

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

ÉCOLE POLYTECHNIQUE DE LOUVAIN

# **UsiGesture: A Structured Method for Engineering Pen-Based Gestures in Graphical User Interfaces**

## **PhD Thesis**

### **Supervisor**

Prof. Jean VANDERDONCKT (ILSM & ICTeam), Université catholique de Louvain

### **Jury**

Prof. Gaëlle CALVARY, LIG, Laboratoire d'Informatique de Grenoble, France (examiner)

Prof. Oscar PASTOR, Polytechnic University of Valencia, Spain (examiner)

Dr. Adrien COYETTE (ILSM), Université catholique de Louvain (reader)

Dr. Suzanne KIEFFER (ILSM), Université catholique de Louvain (reader)

### **President**

Prof. Christophe DE VLEESCHOUWER (ICTeam), Université catholique de Louvain

LOUVAIN-LA-NEUVE

October 2016



# Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Jean Vanderdonckt that supported me during my Ph.D. thesis. He brought all the knowledge, motivation and guidance helping me to accomplish this work.

Besides my advisor, I would like to warmly thank the rest of my thesis committee: Prof. Gaëlle Calvary, Prof. Oscar Pastor, Dr. Adrien Coyette and Dr. Suzanne Kieffer.

I thank the students that helped to create under my supervision the experiments presented in this work: Xavier De Coster, Charles de Groote, Arnaud Destiné, Pierre Deville, Laurent Lamouline, Thibault Leruitte, Vincent Nuttin, David Manuel Beza Quesada, Thomas Borlée, Antonio Ferreira Martinez, Pablo Guerrero Figueroa, Arnaud Pineux, Cédric Snauwaert, Steven An, Bastien Collet, Sylvain Duquenne, Laurent Genard, David Lebrun and Jonathan Moriau.

I also thank my fellow labmates for the useful advices, the stimulating discussions, and all the nice and fun moments spent together.

Last but not least, I thank my friends and family that spiritually supported me from the first day and throughout the whole creation process of this thesis.

The thesis was supported by:

- The UsiXML project standing for USer Interface eXtensible Markup Language. The ITEA2 European Project under contract number ITEA2-2008-0026. January 2010 - March 2013.
- The FEDER Compétitivité eHealth research project (ICTs supporting the patient), under convention n°2299.
- The FIRST Entreprise "outil générique" research project funded by DGO6, Région Wallonne, under convention n°7105.





# Table of contents

|          |                                               |           |
|----------|-----------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                           | <b>1</b>  |
| 1.1      | Context . . . . .                             | 1         |
| 1.2      | Thesis . . . . .                              | 4         |
| 1.2.1    | Motivations . . . . .                         | 4         |
| 1.2.2    | Statement . . . . .                           | 5         |
| 1.2.3    | Scope . . . . .                               | 5         |
| 1.2.4    | Research questions . . . . .                  | 7         |
| 1.2.5    | Contributions . . . . .                       | 7         |
| 1.3      | Reading map . . . . .                         | 8         |
| <b>2</b> | <b>State of the art</b>                       | <b>11</b> |
| 2.1      | Recognition mechanism preliminaries . . . . . | 11        |
| 2.1.1    | Gesture taxonomy . . . . .                    | 11        |
| 2.1.2    | Algorithm taxonomy . . . . .                  | 12        |
| 2.1.3    | Examples of applications . . . . .            | 14        |
| 2.2      | Algorithms . . . . .                          | 16        |
| 2.2.1    | Rubine . . . . .                              | 17        |
| 2.2.2    | One Dollar . . . . .                          | 25        |
| 2.2.3    | Levenshtein . . . . .                         | 33        |
| 2.2.4    | Stochastic Levenshtein . . . . .              | 42        |
| 2.2.5    | Extensions of Rubine . . . . .                | 55        |
| 2.2.6    | Extensions of One Dollar . . . . .            | 56        |
| 2.2.7    | Extensions of original Levenshtein . . . . .  | 57        |
| 2.3      | Models . . . . .                              | 59        |
| 2.3.1    | Cameleon Reference Framework . . . . .        | 60        |
| 2.3.2    | iGesture . . . . .                            | 62        |
| 2.4      | Approaches . . . . .                          | 62        |
| 2.5      | Tools . . . . .                               | 63        |
| 2.5.1    | iGesture . . . . .                            | 63        |
| 2.5.2    | quill . . . . .                               | 64        |
| 2.5.3    | DataManager . . . . .                         | 65        |

## Table of contents

|          |                                                                                                      |            |
|----------|------------------------------------------------------------------------------------------------------|------------|
| 2.5.4    | SOUSA . . . . .                                                                                      | 65         |
| 2.5.5    | SATIN . . . . .                                                                                      | 66         |
| 2.6      | Summary . . . . .                                                                                    | 67         |
| 2.6.1    | Shortcomings . . . . .                                                                               | 68         |
| 2.6.2    | Requirements . . . . .                                                                               | 69         |
| 2.6.3    | Conclusion . . . . .                                                                                 | 70         |
| <b>3</b> | <b>Selection of appropriate recognition mechanism</b>                                                | <b>73</b>  |
| 3.1      | Refined algorithms . . . . .                                                                         | 73         |
| 3.1.1    | Rubine . . . . .                                                                                     | 74         |
| 3.1.2    | One Dollar . . . . .                                                                                 | 74         |
| 3.1.3    | Original and stochastic Levenshtein . . . . .                                                        | 75         |
| 3.2      | Implementation . . . . .                                                                             | 76         |
| 3.2.1    | Tools overview . . . . .                                                                             | 76         |
| 3.2.2    | Software architecture . . . . .                                                                      | 82         |
| 3.2.3    | Software and hardware configuration . . . . .                                                        | 87         |
| 3.3      | Experimental study . . . . .                                                                         | 88         |
| 3.3.1    | Data acquisition . . . . .                                                                           | 88         |
| 3.3.2    | Methodology . . . . .                                                                                | 92         |
| 3.3.3    | Results . . . . .                                                                                    | 97         |
| 3.3.4    | Selection of the most appropriate algorithms . . . . .                                               | 120        |
| 3.3.5    | Recommendations for gesture sets creation . . . . .                                                  | 124        |
| 3.4      | Conclusion . . . . .                                                                                 | 126        |
| <b>4</b> | <b>Conceptual modelling of graphical user interfaces incorporating pen-based gesture recognition</b> | <b>127</b> |
| 4.1      | Introduction . . . . .                                                                               | 127        |
| 4.2      | Concrete User Interface . . . . .                                                                    | 128        |
| 4.3      | Gesture System Modeling . . . . .                                                                    | 130        |
| 4.4      | Gesture ontology . . . . .                                                                           | 131        |
| 4.5      | Conclusion . . . . .                                                                                 | 133        |
| <b>5</b> | <b>Approach and tool</b>                                                                             | <b>135</b> |
| 5.1      | Approach . . . . .                                                                                   | 135        |
| 5.2      | Model . . . . .                                                                                      | 140        |
| 5.2.1    | XWT Package . . . . .                                                                                | 142        |
| 5.2.2    | Behavior extension . . . . .                                                                         | 142        |
| 5.2.3    | Gesture Extension . . . . .                                                                          | 144        |
| 5.2.4    | Feeding and Using the System . . . . .                                                               | 144        |

## Table of contents

|          |                                       |            |
|----------|---------------------------------------|------------|
| 5.3      | Tool . . . . .                        | 145        |
| 5.3.1    | Using the Tool . . . . .              | 146        |
| 5.3.2    | Feeding the Tool . . . . .            | 147        |
| 5.4      | Example . . . . .                     | 149        |
| 5.5      | Conclusion . . . . .                  | 154        |
| <b>6</b> | <b>Validation</b>                     | <b>155</b> |
| 6.1      | Woke experiment . . . . .             | 155        |
| 6.1.1    | Methodology . . . . .                 | 155        |
| 6.1.2    | Results and discussion . . . . .      | 160        |
| 6.2      | Restaurant experiment . . . . .       | 162        |
| 6.2.1    | Methodology . . . . .                 | 162        |
| 6.2.2    | Results and discussion . . . . .      | 166        |
| 6.3      | Algorithms comparison . . . . .       | 167        |
| 6.3.1    | Methodology . . . . .                 | 167        |
| 6.3.2    | Results and discussion . . . . .      | 169        |
| 6.4      | Discussion and Conclusion . . . . .   | 173        |
| <b>7</b> | <b>Conclusion</b>                     | <b>175</b> |
| 7.1      | Summary of contributions . . . . .    | 175        |
| 7.1.1    | Conceptual . . . . .                  | 175        |
| 7.1.2    | Methodological . . . . .              | 176        |
| 7.1.3    | Implementational . . . . .            | 176        |
| 7.2      | Gestures pertinence . . . . .         | 177        |
| 7.3      | Limitations and future work . . . . . | 178        |
| 7.4      | Wrap up . . . . .                     | 179        |
|          | <b>Bibliography</b>                   | <b>188</b> |
| <b>I</b> | <b>Appendices</b>                     | <b>189</b> |
| <b>A</b> | <b>Publications</b>                   | <b>190</b> |
| <b>B</b> | <b>Gestures set</b>                   | <b>191</b> |
| B.1      | Letters . . . . .                     | 191        |
| B.2      | Digits . . . . .                      | 192        |
| B.3      | Action gestures . . . . .             | 192        |
| B.4      | Geometrical shapes . . . . .          | 193        |
| B.5      | Signatures . . . . .                  | 193        |

## Table of contents

|          |                                   |            |
|----------|-----------------------------------|------------|
| <b>C</b> | <b>Set of volunteers</b>          | <b>194</b> |
| <b>D</b> | <b>Meta-models details</b>        | <b>195</b> |
| D.1      | Concrete User Interface . . . . . | 195        |
| D.1.1    | Overview . . . . .                | 195        |
| D.1.2    | Main entities . . . . .           | 195        |
| D.1.3    | ConcreteGraphicalIU . . . . .     | 197        |
| D.1.4    | ConcreteStyle . . . . .           | 208        |
| D.1.5    | ConcreteListener . . . . .        | 210        |
| D.2      | Gesture System Modeling . . . . . | 216        |
| <b>E</b> | <b>Woke experiment models</b>     | <b>218</b> |

## List of Figures

|      |                                                                                                                                                                                                                                                                    |    |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1  | Contributions schema. . . . .                                                                                                                                                                                                                                      | 8  |
| 1.2  | Reading map. . . . .                                                                                                                                                                                                                                               | 9  |
| 2.1  | Some gestures recognized by the Mouse Gestures plug-in. . .                                                                                                                                                                                                        | 15 |
| 2.2  | Letters of the Graffiti alphabet. . . . .                                                                                                                                                                                                                          | 16 |
| 2.3  | Imposed data capture area. . . . .                                                                                                                                                                                                                                 | 16 |
| 2.4  | Features used to identify strokes. . . . .                                                                                                                                                                                                                         | 18 |
| 2.5  | Computation of the features of the Rubine's algorithm :<br>FEATURES_RUBINE( $g$ ). . . . .                                                                                                                                                                         | 23 |
| 2.6  | Learning step of the Rubine's algorithm :<br>LEARNING_RUBINE( $G$ ) . . . . .                                                                                                                                                                                      | 23 |
| 2.7  | Classification of a gesture with the Rubine's algorithm :<br>CLASSIFICATION_RUBINE( $g$ ). . . . .                                                                                                                                                                 | 24 |
| 2.8  | Example of disparities between similar gestures. . . . .                                                                                                                                                                                                           | 26 |
| 2.9  | A star gesture resampled to $N=32, 64$ , and $128$ points. . . . .                                                                                                                                                                                                 | 27 |
| 2.10 | Representation of the indicative angle. . . . .                                                                                                                                                                                                                    | 28 |
| 2.11 | Path-distance as a function of angular rotation away from the<br>$0^\circ$ indicative angle (centered y-axis). . . . .                                                                                                                                             | 29 |
| 2.12 | Preprocessing of a gesture for the One Dollar's algorithm :<br>PREPROCESSING_ONEDOLLAR( $g$ ). . . . .                                                                                                                                                             | 31 |
| 2.13 | Training of the One Dollar's algorithm :<br>TRAINING_ONEDOLLAR( $G$ ). . . . .                                                                                                                                                                                     | 31 |
| 2.14 | Classification of a gesture $g$ with the One Dollar's algorithm :<br>CLASSIFICATION_ONEDOLLAR( $g$ ). . . . .                                                                                                                                                      | 32 |
| 2.15 | Working of the Levenshtein's method. The compass rose de-<br>picted in (a) allows to describe a gesture as chain of characters.<br>In (b), a gesture corresponding to the digit 2. The correspond-<br>ing chain of character is "112222334455555555422222222". . . | 37 |
| 2.16 | Example of comparison of two encoded digits . . . . .                                                                                                                                                                                                              | 38 |
| 2.17 | Transformation of a gesture for the Levenshtein's algorithm :<br>TRANSFORMATION_LEVENSHTein( $g$ ). . . . .                                                                                                                                                        | 40 |

## List of Figures

|      |                                                                                                                  |    |
|------|------------------------------------------------------------------------------------------------------------------|----|
| 2.18 | Training of the Levenshtein's algorithm :<br>TRAINING_LEVENSHTein( $G$ ). . . . .                                | 40 |
| 2.19 | Classification of a gesture $g$ with the Levenshtein's algorithm :<br>CLASSIFICATION_LEVENSHTein( $g$ ). . . . . | 41 |
| 2.20 | The stochastic Levenshtein's algorithm : FORWARD-EVALUATE.                                                       | 50 |
| 2.21 | The stochastic Levenshtein's algorithm :<br>BACKWARD-EVALUATE. . . . .                                           | 51 |
| 2.22 | The stochastic Levenshtein's algorithm :<br>EXPECTATION-MAXIMIZATION. . . . .                                    | 52 |
| 2.23 | The stochastic Levenshtein's algorithm : EXPECTATION-STEP.                                                       | 53 |
| 2.24 | The stochastic Levenshtein's algorithm : MAXIMIZATION-STEP.                                                      | 54 |
| 2.25 | The 8 possibilities for a two-stroke $x$ . The numbered dots indicate stroke order and beginnings. . . . .       | 57 |
| 2.26 | The 8 unistroke permutations for a two-stroke $x$ based on the two-stroke gestures in Figure 2.25. . . . .       | 57 |
| 2.27 | Cameleon Reference Framework. . . . .                                                                            | 61 |
| 2.28 | iGesture user interface. . . . .                                                                                 | 62 |
| 2.29 | iGesture user interface. . . . .                                                                                 | 63 |
| 2.30 | quill user interface. . . . .                                                                                    | 64 |
| 2.31 | DataManager user interface. . . . .                                                                              | 65 |
| 2.32 | Sousa user interface. . . . .                                                                                    | 66 |
| 2.33 | DENIM application. . . . .                                                                                       | 67 |
| 3.1  | Training tab. . . . .                                                                                            | 77 |
| 3.2  | One-by-one example adder tab. . . . .                                                                            | 78 |
| 3.3  | Recognizer tab. . . . .                                                                                          | 79 |
| 3.4  | Windows for algorithm settings. . . . .                                                                          | 79 |
| 3.5  | Customization of the choice of the gesture classes. . . . .                                                      | 81 |
| 3.6  | Window for training options. . . . .                                                                             | 82 |
| 3.7  | Diagram of classes of package <i>algorithm</i> . . . . .                                                         | 83 |
| 3.8  | Diagram of classes for the graphical interface. . . . .                                                          | 84 |
| 3.9  | Diagram of classes for the tests on original algorithms. . . . .                                                 | 86 |
| 3.10 | Diagram of classes for the tests on refined algorithms. . . . .                                                  | 86 |
| 3.11 | Graphic tablet Wacom Intuos3 <sup>®</sup> A4. . . . .                                                            | 87 |
| 3.12 | Configuration of the Wacom driver. . . . .                                                                       | 88 |
| 3.13 | Examples of letters templates presented to the volunteers. . . . .                                               | 90 |
| 3.14 | Examples of action gestures templates presented to the volunteers. . . . .                                       | 90 |
| 3.15 | Environment of the data acquisition. . . . .                                                                     | 91 |

## List of Figures

|      |                                                                                                                                                                              |     |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 3.16 | Results of the analysis by training type for Rubine. . . . .                                                                                                                 | 97  |
| 3.17 | Results of the analysis by training type for OneDollar. . . . .                                                                                                              | 99  |
| 3.18 | Results of the analysis by training type for Levenshtein. . . . .                                                                                                            | 100 |
| 3.19 | Results of the analysis by training type for stochastic Levenshtein. . . . .                                                                                                 | 101 |
| 3.20 | Results of the analysis by family of classes for a training made dependently of the user. . . . .                                                                            | 103 |
| 3.21 | Results of the analysis by family of classes for a training made independently of the user (first method). . . . .                                                           | 104 |
| 3.22 | Results of the analysis by family of classes for a training made independently of the user (second method). . . . .                                                          | 106 |
| 3.23 | Results of the analysis by algorithm for the digits. . . . .                                                                                                                 | 107 |
| 3.24 | Results of the analysis by algorithm for the letters. . . . .                                                                                                                | 108 |
| 3.25 | Results of the analysis by algorithm for the action gestures. . .                                                                                                            | 109 |
| 3.26 | Results of the analysis by algorithm for the geometrical shapes.                                                                                                             | 110 |
| 3.27 | Results for the signatures. . . . .                                                                                                                                          | 111 |
| 3.28 | Results for the execution times. . . . .                                                                                                                                     | 112 |
| 3.29 | Results for the preprocessing of Rubine. . . . .                                                                                                                             | 113 |
| 3.30 | Results for $k$ NN of One Dollar. . . . .                                                                                                                                    | 114 |
| 3.31 | Results for the extensions of preprocessing, integration of new features, consideration of adapted costs and consideration of multi-strokes gestures of Levenshtein. . . . . | 115 |
| 3.32 | Results for the normalizations of Levenshtein. . . . .                                                                                                                       | 116 |
| 3.33 | Results for $k$ NN of Levenshtein. . . . .                                                                                                                                   | 117 |
| 3.34 | Results for the extensions of preprocessing, integration of new features and consideration of multi-strokes gestures of stochastic Levenshtein. . . . .                      | 118 |
| 3.35 | Results for the normalizations of stochastic Levenshtein. . . .                                                                                                              | 119 |
| 3.36 | Results for $k$ NN of stochastic Levenshtein. . . . .                                                                                                                        | 120 |
| 4.1  | CUI Overview. . . . .                                                                                                                                                        | 128 |
| 4.2  | CUI ConcreteGraphicalIU. . . . .                                                                                                                                             | 129 |
| 4.3  | Gesture system modeling. . . . .                                                                                                                                             | 130 |
| 4.4  | Gesture ontology. . . . .                                                                                                                                                    | 132 |
| 5.1  | Method. . . . .                                                                                                                                                              | 139 |
| 5.2  | Model. . . . .                                                                                                                                                               | 141 |
| 5.3  | Tool. . . . .                                                                                                                                                                | 146 |
| 5.4  | Example: first version. . . . .                                                                                                                                              | 151 |

## List of Figures

|      |                                                       |     |
|------|-------------------------------------------------------|-----|
| 5.5  | Example: second version. . . . .                      | 152 |
| 5.6  | Example: third version. . . . .                       | 153 |
| 6.1  | Woke experiment: participants statistics. . . . .     | 157 |
| 6.2  | Woke experiment: participants gender shares. . . . .  | 157 |
| 6.3  | Woke experiment: participants jobs shares. . . . .    | 158 |
| 6.4  | Woke experiment: general interface. . . . .           | 159 |
| 6.5  | Woke experiment: order interface. . . . .             | 160 |
| 6.6  | Woke experiment: classical interface results. . . . . | 160 |
| 6.7  | Woke experiment: gestural interface results. . . . .  | 161 |
| 6.8  | Restaurant experiment: drinks. . . . .                | 163 |
| 6.9  | Restaurant experiment: food. . . . .                  | 164 |
| 6.10 | Restaurant experiment: gesture set. . . . .           | 164 |
| 6.11 | Restaurant experiment: gestures handling. . . . .     | 165 |
| 6.12 | Algorithms comparison: geometrical shapes. . . . .    | 170 |
| 6.13 | Algorithms comparison: arrows. . . . .                | 171 |
| 6.14 | Algorithms comparison: lowercase letters. . . . .     | 171 |
| 6.15 | Algorithms comparison: uppercase letters. . . . .     | 172 |
| D.1  | CUI Overview. . . . .                                 | 195 |
| D.2  | CUI main entities. . . . .                            | 196 |
| D.3  | CUI ConcreteGraphicalIU. . . . .                      | 198 |
| D.4  | CUI ConcreteStyle. . . . .                            | 207 |
| D.5  | CUI ConcreteListener. . . . .                         | 211 |
| D.6  | Gesture system modeling. . . . .                      | 216 |
| E.1  | Woke experiment task model. . . . .                   | 218 |
| E.2  | Woke experiment AUI model. . . . .                    | 218 |



# List of Tables

|     |                                                                                                                                                           |     |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 2.1 | Evolution of the distance of Levenshtein by comparing the string $x = \textit{lire}$ to the string $y = \textit{livre}$ . . . . .                         | 36  |
| 2.2 | Computation of the distance of Levenshtein by comparing the string of the digit 2 : “122355552222” to the string of the digit 8 : “123555321777”. . . . . | 39  |
| 2.3 | Matrix of adapted costs of substitution. . . . .                                                                                                          | 59  |
| 2.4 | Comparison of the existing methods. . . . .                                                                                                               | 68  |
| 3.1 | Original algorithms summary. . . . .                                                                                                                      | 122 |
| 3.2 | Refined algorithms summary. . . . .                                                                                                                       | 125 |
| 6.1 | Restaurant experiments: results. . . . .                                                                                                                  | 167 |
| C.1 | Table summarizing the characteristics of the volunteers. . . . .                                                                                          | 194 |



# Chapter 1 Introduction

This chapter introduces the context of the research and sets the frame of this work: the motivations leading to it, the formalization of its statement and how it is scoped. The underlying research questions are then given followed by the corresponding expected contributions. It finally explains how the thesis is structured.

## 1.1 Context

For all-time, human beings have used gestures in order to make them understandable. These gestures can be various and of different kinds. If the arms and hands are certainly the most frequently used parts of the body to indicate for example a direction, an agreement or disagreement, and so on, the head of the speaker can be turned to someone to indicate who is the listener, the eyes and the mouth can be used as feeling indicator (e.g. happy, sad, angry, ...), etc. But people even use gesture when they know nobody can see them (e.g. on the phone). In their work (van den Hoven and Mazalek [2011]), Elise van den Hoven and Ali Mazalek also describe the gesture also as bringing support beyond the communication itself, since it can facilitate remembering and lighten cognitive loads.

Up to now, lot of efforts have been produced to study the gestures in different fields, including psychology, anthropology and linguistics. Several researchers have tried to define and classify the gestures. One of the first was Efron (Efron [1941]) who identified three phases decomposing the gesture : the first one is the preparation phase occurring just before the actual gesture and corresponding to the move of the hands to the initial location ; the second phase is the stroke phase corresponding to the actual gesture ; the third and last phase is the retraction phase occurring just after the actual gesture and corresponding to the move

## 1 Introduction

of the hands to an undefined position. Additionally, Efron highlighted five different categories for the gesture : physiographics corresponding to a figural representation of an object ; kinetographics corresponding to the representation of an action ; ideographics corresponding to a support of the ideas in the head of the speaker ; deictics corresponding to pointing gestures ; batons correspondings to speech markers. After him, others have continued the investigations like Kendon (Kendon [1988]) who divided the gestures into two categories, depending on the speech or not. Quek (Quek [1995]) made the distinction between intentional gestures (conveying information) and unintentional gestures (conveying no useful information). Grosjean and Kerbrat-Orecchioni (Grosjean and Kerbrat-Orecchioni [1999]) brought the notion of actions that have a practical function and that can be distinguished from the gesture since gestures do not modify the physical world but only carry meaning.

Another classification, belonging to the field of Human-Computer Interaction (HCI), is given by Quek (Quek [1995]) — and then improved in Pavlovic et al. [1997] — who identified two categories : communicative and manipulative gestures. Manipulative gestures act on physical artifacts in the environment while communicative ones are useful for communication. This notion can be linked to the classification introduced by Grosjean and Kerbrat-Orecchioni and is part of the field in which this work will investigate. Gestures handling is a subject widely studied in HCI, for interacting and communicating with computational systems purposes. In order to allow these systems to deal with the gestures, two phases are required. The gesture must be computed and interpreted but before it has to be acquired. For that reason, different means of detection of the gesture are possible, depending on the approach considered.

