617>. 8
- [154] Trygve Reenskaug. Thing-Model-View-Editor and Example from a Planningsysten. Technical report, XEROX PARC, 1979.  
<http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf>. 42
- [155] E. Reiter and R. Dale. Building Applied Natural Language Generation Systems. *Natural Language Engineering*, 3(1):57–87, 1997. 150
- [156] Rhetorical. Real Speak.  
<http://www.rhetorical.com>. 150
- [157] R. Rosenfeld, X. Zhu, A. Toth, S. Shriver, K. Lenzo, and A. Black. Towards a Universal Speech Interface. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP*, 2000. 131

- [158] Ronald Rosenfeld, Dan Olsen, and Alex Rudnicky. Universal Speech Interfaces. *interactions*, 8(6):34–44, 2001. 6
- [159] Ronald Rosenfeld, Xiaojin Zhu, Stefanie Shriver, Arthur Toth, Kevin Lenzo, and Alan W Black. Towards a Universal Speech Interface. In *Proceedings of the Sixth International Conference on Spoken Language Processing*, 2000. 6
- [160] A. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, Xu W., and A. Oh. Creating Natural Dialogs in the Carnegie Mellon Communicator System. In *Proceedings of Eurospeech*, pages 1531–1534, 1999. 5, 130, 135
- [161] M. D. Sadek and R De Mori. In *Spoken Dialogues with Computers*, chapter Dialogue systems, pages 523–561. Ed. Academic Press, London, 1998. 135
- [162] SALTForum. Speech Application Language Tags (SALT) Forum. <http://www.saltforum.org/>. 150, 152
- [163] A. Sears, M. Lin, and A. S. Karimullah. Speech-Based Cursor Control: Understanding The Effects of Target Size, Cursor Speed and Command Selection. *Universal Access in the Information Society*, 2(1):30–43, 2002. 9
- [164] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue. Galaxy-II: A Reference Architecture for Conversational System Development. In *Proc. ICSLP-98*, volume 3, pages 931–934, 1998. 135, 145
- [165] Stephanie Seneff and Joseph Polifroni. Dialog Management in the Mercury Flight Reservation System. In *Proceedings of ANLP/NAACL, Workshop on Conversational Systems*, 2000. 135
- [166] Hilit Serby, Elad Yom-Tov, and Gideon F. Inbar. An Improved P300-Based Brain Computer Interface. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 13(1):89–98, 2005. 4
- [167] Govind Seshadri. Understanding JavaServer Pages Model 2 Architecture. *Java World*, December 1999. available at <http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html>. 43
- [168] Alan Shalloway and James R. Trott. *Design Patterns Explained. A New Perspective on Object-Oriented Design*. Addison-Wesley Professional, 2004. 156
- [169] S. S. Shapiro and M. B. Wilk. An Analysis of Variance Test for Normality (complete samples). *Biometrika*, 52(3,4):591 – 611, 1965. 173
- [170] Chetan Sharma and Jeff Kunins. *VoiceXML: Professional Developer’s Guide with CDROM*. Wiley, 2001. 132



- [171] Stefanie Shriver, Roni Rosenfeld, Xiaojin Zhu, Arthur Toth, Alex Rudnicky, and Markus Flueckiger. Universalizing Speech: Notes from the USI Project. In *Proceedings of the Seventh European Conference on Speech Communication and Technology*, 2001. 6
- [172] Jane Siegel, Elaine Hyder, Jack Moffett, and Elise Nawrocki. IETM Usability: Using Empirical Studies to Improve Performance Aiding. Technical report, Computer Science Department, School of Computer Science, Carnegie Mellon University, May 2001.  
<http://reports-archive.adm.cs.cmu.edu/anon/2001/CMU-CS-01-131.pdf>. 4, 154
- [173] Anoop K. Sinha, Scott R. Klemmer, and James A. Landay. Embarking on Spoken-Language NL Interface Design. *International Journal of Speech Technology*, 5(2):159–169, 2002. 133
- [174] Christian Sipek. Framework for Speech-Enabled Custom User Interfaces on Mobile Devices. Master’s thesis, Technische Universität München, Institut für Informatik, 2002. 12, 83, 160
- [175] Asim Smailagic and Daniel P. Siewiorek. Modalities of interaction with CMU wearable computers. *IEEE Personal Communications*, 3(1):14 – 25, 1996. 49, 82, 132, 194
- [176] H. Soltau and A. Waibel. Specialized Acoustic Models for Hyperarticulated Speech. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2002. 149
- [177] SPIRIT. Automatic Speech Recognition.  
<http://www.spiritdsp.com>. 7, 33
- [178] Guillaume Belrose Stephen Hinde. Computer Pidgin Language: A New Language to Talk to Your Computer? Technical report, Internet Systems and Storage Laboratory, 2001. HPL-2001-182. 131
- [179] Rory Stuart and Gareth Gabrys. A speech compression proposal for directory assistance operators: Goms predictions. In *INTERACT '93 and CHI '93 conference companion on Human factors in computing systems*, pages 159 – 160, 1993. 35
- [180] R. Stuckless. Evaluation of the DragonDictate Discrete Word (speech) Recognition System for its Application with Deaf Adults. Technical report, Rochester, NY: National Technical Institute for the Deaf, 1996. 29
- [181] Inc Sun Microsystems. FreeTTS 1.2 - A speech synthesizer written entirely in the Java programming language.  
<http://freetts.sourceforge.net/docs/index.php>. 150