The most difficult environment to capture gestures is in the 3-dimensional free space. If it could represent the most natural way to interact with interactive systems, it is in practice really difficult to handle the gestures in a suitable way. Different ways of dealing with the gestures have already been investigated like using glove devices or video tracking, but several issues remain unsolved from tracking the position and movement of the hand, to modeling, analyzing and recognizing the gesture (van den Hoven and Mazalek [2011]). Another way of dealing with gestures is interaction with physical artifacts. More and more attention is paid to this field, in particular for commercial applications like Nintendo's Wii Remote that takes advantage of remote controllers in order to fit as well as possible the human gestures. In addition to game controllers, we can hold up as examples the gestures with mobile devices (e.g. by squeezing the mobile device to launch some command) or gestures with batons and wands,

## 1 Introduction

etc.

In order to circumvent the problem of sensing or tracking the gestures, many researches have been produced on 2-dimensional surface. Indeed, the big advantage of the surface is to constrain the gesture by eliminating one dimension. Additionally, the beginning and the end of the gesture are easily detectable since they usually correspond to the beginning and the end of the stroke. However, all the technical issues are not resolved and it still remains some difficulties to distinguish for example a command from an ink stroke. Solutions exist but are not always totally convenient.

2-dimensional gestures certainly represent the most explored way of capturing gestures in HCI, from the first applications based on stylus interaction to the current ones allowing to use the fingers (e.g. the iPhone applications). In comparison with the fingers, the pen or stylus has the advantage of higher precision because of its small tip. One of the first well-known examples of 2D gesture recognition for pen-based digital interaction, coming from the 1960s, is the SketchPad application (Sutherland [1963]) which is a tool for computer-assisted conception. It is considered as the pioneer of the Graphical User Interface and the object-oriented programming. Another well-known language is GRAIL (Graphical Input Language) adapted for one of the first graphics tablet, the RAND Tablet (Ellis et al. [1969]). In the 1980s, researchers of the University of Toronto used gestures captured through graphics tablet to allow the use of commands in drawing program (Buxton et al. [1983]) or in the musical field to assist a composer in the editing of a file of information detailing a musical score (Buxton et al. [1985]). Researchers at IBM also used pen-based gesture input for editing operations in different applications like a drawing tool, a spreadsheet, a music editor or an equation editor (Wolf et al. [1989]). In 1991, Rubine presented one of the first published algorithm for the recognition of mouse and pen-based gestures, used in its toolkit GRANDMA allowing to rapidly adding gestures to direct manipulation interfaces (Rubine [1991]). Then the arrival of the personal digital assistants (PDAs) contributed to the popularization of pen-based gesture interaction. One of the first ones was the Apple Newton MessagePad in 1992, followed a few years later by Palm supported by the recognition system Graffiti (Mackenzie and Zhang [1997]). Gesture interaction on wider displays was also investigated with pen-based whiteboard systems such as the Tivoli application on the Xerox LiveBoard (T. P. Moran [1997]) providing interaction techniques that enable groups of users in informal meetings to easily organize and rearrange material and to manage the space on the board. In 2004 was presented a new interaction

technique, Stitching (Hinckley et al. [2004]), allowing users to combine pen-operated mobile devices, with pen gestures able to span multiple displays.

## 1.2 Thesis

### 1.2.1 Motivations

In van den Hoven and Mazalek [2011], we can see that gestures play an important role in communication by improving and facilitating it. They support the listener, who is trying to understand the speaker but they also support the speaker, by facilitating the conceptualization and verbalization of messages and reducing cognitive load. Several examples illustrate this assertion like the article of Graham and Argyle (Graham and Argyle [1975]) showing that speakers who are restricted in their gesturing use more words and give poorer directions, or the article of Rime and Schiaratura (Rime and Shiaratura [1991]) underlying the correlation between the reduction of the vividness of imagery and the restriction of the gestures. In human computer interaction, gestures are also used to facilitate communication with digital applications, since their expressive nature can enable less constraining and more intuitive digital interactions than conventional user interfaces.

As explained in Hong et al. [2002], three types of persons are involved in such applications : the end-user, that accomplishes some task with the technology; the designer, that creates the interface, evaluates it and iteratively improves it ; the programmer, that writes good code and debugs it. Considering the gestures, the usability and the powerfulness would be improved for the end-user by incorporating them. Nevertheless, the difficulty met by the programmer is to know what is the best way and the best conditions to recognize them to incorporate in the user interface. Because a lot of possibilities exist, several tools provide support for testing gesture recognition algorithms. Certain of them even allow to produce complete comparisons between these algorithms. Other tools help the designer for the creation of user interfaces. However, to our knowledge, none of them offers the possibility to design interfaces directly integrating the use of gestures, acting as decision provider for the programmer in order to help him to choose the best way to incorporate gestures in applications and allowing the designer to create user interfaces.

## 1 Introduction

Such a tool would require a strong underlying modelling in order to let engineers, programmers and designers create user interfaces incorporating pen-based gestures in a consistent and powerful way. Additionally, good modelling confers understandability and better extendibility. To our knowledge, very few tools are supported (at least publically) by a strong set of models. iGesture (Signer et al. [2007]) gives an interesting representation for the recognition mechanism, but not linked to a strong modellisation for graphical user interfaces.

A tool alone is often not sufficient to be used in an appropriate way. Guidance is most of time required in order to exploit the tool at its full capacities. To our knowledge, no tool is supported and structured by a strong and systematic approach that would allow the different roles involved the creation of user interfaces incorporating pen-based gestures to smoothly collaborate around the proposed tool.

Regarding the importance of the gestures, creating a method, a framework, and modelling helping the engineers, programmers and the designers in their task of producing user interfaces incorporating gestures would constitute an interesting way of investigation.

### 1.2.2 Statement

In order to provide designers and developers with methodological guidance for incorporating pen-based gestures into graphical user interfaces, a structured method is proposed that consists of: a Concrete User Interface model supporting gestures, a step-wise approach based on this model, and a supporting software environment.

### 1.2.3 Scope

Lots of different aspects belong to the field of gesture recognition and user interface design. The thesis focusses on some of them :

- The primary goal of the thesis is to provide easy ways to create UIs incorporating pen-based gestures through a set of tools supporting approaches and models. The targeted field the thesis belongs to is the **Engineering of Interactive Systems**.

## 1 Introduction

- User interfaces can be concretized in several ways called modalities. A modality represents the way the user interacts with the application. It can be graphical, vocal, haptic, etc. During this thesis, the efforts will be concentrated on the integration of the gestures in the most common modality : the **graphical modality**. Since this modality is probably the most frequently used on most types of interactive systems, software based or not, only Graphical User Interfaces will be considered from now on. Incorporation of gestures in multimodal user interfaces is another goal.
- Gesture interaction is an interaction modality in which gesture is the primary way to interact with a computer based system, provided that gesture recognition is efficient (Wikipedia). Gesture interaction can occur :
  - on 2D surfaces (e.g., tablet pc, graphic tablet, tactile screens, multi-touch environment) or in 3D spaces (e.g., virtual reality, augmented reality) ;
  - with different input devices (e.g., stylus, pen, 3D mouse, phantoms, glove) ;
  - with or without hand gestures ;
  - with or without fingers;
  - etc.

We hereby define **pen-based gestures on 2D surfaces** as the specialization of gesture interaction on any 2D surface with a pen as input device, without using any finger or hand, for a single user (i.e. in a non collaborative task). The scope of this thesis is delineated by this. We also do not consider here handwriting and signature recognition.

- Several roles are involved in the whole life cycle of user interfaces. The thesis specifically targets to help the **engineers**, the **programmers** and **designers**. The end users takes an active part in this life cycle and are considered at some point in the thesis, but the different contributions do not specifically apply to them.
- The goal of the thesis is to **facilitate the process of user interfaces creation** and not to prove the quality of these interfaces will be better (even if it is an expected consequence).



### 1.2.4 Research questions

The thesis is aimed at providing guidance around the creation process of user interfaces integrating pen-based gestures recognition. This task can be very time-consuming and difficult depending on the background of the person(s) in charge of the creation, leading to different underlying research questions.

A first question is to understand the concepts relative to a user interface integrating gestures: the different components required to achieve this interface and the way they interact with each other. We are hereby interested to provide a way to document and easily describe such interfaces.

A second question concerns the interaction between the different protagonists implicated in the process of creation of such user interfaces: the different roles required and how they can coordinate to achieve this task in the most efficient way. We are hereby interested to elaborate a systematic approach providing these protagonists with a canvas to follow and adapt depending on their specific needs, allowing them to enact a fruitful collaboration.

The third question is related to the concrete achievement of the user interface integrating pen-based gestures following the pre-determined concepts and approach. We are hereby interested to propose an effective tool allowing to implement the concepts and approach proposed in the global method.

### 1.2.5 Contributions

Following the research question, the aimed contributions can be divided in three main categories.

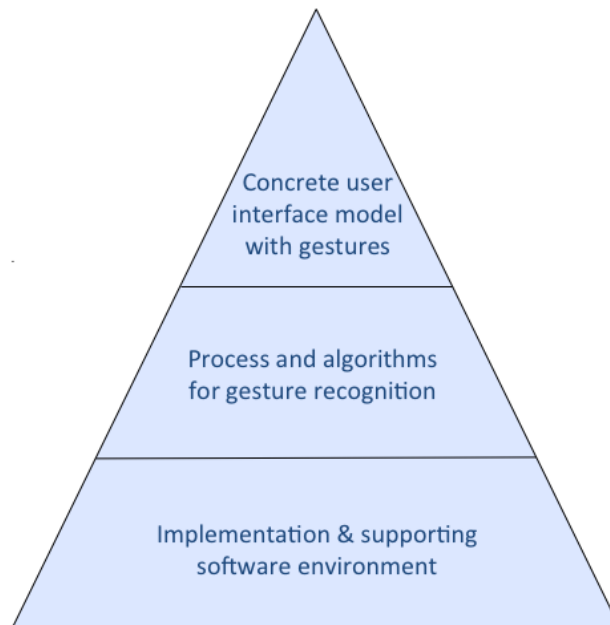
On a conceptual point of view, a strong modelling must be proposed to describe any user interface integrating pen-based gestures recognition. The goal is to provide the different creators with the ability to deeply understand what is the composition of such user interfaces and how the different components work together to form the targeted user interface.

On a methodological point of view, the thesis is aimed at providing a structured approach to support the different steps involved in the process of creation of pen-based user interfaces. The objective is to formally state these steps and propose a clear delimitation of the roles with the interaction between them.

## 1 Introduction

On an implementational point of view, the goal of this thesis is to provide a tool supporting different protagonists in the whole process of creating user interfaces incorporating pen-based gesture recognition, in adequation with the proposed modelling and approach. This tool should be powerful and extensible in order to support any needs not addressed yet. In addition, it must be very easy to use for people not necessarily skilled in programming.

Figure 1.1 gives a schema of these contributions.



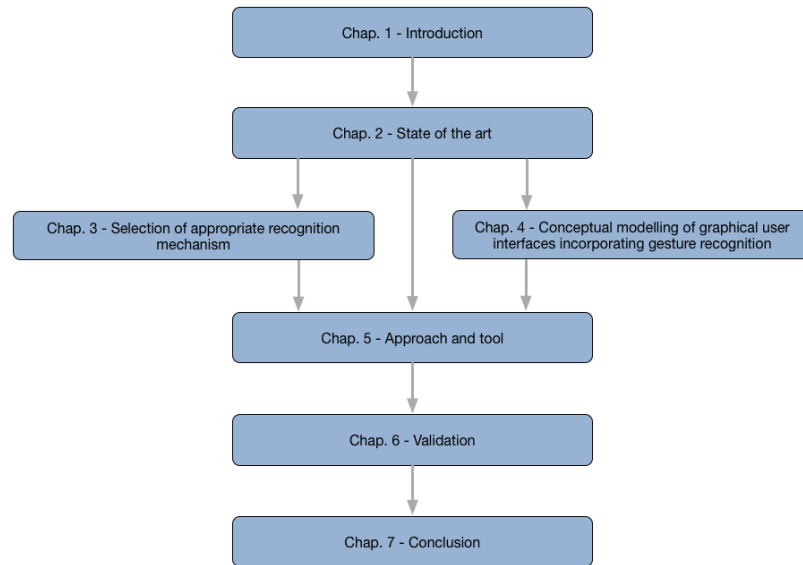
**Figure 1.1** Contributions schema.

### 1.3 Reading map

Figure 1.2 depicts the structure of this thesis. It gives an high-level overview on the way the different chapters are articulated with their dependencies.

**Chapter 1** introduces the context of the research and sets the frame of this work: the motivations leading to it, the formalization of its statement and how it is scoped. The underlying research questions are then given followed by the corresponding expected contributions. It finally explains how the thesis is structured.

## 1 Introduction



**Figure 1.2** Reading map.

Based on the research questions, **Chapter 2** presents the state of the art related to this research. Some thoughts are first shared about the recognition mechanisms necessary behind user interfaces integrating gesture recognition. As part of these mechanisms, some existing algorithms along with possible extensions are explicated. Developers can implement these recognition mechanisms in user interfaces from scratch, or rely on existing assets including modelling, methodologies and tools. Different pieces of such assets are then described in the state of the art. The chapter is concluded with a comparison of the existing tools regarding the elicited research questions, leading to shortcomings and requirements.

**Chapter 3** discusses new original extensions of the existing algorithms described in the state of the art. It then gives an overview of the tool created to perform comparative studies to select the best recognition mechanisms depending on the requirements of the application. This tool includes different implementations of algorithms and extensions, as well as gestures datasets. An experimental study led with the help of this platform is then described, allowing to acquire gestures and select the most appropriate algorithms depending on different factors.

**Chapter 4** shows the models elaborated in this thesis. The first one allows

## 1 Introduction

to describe any graphical user interface at the concrete level. The second one allows to extend the first model to describe the components necessary to add gesture recognition capabilities to the interface.

**Chapter 5** brings an approach specific for the creation of user interfaces integrating gesture recognition. It then describes a tool elaborated to actually create this kind of user interfaces in an easy way. This tool supports the specific approach, is based on a specific instantiation of the models presented in chapter 4 and integrates the recognition mechanisms presented in chapter 3. This chapter finally shows through a step-by-step example how to use this tool.

**Chapter 6** gives some elements of validation. It does not stand as an ultimate evidence for the improvements of the produced user interfaces (even if it is an expected result). Instead, the chapter explains two different experiments showing how the created tool, approach and models can help during the process of creating user interfaces integrating gesture recognition.

**Chapter 7** concludes by summarizing the different contributions brought through this work and discusses the potential improvements and future work.

## Chapter 2 State of the art

The previous chapter allowed to list the research questions and aimed contributions. This chapter details a review of the corresponding related work. It first presents the choices made for the recognition mechanisms. The selected algorithms are then detailed, before a review of the existing tools, approaches and models allowing creators of user interface to incorporate pen-based gestures in their interfaces. It is concluded by describing the shortcomings identified in this review and their corresponding requirements.

### 2.1 Recognition mechanism preliminaries

In this section we explain the choices made for the two main components of the recognition mechanisms: the gestures and the algorithms. We then show some direct applications that can be made of this recognition applied to user interfaces incorporating pen-based gesture recognition.

#### 2.1.1 Gesture taxonomy

Gestures can be defined as movements. For example, movements of the upper limbs or the head allow to execute tasks or express emotional states. Cadoz and Wanderley (Cadoz and Wanderley [2000]) characterized a gesture by three major functions: (i) an ergotic function, which could modify the environment on which the gesture is applied; (ii) an epistemic function, which affects the perception of this environment, and (iii) a semiotic function, which is assumed to communicate information. Any gesture could assume one to three of these functions.

Several kinds of gesture types exist and are potentially interesting. During this study, only the handwritten movements will be considered. In these category

## 2 State of the art

of gesture, it is possible to distinguish different types of symbols that can be categorized into families :

- letters and digits : lower case or capital letters of the alphabet, letters with or without accent, etc. ;
- action gestures : segment of lines drawn in a particular direction, broken line, etc. ;
- geometrical shapes : square, circle, etc. but also doubled geometrical shapes ;
- signatures.

Furthermore, handwritten gestures can be realized with several strokes.

In this thesis, two big classes of gestures are considered. The first ones are the gestures already sketched and contained in a database. They constitute the core of the training and will be called training gestures or learning gestures. The second class of gestures are the gestures we want to classify. They will be called input gestures or gestures to recognize.

### 2.1.2 Algorithm taxonomy

Several different types of gesture recognition algorithms exist. Some of them are only based on geometric features while other ones takes into account the dynamic of the gesture. The way the data are processed varies from an algorithm to another.

The purpose is to firstly develop a gesture recognition platform in order to evaluate certain algorithms and the impact of some characteristics on the classification. The first step is thus to choose the algorithms to be tested.

The first algorithm chosen is the algorithm developed by Dean Rubine (Rubine [1991]). This algorithm has been widely used in different applications (e.g., Jr. et al. [1999], Chatty and Lecoanet [1996]). It is based on geometric and dynamic features of the gestures. For each of the learning gesture classes, a mean feature vector is computed from which are derived weights. For a new input gesture, scores can be obtained for each class by combining its own feature vector with the weights for the considered class. The input gesture is

## 2 State of the art

classified by choosing the class obtaining the best score. The simplicity of the algorithm, its theoretical performances and its celebrity are the main arguments for choosing it.

A second type of algorithm, only based on geometrical features, is the algorithm of Levenshtein (Levenshtein [1966], [Gusfield, 1997, Chap. 11, p. 216], Coyette et al. [2007], Soukoref and MacKenzie [2001]). It differs from the Rubine's algorithm in the way they process the geometrical features. In the Levenshtein's algorithm, the geometrical features are not considered as themselves but are rather extracted from the point-by-point directions of the gesture. More precisely, each gesture is represented by a chain of character built with symbols associated with the directions. The Levenshtein's algorithm allows then to compare the chain of characters of the input gesture to each chain of characters of the training gestures. The classification is realized by choosing the class of the closest gesture found by the algorithm.

In the field of algorithms based on geometrical properties of the gestures, it also exists the One Dollar's algorithm (Wobbrock et al. [2007]). This algorithm is presented as being simple to implement, cheap (do not need a huge number of training examples) and giving good results. The algorithm starts by executing some preprocessing on the training gestures and the input gesture. The first step of the transformation is a resampling, a rotation, a rescaling, and a translation. The input gesture can then be compared to each training gesture by firstly rotating (second step of the transformation) and then scoring it. The score simply corresponds to the distance between the two gestures. The class attributed is thus the class of the closest gesture found by the algorithm.

These three selected algorithms are different but are all mainly directly based on geometric and dynamic features (it is not totally true for the Rubine's algorithm which use a statistical method in order to compute the weights during the training phase). It should be interesting to compare them to an algorithm totally based on a statistical method. Such an approach can be provided by artificial neural networks ([Mitchell, 1997, chap. 4]). This technique has already been studied (Goh et al. [1997]) and gave good results. However, it is reasonably not possible to use this kind of algorithm since the required number of learning data and the announced time of execution are too huge. Another algorithm must thus be explored.

The Levenshtein's algorithm uses chains of characters in order to represent the gestures and their point-by-point directions. With this kind of chains, it is possible to applicate the hidden Markov models (Durbin et al. [1998], Artieres

## 2 State of the art

et al. [2007], Okumur et al. [2005]). Following this idea leads to the stochastic Levenshtein's algorithm. Without going into the details, this algorithm allows to improve the initial version of the Levenshtein's algorithm by automatically learning some weights rather than considering fixed ones. The method is totally based on statistics, with a kind of one-state hidden Markov model and has been firstly presented for the word pronunciation learning (Ristad and Yianilos [1998]) and later for the writing recognition (Oncina and Sebban [2006]). This algorithm only based on statistics is thus chosen for its statistical properties but also because it is a natural improvement of the initial version of the Levenshtein's algorithm, which is interesting to consider.

### 2.1.3 Examples of applications

#### Mouse Gestures

Mouse Gestures is a plug-in for the web browser Mozilla Firefox<sup>®</sup>. It also exists for all the browsers of the Mozilla<sup>®</sup> suite and for the mail client Mozilla Thunderbird<sup>®</sup>.

Mouse Gestures allows to execute commands in a very simple and quick way, without using keyboard, menus or tool bars. The user holds one of the mouse buttons (generally the right one) and move the mouse in order to create a gesture. If the gesture is recognized, the action associated is executed. It improves the comfort of navigation since the user can initiate a huger panel of actions by only using its mouse.

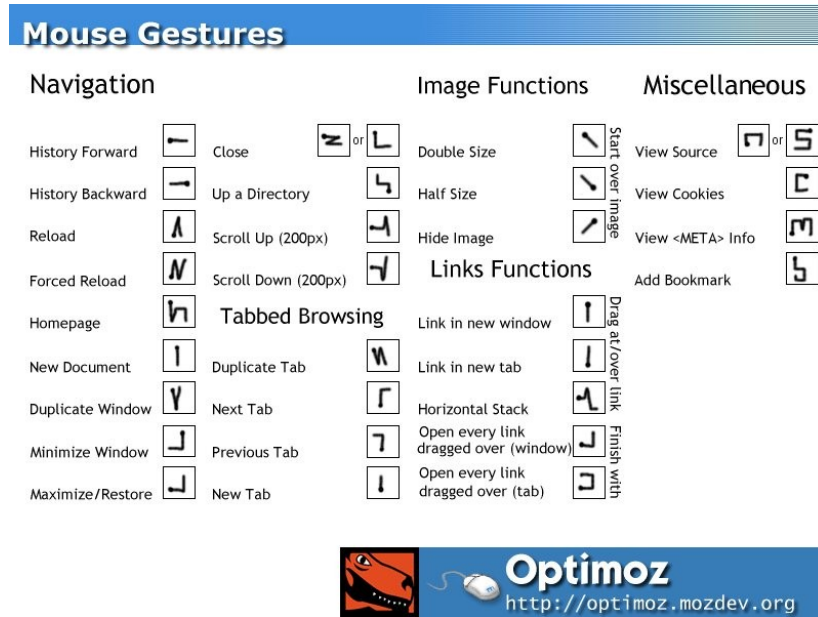
Some gestures natively recognized by the application are presented in Figure 2.1. The whole set of gestures taken into account by Mouse Gestures is available at the address [Mozilla].

#### SketchiXML

SketchiXML (Coyette et al. [2004]) is a multi-agent design tool for sketching user interfaces based on the UIDL (User Interface Description Language) UsiXML [UsiXML]. It allows a designer to sketch a user interface at different levels of details and provides support for different contexts of use.



## 2 State of the art



**Figure 2.1** Some gestures recognized by the Mouse Gestures plug-in.

SketchiXML is conceived to provide informal specifications of the interface and improve the dialog between designer and final user.

The behaviour of the application can be defined with a set of parameters. For example, the user can define if a recognized shape should be displayed differently, or can define the required quality.

When the design is finished, it is analyzed in order to produce the specification of the interface independently of some context, including the platform and the user. These specifications are used to progressively derive one or several interfaces, or one or several users, platforms and environments.

The next step in the elaboration process consists in importing the generated specifications in a high-fidelity editor like GrafiXML (Michotte and Vanderdonckt [2008]). Using this kind of tool allows to add attributes that could not be specified during the sketching phase.

### Graffiti

Graffiti is an alphabet used in the PDAs industry, particularly by Palm [Palm]. The alphabet, whose the "letters" part is depicted in Figure 2.2, allows the users

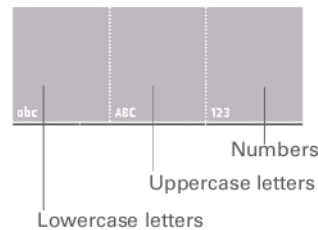
## 2 State of the art

to write in a natural way on their PDAs. Each symbol is associated with a letter of the Roman alphabet and is composed — in the first version of Graffiti — of a unique stroke. The user must thus learn this alphabet but this task seems not too hard since the symbols look alike : after a training of five minutes, a recognition rate of 97% is reached (Mackenzie and Zhang [1997]).



**Figure 2.2** Letters of the Graffiti alphabet.

Some particular areas of the PDAs screen are devoted to the capture of these gestures, generally the bottom. Depending the part of the area used, different characters can be recognized. For example, in extended mode, the left part of the area is assigned for the lower case letter recognition, the middle part for the capital letters, and the right part for the digits (Figure 2.3).



**Figure 2.3** Imposed data capture area.

The gesture recognition method is quite simple. The data capture area is marked out in squares and a gesture is recognized if the stroke pass by certain squares in a precise order.

## 2.2 Algorithms

This section describes in details the algorithms and extensions selected to be implemented and tested in the study. Many other algorithms and extensions ex-

ist, like SIGER (SIGeR) or Rubine3D (Rubine3D), but the goal of this section is to focus on the four types of algorithms described in the algorithm taxonomy.

### 2.2.1 Rubine

#### Description

The Rubine algorithm described in Rubine [1991] is a statistical-based recognition method. It allows analyzing unique stroke gestures. The algorithm relies on two phases that are independent and short to implement: the training and the classification.

Each gesture is represented by a table  $g$  composed of  $P$  points of coordinate  $(x_p, y_p)$  which are associated with a timestamp  $t_p$  :

$$g_p = (x_p, y_p, t_p) \quad 0 \leq p < P$$

The data are firstly cleaned before the gesture can be used for a training or in a classification. This cleaning consists of rejecting points whose the predecessors are located 3 pixels away or less. The goal is to avoid trembling during the gesture sketching. The recognition problem can be defined as follows: there is a set of  $C$  gesture classes, numbered from 0 to  $C - 1$ . Each class is defined by a set of gestures-examples. Given a new gesture  $g$ , the class of  $g$  must be determined. In other words a class whose the members are the closest to  $g$  following a criterion explained after.

The statistical recognition is realized in two steps. In a first time, a feature vector  $\mathbf{f} = [f_1, \dots, f_n]$  is extracted from the encoded gesture. Then this vector  $\mathbf{f}$  is classified as one of the  $C$  different possible gestures thanks to a linear function.

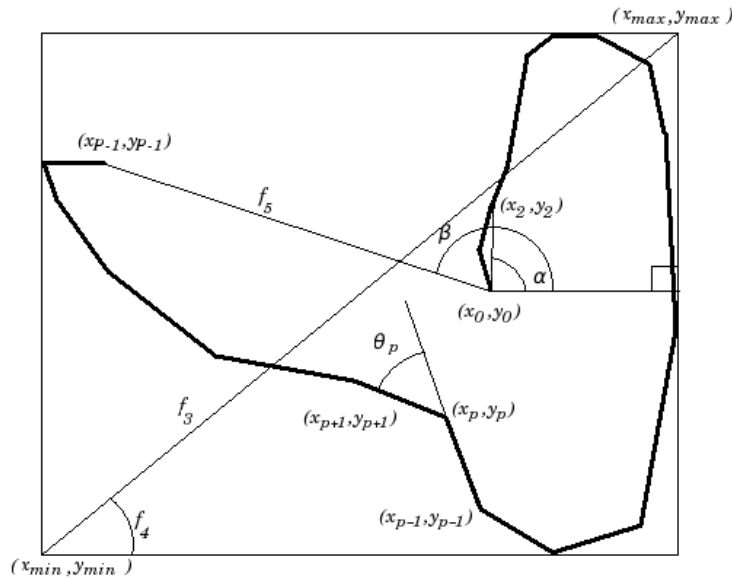
**Features** As mentioned, Rubine is based on a feature vector to classify the gestures. In his article (Rubine [1991]), the author describes the criteria on which he relied in order to select 13 features. Here are the criteria :

- each feature must be incrementally computable in constant time for each input point of the gesture. This property allows arbitrarily large gestures to be handled as efficiently as small ones;

## 2 State of the art

- since the classification algorithm performs poorly when a class has a feature with a multimodal distribution, a small change in the input should result in a correspondingly small change in each feature;
- there should be enough features to provide differentiation between all gestures that might reasonably be expected, but, for efficiency reasons, there should not be too many.

Figure 2.4 shows the actual features used, both geometrically and algebraically.



**Figure 2.4** Features used to identify strokes.

- $f_1$  et  $f_2$  : the cosine and the sine of the initial angle of the stroke,

$$f_1 = \cos \alpha = (x_2 - x_0) / \sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}$$

$$f_2 = \sin \alpha = (y_2 - y_0) / \sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}$$

For these two features, the third point of the gesture is used instead of considering the angle between the two first points in order to avoid noise in the results. Additionally, the angle's cosine and sine are used as features rather than the angle itself to avoid a discontinuity as the angle passes through  $2\pi$  and wraps to 0.

Let a *bounding box* be the smallest rectangular box holding all the gesture.

## 2 State of the art

- $f_3$  : the length of the bounding box diagonal :

$$f_3 = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2}$$

- $f_4$  : the angle of the bounding box diagonal :

$$f_4 = \arctan \frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

- $f_5$  : the distance between the first and the last point of the gesture :

$$f_5 = \sqrt{(x_{P-1} - x_0)^2 + (y_{P-1} - y_0)^2}$$

- $f_6$  et  $f_7$  : the cosine and the sine of the angle between the first and the last point of the gesture :

$$f_6 = \cos \beta = (x_{P-1} - x_0) / f_5$$

$$f_7 = \sin \beta = (y_{P-1} - y_0) / f_5$$

- $f_8$  : the total length of the gesture.

Let  $\Delta x_p = x_{p+1} - x_p$  and  $\Delta y_p = y_{p+1} - y_p$ .

$$f_8 = \sum_{p=0}^{P-2} \sqrt{\Delta x_p^2 + \Delta y_p^2}$$

Let  $\theta_p$  be the angle between the segment joining the point  $p$  to the point  $p+1$  :

$$\theta_p = \arctan \frac{\Delta x_p \Delta y_{p-1} - \Delta x_{p-1} \Delta y_p}{\Delta x_p \Delta x_{p-1} + \Delta y_p \Delta y_{p-1}}$$

- $f_9$  : the total angle traversed :

$$f_9 = \sum_{p=1}^{P-2} \theta_p$$

- $f_{10}$  : the sum of the absolute value of the angle at each point of the gesture :

$$f_{10} = \sum_{p=1}^{P-2} |\theta_p|$$

## 2 State of the art

- $f_{11}$  : the sum of the squared value of the angle at each point of the gesture :

$$f_{11} = \sum_{p=1}^{P-2} \theta_p^2$$

The feature  $f_{11}$  gives a piece of information on *the sharpness* of the gesture. It allows to distinguish smooth gestures and those with sharp angles, such as "U" and "V".

Soit  $\Delta t_p = t_{p+1} - t_p$ .

- $f_{12}$  : the maximum speed (squared) of the gesture :

$$f_{12} = \max_{p=0}^{P-2} \frac{\Delta x_p^2 + \Delta y_p^2}{\Delta t_p^2}$$

- $f_{13}$  : the duration of the gesture :

$$f_{13} = t_{P-1} - t_0$$

$f_{12}$  and  $f_{13}$  features are dynamic components and allow the representation of a whole gesture instead of only its static image. Some applications could deactivate the use of these features.

The aforementioned feature set was empirically determined by Dean Rubine to work well on number of different gestures sets. Unfortunately, there are cases in which the features fail to distinguish between obviously different gestures (e.g. because the features take no account of the ordering of angles in a gesture). In such cases an additional feature may be added to discriminate between the thus far indistinguishable gestures.

The following description of the classification and training steps, as the outliers rejection, are the ones from Rubine [1991].

**Gesture classification** Given the feature vector  $\mathbf{f}$  computed  $g$ , the classification algorithm is quite for an input gesture simple and efficient. Associated with each gesture class is a linear evaluation function over the features. Gesture class  $c$  has weights  $w_{ci}$  for  $0 \leq i \leq F$ , where  $F$  is the number of features,

## 2 State of the art

currently 13. (Per-gesture-class variables are written with hatted subscripts to indicate their class.) The evaluations,  $v_{\hat{c}}$ , are calculated as follows :

$$v_{\hat{c}} = w_{\hat{c}0} + \sum_{i=1}^F w_{\hat{c}i} f_i \quad 0 \leq c < C \quad (2.1)$$

The classification of gesture  $g$  is simply the  $c$  which maximizes  $v_{\hat{c}}$ .

**Training** The training algorithm is a classic linear discriminator. The training problem is to determine the weights  $w_{\hat{c}i}$  from the example gestures.

Let  $f_{\hat{c}ei}$  be the  $i^{\text{th}}$  feature of the  $e^{\text{th}}$  example of gesture class  $c$ ,  $0 \leq e < E_{\hat{c}}$ , where  $E_{\hat{c}}$  is the number of training examples of class  $c$ . The sample estimate of the mean feature vector per class,  $\bar{f}_{\hat{c}i}$ , is simply the average of the features in the class :

$$\bar{f}_{\hat{c}i} = \frac{1}{E_{\hat{c}}} \sum_{e=0}^{E_{\hat{c}}-1} f_{\hat{c}ei}$$

The sample estimate of the covariance matrix of class  $c$ ,  $\Sigma_{\hat{c}ij}$ , is computed as :

$$\Sigma_{\hat{c}ij} = \sum_{e=0}^{E_{\hat{c}}-1} (f_{\hat{c}ei} - \bar{f}_{\hat{c}i})(f_{\hat{c}ej} - \bar{f}_{\hat{c}j})$$

For convenience in the next step, the usual  $1/(E_{\hat{c}} - 1)$  factor has not been included in  $\Sigma_{\hat{c}ij}$ . The  $\Sigma_{\hat{c}ij}$  are averaged to yield  $\Sigma_{ij}$ , an estimate of the common covariance matrix.

$$\Sigma_{ij} = \frac{\sum_{c=0}^{C-1} \Sigma_{\hat{c}ij}}{-C + \sum_{c=0}^{C-1} E_{\hat{c}}} \quad (2.2)$$

The sample estimate of the common covariance matrix  $\Sigma_{ij}$  is then inverted. The result is denoted  $\Sigma^{-1}$ . The weights  $w_{\hat{c}j}$  are computed from the estimates as follows :

$$w_{\hat{c}j} = \sum_{i=1}^F (\Sigma^{-1})_{ij} \bar{f}_{\hat{c}i} \quad 1 \leq j \leq F$$

$$w_{\hat{c}0} = -\frac{1}{2} \sum_{i=1}^F w_{\hat{c}i} \bar{f}_{\hat{c}i}$$

A singular matrix can usually be handled by discarding a subset of the features.

## 2 State of the art

**Rejection** A linear classifier will always classify a gesture  $g$  as one of the  $C$  gesture classes. This part presents methods for rejecting ambiguous gestures and outliers.

Intuitively, if there is a near tie for the maximum per-class evaluation function  $v_i$  the gesture is ambiguous. Given a gesture  $g$  with feature vector  $\mathbf{f}$  classified as class  $i$  ( $v_i > v_j$  for all  $j \neq i$ ) :

$$\bar{P}(\hat{i}|g) = \frac{1}{\sum_{j=0}^{C-1} e^{v_j - v_i}}$$

is an estimate of the probability that  $g$  was classified correctly. Rejecting gestures in which  $\bar{P}(\hat{i}|g) < 0.95$  works well in practice.

The Mahalanobis distance Duda and Hart [1993] can be used to determine the number of standard deviations a gesture  $g$  is away from the mean of its chosen class  $i$ .

$$\delta^2 = \sum_{j=1}^F \sum_{k=1}^F (\Sigma^{-1})_{jk} (f_j - \bar{f}_{ij})(f_k - \bar{f}_{ik})$$

Rejecting gestures for which  $\delta^2 > \frac{1}{2}F^2$  eliminates the obvious outliers. Unfortunately, this thresholding also tends to reject many seemingly good gestures, making it less than ideal.

### Pseudo-code

Three parts can be distinguished in the algorithm :

1. FEATURES\_RUBINE( $g$ ) (Figure 2.5) that computes the features vector of a gesture  $g$  ;
2. LEARNING\_RUBINE( $G$ ) (Figure 2.6) that computes the weights for each available gesture class in the provided learning set  $G$  ;
3. CLASSIFICATION\_RUBINE( $g$ ) (Figure 2.7) that classifies a gesture  $g$  after being trained by the learning algorithm.

### Advantages

A first advantage of the Rubine's algorithm is the short and independent phases of training and classification. Indeed only ten to fifteen examples are sufficient



## 2 State of the art

---

**FEATURES\_RUBINE( $g$ )**

---

**Input:** a gesture  $g$   
**Output:** the feature vector  $\mathbf{f}$  of the gesture  $g$

```

1   $\mathbf{f}[0] \leftarrow f_1 \leftarrow$  cosine of the initial angle of  $g$ 
2   $\mathbf{f}[1] \leftarrow f_2 \leftarrow$  sine of the initial angle of  $g$ 
3   $\mathbf{f}[2] \leftarrow f_3 \leftarrow$  length of the bounding box diagonal of  $g$ 
4   $\mathbf{f}[3] \leftarrow f_4 \leftarrow$  angle of the bounding box diagonal of  $g$ 
5   $\mathbf{f}[4] \leftarrow f_5 \leftarrow$  distance between the first and the last point of  $g$ 
6   $\mathbf{f}[5] \leftarrow f_6 \leftarrow$  cosine of the angle between the first and the last point of  $g$ 
7   $\mathbf{f}[6] \leftarrow f_7 \leftarrow$  sine of the angle between the first and the last point of  $g$ 
8   $\mathbf{f}[7] \leftarrow f_8 \leftarrow$  total length of  $g$ 
9   $\mathbf{f}[8] \leftarrow f_9 \leftarrow$  total angle traversed of  $g$ 
10  $\mathbf{f}[9] \leftarrow f_{10} \leftarrow$  sum of the absolute value of the angle at each point of  $g$ 
11  $\mathbf{f}[10] \leftarrow f_{11} \leftarrow$  sum of the squared value of the angle at each point of  $g$ 
12  $\mathbf{f}[11] \leftarrow f_{12} \leftarrow$  maximum speed (squared) of  $g$ 
13  $\mathbf{f}[11] \leftarrow f_{13} \leftarrow$  duration of  $g$ 
14 return  $\mathbf{f}$ 

```

---

**Figure 2.5** Computation of the features of the Rubine's algorithm :  $\text{FEATURES\_RUBINE}(g)$ .

---

**LEARNING\_RUBINE( $G$ )**

---

**Input:** a gesture set  $G$   
**Result:** the algorithm is trained on the gesture set  $G$

```

1  while  $G \neq \emptyset$  do
2       $g \leftarrow$  a gesture of  $G$ 
3       $G \leftarrow G \setminus g$ 
4       $\mathbf{f} \leftarrow \text{FEATURES\_RUBINE}(g)$ 
5      Compute the estimate sample of the mean feature vector for the gestures
        of the same class than  $g$  that are already removed from  $G$ 
6      Compute the estimate sample of the covariance matrix (cf. equation 2.2)
7      Compute the weights allowing the classification based of the inverted
        estimate sample of the covariance matrix
    end

```

---

**Figure 2.6** Learning step of the Rubine's algorithm :  $\text{LEARNING\_RUBINE}(G)$

---

**CLASSIFICATION\_RUBINE**( $g$ )

---

**Input:** a gesture  $g$   
**Data:** the algorithm is trained on a gesture set  $G$   
**Output:** the class  $gClass$  determined from the gesture  $g$

```

1  f  $\leftarrow$  FEATURES_RUBINE( $g$ )
2  ( $gClass, gScore$ )  $\leftarrow$  ( $\emptyset, -\infty$ )
3  foreach gesture class  $c_i \in C$ , the set of trained class do
4      Compute the score  $s$  of the gesture  $g$  compared to the weights of the class
         $c_i$  (cf. equation 2.1)
5      if  $gScore < s$  then
6          | ( $gClass, gScore$ )  $\leftarrow$  ( $c_i, s$ )
          end
      end
9  return  $gClass$ 

```

---

**Figure 2.7** Classification of a gesture with the Rubine's algorithm :  
 CLASSIFICATION\_RUBINE( $g$ ).

to correctly train the algorithm and obtain satisfying recognition rates (until 99%).

The basic features described by Dean Rubine are geometric and algebraic features and are easily computable. The comparison of these features is quite easy since they are represented by numerical values. This implies quick training and classification steps.

A second good point of the algorithm is its modularity. The features of a gesture can be easily modified. Furthermore the number of criteria is not fixed, implying extensibility of the tool.

### Limitations

Because of its modularity, the Rubine's algorithm does not have real limitations. Perhaps Its conception for uni-stroke gesture could be a weakness but there exist means for adapting the method to multi-strokes gesture.

Nevertheless we can point out that the rejection criterion tends to eliminate too much good gestures. The goal is obviously to only select the outliers. Actually, as the rejection criterion is optional, using it should be carefully discussed

## 2 State of the art

depending on the involved application. For example, if the application allows an easy cancel operation, then it could be interesting to discard the rejection criterion. In this case, when a gesture is wrongly recognized, then time needed to cancel and try again the gesture will be shorter than the time needed to sketch again the gesture considered as outliers by the algorithm.

### Summary

The Rubine's algorithm is based on the features of the gestures. It requires a quite small number of examples (from 15 to 20) per gesture class and is simple to implement.

For this method, a gesture set consists in different gesture classes. A gesture class is characterized by a training set of gestures of the same class (for example the letter "a"). In order to classify the gestures, the algorithm uses a feature set describing them: each gesture is associated with a feature vector. Additionally, weights are computed for each gesture class during the training phase. Based on these weights and on its feature vector, an input gesture can be scored for each gesture class. The class maximizing the score will be chosen.

In the original version of the algorithm, 13 different features are taken into account like the initial angle, the duration or the length of the gesture, ... but these features can be modified, replaced or increased in function of the needs.

The results of Rubine's algorithm are really convincing. Depending on the number of gesture classes to recognize, rates can reach 96 to 99%.

### 2.2.2 One Dollar

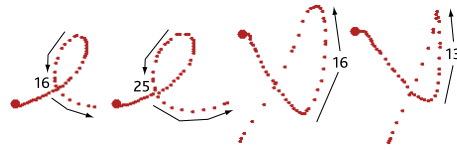
#### Description

The One Dollar algorithm, described in Wobbrock et al. [2007], is based on the comparison of point sets representing the gestures. The input gesture is compared to each training gesture (or template), the closest one being used to determine the class of the input gesture.

Comparing two gestures in their initial form point by point would not be pertinent. For two similar gestures, the user speed and the sensing hardware and

## 2 State of the art

software can induce different rates at which the points of the gestures appear. Furthermore, the length, the orientation, ... can vary.



**Figure 2.8** Example of disparities between similar gestures.

For example, let us consider figure 2.8. The two first sets of points define the same gesture but with different rates. The two last sets of points define another same gesture but with different lengths. Also, the two first gestures can be made similar to the two following ones using a  $90^\circ$  clockwise turn. Reflecting on these issues and on a desire for simplicity, the authors formulated eight criteria for their algorithm. One Dollar's algorithm must :

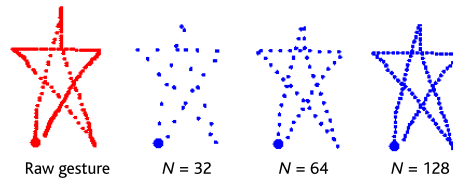
- be resilient to variations in sampling due to movement speed or sensing;
- support optional and configurable rotation, scale, and position invariance;
- require no advanced mathematical techniques (e.g., matrix inversions, derivatives, integrals);
- be easily written in few lines of code;
- allow developers and application end-users to "teach" it new gestures with only one example;
- return an  $N$ -best list with sensible  $[0..1]$  scores that are independent of the number of input points;
- provide recognition rates that are competitive with more complex algorithms previously used in HCI to recognize certain types of uni-stroke gestures useful for making selections, executing commands, or entering symbols (described in Wobbrock et al. [2007]).

Based on these principles, the authors imagined the four-step One Dollar's algorithm. Raw input points, whether those of gestures meant to serve as templates, or those of candidate gestures attempting to be recognized, are initially

## 2 State of the art

treated the same : they are resampled, rotated once, scaled, and translated. Candidate points  $C$  are then scored against each set of template points  $T_i$  over a series of angular adjustments to  $C$  that finds its optimal angular alignment to  $T_i$ . Each of these steps is explained in more details below with the description provided in Wobbrock et al. [2007].

**Step 1 : Resample the Point Path** As noted in the previous section, gestures in user interfaces are sampled at a rate determined by the sensing hardware and input software. Thus, movement speed will have a clear effect on the number of input points in a gesture. To make gesture paths directly comparable even at different movement speeds, gestures are first resampled such that the path defined by their original  $M$  points is defined by  $N$  equidistantly spaced points (Figure 2.9).



**Figure 2.9** A star gesture resampled to  $N=32$ , 64, and 128 points.

Using an  $N$  that is too low results in a loss of precision, while using an  $N$  that is too high adds time to path comparisons. In practice, a  $N$  around 64 was found as adequate.

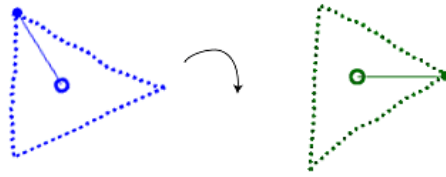
To resample, we first calculate the total length of the  $M$ -point path. Dividing this length by  $(N - 1)$  gives the length of each increment,  $I$ , between  $N$  new points. Then the path is stepped through such that when the distance covered exceeds  $I$ , a new point is added through linear interpolation. At the end of this step, the candidate gesture and any loaded templates will all have exactly  $N$  points. This will allow to measure the distance from  $C[k]$  to  $Ti[k]$  for  $k = 1$  to  $N$ .

**Step 2 : Rotate once based on the "indicative angle"** With two paths of ordered points, there is no closed-form solution for determining the angle to which one set of points should be rotated to best align with the other. Although there are complex techniques based on moments, these are not made

## 2 State of the art

to handle ordered points. The One Dollar's algorithm therefore searches over the space of possible angles for the best alignment between two point-paths. Although for many complex recognition algorithms an iterative process is prohibitively expensive, One Dollar is fast enough to make iteration useful. In fact, even naïvely rotating the candidate gesture by  $+1^\circ$  for  $360^\circ$  is fast enough for interactive purposes with 30 templates. However, it can do better than brute force with a "rotation trick" that makes finding the optimal angle much faster.

First, a gesture's *indicative angle* is found, which is defined as the angle formed between the centroid of the gesture and the gesture's first point. Then the gesture is rotated so that this angle is at  $0^\circ$  (Figure 2.10).



**Figure 2.10** Representation of the indicative angle.

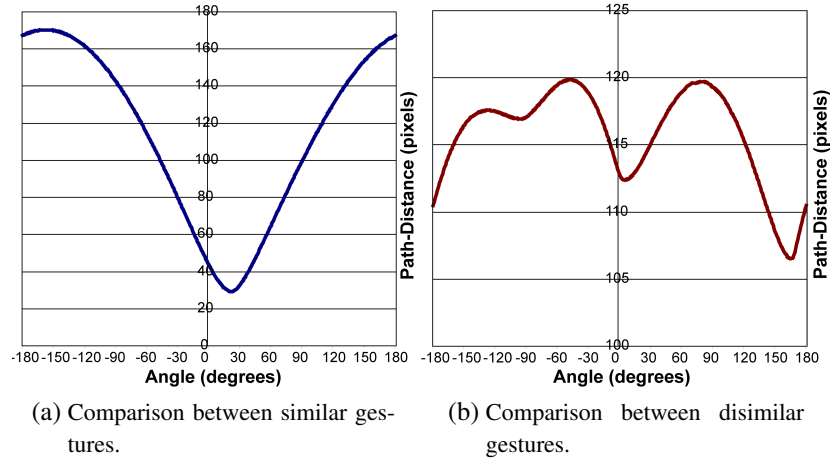
**Step 3 : Scale and translate** After rotation, the gesture is scaled to a *reference square*. By scaling to a square, we are scaling non-uniformly. This will allow to rotate the candidate about its centroid and safely assume that changes in pairwise point-distances between  $C$  and  $T_i$  are due only to rotation, not to aspect ratio. Of course, non-uniform scaling introduces some limitations, which will be discussed below.

After scaling, the gesture is translated to a reference point. For simplicity, the translation of gesture is chosen so that its centroid is at  $(0,0)$ .

**Step 4 : Find the optimal angle for the best score** At this point, all candidates  $C$  and templates  $T_i$  have been treated the same: resampled, rotated once, scaled, and translated. However, after rotating the indicative angles of all gestures to  $0^\circ$  as explained during step 2, there is no guarantee that two gestures  $C$  and  $T_i$  will be aligned optimally.  $C$ 's angle must therefore be tuned so that  $C$ 's path-distance to  $T_i$  is minimized. As mentioned, a brute force scheme could rotate  $C$  by  $+1^\circ$  for all  $360^\circ$  and take the best result. Although this method is guaranteed to find the optimal angle to within  $0.5^\circ$ , it is unnecessarily slow. The number of iterations used to find the optimal angle should thus be reduced.

## 2 State of the art

For this purpose, it has been experimentally determined that for comparison between similar gesture there was always a global minimum and no local minima in the graphs of path-distance as a function of angle (Figure 2.11 (a)).



**Figure 2.11** Path-distance as a function of angular rotation away from the  $0^\circ$  indicative angle (centered y-axis).

Therefore, a first improvement over the brute force approach would be hill climbing : rotate  $C$  by  $\pm 1^\circ$  for as long as  $C$ 's path-distance to  $T_i$  decreases.

However, although hill climbing is efficient for similar gestures, it is not efficient for dissimilar ones. Firstly, in this case the indicative angle is not a good approximation for the optimal angle, which will require more iterations to find it. Secondly, many dissimilar gesture pairs have local minima in their path-distance graphs (Figure 2.11 (b)), which is a problem for the stop criterion of this method.

Finally, since there is many more comparisons of a candidate to dissimilar templates than to dissimilar ones, it is chosen to use a strategy that performs slightly worse than fill climbing for similar gestures but far better for dissimilar ones. The strategy is *Golden Section Search (GSS)*, described in Press et al. [1992].