- [182] I.E. Sutherland. SketchPad: A Man-Machine Graphical Communication System. In *Proceedings of AFIPS Spring Joint Computer Conference*, pages 329–346, 1963. 2
- [183] Yoshiyuki Takada, Takao Miyahara, Tatsuya Tanaka, Takashi Ohyama, and Yoshio Nakamura. Modulation of H Reflex of Pretibial Muscles and Reciprocal Ia Inhibition of Soleus Muscle During Voluntary Teeth Clenching in Humans. *Journal of Neurophysiology*, 83(4):2063–2070, 2000. 4
- [184] Andrew S. Tanenbaum. *Computernetzwerke*. Prentice Hall, 1998. 157
- [185] Stefanie Tomko and Roni Rosenfeld. Shaping Spoken Input in User-Initiative Systems. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP '04*, 2004. 131
- [186] Stefanie Tomko and Roni Rosenfeld. Speech Graffiti vs. Natural Language: Assessing the User Experience. In *Proceeding of the Human Language Technology Conference / North American Chapter of the Association for Computational Linguistics , HLT/NAACL*, 2004. 6, 131
- [187] David Traum and Staffan Larsson. *Smith and Kuppevelt (eds.): Current and New Directions in Discourse & Dialogue*, chapter The Information State Approach to Dialogue Management, pages 325–353. Kluwer Academic Publishers, 2003. 134
- [188] Omer Tsimhoni, Paul Green, and Jennifer Lai. Listening to Natural and Synthesized Speech while Driving: Effects on User Performance. *International Journal of Speech Technology*, 5(2):159–169, 2002. 150
- [189] J. W. Tukey, B. P Bogert, and M. J. R. Healy. The Quefreny Analysis of Time Series for Echoes: Cepstrum, Pseudo-autocovariance, Cross-cepstrum, and Saphe cracking. In *Proceedings of the Symposium on Time Series Analysis (M. Rosenblatt, Ed)*, pages 209–243, 1963. 7
- [190] Christian Ullenboom. *Java ist auch eine Insel. Programmieren mit der Java Standard Edition Version 5*. Galileo Computing, 2006. previous edition available under <http://www.galileocomputing.de/openbook/javainsel4/>. 43, 70
- [191] Voxeo. Nuance VoiceXML Reference:usingso. <http://community.voxeo.com/vxml/docs/nuance20/usingSO.html>. 132
- [192] W3C. Speech Synthesis Markup Language (SSML) Version 1.0. <http://www.w3.org/TR/speech-synthesis/>. 150
- [193] M. Walker, L. Hirschman, and J. Aberdeen. Evaluation for DARPA Communicator Spoken Dialogue Systems. In *In Proc. Second International Conference on Language Resources and Evaluation, Athens, Greece., 2000*. 135

- [194] Xu Wei and Alexander I. Rudnicky. Task-based Dialog Management using an Agenda. In *ANLP/NAACL Workshop on Conversational Systems*, pages 42–47, 2000. 154
- [195] Greg Welch and Gary Bishop. Course 8 – An introduction to the Kalman filter. *SIG-GRAPH 2001 Courses*, 2001. 5
- [196] A.T Welford. *Fundamentals of Skill*. London: Methuen, 1968. 104
- [197] Detmer WM, Shiffman S, Wyatt JC, riedman CP, Lane CD, and Fagan LM. A Continuous-speech Interface to a Decision Support System, part 2: an Evaluation using a Wizard-of-Oz Experimental Paradigm. *J Am Med Inform Assoc*, 2:46–57, 1995. 30
- [198] C. Zinn, J. D. Moore, M. G. Core, S. Varges, and K. Porayska-Pomsta. The BE&E Tutorial Learning Environment (BEETLE). In *Proceedings of Diabruck, the 7th Workshop on the Semantics and Pragmatics of Dialogue*, August 2003. 136
- [199] V. Zue, S. Seneff, J. Glass, L. Hetherington, E. Hurley, H. Meng, C. Pao, J. Polifroni, R. Schloming, and P. Schmid. From Interface to Content: Translingual Access and Delivery of On-line Information. In *Proc. Eurospeech*, pages 2047–2050, August 1997. 135
- [200] John Zukowski. *The Definitive Guide to Java Swing*. Apress, 2005. 15, 70