**Gesture classification** Using this technique, a candidate  $C$  can be compared to each stored template  $T_i$  to find the average distance  $d_i$  between corre-

## 2 State of the art

sponding points :

$$d_i = \frac{\sum_{k=1}^N \sqrt{(C[k]_x - T_i[k]_x)^2 + (C[k]_y - T_i[k]_y)^2}}{N} \quad (2.3)$$

This equation defines  $d_i$ , the *path-distance* between  $C$  and  $T_i$ . The template  $T_i$  with the least path-distance to  $C$  is the result of the recognition. This minimum path-distance  $d_i^*$  is converted to a  $[0..1]$  score using :

$$score = 1 - \frac{d_i^*}{0.5 \sqrt{size_1^2 + size_2^2}} \quad (2.4)$$

In this equation,  $size_1$  and  $size_2$  are the lengths of the sides of the reference squares used in step 3. Usually, these lengths are equal.

### Pseudo-code

Three parts can be distinguished in the algorithm :

1. PREPROCESSING\_ONEDOLLAR( $g$ ) (Figure 2.12) that executes a pre-processing on the gesture  $g$  given as parameter ;
2. TRAINING\_ONEDOLLAR( $G$ ), (Figure 2.13), that transforms and stores each gesture  $g$  of the learning set  $G$  given as parameter ;
3. CLASSIFICATION\_ONEDOLLAR( $g$ ) (Figure 2.14), that classifies a gesture  $g$  after being trained by the training algorithm.

### Advantages

The first big advantage of One Dollar's algorithm is the very small number of training examples required to obtain good recognition rates. Only one template can be sufficient to have a pretty good recognition rate while with three or four templates it is possible to reach very good precision (less than 1% of error rate).

The second advantage is thus the good recognition rate offered by the algorithm (independently of the number of training examples required). Unfortunately it



---

**PREPROCESSING\_ONEDOLLAR( $g$ )**

---

**Input:** a gesture  $g$

**Output:** the gesture  $g$  preprocessed :  $g'$

- 1  $g' \leftarrow$  Resampling of  $g$
  - 2  $g' \leftarrow$  Rotation of  $g'$  of the opposite of the indicative angle of  $g'$
  - 3  $g' \leftarrow$  Resampling of  $g'$
  - 4  $g' \leftarrow$  Translation of the centroid of  $g'$  to the origin
  - 5 **retourner**  $g'$
- 

**Figure 2.12** Preprocessing of a gesture for the One Dollar's algorithm : **PREPROCESSING\_ONEDOLLAR( $g$ )**.

---

**TRAINING\_ONEDOLLAR( $G$ )**

---

**Input:** a gesture set  $G$

**Output:** the string set  $T$  corresponding to the transformation of  $G$

- 1  $T \leftarrow \emptyset$ , the training data set
  - 2 **foreach**  $g_i \in G$  **do**
  - 3      $T \leftarrow T \cup \{\text{PREPROCESSING\_ONEDOLLAR}(g_i)\}$
  - end**
- 

**Figure 2.13** Training of the One Dollar's algorithm : **TRAINING\_ONEDOLLAR( $G$ )**.

## 2 State of the art

---

**CLASSIFICATION\_ONEDOLLAR( $g$ )**

---

**Data:** the string set  $T$  corresponding to the transformation of  $G$   
**Input:** a gesture  $g$   
**Output:** the class  $c$  of the gesture  $g$

```

1   $best \leftarrow 0$ , initialization of the best recognition score for the gesture  $g$ 
2   $g' \leftarrow \text{PREPROCESSING\_ONEDOLLAR}(g)$ 
3  foreach  $t_i \in T$  do
4      Find the angle giving the best distance  $d_i$  between  $g'$  et  $t_i$ 
5      Compute the score  $s_i$  based on  $d_i$ 
6      si  $s_i > best$  alors
7           $best \leftarrow s_i$ 
8           $c \leftarrow \text{class of } t_i$ 
      fin
end

```

---

**Figure 2.14** Classification of a gesture  $g$  with the One Dollar's algorithm :  $\text{CLASSIFICATION\_ONEDOLLAR}(g)$ .

will be seen in the next section that it is not always the case for all the gesture types.

Furthermore, the method allows to differentiate gestures that would be executed in different ways (for example, a square drawn clockwise and counter-clockwise).

Finally, the speed reached by One Dollar's algorithm is a non negligible quality. Indeed a very short time is required in order to classify a gesture, even more since it is possible to do without the Golden Section Search without big loss of precision for finding the optimal angle.

### Limitations

The One Dollar recognizer is a geometric template matcher, which means that candidate strokes are compared to previously stored templates, and the result produced is the closest match in 2-D Euclidean space. The algorithm is thus unable to take into account interesting features like pressure or speed movement that maybe would reveal similarities or dissimilarities.

## 2 State of the art

Furthermore, to facilitate pairwise point comparison, the default One Dollar's algorithm is rotation, scale and position invariant. While this provides tolerance to gesture variation, it means that One Dollar cannot distinguish gestures whose identities depend on specific orientations, aspect ratios, or locations. For example, separating squares from rectangles, circles from ovals, or up-arrows from down-arrows is not possible without modifying the algorithm.

### Summary

The One Dollar's algorithm is an algorithm based on 2-D geometric distances. It compares the input gesture to each gesture of a template set in order to determine which is the closest one and select its class. The distance is found by computing the mean on the addition point-by-point of the distances of between each gesture pair.

Before computing this distance, the algorithm proceeds to a preprocessing of all the gestures (input as templates) : a resampling to attribute the same number of points to each gesture; a first rotation ensuring that the indicative angle (angle between the centroid and the first point of the gesture) is  $0^\circ$ . Finally, by performing the Golden Section Search technique, the input gesture is rotated for each pair formed with each template in order to find the optimal angle.

One Dollar offers a good and fast recognition. Also, the algorithm is cheap since it does not need a lot of training examples. Unfortunately, the algorithm does not allow to take into account other features than the geometric ones. Furthermore, if the invariance as regards of the rotation, scale and position is an advantage in most of situations, it can represent limitations when comparing gestures that use these features to differentiate.

### 2.2.3 Levenshtein

#### Description

The algorithm presented in this section is based on the *distance of Levenshtein* (also known as *edit distance*). The method was described for the first time by Levenshtein [1966] and then adapted to the computation of the distance between two chains of characters (Gusfield [1997]) and finally to the gesture

## 2 State of the art

recognition (Coyette et al. [2007]). This distance allows to measure the dissimilarity degree between two chains of characters following pre-established criterion that will be detailed in the next sections. The main points of discussions concern the modelling of the problem by comparing chains of characters and the parameter setting of the distance of Levenshtein.

**Distance of Levenshtein** Determining the distance of Levenshtein is direct application to the dynamic programming. The main characteristic of the dynamic computing is the reuse of an optimal solution found for a subproblem in order to determine the optimal solution of the considered problem.

The computation of the distance of Levenshtein is thus based on this principle in order to find the distance between two chains of characters. This distance corresponds to the minimal number of insertions, deletions and substitutions to make in order to pass from the first chain of characters to the second one.

To formalize the distance of Levenshtein, let  $x$  and  $y$  be the two chains of characters (or *string*) to compare. We note  $|x|$  and  $|y|$  their respective length. The variable  $x_i$  ( $y_i$ ) symbolizes the  $i_{th}$  character of  $x$  ( $y$ ) with  $x_0$  ( $y_0$ ) representing the empty symbol. The strings can then be noted as follows :

$$\begin{cases} x &= x_1, x_2, \dots, x_{|x|} \\ y &= y_1, y_2, \dots, y_{|y|} \end{cases}$$

A substring of  $x$  representing the characters from  $x_i$  to  $x_j$  is defined by  $x_i^j = x_i, x_{i+1}, \dots, x_j$ .

The goal is to read one by one the characters of the string  $x$  in order to progressively build the string  $y$  following three operations :

- *insertion* of a character in  $y$  ;
- *deletion* of a character from  $x$  ;
- *substitution* of a character in  $y$  by a character from  $x$ .

Also, let us define two conventions :

- $x_i^{|x|}$  means that the  $i - 1$  first characters of  $x$  are read (and thus removed) in order to be supplied to  $y$

## 2 State of the art

- $y_0^j$  means that the  $j$  first characters of  $y$  are transcribed

Characters are thus progressively taken from  $x$  to write  $y$ .

The three operations previously defined can be formalized as follows :

- *insertion*:  $(x_i^{|x|}, y_0^{j-1}) \longrightarrow (x_i^{|x|}, y_0^j)$  ;
- *deletion*:  $(x_{i-1}^{|x|}, y_0^j) \longrightarrow (x_i^{|x|}, y_0^j)$  ;
- *substitution*:  $(x_{i-1}^{|x|}, y_0^{j-1}) \longrightarrow (x_i^{|x|}, y_0^j)$ .

A cost is associated with each of these three operations. In the basic version — described here — of distance of Levenshtein, its value 1 for an insertion, deletion, or substitution applied to two different characters, and 0 for a substitution applied to two same characters :

$$\begin{cases} \delta_{ins} y_i = 1 \\ \delta_{del} x_i = 1 \\ \delta_{sub} (x_i, y_j) = \delta_{ij} \end{cases}$$

With

$$\delta_{ij} = \begin{cases} 1 & \text{if } x_i \neq y_j \\ 0 & \text{if } x_i = y_j \end{cases}$$

It is now possible to formalize the optimal distance  $D^*$  for each step of the algorithm.

Initially :

$$D^*(x_0^{|x|}, y_0^0) = 0$$

At each step the optimal distance is written :

$$D^*(x_i^{|x|}, y_0^j) = \min \begin{cases} D^*(x_i^{|x|}, y_0^{j-1}) + 1 \\ D^*(x_{i-1}^{|x|}, y_0^j) + 1 \\ D^*(x_{i-1}^{|x|}, y_0^{j-1}) + \delta_{ij} \end{cases}$$

And finally, the distance of Levenshtein  $dL$  corresponds to the optimal distance when the whole string  $x$  is used to create the string  $y$  :

$$dL(x, y) = D^*(x_{|x|}^{|x|}, y_0^{|y|})$$

## 2 State of the art

Let us concretize this notion by considering an example. Let  $x$  be the string “*lire*” and  $y$  the string “*livre*”. The table 2.1 illustrates all the possibilities for the evolution of the algorithm. The horizontal movement corresponds to the insertion of a character in  $y$ , the vertical movement corresponds to the deletion of a character from  $x$ , and the diagonal movement represents the substitution of a character of  $y$  by a character of  $x$ .

|   |        | y      |   |   |   |   |   |
|---|--------|--------|---|---|---|---|---|
|   |        | $\phi$ | l | i | v | r | e |
| x | $\phi$ | 0      | 1 | 2 | 3 | 4 | 5 |
|   | l      | 1      | 0 | 1 | 2 | 3 | 4 |
|   | i      | 2      | 1 | 0 | 1 | 2 | 3 |
|   | r      | 3      | 2 | 1 | 1 | 1 | 2 |
|   | e      | 4      | 3 | 2 | 2 | 2 | 1 |

**Table 2.1** Evolution of the distance of Levenshtein by comparing the string  $x = \textit{lire}$  to the string  $y = \textit{livre}$ .

The final result gives a distance of Levenshtein equalling 1. In order to be aware of the sequence of operations, the algorithm must hold for each step a link to the preceding state. For this example, the algorithm executes the following steps :

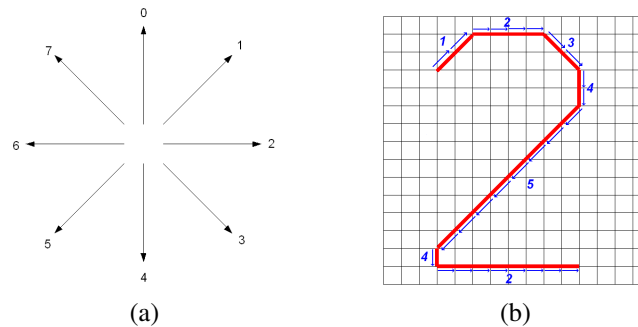
- $dL(\phi, \phi) = 0$
- substitution of  $l$  by  $l \rightarrow dL(l, l) = dL(\phi, \phi) + 0 = 0$
- substitution of  $i$  by  $i \rightarrow dL(li, li) = dL(l, l) + 0 = 0$
- insertion of  $v$  in  $y \rightarrow dL(li, liv) = dL(li, li) + 1 = 1$
- substitution of  $r$  by  $r \rightarrow dL(lir, livr) = dL(li, liv) + 0 = 1$
- substitution of  $e$  by  $e \rightarrow dL(lire, livre) = dL(lir, livr) + 0 = 1$

Thanks to dynamic programming, the computation of the distance of Levenshtein can be done in a time  $O(|x| \cdot |y|)$ .

**Transformation of the gestures into chains of characters** In order to apply the comparison with the distance of Levenshtein on gestures, they have to be transformed into chains of characters. For this purpose it has been chosen to considerate the directions between the points of the gestures. Each

## 2 State of the art

segment linking two points is represented by a digit between 0 and 7. These digits correspond to the directions of the compass rose (North, North-East, East, South-East, South, South-West, West, North-West) represented in Figure 2.15a. Figure 2.15b illustrates the transformation by showing the digit 2 drawn on a grid. The corresponding string is “11222233445555555422222222”.



**Figure 2.15** Working of the Levenshtein’s method. The compass rose depicted in (a) allows to describe a gesture as chain of characters. In (b), a gesture corresponding to the digit 2. The corresponding chain of character is “11222233445555555422222222”.

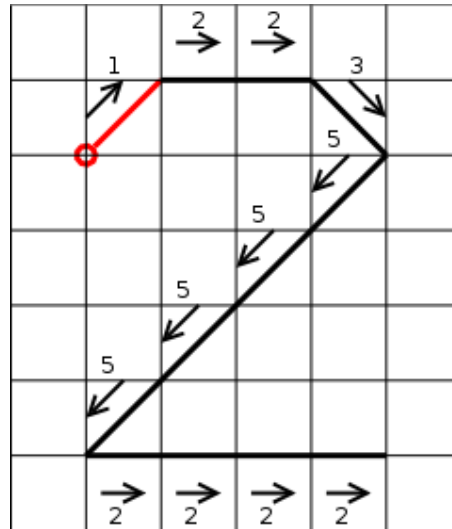
**Application to the gesture recognition** Let us consider the illustrating example represented in Figure 2.16. The digits 2 and 8 are decomposed into segments by following the compass rose depicted in Figure (a). The starting point of the gestures are represented by the red circle and the first directions are the red lines. The digit 2 can thus be translated into the string “122355552222” while the digit 8 corresponds to “123555321777”.

In order to compute the distance of Levenshtein between these two strings, a table is built in the same way than previously shown. It is depicted in Table 2.2 and the found distance separating the gestures is 6. Several different paths are possible to find this value, but the important here is to find the value and not to know the way used to find it.

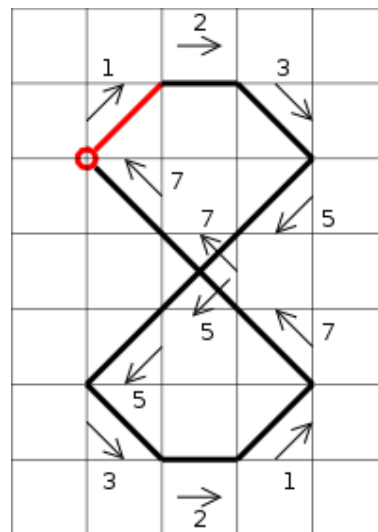
## Pseudo-code

Three parts can be distinguished in the algorithm :

## 2 State of the art



(a)



(b)

**Figure 2.16** Example of comparison of two encoded digits



## 2 State of the art

|        | $\phi$       | 1            | 2            | 3            | 5            | 5            | 5            | 3            | 2            | 1            | 7            | 7            | 7            |
|--------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| $\phi$ | <div>0</div> | 1            | 2            | 3            | 4            | 5            | 6            | 7            | 8            | 9            | 10           | 11           | 12           |
| 1      | 1            | <div>0</div> | 1            | 2            | 3            | 4            | 5            | 6            | 7            | 8            | 9            | 10           | 11           |
| 2      | 2            | <div>1</div> | <div>0</div> | 1            | 2            | 3            | 4            | 5            | 6            | 7            | 8            | 9            | 10           |
| 2      | 3            | 2            | <div>1</div> | 1            | 2            | 3            | 4            | 5            | 5            | 6            | 7            | 8            | 9            |
| 3      | 4            | 3            | 2            | <div>1</div> | 2            | 3            | 4            | 4            | 5            | 6            | 7            | 8            | 9            |
| 5      | 5            | 4            | 3            | 2            | <div>1</div> | 2            | 3            | 4            | 5            | 6            | 7            | 8            | 9            |
| 5      | 6            | 5            | 4            | 3            | 2            | <div>1</div> | 2            | 3            | 4            | 5            | 6            | 7            | 8            |
| 5      | 7            | 6            | 5            | 4            | 3            | 2            | <div>1</div> | 2            | 3            | 4            | 5            | 6            | 7            |
| 5      | 8            | 7            | 6            | 5            | 4            | 3            | 2            | <div>2</div> | 3            | 4            | 5            | 6            | 7            |
| 2      | 9            | 8            | 7            | 6            | 5            | 4            | 3            | 3            | <div>2</div> | <div>3</div> | 4            | 5            | 6            |
| 2      | 10           | 9            | 8            | 7            | 6            | 5            | 4            | 4            | 3            | <div>3</div> | <div>4</div> | 5            | 6            |
| 2      | 11           | 10           | 9            | 8            | 7            | 6            | 5            | 5            | 4            | 4            | <div>4</div> | <div>5</div> | 6            |
| 2      | 12           | 11           | 10           | 9            | 8            | 7            | 6            | 6            | 5            | 5            | 5            | <div>5</div> | <div>6</div> |

**Table 2.2** Computation of the distance of Levenshtein by comparing the string of the digit 2 : “122355552222” to the string of the digit 8 : “123555321777”.

## 2 State of the art

1. TRANSFORMATION\_LEVENSHTein( $g$ ), (Figure 2.17) that transforms the gesture  $g$  given as parameter in a chain of orientation characters ;
2. TRAINING\_LEVENSHTein( $G$ ), (Figure 2.18) that transforms and stores each gesture  $g$  of the learning set  $G$  given as parameter ;
3. CLASSIFICATION\_LEVENSHTein( $g$ ), (Figure 2.19) that classifies a gesture  $g$  after being trained by the training algorithm.

---

### TRANSFORMATION\_LEVENSHTein( $g$ )

---

**Input:** a gesture  $g$

**Output:** the gesture  $g$  transformed :  $g'$

```

1   $g' \leftarrow \emptyset$ 
2  // point  $p_i$  of  $g$ ,  $1 < i < n$ ,  $p_0$  is the origin point and  $p_n$  the last point of  $g$ 
3  for  $i$  from 1 to  $n$  do
4       $dir \leftarrow$  Compute the direction from the point  $p_{i-1}$  to the point  $p_i$ 
5       $dcr \leftarrow$  Choose the closest direction of  $dir$  on the compass rose //(see
        Figure 2.15a)
6       $g' \leftarrow$  concatenation( $g', dcr$ )
    end
8  return  $g'$ 
```

---

**Figure 2.17** Transformation of a gesture for the Levenshtein's algorithm :  
TRANSFORMATION\_LEVENSHTein( $g$ ).

---

### TRAINING\_LEVENSHTein( $G$ )

---

**Input:** a gesture set  $G$

**Output:** the string set  $T$  corresponding to the transformation of  $G$

```

1   $T = \emptyset$ , the training data set
2  foreach  $g_i \in G$  do
3       $T \leftarrow T \cup \{\text{TRANSFORMATION\_LEVENSHTein}(g_i)\}$ 
    end
```

---

**Figure 2.18** Training of the Levenshtein's algorithm :  
TRAINING\_LEVENSHTein( $G$ ).

---

**CLASSIFICATION\_LEVENSHTEIN( $g$ )**

---

**Data:** the string set  $T$  corresponding to the transformation of  $G$   
**Input:** a gesture  $g$   
**Output:** the class  $c$  of the gesture  $g$

```

1   $g' \leftarrow \text{TRANSFORMATION\_LEVENSHTEIN}(g)$ 
2   $best \leftarrow dL_0$ , initialization of the best distance of Levenshtein for the gesture  $g$ 
3   $c \leftarrow \text{class of } t_0$ , initialization of the class for the gesture  $g$ 
4  foreach  $t_i \in T$  do
5      Compute the distance of Levenshtein  $dL_i$  between  $g'$  and  $t_i$ 
6      if  $dL_i < best$  then
7           $best \leftarrow dL_i$ 
8           $c \leftarrow \text{class of } t_i$ 
9      end
10 end
11 return  $c$ 

```

---

**Figure 2.19** Classification of a gesture  $g$  with the Levenshtein's algorithm :  $\text{CLASSIFICATION\_LEVENSHTEIN}(g)$ .

### Advantages

A first advantage of the Levenshtein's algorithm is its ease of implementation. The transformation of the gesture into a chain of character is simple and directly based on the directions between the points of the gesture. Also, the classification is quite easy since it only requires a comparison of the distance between each input - template pair.

A second quality of the Levenshtein method is its ability to differentiate same gestures but executed in different ways (for example, a square drawn clockwise and counter-clockwise).

A third quality is the fastness of execution. The transformation of the gestures is really quick since it is only a correspondence between directions and characters. Furthermore, the time needed for computing the distances of Levenshtein is bounded thanks to the use of dynamic programming techniques.

## 2 State of the art

### Limitations

In most cases, the gestures are quite small and transformation step is fast. However, since the time required for this step is directly dependent of the number of points in the gesture, if this number of points becomes very huge, the time needed for the step can raise linearly.

Another drawback is also introduced by the length of the training examples. A small and a big occurrence of a similar gesture can be recognized dissimilar since they will require many deletion and insertion operations.

### Summary

The Levenshtein's algorithm is an algorithm directly based on the directions of the gestures in order to generate chains of characters symbolizing these directions. Dynamic programming techniques allow to compute the distance of Levenshtein between two chains of characters by considering the number of insertions, deletions, and substitutions to pass from one chain to the other one. By comparing the distances between the input gesture and each of the training gestures, its classification is found by taking the class of the closest training example.

The algorithm can give very good recognition rates if the number of training examples is sufficiently huge and covers most of the possible gestures. Unfortunately, it implies a huge number of comparisons, which is not necessarily compatible with good speed.

Finally, the Levenshtein's algorithm take into account the directions of the gesture but is not aware of other features like the dynamic of the gesture. However, the algorithm could be adapted by adding some parameters like pressure variation, speed variation, etc.

### 2.2.4 Stochastic Levenshtein

#### Description

The basic distance of Levenshtein (or edit distance) described in the previous section has arbitrarily fixed costs allowing to model operations of insertion,

## 2 State of the art

deletion and substitution. The purpose of the approach presented in this section is to deal with automatically adjusted costs that improve the results of the original algorithm in the best way.

As introduced by Oncina et Sebban in their article (Oncina and Sebban [2006]), if the gestures transformations are based on a random phenomenon and then on an underlying probability distribution, edit operations become random variables. The resulting similarity measure is then called the *stochastic edit distance*. An efficient way to model this distance consists in viewing it as a stochastic transduction between the input and output alphabets. One of the first published algorithm to automatically learn the parameters of a stochastic transducer has been proposed by Ristad and Yianilos ((Ristad and Yianilos [1998])). They provide a stochastic model which allows us to learn a stochastic edit distance, in the form of a memoryless transducer (i.e. with only one state), from a corpus of similar examples, using the Expectation-Maximization (EM) algorithm. Ristad and Yianilos define the stochastic edit distance between two strings  $x$  and  $y$  as (the minus logarithm of) the *joint* probability of the pair  $(x, y)$ .

The formalization and description of the algorithm given below are the ones defined by Oncina and Sebban in Oncina and Sebban [2006].

**Notations** An alphabet  $X$  is a finite nonempty set of symbols.  $X^*$  denotes the set of all finite strings over  $X$ . Let  $x \in X^*$  be an arbitrary string of length  $|x|$  over the alphabet  $X$ . In the following, unless stated otherwise, symbols are indicated by  $a, b, \dots$ , strings by  $u, v, \dots$  and the empty string by  $\epsilon$ .  $\mathbb{R}^+$  is the set of non negative reals. Let  $f$  be a function, from which  $[f(x)]_{\pi(x, \dots)}$  is equal to  $f(x)$  if the predicate  $\pi(x, \dots)$  holds and 0 otherwise, where  $x$  is a (set of) dummy variable(s).

A string edit distance is characterized by a triple  $(X, Y, c_e)$  consisting of the finite alphabets  $X$  and  $Y$  and the primitive cost function  $c_e : E \rightarrow \mathbb{R}^+$  where  $E = E_s \cup E_d \cup E_i$  is the alphabet of primitive edit operations,  $E_s = X \times Y$  is the set of substitutions,  $E_d = X \times \{\epsilon\}$  is the set of deletions,  $E_i = \{\epsilon\} \times Y$  is the set of insertions. Each such triple  $(X, Y, c_e)$  induces a distance function  $d : X^* \times Y^* \rightarrow \mathbb{R}^+$  that maps a pair of strings to a non negative real value. The edit distance  $d(x, y)$  between two strings  $x \in X$  and  $y \in Y$  is defined recursively

## 2 State of the art

as :

$$d(x, y) = \min \begin{cases} [c_e(a, b) + d(x', y')]_{x=x'a \wedge y=y'b} \\ [c_e(a, b) + d(x', y)]_{x=x'a} \\ [c_e(\epsilon, b) + d(x, y')]_{y=y'b} \end{cases}$$

**Joint memoryless transducer** A joint memoryless transducer defines a joint probability distribution over the pairs of strings. It is denoted by a tuple  $(X, Y, c, \gamma)$  where  $X$  is the input alphabet,  $Y$  is the output alphabet,  $c$  is the *primitive* joint probability function,  $c : E \rightarrow [0, 1]$  and  $\gamma$  is the probability of the termination symbol of a string. As  $(\epsilon, \epsilon) \notin E$ , in order to simplify the notations,  $c(\epsilon, \epsilon)$  and  $\gamma$  are going to be used as synonyms.

Let us assume for the moment that we know the probability function  $c$  (in fact, it will be learned later). We are then able to compute the joint probability  $p(x, y)$  of a pair of strings  $(x, y)$ . Actually, the joint probability  $p : X^* \times Y^* \rightarrow [0, 1]$  of the strings  $x, y$  can be recursively computed by means of an auxiliary function (forward)  $\alpha : X^* \times Y^* \rightarrow \mathbb{R}^+$  as :

$$\begin{aligned} \alpha(x, y) = & [1]_{x=\epsilon \wedge y=\epsilon} \\ & + [c(a, b) \cdot \alpha(x', y')]_{x=x'a \wedge y=y'b} \\ & + [c(a, \epsilon) \cdot \alpha(x', y)]_{x=x'a} \\ & + [c(\epsilon, b) \cdot \alpha(x, y')]_{y=y'b} , \end{aligned}$$

And then :  $p(x, y) = \alpha(x, y)\gamma$ .

In a symmetric way  $p(x, y)$  can be recursively computed by means of an auxiliary function (backward)  $\beta : X^* \times Y^* \rightarrow \mathbb{R}^+$  as :

$$\begin{aligned} \beta(x, y) = & [1]_{x=\epsilon \wedge y=\epsilon} \\ & + [c(a, b) \cdot \beta(x', y')]_{x=ax' \wedge y=by'} \\ & + [c(a, \epsilon) \cdot \beta(x', y)]_{x=ax'} \\ & + [c(\epsilon, b) \cdot \beta(x, y')]_{y=by'} \end{aligned}$$

And then :  $p(x, y) = \beta(x, y)\gamma$ .

Both functions (forward and backward) can be computed in  $O(|x| \cdot |y|)$  time using a dynamic programming technique. This model defines a probability distribution over the pairs  $(x, y)$ . More precisely,

$$\sum_{x \in X^*} \sum_{y \in Y^*} p(x, y) = 1 ,$$

## 2 State of the art

that is achieved if the following conditions are fulfilled :

$$\begin{aligned} \gamma > 0; c(a, b), c(\epsilon, b), c(a, \epsilon) &\geq 0 \quad \forall a \in X, b \in Y; \\ \sum_{\substack{a \in X \cup \{\epsilon\} \\ b \in Y \cup \{\epsilon\}}} c(a, b) &= 1. \end{aligned}$$

Given  $p(x, y)$ , we can then compute, as mentioned in Ristad and Yianilos [1998], the stochastic edit distance  $d_s(x, y)$  between  $x$  et  $y$  which is defined as the negative logarithm of the probability of the string pair  $p(x, y)$  according to the memoryless stochastic transducer :

$$d_s(x, y) = -\log p(x, y) \quad \forall x \in X^*, \forall y \in Y^*$$

The following step consists in learning the parameters  $c(a, b)$  of the memoryless transducer, i.e. the primitive edit costs. Let  $S$  be a finite set of  $(x, y)$  pairs of similar strings. Ristad et Yianilos propose to use the expectation-maximization (EM) algorithm to find the optimal joint stochastic transducer. The EM algorithm consists in two steps (expectation and maximization) that are repeated until a convergence criterion is achieved. For the thesis purposes, this convergence criterion is chosen to be the absence of consequent variation (less than 1% of the preceding value) of  $\alpha(x, y)$  (or  $\beta(x, y)$ ).

Given an auxiliary  $(|X| + 1) \times (|Y| + 1)$  matrix  $\delta$ , the expectation step can be described as follows :

$$\begin{aligned} \forall a \in X, b \in Y, \\ \delta(a, b) &= \sum_{(xax', yby') \in S} \frac{\alpha(x, y) c(a, b) \beta(x', y') \gamma}{p(xax', yby')}, \\ \delta(\epsilon, b) &= \sum_{(xx', yby') \in S} \frac{\alpha(x, y) c(\epsilon, b) \beta(x', y') \gamma}{p(xx', yby')}, \\ \delta(a, \epsilon) &= \sum_{(xax', yy') \in S} \frac{\alpha(x, y) c(a, \epsilon) \beta(x', y') \gamma}{p(xax', yy')}, \\ \delta(\epsilon, \epsilon) &= \sum_{(x, y) \in S} \frac{\alpha(x, y) \gamma}{p(x, y)} = |S|. \end{aligned}$$

And the maximization :

$$\begin{aligned} c(a, b) &= \frac{\delta(a, b)}{N} \quad \forall a \in X \cup \{\epsilon\} \quad \forall b \in Y \cup \{\epsilon\} \\ \text{avec} \quad N &= \sum_{\substack{a \in X \cup \{\epsilon\} \\ b \in Y \cup \{\epsilon\}}} \delta(a, b) \end{aligned}$$

## 2 State of the art

Unfortunately, the algorithm just described suffers of a big limitation : by learning the primitive joint probability function  $c(a, b)$ , Ristad et Yianilos learn at the same time the marginal distribution  $c(a) = \sum_{b \in Y \cup \{\epsilon\}} c(a, b)$ . The learned edition costs (and the stochastic edit distance) are then dependent of the a priori distribution of the input strings, that is obviously awkward. To free of this statistical bias, we have to learn the primitive conditional probability function independently of the marginal distribution. That is the goal of the next section.

**Conditional memoryless transducer** A conditional memoryless transducer is denoted by a tuple  $(X, Y, c, \gamma)$  where  $X$  is the input alphabet,  $Y$  is the output alphabet,  $c$  is the *primitive* conditional probability function  $c : E \rightarrow [0, 1]$  and  $\gamma$  is the probability of the termination symbol of a string. As in the joint case, since  $(\epsilon, \epsilon) \notin E$ , in order to simplify the notations,  $c(\epsilon, \epsilon)$  and  $\gamma$  are going to be used as synonyms.

The probability  $p : X^* \times Y^* \rightarrow [0, 1]$  of the string  $y$  assuming the input one was a  $x$  (noted  $p(y|x)$ ) can be recursively computed by means of an auxiliary function (forward)  $\alpha : X^* \times Y^* \rightarrow \mathbb{R}^+$  as :

$$\begin{aligned} \alpha(y|x) = & [1]_{x=\epsilon \wedge y=\epsilon} \\ & + [c(b|a) \cdot \alpha(y'|x')]_{x=x'a \wedge y=y'b} \\ & + [c(\epsilon|a) \cdot \alpha(y|x')]_{x=x'a} \\ & + [c(b|\epsilon) \cdot \alpha(y'|x)]_{y=y'b} , \end{aligned}$$

And then :  $p(y|x) = \alpha(y|x)\gamma$ .

In a symmetric way,  $p(y|x)$  can be recursively computed by means of an auxiliary function (backward)  $\beta : X^* \times Y^* \rightarrow \mathbb{R}^+$  as :

$$\begin{aligned} \beta(y|x) = & [1]_{x=\epsilon \wedge y=\epsilon} \\ & + [c(b|a) \cdot \beta(y'|x')]_{x=ax' \wedge y=by'} \\ & + [c(\epsilon|a) \cdot \beta(y|x')]_{x=ax'} \\ & + [c(b|\epsilon) \cdot \beta(y'|x)]_{y=by'} . \end{aligned}$$

And then :  $p(y|x) = \beta(y|x)\gamma$ .



## 2 State of the art

As in the joint case, both functions can be computed in  $O(|x| \cdot |y|)$  time using a dynamic programming technique. In this model a probability distribution is assigned conditionally to each input string. Then

$$\sum_{y \in Y^*} p(y|x) \in \{1, 0\} \quad \forall x \in X^*$$

The 0 is in the case the input string  $x$  is not in the domain of the function.<sup>1</sup> It can be show that the normalization of each conditional distribution can be achieved if the following conditions over the function  $c$  and the parameter  $\gamma$  are fulfilled (for more details, see Oncina and Sebban [2006]) :

$$\begin{aligned} \gamma > 0; c(a, b), c(\epsilon, b), c(a, \epsilon) &\geq 0 \quad \forall a \in X, b \in Y; \\ \sum_{b \in Y} c(b|\epsilon) + \sum_{b \in Y} c(b|a) + c(\epsilon|a) &= 1 \quad \forall a \in X; \sum_{b \in Y} c(b|\epsilon) + \gamma = 1. \end{aligned}$$

As in the joint case, the expectation-maximization algorithm can be used in order to find the optimal parameters. The expectation step deals with the computation of the matrix  $\delta$  :

$$\begin{aligned} \forall a \in X, b \in Y, \\ \delta(b|a) &= \sum_{(xax', yby') \in S} \frac{\alpha(y|x)c(b|a)\beta(y'|x')\gamma}{p(yby'|xax')}, \\ \delta(b|\epsilon) &= \sum_{(xx', yby') \in S} \frac{\alpha(y|x)c(b|\epsilon)\beta(y'|x')\gamma}{p(yby'|xx')}, \\ \delta(\epsilon|a) &= \sum_{(xax', yy') \in S} \frac{\alpha(y|x)c(\epsilon|a)\beta(y'|x')\gamma}{p(yy'|xax')}, \\ \delta(\epsilon|\epsilon) &= \sum_{(x,y) \in S} \frac{\alpha(y|x)\gamma}{p(y|x)} = |S|. \end{aligned}$$

---

<sup>1</sup>If  $p(x) = 0$  then  $p(x, y) = 0$  and as  $p(y|x) = p(x, y)/p(x)$  we have  $\frac{0}{0}$  indeterminism. The chosen solution is to take  $\frac{0}{0} = 0$  in order to maintain  $\sum_{y \in Y^*} p(y|x)$  finite.

## 2 State of the art

The maximization step allows to deduce the current edit costs. :

$$\begin{aligned} c(b|a) &= \frac{\delta(b|a)}{N(a)} \frac{N - N(\epsilon)}{N}, \\ c(b|\epsilon) &= \frac{\delta(b|\epsilon)}{N}, \\ c(\epsilon|a) &= \frac{\delta(\epsilon|a)}{N(a)} \frac{N - N(\epsilon)}{N}, \\ \gamma &= \frac{N - N(\epsilon)}{N}, \end{aligned}$$

where

$$\begin{aligned} N &= \sum_{\substack{a \in X \cup \{\epsilon\} \\ b \in Y \cup \{\epsilon\}}} \delta(b|a), \\ N(a) &= \sum_{b \in Y \cup \{\epsilon\}} \delta(b|a), \\ N(\epsilon) &= \sum_{b \in Y} \delta(b|\epsilon) \end{aligned}$$

**Application to the gesture recognition** As explained in Ristad and Yianilos [1998], the joint probabilities are not adapted to classification. Indeed they allow to determine a distance between strings, but from a corpus of similar strings. But the goal is here to categorize strings on a classifier trained with strings of different classes in order to determine the class of the input strings. That is what the conditional memoryless transducer proposes.

This conditional memoryless transducer is trained on finite set  $S$  of string pairs (strings representing the gestures and generated with the directions in the same way than explained for the original version of Levenshtein's algorithm). In order to understand how the string pairs set is built, let us assume an example for which  $i$  different classes exist. In each class set, several examples are selected in order to form a learning set  $LS$  used during the training set. This set is composed of  $i$  subsets  $LS_i$ , each subset holding the learning example chosen for corresponding class. For each learning subset  $LS_i$  a set of pairs  $PS_i$  is built in the form  $(x, NN(x))$ ,  $\forall x \in LS_i$ , where  $NN(x) = \underset{y \in LS_i - \{x\}}{\operatorname{argmin}} d_{lev}(x, y)$  and  $d_{lev}$  is the classical distance of Levenshtein. In other words, each example is associated with a second example that is the closest one among its original class learning subset and by considering the classical distance of Levenshtein.

Once these pair subsets constituted, the conditional transducer is trained on  $\cup_i PS_i$  while an input gesture  $x'$  is classified by maximizing the probability

## 2 State of the art

$p(y|x')$  provided by the conditional transducer  $y \in \cup_i L S_i$ .

### Pseudo-code

The pseudo-code of the algorithm is decomposed into five parts :

1. FORWARD-EVALUATE( $x^T, y^V, c$ ) (Figure 2.20) that computes “forward” the dynamic programming table allowing to determine the probability of  $y^V$  conditioned to  $x^T$  ;
2. BACKWARD-EVALUATE( $x^T, y^V, c$ ) (Figure 2.21) that computes “backward” the dynamic programming table allowing to determine the probability of  $y^V$  conditioned to  $x^T$  ;
3. EXPECTATION-MAXIMIZATION( $c, S$ ) (Figure 2.22) that optimizes the primitive edit costs of  $c$  thanks to the training pair set  $S$  ;
4. EXPECTATION-STEP( $x^T, y^V, c, \delta, S$ ) (Figure 2.23) that computes the auxiliary primitive edit costs of  $\delta$  for a step of the expectation-maximization algorithm ;
5. MAXIMIZATION-STEP( $c, \delta$ ) (Figure 2.24) that maximizes the primitive edit costs of  $c$  in function of  $\delta$  for a step of the expectation-maximization algorithm.

### Advantages

The stochastic Levenshtein’s algorithm allows to optimize the weights for training data. Its big advantage is thus to improve the recognition rates in relation to the classical algorithm.

### Limitations

The slowness is the major drawback of the stochastic Levenshtein’s algorithm. The expectation-maximization step takes a lot of time. It represents a problem for reaching the best working of the algorithm since a lot of training examples are needed.

## 2 State of the art

---

**FORWARD-EVALUATE**( $x^T, y^V, c$ )

---

**Input:**  $y^V$  the string for which we want to know the probability conditioned to  $x^T$   
 $c$  the primitive edit costs table

**Output:** the table holding the dynamic programming steps of the forward function and in which we can find  $y^v|x^t$  at index (t,v)

```

1   $\alpha_{0,0} \leftarrow 1$ 
2  for  $t \leftarrow 0$  to  $T$  do
3      for  $v \leftarrow 0$  to  $V$  do
4          if  $v > 1$  or  $t > 1$  then
5               $\alpha_{t,v} \leftarrow 0$ 
6          end
7          if  $v > 1$  then
8               $\alpha_{t,v} \leftarrow \alpha_{t,v} + c(y_v|\epsilon)\alpha_{t,v-1}$ 
9          end
10         if  $t > 1$  then
11              $\alpha_{t,v} \leftarrow \alpha_{t,v} + c(\epsilon|x_t)\alpha_{t-1,v}$ 
12         end
13         if  $v > 1$  et  $t > 1$  then
14              $\alpha_{t,v} \leftarrow \alpha_{t,v} + c(y_v|x_t)\alpha_{t-1,v-1}$ 
15         end
16     end
17 end
18  $\alpha_{T,V} \leftarrow \alpha_{T,V} * c(\epsilon|\epsilon)$ 
19 return  $\alpha$ 

```

---

**Figure 2.20** The stochastic Levenshtein's algorithm : FORWARD-EVALUATE.

## 2 State of the art

---

**BACKWARD-EVALUATE**( $x^T, y^V, c$ )

---

**Input:**  $y^V$  the string for which we want to know the probability conditioned to  $x^T$   
 $c$  the primitive edit costs table

**Output:** the table holding the dynamic programming steps of the backward function and in which we can find  $y^v|x^t$  at index (t,v)

```

1   $\beta_{0,0} \leftarrow 1$ 
2  for  $t \leftarrow T$  to 0 do
3      for  $v \leftarrow V$  to 0 do
4          if  $v < 1$  or  $t < 1$  then
5               $\beta_{t,v} \leftarrow 0$ 
6          end
7          if  $v < 1$  then
8               $\beta_{t,v} \leftarrow \beta_{t,v} + c(y_{v+1}|\epsilon)\beta_{t,v+1}$ 
9          end
10         if  $t < 1$  then
11              $\beta_{t,v} \leftarrow \beta_{t,v} + c(\epsilon|x_{t+1})\beta_{t+1,v}$ 
12         end
13         if  $v < 1$  et  $t < 1$  then
14              $\beta_{t,v} \leftarrow \beta_{t,v} + c(y_{v+1}|x_{t+1})\beta_{t+1,v+1}$ 
15         end
16     end
17 end
18  $\beta_{T,V} \leftarrow \beta_{T,V} * c(\epsilon|\epsilon)$ 
19 return  $\beta$ 

```

---

**Figure 2.21** The stochastic Levenshtein's algorithm :  
 BACKWARD-EVALUATE.

---

EXPECTATION-MAXIMIZATION( $c, S$ )

---

**Input:**  $c$  the primitive edit costs table  
 $S$  the training string pair set  
**Result:** the primitive edit costs of  $c$  are optimized

```

1  repeat
2      foreach  $a \in E$  do
3          foreach  $b \in E$  do
4               $\delta(b|a) \leftarrow 0$ 
          end
      end
7     for  $i \leftarrow 1$  to  $n$  do
8         EXPECTATION-STEP( $x^{T_i}, u^{V_i}, c, \delta, S$ )
      end
10    MAXIMIZATION-STEP( $c, \delta$ )
until convergence

```

---

**Figure 2.22** The stochastic Levenshtein's algorithm :  
 EXPECTATION-MAXIMIZATION.

### Summary

The selected stochastic Levenshtein's algorithm is based on the conditional probabilities. Firstly because of the bias introduced by the joint probabilities and secondly because the algorithm based on the joint probabilities is not adapted for the strings of different classes.

The stochastic Levenshtein's algorithm is probabilistic. It takes a string pair set as input. These strings are elaborated with the same technique used for classical Levenshtein's algorithm, i.e. by considering the directions between each pair of successive points of the gesture in order to assign them a digit between 0 and 7 regarding the compass rose depicted in Figure 2.15a. Each string is then associated with another string belonging to the same class and which is the closest one regarding the classical distance of Levenshtein. These set of pairs represents the learning set and is provided to the conditional transducer in order to perform the training.

The goal of this training is to optimize, for the set of learning string pairs, the probabilities to pass from a symbol of the first string to a symbol of the second

## 2 State of the art

---

**EXPECTATION-STEP**( $x^T, y^V, c, \delta, S$ )

---

**Input:**  $y^V$  and  $x^T$  two strings  
 $c$  the primitive edit costs table  
 $\delta$  the auxiliary primitive edit costs table  
 $S$  the training string pair set

**Result:** the auxiliary primitive edit costs of  $\delta$  are optimized

```

1   $\alpha \leftarrow \text{FORWARD-EVALUATE}(x^T, y^V)$ 
2   $\beta \leftarrow \text{BACKWARD-EVALUATE}(x^T, y^V)$ 
3  if  $\alpha_{T,V} == 0$  then
4    return
5  end
6   $\delta(\epsilon|\epsilon) \leftarrow |S|$ 
7  for  $t \leftarrow 0$  to  $T$  do
8    for  $v \leftarrow 0$  to  $V$  do
9      if  $t > 0$  then
10        $\delta(\epsilon|x_t) \leftarrow \delta(\epsilon|x_t) + \alpha_{t-1,v} c(\epsilon|x_t) \beta_{t,v} / \alpha_{T,V}$ 
11      end
12      if  $v > 0$  then
13        $\delta(y_v|\epsilon) \leftarrow \delta(y_v|\epsilon) + \alpha_{t,v-1} c(y_v|\epsilon) \beta_{t,v} / \alpha_{T,V}$ 
14      end
15      if  $t > 0$  et  $v > 0$  then
16        $\delta(x_t, y_v) \leftarrow \delta(x_t, y_v) + \alpha_{t-1,v-1} c(x_t, y_v) \beta_{t,v} / \alpha_{T,V}$ 
17      end
18    end
19  end

```

---

**Figure 2.23** The stochastic Levenshtein's algorithm : EXPECTATION-STEP.

## 2 State of the art

---

**MAXIMIZATION-STEP**( $c, \delta$ )

---

**Input:**  $c$  the primitive edit costs table  
 $\delta$  the auxiliary primitive edit costs table  
**Result:** the primitive edit costs of  $c$  are maximized regarding  $\delta$

```

1   $N \leftarrow 0$ 
2  forall the  $a \in E \cup \epsilon$  do
3     $N(a) \leftarrow 0$ 
4  end
5  forall the  $a \in E \cup \epsilon$  do
6    forall the  $b \in E \cup \epsilon$  do
7       $N \leftarrow N + \delta(b|a)$ 
8    end
9  end
10 forall the  $a \in E$  do
11   forall the  $b \in E \cup \epsilon$  do
12      $N(a) \leftarrow N(a) + \delta(b|a)$ 
13   end
14 end
15 forall the  $b \in E$  do
16    $N(\epsilon) \leftarrow N(\epsilon) + \delta(b|\epsilon)$ 
17 end
18 forall the  $a \in E$  do
19   forall the  $b \in E$  do
20      $c(b|a) \leftarrow \frac{\delta(b|a)}{N(a)} \frac{N - N(\epsilon)}{N}$ 
21   end
22 end
23 forall the  $b \in E$  do
24    $c(b|\epsilon) \leftarrow \frac{\delta(b|\epsilon)}{N}$ 
25 end
26 forall the  $a \in E$  do
27    $c(\epsilon|a) \leftarrow \frac{\delta(\epsilon|a)}{N(a)} \frac{N - N(\epsilon)}{N}$ 
28 end
29  $c(\epsilon|\epsilon) \leftarrow \frac{N - N(\epsilon)}{N}$ 

```

---

**Figure 2.24** The stochastic Levenshtein's algorithm : MAXIMIZATION-STEP.



## 2 State of the art

string, for each symbol of the alphabet. The insertions and deletions are also concerned since the empty are taken into account in addition to the alphabet. These probabilities correspond to the primitive edit costs for which we want determine the matrix.

In order to optimize these probabilities, the algorithm uses auxiliary functions. These auxiliary functions are the *forward* and *backward* functions allowing both to compute the probabilities of a string conditioned to a second one. The forward function realizes this operation by computing — as indicated by its name — forward, meaning by starting from the first symbol of the string to end at the last one while the backward function performs the opposite by going from the last symbol to the first one. These two functions are based on the costs table ; they are both recursive and can be computed with dynamic programming techniques.

The primitive edit costs are determined with the help of an expectation-maximization algorithm. It consists in an expectation step and a maximization step repeated until a stop criterion is met. The expectation step allows to determine an auxiliary costs matrix whose the auxiliary costs are processed during the maximization step in order to determine the real primitive edit costs matrix. The stop criterion is chosen to be the absence of consequent variation (less than 1%) of the conditional probabilities between two successive steps of the algorithm.

Once the costs matrix determined, the training is finished and the recognition can begin. For this purpose, the probabilities of the input gesture conditioned to each training gesture are computed by using the forward and backward function. The class attributed to the input gesture is the class of the training gesture maximizing the conditional probability.

In the end, the stochastic Levenshtein's algorithm can offer very good recognition rates but can suffer of slowness during its training step since a lot of training examples are needed to reach its best working.

### 2.2.5 Extensions of Rubine

#### E-Rubine

E-Rubine is an algorithm elaborated thanks to iGesture (Signer et al. [2007]). It improves the basic Rubine's algorithm by relying on an extended set of fea-

## 2 State of the art

tures. These features support both single-stroke and multi-stroke gestures and are here elicited.

### Single-stroke

- Number of stop points
- Distance from start to centre point in relation to the diagonal
- Direction of the first half of the stroke
- Direction of the second half of the stroke
- Angle between the first and the second half of the stroke
- Distance from start to end point in relation to the diagonal

### Multi-stroke

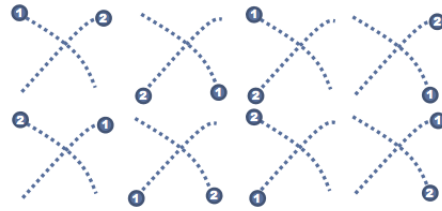
- Number of strokes
- Straightness
- Total distance between strokes
- Angle between strokes
- Stroke distance in relation to each other

## 2.2.6 Extensions of One Dollar

### N Dollars

N Dollars algorithm (Wobbrock and Anthony [2010]) extends the One Dollar algorithm to handle multi-stroke gestures. To achieve that, N Dollars computes and stores each multistroke's unistroke permutations. Each permutation represents one possible combination of stroke order and direction, i.e. one of the 8 possibilities in Figure 2.25, which is then made into a unistroke by simply connecting the endpoints of component strokes.

## 2 State of the art



**Figure 2.25** The 8 possibilities for a two-stroke  $x$ . The numbered dots indicate stroke order and beginnings.

This is, in effect, treating a multistroke as if it were fundamentally a unistroke, but where part of the stroke is made away from the sensing surface, i.e., by following the user's hand. Thus, a two-stroke “x” permutes into the 8 unistrokes of Figure 2.26.

Each permutation is then handled as an unistroke gesture using Euclidean distance as in One Dollar. The score for a multistroke template is the best of its unistroke scores of permutations.

This improved variant of One Dollar algorithm is not be considered in the comparative study as it was released after it was conducted.



**Figure 2.26** The 8 unistroke permutations for a two-stroke  $x$  based on the two-stroke gestures in Figure 2.25.

### 2.2.7 Extensions of original Levenshtein

#### Normalization of the distance

The distance of Levenshtein previously described works very well for many applications. However, it is not advised for other ones such as recognizing noisy subsequences since they can suffer of big differences of lengths. It should be thus convenient to introduce some normalizations of the chains of characters representing the gestures in order to take their length into account in a better

## 2 State of the art

way. It is obvious that an error in the comparison of two small sequences is more critical than in two big ones.

Yujian and Bo describe in their article (Yujian and Bo [2007]) two well-known methods of normalization, and a new method called *Normalized Generalized Levenshtein Distance (N-GLD)*. The advantage of their new method is that it fulfills the triangle inequality and represents a genuine metric between two strings.

Yujian and Bo define the Generalized Levenshtein Distance  $GLD(X, Y)$  by the minimum cost of transforming one string into another one through a sequence of weighted edit operations. More formally, for an editing path  $P_{X,Y}$  between  $X$  and  $Y$ , and a weight  $W(P_{X,Y})$  representing the addition of the weights of the edit transformations between  $X$  and  $Y$ , we have :

$$GLD(X, Y) = \min \{W(P_{X,Y})\}$$

The first normalization method based on the editing path lengths can be defined as :

$$NED_1(X, Y) = \min \left\{ \frac{W(P_{X,Y})}{L(P_{X,Y})} \right\}$$

where  $L(P_{X,Y})$  is the number of elementary edit operations described by  $P_{X,Y}$

The second method, based on the string lengths can be defined as :

$$NED_2(X, Y) = \min \left\{ \frac{W(P_{X,Y})}{|X| + |Y|} \right\}$$

In order to obtain the new metric N-GLD, let us first define the *Generalized Levenshtein Similarity (GLS)* between  $X$  and  $Y$  as :

$$GLS(X, Y) = \frac{\alpha(|X| + |Y|) - GLD(X, Y)}{2}$$

where  $\alpha$  is the maximum cost between insertion and deletion.

The metric N-GLD can then be defined as :

$$N-GLD = \frac{GLS(X, X) + GLS(Y, Y) - 2GLS(X, Y)}{GLS(X, X) + GLS(Y, Y) - GLS(X, Y)}$$

The results obtained by Yujian and Bo revealed relatively similar accuracies for the different normalizations, but with better rates than for the Levenshtein without normalization. The N-GLD metric yields slightly better results if the triangle inequality is violated to a certain degree.

## 2 State of the art

### Modification of the substitution costs

In the original version of the Levenshtein algorithm, the insertion costs as the deletion costs of symbol are fixed to a value of 1 while the substitution costs between two symbols can be 0 or 1 depending on the similarity or dissimilarity of the symbols. The idea for this extension (suggested in Oncina and Sebban [2006]) is to fix the substitution costs in order to improve the efficiency of the algorithm. This modification is only suited in the particular case where the chains of characters are built with the point-by-point directions of the gestures (see compass rose of directions in Figure 2.15a). The costs are thus modified in a way that their value represents the difference between directions. In other words, the more two directions are opposite, the more the weight of substituting one direction the other one is big. The matrix presented in Table 2.3 shows the different costs of substitution depending of the involved directions.

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 3 | 2 | 1 |
| 1 | 1 | 0 | 1 | 2 | 3 | 4 | 3 | 2 |
| 2 | 2 | 1 | 0 | 1 | 2 | 3 | 4 | 3 |
| 3 | 3 | 2 | 1 | 0 | 1 | 2 | 3 | 4 |
| 4 | 4 | 3 | 2 | 1 | 0 | 1 | 2 | 3 |
| 5 | 3 | 4 | 3 | 2 | 1 | 0 | 1 | 2 |
| 6 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | 1 |
| 7 | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 |

**Table 2.3** Matrix of adapted costs of substitution.

## 2.3 Models

The development of user interfaces (UIs), ranging from early requirements to software obsolescence, has become a time-consuming and costly process. Typically, the graphical user interface (GUI) of an interactive system represents about 48% of the source code, requires about 45% of the development time and 50% of the implementation time, and covers 37% of the maintenance time (Myers and Rosson [1992]).

Model-Based User Interface Development (MBUID) is one approach that aims at decreasing the effort needed to develop user interfaces while ensuring quality. The purpose of Model-Based Design is to identify high-level models that

## 2 State of the art

allow designers to specify and analyse interactive software applications from a more semantic oriented level rather than starting immediately to address the implementation level. This allows them to concentrate on more important aspects without being immediately confused by many implementation details and then to have tools which update the implementation in order to be consistent with high-level choices.

Different frameworks have been developed to conceptually capture the important aspects of a MBUID process. Among them, one is now widely accepted in the HCI Engineering community as a reference for structuring and classifying model-based development processes of user interfaces that support multiple contexts of use: the Cameleon Reference Framework.

Another interesting contribution is the iGesture model. It is used to describe the gesture modelling under the iGesture framework.

### 2.3.1 Cameleon Reference Framework

The Cameleon Unified Reference Framework was produced by the EU-funded Cameleon FP5 Project (FP5-IST4-2000-30104). CAMELEON (Calvary et al. [2003], UsiXML-D1.3, recommended by W3C in Fonseca et al.), describes a framework that serves as a reference for classifying user interfaces supporting multiple targets, or multiple contexts of use in the field of context-aware computing. The CAMELEON Framework provides a unified understanding of context-sensitive user interfaces rather than a prescription of various ways or methods of tackling different steps of development. Rather, the framework structures the development life cycle into four levels of abstraction, from the task specification to the running interface (see Figure 2.27):

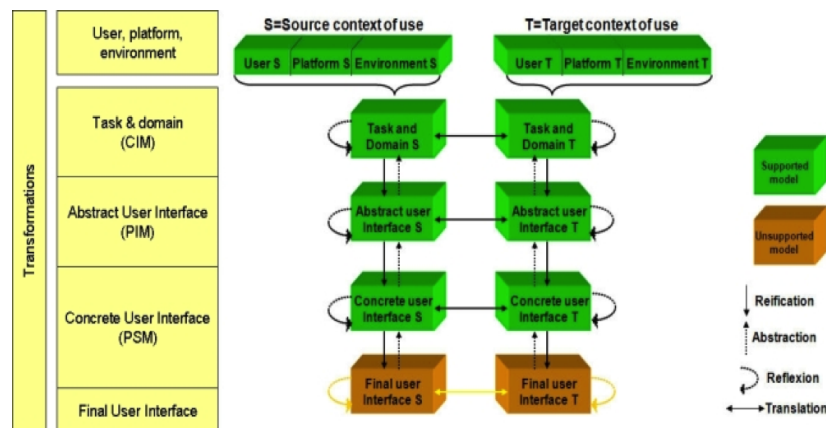
- The Task and Concepts level (corresponding to the Computational-Independent Model–CIM–in MDE) which considers the logical activities that need to be performed in order to reach the users' goals. Often they are represented hierarchically along with indications of the temporal relations among them and their associated attributes.
- The Abstract User Interface (AUI) (corresponding to the Platform-Independent Model–PIM– in MDE) is an expression of the user interface in terms of interaction spaces (or presentation units), independently

## 2 State of the art

of which interactors are available and even independently of the modality of interaction (graphical, vocal, haptic ...). An interaction space is a grouping unit that supports the execution of a set of logically connected tasks.

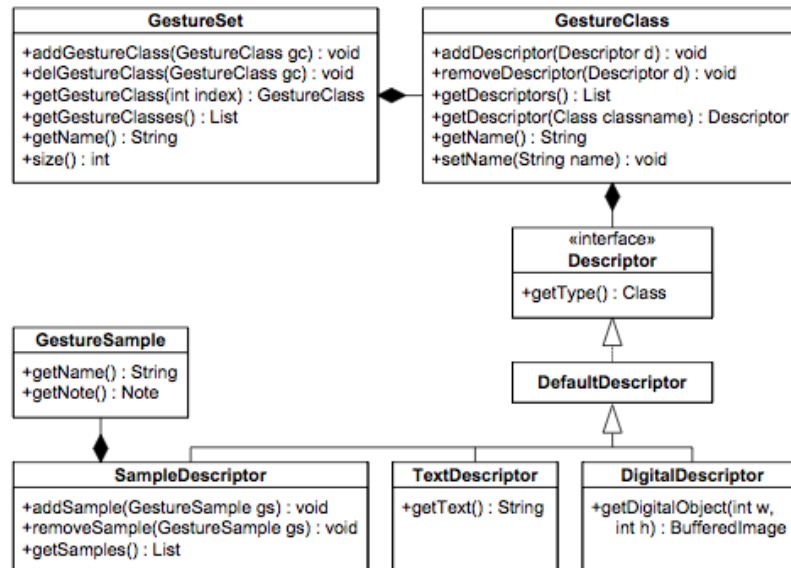
- The Concrete User Interface (CUI) (corresponding to the Platform-Specific Model–PSM– in MDE) is an expression of the user interface in terms of “concrete interactors”, that depends on the type of platform and media available and has a number of attributes that define more concretely how it should be perceived by the user. “Concrete interactors” are, in fact, an abstraction of actual user interface components generally included in toolkits.
- The Final User Interface (FUI) (corresponding to the code level in MDE) consists of source code, in any programming language or mark-up language (e.g. Java or HTML5). It can then be interpreted or compiled. A given piece of code will not always be rendered on the same manner depending on the software environment (virtual machine, browser ...). For this reason, we consider two sublevels of the FUI: the source code and the running interface.

These levels are structured with a relationship of reification going from an abstract level to a concrete one and a relationship of abstraction going from a concrete level to an abstract one. There can be also a relationship of translation between models at the same level of abstraction.



**Figure 2.27** Caméléon Reference Framework.

### 2.3.2 iGesture



**Figure 2.28** iGesture user interface.

Figure 2.28 shows the gesture modelling used by iGesture (Signer et al. [2007]). The **GestureClass** class represents an abstract gesture characterised by its name and a list of descriptors. A set of gesture classes is grouped in a **GestureSet** which can then be used to initialize an algorithm. The **Descriptor** interface has to be implemented by any gesture descriptor. A single **GestureSample** is an instance of a gesture and contains the note captured by an input device. In addition to the sample descriptor, a textual description specifying the directions between characteristic points of a gesture can be used, as well as a digital descriptor representing the gesture in terms of a digital image.

## 2.4 Approaches

User interface development methodologies, processes, and development life cycles have been already the subject of extensive research and comparisons. Among them are Kikuchi et al., Harston and Boehm-Davis and Humayoun et al.. The application of mainstream software development methodologies (e.g., a list of them is available at <http://www.itinfo.am/eng/software->



## 2 State of the art

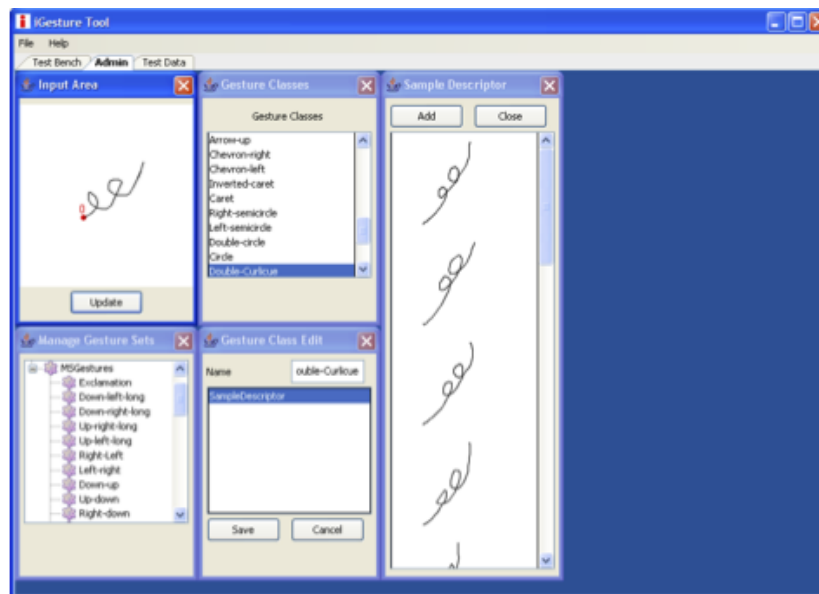
development-methodologies/) borrowed from software engineering to user interface development (e.g., agile methods or extreme programming) falls in the same case. Since these comparisons are beyond the scope of this thesis, we refer to the literature for further details.

### 2.5 Tools

This section gives an overview of the existing tools providing help at one or different levels during the creation of user interfaces integrating pen-based gesture recognition.

#### 2.5.1 iGesture

iGesture (Signer et al. [2007]) is a Java-based gesture recognition framework focusing on extensibility and cross-application reusability by providing an integrated solution that includes the iGesture recognition framework as well as the iGesture tool component for the creation of customized gesture sets.

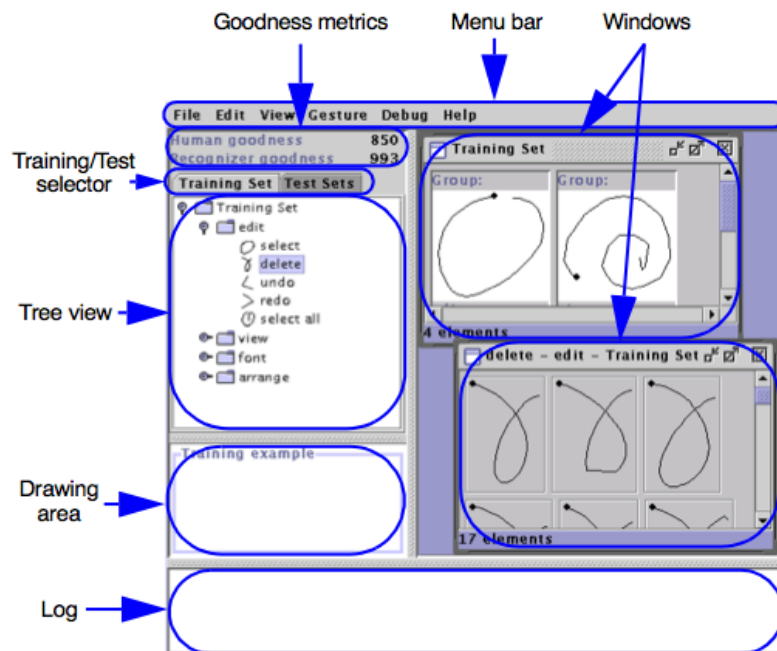


**Figure 2.29** iGesture user interface.

Figure 2.29 depicts the user interface (admin part) of the tool. Its first goal is to provide the functionality to acquire gestures from input devices. It is also possible to incorporate recognizers and test them with the acquired gestures.

## 2.5.2 quill

Users of current systems sometimes find that gestures are hard to remember and are often misrecognized. Quill (Long et al. [2001]) assists designers in creating and improving gestures for pen-based interfaces, both in terms of computer recognizability and likelihood of confusion among gestures by people. quill helps designers improve their gestures by warning them about potential problems with their gestures and offering advice about how to fix them.



**Figure 2.30** quill user interface.

Figure 2.30 shows an overview of the quill interface. It allows to create gestures and organize them in sets. With the gesture sets, recognition can be tested through the drawing area. The result of the recognition is then shown in the log console. Throughout this process, quill monitors the gestures the designer enters, and analyzes them for problems. If it discovers a problem with human

## 2 State of the art

similarity, it is reported immediately. Additionally, quill provides an indication of the quality of the gesture set in terms of two *goodness metrics*. One metric measures how well the recognizer will be able to disambiguate the gestures, and the other indicates how good the gestures are for humans.

### 2.5.3 DataManager

DataManager (Schmieder et al. [2009]) is a toolkit to automatically evaluate recognition algorithms. Its provides a way to compare different algorithms and determine which is the best one for new sketch domains.

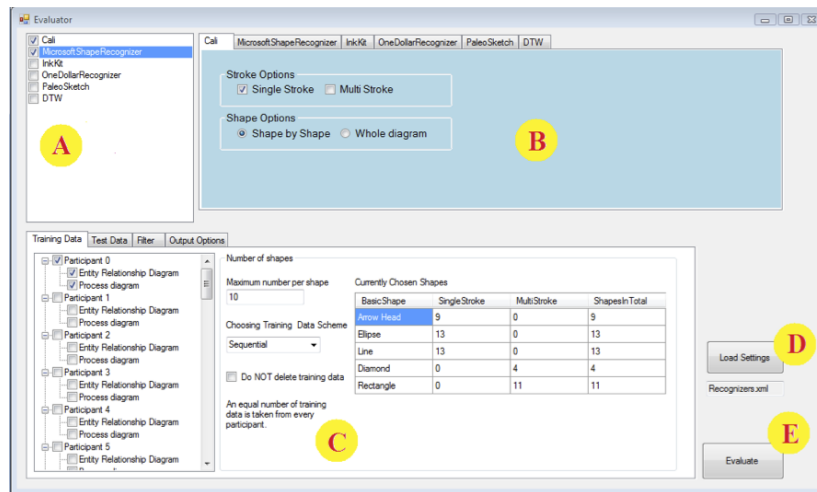


Figure 2.31 DataManager user interface.

Figure 2.31 gives a representation of DataManager tool. To evaluate new algorithms, the platform can be employed by simply integrating the new algorithm and testing it on data other recognizers have been tested on. Additionally, by providing the means to collect new data and label it, new domains can be tested on the implemented algorithms to determine the most appropriate one.

### 2.5.4 SOUSA

Although existing domain-specific datasets are readily available, most sketch recognition researchers are forced to collect new data for their particular domain. Sousa (Paulson et al. [2008]) is a general-purpose sketch collection and

## 2 State of the art

verification tool that allows researchers to design custom user studies through an online applet.

The screenshot shows the 'User Study Setup' applet window. It has two tabs: 'Collection Study' (selected) and 'Verification Study'. The 'Save Directory' field is set to 'http://srLcsdl.tamu.edu/userstudy/mathsigns'. The 'Study Title' is 'Math Signs'. The 'Basic XML Format' radio button is selected. Under 'User Information', there are three items: 'Item 1: Method of input (pen or mouse?)', 'Item 2: Age', and 'Item 3: Occupation', each with a text input field and a button labeled 'X'. An 'Add Info...' button is below. The 'Shape Information' section contains a table with four rows of shape definitions. At the bottom are 'Add Shape...', 'New', 'Load', and 'Save' buttons. The status bar at the bottom says 'Java Applet Window'.

| Description                        | Directory | NumToDraw | MaxStrokes | Images | X |
|------------------------------------|-----------|-----------|------------|--------|---|
| Description 1: Addition sign       | add       | 3         | -1         | 1      | X |
| Description 2: Subtraction sign    | sub       | 3         | -1         | 1      | X |
| Description 3: Multiplication sign | mult      | 6         | -1         | 2      | X |
| Description 4: Division sign       | div       | 6         | -1         | 2      | X |

**Figure 2.32** Sousa user interface.

Figure 2.32 shows a screenshot of the setup applet for creating a collection study through Sousa tool. It allows users to add shapes and label them with various informations.

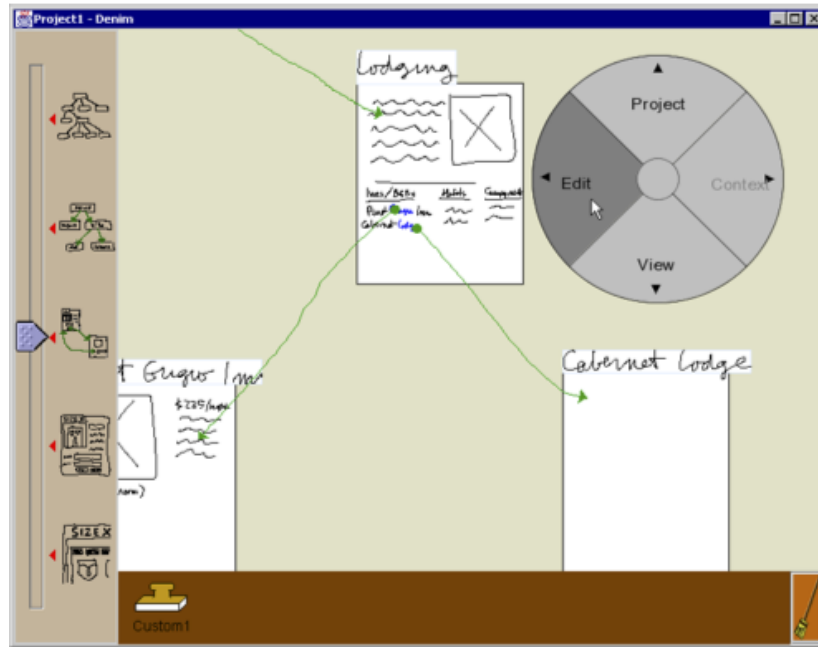
### 2.5.5 SATIN

SATIN (Hong and Landay [2000]) was created to facilitate the creation of pen-based application. More specifically, it is Java-based toolkit designed to support the creation of applications that leverage the informal nature of pens. This support includes a scenegraph for manipulating and rendering objects, support for zooming and rotating objects, switching between multiple views of an object, integration of pen input with interpreters, libraries for manipulating ink strokes, widgets optimized for pens, and compatibility with Java's Swing toolkit. SATIN includes a generalized architecture for handling pen input, consisting of recognizers, interpreters, and multi-interpreters.

SATIN is not intended to be used as itself to easily create user interfaces integrating gesture recognition with the help of WYSIWYG editor. However, it

## 2 State of the art

is a very flexible and extensible toolkit allowing to add for example custom recognizers. An example of application built on top of SATIN is DENIM (see Figure 2.33), a website design tool aimed at the early stages of information, navigation, and interaction design.



**Figure 2.33** DENIM application.

## 2.6 Summary

Different models and tools exist to support pen-based at different stages of the constitution of a gesture recognition mechanism: creation of gesture, labelling of gesture, constitution of datasets, creation of recognition algorithms, training of recognizers, gesture testing regarding the recognizers. Table 2.4 has been expressed based on Harvey's balls (Kellen) and shows the strengths and weaknesses of the different frameworks presented in the state of the art. The more a ball is filled, the more a feature is supported by the corresponding framework.

We then elicit the shortcomings we identified and derive the requirements aimed at addressing them, and finally conclude.

## 2 State of the art

|                                       | iGesture                                                                          | quill                                                                             | D.M.                                                                               | SOUSA                                                                               | SATIN                                                                               |
|---------------------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| PBUI specification language           |  |  |  |  |  |
| Guidance on team collaboration        |  |  |  |  |  |
| Guidance on development life-cycle    |  |  |  |  |  |
| Algorithmic knowledge                 |  |  |  |  |  |
| Usability knowledge                   |  |  |  |  |  |
| Software support for PBUI development |  |  |  |  |  |
| Software repositories of PBUI assets  |  |  |  |  |  |

**Table 2.4** Comparison of the existing methods.

### 2.6.1 Shortcomings

This part describes the shortcomings identified in the state of the art, divided into the conceptual, methodological and implementational points of view.

#### Conceptual

- *Lack of language enabling designers to specify gestural interfaces.* To our knowledge, very few languages allow to formally specify a user interface integrating pen-based gesture recognition. This leads to limited or absence of conceptual modelling of such interfaces.

#### Methodological

- *Limited methodological guidance for designing and developing interfaces integrating pen-based gestures.* The roles of the persons involved in the creation of user interfaces integrating pen-based gestures recognition are often not well delimited. This leads to a bad separation of roles and non efficient collaboration.
- *Limited knowledge on sketch recognition algorithms for user interface designers and developers.* Designers and developers rarely have knowledge specific to sketch recognition algorithms. More particularly, they

## 2 State of the art

face the difficulty of selecting the best algorithm regarding the UI they want to create, the difficulty of parametrizing and optimizing the algorithms, or combining them to obtain better results. Moreover, only a few comparative studies exist for helping them in the hard task of calibrating the recognition mechanism.

- *Limited involvement of usability knowledge of user interface including pen-based gestures.* Little or no usability guidelines are provided for selecting the right gestures for the right interfaces and context. This leads to a potential lack of user interaction for the end users.

### Implementational

- *Limited software supporting the development lifecycle of user interfaces integrating pen-based gestures.* To our knowledge, very few tools support the whole lifecycle of the process of creation of user interfaces integrating pen-based gesture recognition. This leads to a loss of time for developer that then need to create such interface with their own tool not suited for this specific purpose.
- *Incompleteness for all knowledge required for creating user interfaces integrating pen-based gestures.* The knowledge required to create user interfaces integrating pen-based gestures is specific and complex. Besides the usual knowledge associated to classical graphical user interfaces, it consists in the algorithmic behind the recognition engine and the repository of gestures to be recognized. These assets are complex to build, requiring time and skills not always available.

### 2.6.2 Requirements

Based on the shortcomings previously stated, this part elicits the expected requirements concretized as a model (conceptual), approach (methodological) and tool (implementational).

#### Conceptual

- *Concrete level of abstraction.* The model should capture gestural interfaces at the concrete level of abstraction.

## 2 State of the art

- *Gesture integration.* The model should support integrating gestures with the rest of the user interface development.

### Methodological

- *Explicit collaboration guidance.* The method should provide creators of gestural user interfaces with explicit guidance on how to enact the development lifecycle.
- *Explicit recognition mechanisms guidance.* The method should help the creators of user interfaces integrating pen-based gesture recognition to select the right recognition mechanism (recognition algorithm and gesture dataset) for the right use case.

### Implementational

- *Concrete level of abstraction support.* The tool should support the designers and developers for creating user interfaces at the concrete level of abstraction.
- *Gesture integration support.* The tool should support the designers and developers for the full process of integrating gestures in user interfaces.
- *Recognition mechanism extendibility.* The tool should enable gesture experts to enrich its recognition mechanisms with new algorithmic knowledge and new gesture datasets.

### 2.6.3 Conclusion

Different existing frameworks provide assistance for some part of the process leading to the creation of user interfaces integrating pen-based gesture recognition. However, to our knowledge, no method fully supports the whole set of steps required to create the pen-based recognition mechanisms and let teams of people with different roles (gesture specialists, programmers, designers) easily include them in graphical user interfaces. The three next chapters present the contributed work in this direction: Chapter 3 shows a way to understand and



## 2 State of the art

parametrize the right recognition mechanisms for the right use case, Chapter 4 details the proposed modelling to support the conception of such user interfaces, and Chapter 5 wraps this knowledge in a tool to actually enable the creation of pen-based user interfaces.



## **Chapter 3** Selection of appropriate recognition mechanism

In order to support the inclusion of pen-based gestures in user interfaces, we first need to set up a recognition mechanism. The goal of this chapter is to constitute a dataset of useful gestures to include in user interfaces, choose different algorithms to be used to perform recognition on these gestures, and then provide a way to the actors involved in the creation of pen-based user interfaces to select the right gestures and algorithms best suited for their needs.

The selected gestures and algorithms were already described in the state of the art. During their implementation, we noticed different variations of these algorithms that can be very useful for pen-based recognition. The following sections first describe these refinements, then explain the implementation of the tool used for exploring the different recognition mechanisms, and finally provide a comparative study to guide the pen-based user interface creators selecting the right recognition mechanism depending on the context of their application.

### **3.1 Refined algorithms**

This section presents the imagined refinements of the algorithms of Rubine, One Dollar and Levenshtein (classical and stochastic).

### 3 Selection of appropriate recognition mechanism

#### 3.1.1 Rubine

##### Consideration of new features

Pressure, inclination, or orientation of the pen are measurable features thanks to the graphics tablet. Extend the feature vector of Rubine in order to integrate them can thus represent a potential improvement. However, because of the difficulties met to integrate the feature in a pertinent way, these extension has not been selected.

##### Gesture preprocessing

One Dollar's algorithm executes four types of gesture preprocessing before comparing them : a resampling, a rotation, a rescaling and a translation. Testing these transformations on gestures before executing the Rubine's algorithm constitutes an interesting modification to observe.

#### 3.1.2 One Dollar

##### Other Distances

Once the gestures preprocessed, the One Dollar's algorithm proceeds to a point-by-point quadratic distance between the input gesture and the training gestures in order to classify the input gesture. Even if this kind of distance allows to model 2D distances and constitutes the principle of One Dollar, other distances could be tested. This is what is proposed in the next section with the modification attempting to preprocess the gestures before using the distance of Levenshtein.

##### Variation of the number of nearest neighbours

The One Dollar classification of an input gesture consists in attributing it the class of the closest training gesture considering point-by-point quadratic distance. Instead of only taking one training example into account, a possible improvement can be considering more training examples. In this way, the more

### 3 Selection of appropriate recognition mechanism

represented class inside these closest training gestures would be selected. This method is known as *kNN*, the *k-Nearest neighbours* method. The advantage of this method is that it does not suffer from any high additional computation time since the results for the different numbers of nearest neighbours can be obtained in only one iteration. Indeed, the computation consists in elaborating a list of training gestures sorted by their distance to the input gesture : the first item is considered for  $kNN = 1$ , the two first ones for  $kNN = 2$ , etc.

#### 3.1.3 Original and stochastic Levenshtein

##### Consideration of new features

As for Rubine's algorithm, integrate pressure, inclination, and orientation of the pen can be useful. However, the way the algorithm will be impacted is different. For both original and stochastic versions of Levenshtein, these features will be modeled through different generations of the chains of character representing the gestures. More particularly, the symbols of the chains will be associated with point-by-point differences of pressure, inclination, or orientation rather than point-by-point directions. Furthermore, combination of these characteristics can be examined.

##### Multi-stroke gestures

During the sketching of a gesture, the pen can sometimes be lifted up, involving the apparition of a gap between two successive points. The gesture becomes no longer uni-stroke and is then multi-stroke. In the initial version of the original and stochastic versions of the Levenshtein's algorithm, two successive points surrounding a gap are processed in the same way than the other point, i.e. by attributing one symbol associated to the direction between them in the chains of characters representing the gesture. For this refinement of the algorithm, the idea is to model the gap in order it would be taken into account. For this purpose, a special symbol representing a piece of gap will be added to the chain of characters a number of times when the pen is lifted up. This number of times depends on some adding frequency (to be determined empirically) and the distance between the two successive points surrounding the gap.

### 3 Selection of appropriate recognition mechanism

#### Gesture preprocessing

One Dollar's algorithm executes four types of gesture preprocessing before comparing them : a resampling, a rotation, a rescaling and a translation. As for Rubine's algorithm, testing these transformations on gestures before executing the original and stochastic version of Lenvenshtein's algorithm constitutes an interesting modification to observe.

## 3.2 Implementation

This section presents the tool created to explore and benchmark different recognition mechanisms.

### 3.2.1 Tools overview

#### Generalities

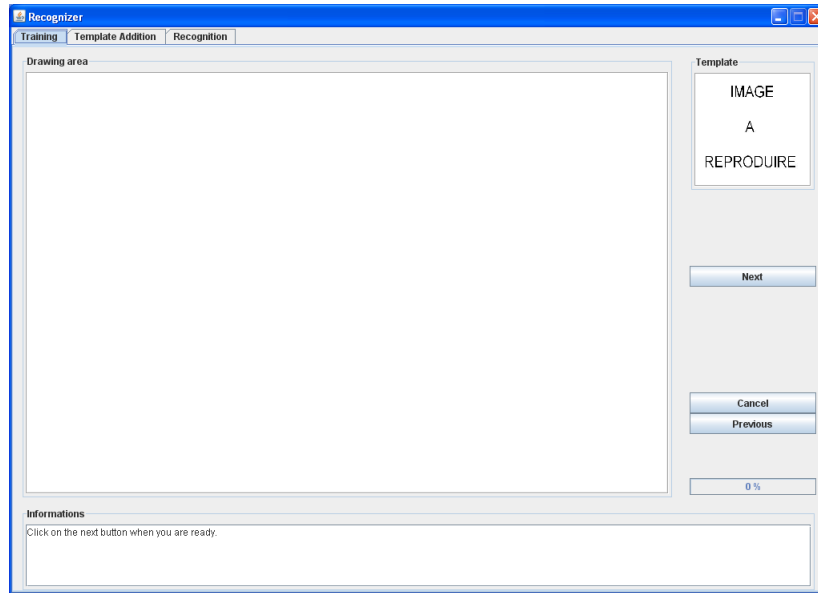
The tool is composed of a graphical part and a non-graphical part. The graphical interface is divided into three tabs. The first one is intended to train the system by the user by allowing it to reproduce a set of templates. The second tab also deals with training of the system since it allows a more "manually" training by allowing the user to add examples of the class of its choice one-by-one. The third tab concerns the recognition in which the user can specify several options (for the algorithm, the type of training or the type of gestures) and sketch a gesture in order to test the recognition. Finally, the non-graphical part is the benchmarker, i.e. a set of Java classes through which the comparative study can be performed.

#### Training

Figure 3.1 shows the tab allowing the user to add a complete set of training examples of each class.

The user has to reproduce the template (displayed in the top-right of the window) into the sketching zone (main area of the window). Then the user can

### 3 Selection of appropriate recognition mechanism



**Figure 3.1** Training tab.

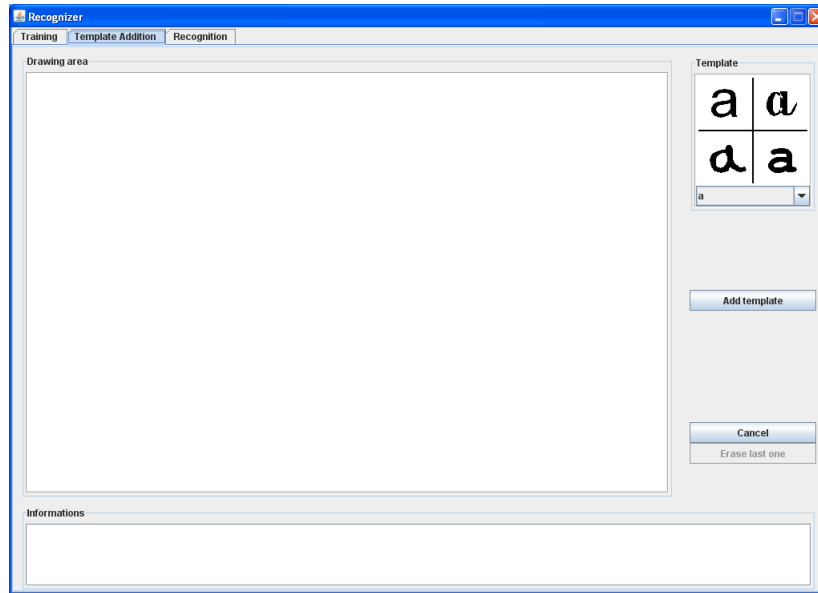
press the button *Next* in order to validate its sketch and go to the next template. It can also press the button *Clear* in order to clear the sketching zone and draw again the template. The button *Previous* gives the possibility to come back to the previous template and sketch it again (e.g. when the user realizes that its sketch is wrong just after validated it). Just below the buttons, a progress bar indicates to the user the proportion of templates already drawn and that it still has to draw. The number of templates is 30 for the signatures and 10 for all the other classes, with a total of 640 — these templates will be explained in more details in section 3.3.1. Finally the console displays each template in English in order to avoid all ambiguity (e.g., the letter “o” and the digit “0”).

#### One-by-one example adder

Figure 3.2 shows the tab allowing the user to add one by one training examples for a desired class.

As for the complete training, the template is displayed in the top-right of the window. The difference is that the class of the template is chosen by the user through the combobox just below. The user can validate its sketching with the button *Add template*, that is then associated with the chosen class. The button *Clear* allows to clear the drawing area and the button *Redraw last example*

### 3 Selection of appropriate recognition mechanism



**Figure 3.2** One-by-one example adder tab.

gives the possibility to come back to the last example drawn in order to draw it again. The console displays information about the example addition.

#### Recognizer

Figure 3.3 shows the tab allowing the user to proceed to the recognition of an example.

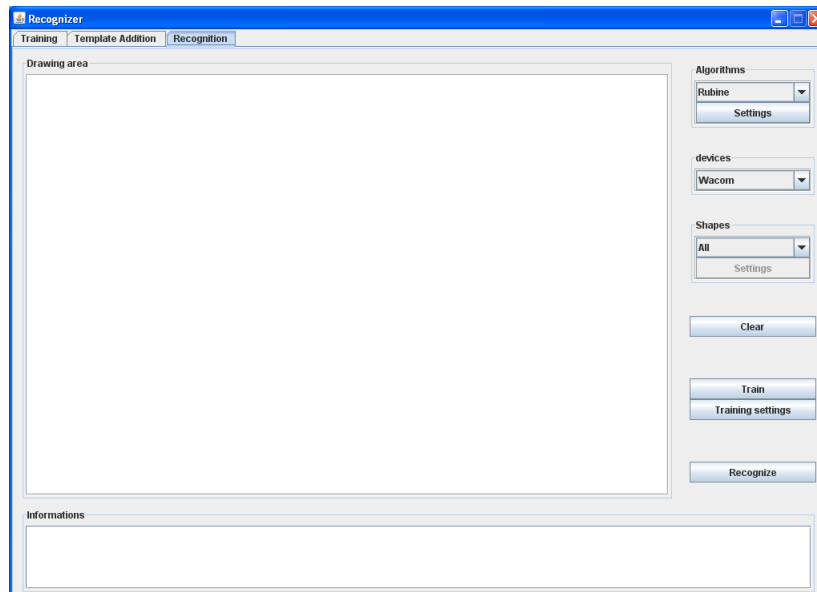
The first menu at the top-right of the window can be used to specify the algorithm among Rubine, One Dollar, Levenshtein and stochastic Levenshtein. The button *Settings* just below opens a new window in which the algorithm options can be configured.

The window presented in Figure 3.4(a) allows to configure the Rubine's algorithm. The available options concern the gesture preprocessing : resampling, rotation, rescaling and translation.

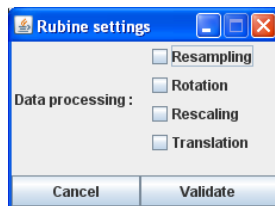
The window presented in Figure 3.4(b) allows to configure the One Dollar's algorithm. The only available option is the variation of the number of nearest neighbours to take into account during the recognition.



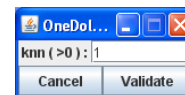
### 3 Selection of appropriate recognition mechanism



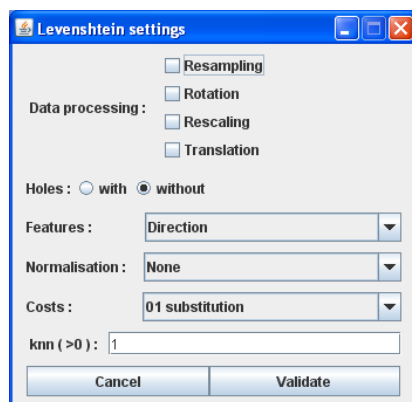
**Figure 3.3** Recognizer tab.



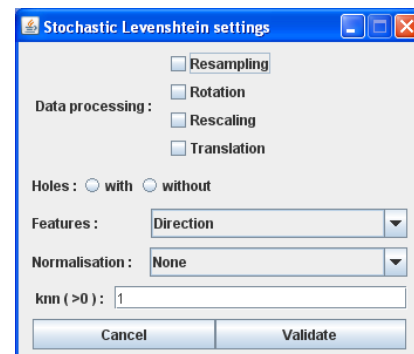
(a) Rubine



(b) One Dollar



(c) Levenshtein



(d) Levenshtein stochastic

**Figure 3.4** Windows for algorithm settings.

### 3 Selection of appropriate recognition mechanism

The window presented in Figure 3.4(c) allows to configure the original Levenshtein's algorithm. Here are the descriptions of the different available options :

- Data processing : as for Rubine's algorithm, preprocessing can be applied on gestures ;
- Holes : holes between the different strokes of the gesture can be taken into account if option *with* is selected, or not taken into account if option *without* is selected ;
- Features : the combobox gathers the different possibilities of features to be considered for the transformation of the gesture into string and are the directions (default), the pressure (normalized from 0 to 7), the inclination of the pen (normalized from 0 to 7), the orientation of the pen (normalized from 0 to 7), the difference between the pressures of two successive points (0 for increase, 1 for equality, 2 for decrease), the difference between the inclinations of the pen of two successive points (0 for increase, 1 for equality, 2 for decrease), the difference between the orientation of the pen of two successive points (0 for increase, 1 for equality, 2 for decrease), a combination between direction and difference of pressures (24 different symbols used) ;
- Normalization : the combobox gathers the three normalization for Levenshtein's algorithm explained before ;
- Costs : the combobox gives the possibility to consider basic substitution costs (0 for same symbols, 1 for different ones) or substitution costs adapted to directions (see table 2.3) ;
- Knn : the text area allows to specify the number of nearest neighbours to take into account.

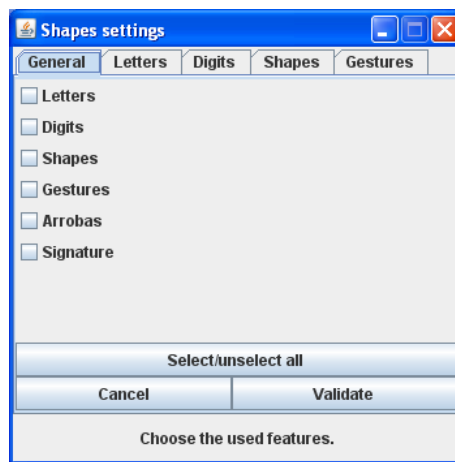
Furthermore, let us notice that the operations of rotation, rescaling and translation, as well as the consideration of adapted costs table can only be realized for the basic directions (default choice of Features option) and the uni-stroke gestures (option *without* of Holes option).

The window presented in Figure 3.4(d) allows to configure the stochastic Levenshtein's algorithm. The options are the same than the options available for the original version of Levenshtein's algorithm except for the costs.

### 3 Selection of appropriate recognition mechanism

Once the algorithm and its options specified, the next combobox of the recognition tab propose to choose the input device. Currently, the only available possibility is the Wacom<sup>®</sup> tablet but the tool could be extended in order to accept other devices.

The classes to consider for the recognition can be selected through the next combobox giving the choice between all classes, a particular family of classes (i.e. letters, digits, action gestures, geometrical shapes or signatures), or a total customization. This total customization can be specified by pressing the button *Settings* when *Customize* is selected, causing the opening of a new window (see Figure 3.5 in which each class can be manually chosen).

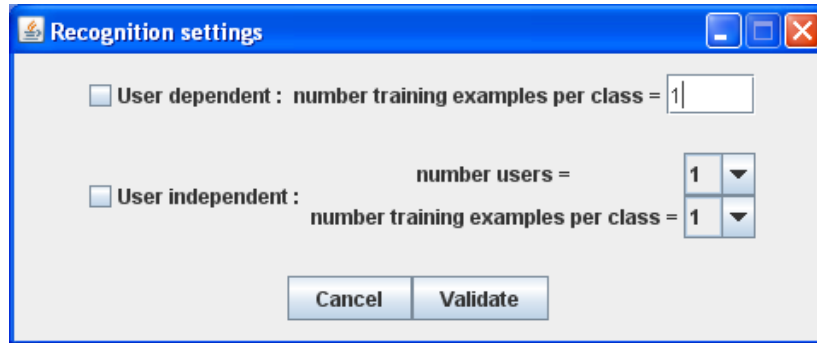


**Figure 3.5** Customization of the choice of the gesture classes.

Four buttons appears after the comboboxes. The first button *Clear* allows to clear the sketching area as in the two other tabs. The button *Train* launches the training phase. It is independent of the recognition phase in order to avoid training again the algorithm each time the user wants to test the recognition of a gesture for the same training options. Before launching the training, the options can be specified in the window opened by pressing the button *Training settings* (see Figure 3.6).

Selecting the *User dependent* option means that the examples sketched by the user are taken into account. The number of examples to consider for each class can be specified. If for some class this number are huger than the number of available examples, then all the existing examples will be used for this class. Selecting the *User independent* option means that the examples of the database are taken into account. The number of users as well as the number of training

### 3 Selection of appropriate recognition mechanism



**Figure 3.6** Window for training options.

examples can be specified.

Finally, the last button *Recognize* allows to proceed to the recognition phase. At first time, some information like the selected algorithm and its options are displayed in the console, then the recognition algorithm is actually launched and finally the result is displayed.

#### Benchmarkmarker

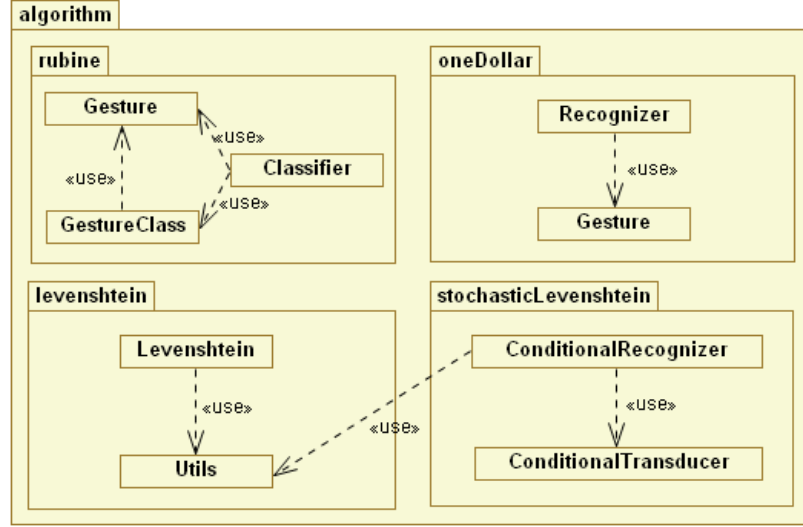
This part of the tool does not have graphical representation. It consists in a set of Java classes allowing to produce a benchmarking over the different algorithms. For this purpose, some specific methodology has been chosen for this study and will be explained in details in section 3.3.2. The architecture of the Java classes is described in the next section.

### 3.2.2 Software architecture

The diagram of classes depicted in Figure 3.8 represents the overall architecture of the system implemented in Java and more particularly with the Swing package. This diagram is a little bit simplified and only focus on important information : less important classes are not present and inheritance or implementation relationships are not showed.

The package *gui* constitutes the core of the graphical interface. The main class *GUI* represents the main window of the graphical interface. The classes *Recognizer*, *Trainer* and *TemplateAdder* are customized panels contained in the three

### 3 Selection of appropriate recognition mechanism



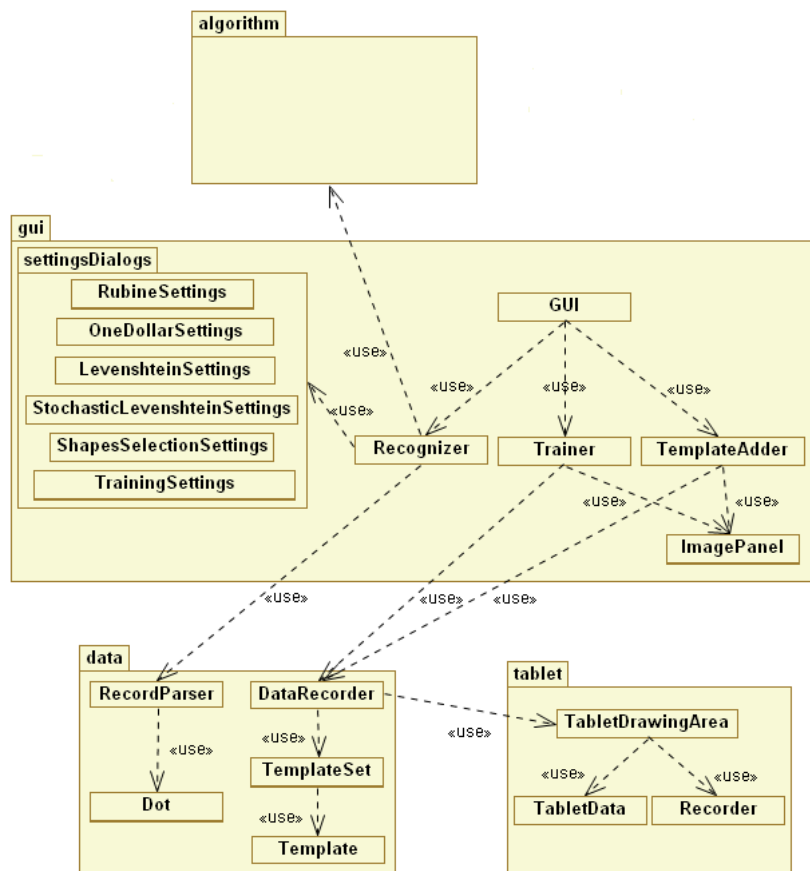
**Figure 3.7** Diagram of classes of package *algorithm*.

different tabs of the main window. The classes *Trainer* and *TemplateAdder* use both the class *ImagePanel* that is also a customization of a panel, allowing to display an image (the template) at the top-right of the two tabs. The subpackage *settingsDialogs* includes the different dialog windows allowing to configure the algorithms. This subpackage is thus used by the class *Recognizer*.

The recognition mechanism is implemented in the package *algorithm* detailed in Figure 3.7. Each algorithm has its own subpackage in which the main classes are represented. Rubine and OneDollar<sup>1</sup> have both a class *Gesture* allowing the representation of a gesture and its features while Levenshtein and stochastic Levenshtein do not need such a class because they directly deal with the string representation of the gestures. For Rubine, *GestureClass* gathers several gestures belonging to the same class and the *Classifier* takes as input different *GestureClass* for the training and a *Gesture* to be recognized. OneDollar adds directly the *Gesture* to its *Recognizer* in order to preprocess the learning gestures as explained in the theory. Concerning Levenshtein, the class *Utils* allows several operations of generation and transformation of the gesture in chains of characters. The class *Levenshtein* uses it to perform processing on gestures and proceed to training and recognition by computing Levenshtein distances. The computation of the string is based on the same class for the

<sup>1</sup>The algorithms of Rubine and One Dollar have been implemented and provided by Dr. Adrien Coyette.

### 3 Selection of appropriate recognition mechanism



**Figure 3.8** Diagram of classes for the graphical interface.

### 3 Selection of appropriate recognition mechanism

algorithm of stochastic Levenshtein. The class *ConditionalTransducer* allows the optimization of the weights on a group of training pairs of strings. It helps the *ConditionalRecognizer* in order to compute the matrix of substitution costs.

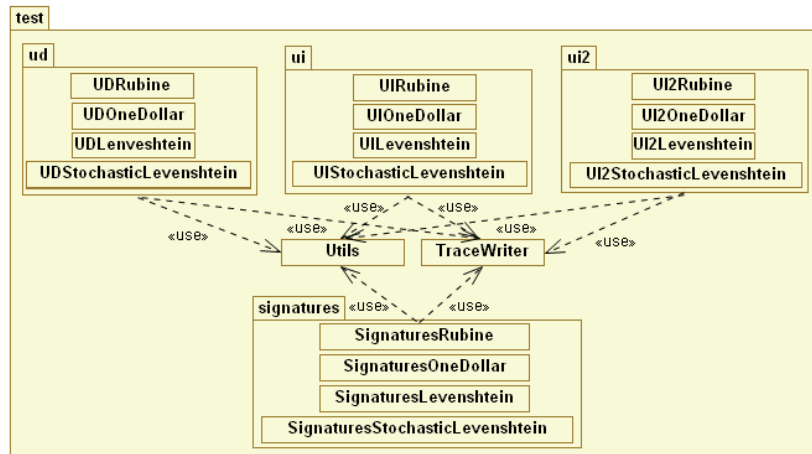
The communication between the interface and the graphic tablet is realized with the classes defined in the package *tablet*. However, the interface does not directly deal with this package but uses the package *data* and more particularly the class *DataRecorder*. This class allows to configure an object *TabletDrawingArea* in order to catch input data from the tablet and, with the help of *Template* contained in *TemplateSet*, record the data in the right place in the right formatted files (text files whose each line represents one point of the gesture). The class *RecordParser* is useful to parse these formatted structure and transform them into appropriated structures, i.e. Java structures containing *Dot* objects. The class *Dot* allows to abstract a dot and store all the provided information on it like spatial coordinates, temporal information, pressures, etc.

The class *TabletDrawingArea* previously quoted is important for the graphical interface and the use of the tablet. It decides which data packet is taken into account in function of the pen touches the tablet or not. Points corresponding to data packets taken into account are drawn by the *TabletDrawingArea*.

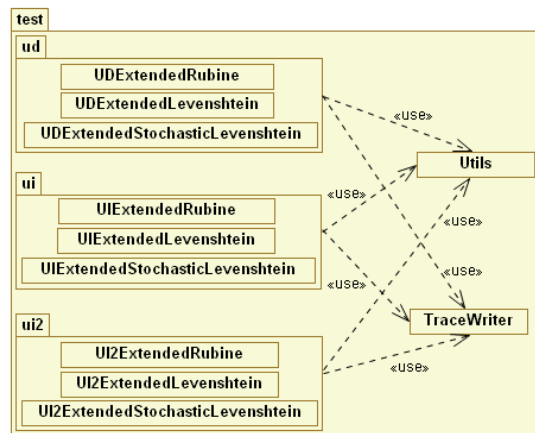
In order to be constantly aware of new possible arrival of data, a thread is running for each instance of the class *TabletDrawingArea*. It checks each millisecond in the cache memory of the tablet if new data are available and in this case sends them to its associated instance. Collecting the data from the cache memory is done through the native Java interface *Jwintab* available on the website [JWinTab].

As a result, 52 classes have been used to create the platform, including 12099 lines of code written in Java (except for *Jwintab* package programmed in C). Figure 3.9 shows the diagram of classes for the tests on original algorithms and Figure 3.10 shows the diagram of classes for the tests on refined algorithms. For clarity purposes, they are depicted on two different diagrams but the classes are actually included in the same subpackages : *test.ud*, *test.ui* and *test.ui2* are the same subpackages on the two diagrams. Without going into the details (experimental methodology will be described in details in section 3.3.2) the classes are gathered by training types : *ud* for user dependent method, *ui* and *ui2* for two different user independent methods. The signatures are an exception since this family of class are not tested in the same way than the other symbols. Each class corresponds to a specific original or refined algorithm and takes a digit as parameter in order to specify the family of class to consider (1

### 3 Selection of appropriate recognition mechanism



**Figure 3.9** Diagram of classes for the tests on original algorithms.



**Figure 3.10** Diagram of classes for the tests on refined algorithms.



### 3 Selection of appropriate recognition mechanism

for digits, 2 for letters, 3 for action gestures, 4 for geometrical shapes).

The two classes *Utils* and *TraceWriter* provide tools to the test classes. The first one gathers a set of operations allowing to transform the names of the classes in digit and conversely (useful for the iterations), to compute the confusion matrices (for producing some statistics), etc. The second one allows to produce a trace file in which are listed the raw data : for each number of training examples and for each number of nearest neighbors is provided the confusion matrix on all the users, its LaTeX format, and the classification rates (global and by user).

#### 3.2.3 Software and hardware configuration

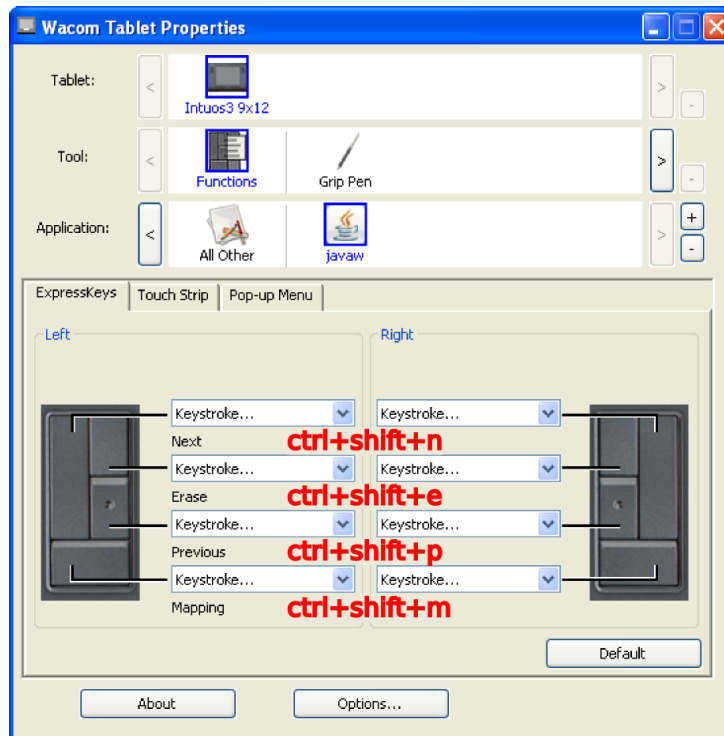
The input device used during this study is a graphic tablet Wacom Intuos3<sup>®</sup> A4 (9x12 inches) (see Figure 3.11). The resolution of the tablet is 200 lines per millimeter on an active surface of 230mm x 305mm. It provides the position, the inclination and the orientation of the pen, as well as its pressure (normalized from 0 to 1023).



**Figure 3.11** Graphic tablet Wacom Intuos3<sup>®</sup> A4.

### 3 Selection of appropriate recognition mechanism

The platform is conceived for the Windows XP<sup>®</sup> operating system and requires Java 1.6+. The use of the Wacom driver is necessary and must be configured as indicated in Figure 3.12 : a keystroke must be associated at each button of the tablet (e.g., the *Next* buttons are associated to the sequence “ctrl+shift+n”).



**Figure 3.12** Configuration of the Wacom driver.

## 3.3 Experimental study

This section details the comparative study produced with the help of the tool described in the previous sections. It first describes how the gesture dataset was acquired, what methodology was used to conduct the study, and then shows and discusses the results.

### 3.3.1 Data acquisition

For producing an experimental study, input data is required to train and test the algorithms. Unfortunately, many databases can be found but suffers from lack

### 3 Selection of appropriate recognition mechanism

of information. For example, IAM [IAM database] contains lots of samples based on the contribution of more than 500 persons, more than 1500 scanned sheets, 10 000 isolated lines of text and 100 000 isolated words, but is only usable for offline recognition. Other databases such as UniPen [Unipen], UCI UJIPen [UJIPen] would be more useful as they provide much more information such as the complete dots set describing the gesture, temporal information, pressure, pen inclination, etc. However these databases only include letters and numbers. With regards to the initial purpose we had no other choice than building our own database so as to integrate geometrical shapes, signatures and action commands, in addition to the letters and numbers.

#### Description of the gestures

The choice of the gestures to obtain was dictated by the feasibility of the encoding. It was not possible to ask to the volunteers to spend more than one hour sketching gestures because of time constraint and risk of lack of attention. The set of gestures was thus reduced to its minimum, while considering all the various types of gestures. The templates used by the training platform are depicted in Appendix B and are composed of :

- the 26 lower case letters ;
- the digits from 0 to 9 ;
- 17 action gestures ;
- 8 geometrical shapes ;
- the signature of the user.

To ensure a sufficient training of the algorithms, the volunteers were asked to provide 10 samples for each gesture, and 30 for each signature. A total of 30 volunteers participated to the database development. Each gesture was then reproduced 300 times, except the signatures of the 30 participants.

#### Acquisition progress

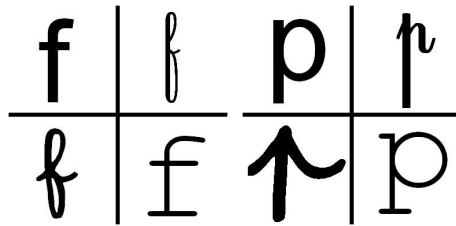
The volunteers were asked to produce the gesture randomly, in order to avoid quality distortion for some gestures. Indeed, during the experiment, the user

### 3 Selection of appropriate recognition mechanism

is likely to take confidence and goes faster and faster trying to end quicker. Mixing the data makes it possible to distribute the various moments of attention and lassitude, but also to avoid a biased “noise”.

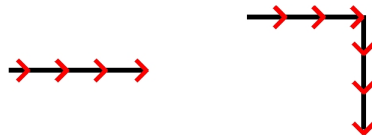
A few rules were also elicited for the gesture acquisition :

- Participants were requested to avoid making erasures. If necessary, they were asked to erase the complete gesture and to start again. The dimension of the gestures and their position on the screen were free ;
- Participants were told it was not a contest. The main purpose was to be as natural as possible ;
- Four representations were depicted for each template in order to enforce volunteer to use their own writing and not copy the image they see (see Figure 3.13) ;



**Figure 3.13** Examples of letters templates presented to the volunteers.

- Action commands were represented with a black line, and red arrows to indicate the drawing direction (see Figure 3.14). Furthermore, the action gestures were imposed to be uni-stroke ;



**Figure 3.14** Examples of action gestures templates presented to the volunteers.

### 3 Selection of appropriate recognition mechanism

- Geometrical shapes could be drawn with several lines (except for the circle). Participants were informed that the precision and artistic aspects were clearly secondary ;
- The signature did not require any particular constraint.

Before the data acquisition, some technical instructions have been provided to each volunteer. Also, they were allowed to train on the platform during a few minutes.

The number of gestures to reproduce was calibrated in order to reach an average time of 30 minutes per volunteer. Finally, a mean of 29 minutes 45 seconds was obtained with a minimum of 13 minutes, a maximum of 48 minutes and a standard deviation of 8 minutes 20 seconds.

#### Conditions

The data acquisition took place in similar conditions for all the volunteers. The place was not always the same but was chosen in respect of some conditions, in particular quiet and without external perturbations. The office of Prof. Jean Vanderdonckt and the computer science room of Reaumur building were retained. A picture of this room as well as the device used is represented in Figure 3.15. The laptop is equipped of a Intel® Pentium® M 1.6GHz processor, 2Gb DDR of RAM and a 15 inches screen with a resolution of 1400x1050 pixels.



**Figure 3.15** Environment of the data acquisition.

### 3 Selection of appropriate recognition mechanism

#### Sample population

The 30 persons that participated to the experience were all volunteers. The age varied between 18 and 55 years with a mean of 28 years and a standard deviation of 10 years and 9 months. The group was composed of 16 women and 14 men, in which 26 right-handed and 4 left-handed persons. Their field of activities was well different with persons dealing with sciences, psychology, biology, architecture, etc. Among them, only one does not use a computer regularly and 9 that had already manipulated tools with pen, as PDA, pocket PC, ... Additionally, 2 persons are used to the graphic tablets. A table summarizing the characteristics of the volunteers is available in Appendix C.

#### 3.3.2 Methodology

##### Original algorithms

The comparison of the four selected algorithms required a rigorous and identical methodology for each of them in order to obtain pertinent results. Provided that the slowest algorithm is stochastic Levenshtein, the possibilities concerning the execution time of the tests were determined by this algorithm. Certain choices were thus made to execute the tests in reasonable time.

The families of classes were never mixed. All the comparisons were always produced among the examples of a same family, for each family. This choice resulted in a drastic reduction of the execution time.

The constituted database is composed of 30 signatures by user, and 10 examples for the other classes. Two types of tests were realized : tests when the algorithms are trained with examples of the user that is used for the recognition (called in the followings tests with training or made dependently of the user), and tests when the algorithms are only trained with examples of other users than the one used for the recognition (called in the followings tests without training or made independently of the user). The statistical method chosen for producing the comparative study is a kind of cross-validation, which will be explained later. A maximum of 9 training examples were thus considered during the study since this number can not be exceeded for the tests with training. Indeed, 10 examples are available for each user and minimum 1 is required for the recognition.

### 3 Selection of appropriate recognition mechanism

**Tests with training** For each test, the number of training examples varied between 1 and 9 for One Dollar and original Levenshtein's algorithms, between 2 and 9 for Rubine and stochastic Levenshtein's algorithms. For Rubine, it can be explained by considering equation 2.2 which proceeds to a division by the number of classes subtracted to the total number of examples provided for each class. With only one example by class it comes to divide by zero, which is not possible. For stochastic Levenshtein, two examples of the same class are needed since pairs of strings belonging to the same class are constituted during the training phase.

Five users were used for each test. For each these users, the different algorithms were trained with a number of examples varying between 1 (or 2) and 9, the remaining examples being used for recognition. In order to obtain the more possible pertinent results for a fixed number  $n$  of training examples by class, train the algorithms with all the possible combinations of  $n$  examples among the 10 ones available would have been ideal. However, it would have imply a too huge number of possibilities. In place of a full cross-validation, a *moving window* was chosen in order to produce the tests. To illustrate this concept, let us consider a number of training examples by class  $n = 4$  : for a given class  $c$ , the examples  $c_1, c_2, c_3$  and  $c_4$  are firstly considered for the training (examples  $c_5$  to  $c_{10}$  are thus used for recognition), then the examples  $c_2$  to  $c_5$ , then the examples  $c_3$  to  $c_6$ , and so on until  $c_4, c_1, c_2$  and  $c_3$ . In a general way, for a number  $n$  of training examples by class, each example is considered  $n$  times for the training.

A confusion matrix was thus built for each number of examples by class  $n$ . The confusion matrix is a tool allowing to measure the quality of the classification. Each column of the matrix represents the number of examples of an estimated class while each line represents the number of the examples of a real class. It is helpful to determine the recognition rate of  $n$  as well as the way each class is categorized.

**Tests without training** Two different methods were considered for the tests without training of the user. The first one took five users into consideration whose one was used for the training all the other ones for the recognition. The second method took ten users into consideration whose one was used for recognition and all the other ones for the training.

The procedure imagined for the first method is similar to the procedure used for a training made dependently of the user. The algorithms are trained each

### 3 Selection of appropriate recognition mechanism

time on one user in the same way than described just before and the difference comes from the examples used for recognition not taken from the same user but other ones. In order to avoid a too long execution time, it was chosen to consider only the examples whose number was not chosen for the training. A confusion matrix was computed for each number of training examples, as well as a classification rate for each user tester. Furthermore, a statistic allowing to visualize the classification rate by user used for the training is provided.

The second method for a training made independently of the user concerned a set of ten users in which nine of them were used for the training and the remaining one for the recognition. As for previous trainings, the number of examples by class varied between 1 (or 2) and 9. For a given class  $c$  and a number of training examples by class  $n$ , the way the examples were selected was to always take the first one  $c_1$  into account for  $n$  of the nine users considered for the training. In other words, the *moving window* described before is not used on the examples themselves anymore but rather on the users considered. Let us illustrate this for a given class  $c$ , a number of examples by class  $n$  and ten users. Provided the first example of the class is always taken into account, let  $u_x$  be the first example of class  $c$  for the  $x^{th}$  user. Let us then suppose that the first user and thus  $u_1$  is used for the classification, then  $u_2$ ,  $u_3$ ,  $u_4$  and  $u_5$  are considered for the training, then  $u_3$  to  $u_6$ , and so on until  $u_{10}$ ,  $u_2$ ,  $u_3$  and  $u_4$ .

**Signatures** The family of classes of signatures represents a specific case. A class of this family corresponds to the signature of a particular user. Training made dependently or independently of the user can not be considered here since it would have no sense trying to recognize the signature of a person with an algorithm not trained with signatures of the same person. The goal is to test the recognition of a signature among a set of different signatures. For this purpose, a set of ten users was considered and the number of signatures considered for the training varied between 1 (or 2) and 29 since the number of available signatures for each user is 30.

#### Refined algorithms

The tests for the refined algorithms were produced in conditions similar to those described for the original algorithms : one method of training dependent of the user and two independent of the user, for a number of training examples



### 3 Selection of appropriate recognition mechanism

varying between 1 (or 2) and 9. However, all the families of classes were not tested. Only the digits were considered since they represent the more common symbols and allow faster recognition thanks to their reduced number of different classes. The refinements tested are explained algorithm by algorithm in the followings.

**Rubine** The preprocessing of the data was added to the original version of the Rubine's algorithm. The gestures were thus either resampled, reoriented or rescaled (one operation at a time). The operation of translation was not tried since it is already included in the original Rubine's algorithm.

**One Dollar** The chosen modification for the One Dollar's algorithm is the variation of the number of nearest neighbors. The original version of the algorithm classifies a new gesture with the class of the closest training example considering the Euclidean distance. Instead of only considering one training example, this number varied between 1 and 9.

**Original Levenshtein** The algorithm of Levenshtein (original as stochastic) was the most experimented algorithm for modifications.

Firstly a preprocessing of the data was executed : a resampling as well as a reorientation. The rescaling and the translation were not kept since these operations do not modify the point-by-point directions and thus strings representing the gestures.

The second feature interesting to take into account is the holes in the multi-strokes gestures. In the original version, the direction between two points surrounding a hole was processed in the same way than two successive points, i.e. by attributing the direction of the compass rose corresponding as well as possible. In order to consider a hole between two points, an imaginary line was drawn between them. During the generation of the string representing the gesture, a new symbol ("x" in the implementation) was inserted to symbolize the hole. The number of symbols added depends of the length of the line and the frequency of addition (determined empirically).

Since significant ameliorations were observed for the consideration of the holes, but also for the resampling, these two refinements were tried at the same time.

### 3 Selection of appropriate recognition mechanism

The inclusion of the pressure was also studied. For this purpose, the range of possible values for the pressure was normalized in order to correspond to an alphabet of 8 symbols (0 to 7 as for the directions). A set of 8 intervals was thus built, for which the first interval is between 0 and  $MAX\_PRESSURE/8$ , the second interval between  $MAX\_PRESSURE/8$  and  $2 * MAX\_PRESSURE/8$ , and so on until the eighth interval between  $7 * MAX\_PRESSURE/8$  and  $MAX\_PRESSURE$ . Thanks to these intervals, it is possible to build the string representing the gesture by attributing a symbol in function of the interval to which the pressure of a point belongs. The same process was also applied for the inclination and the orientation of the pen.

The non-very concluding results for the inclusion of these three features are not shown in the followings. However, another way to take them into account was imagined. Instead of considering the normalization of the values themselves, the differences point-by-point of the value were treated. The alphabet used for the generation of the strings was thus composed of three symbols : 0 for an increasing, 1 for no change, and 2 for a decreasing.

The results were a little bit better but unfortunately still not sufficient to be shown here. Finally, a last idea was tested for these features : the combination of the point-by-point orientations and the differences of the features giving the best results, i.e. the pressure. For this purpose, an alphabet of 24 symbols was required. Indeed, the 8 possibilities of orientations multiplied by the 3 possibilities of differences of pressures result as 24 possibilities.

The original algorithm of Levenshtein was also tested by adapting it for the three normalizations described in the state of the art.

Levenshtein considers only one closest training example in order to classify a new gesture. As for the algorithm of One Dollar, a variation of the number of nearest neighbors was tested.

Finally, the basic costs used for the Levenshtein's algorithm are 0 for a substitution between two same symbols and 1 for suppression, an insertion or a substitution between two different symbols. The adapted costs matrix presented in Table 2.3 was tried in order to improve the substitution costs.

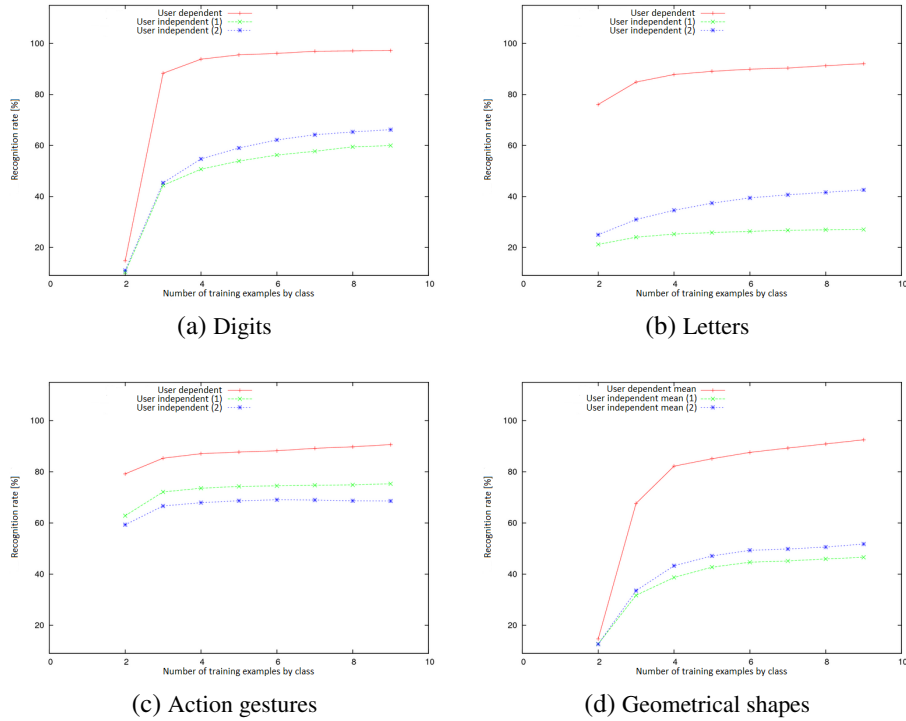
**Stochastic Levenshtein** The refinements tried for the stochastic Levenshtein's algorithm are the same than those tried for the original version of Levenshtein, except the adapted costs matrix since the adaptation of the costs is the purpose of the stochastic Levenshtein's algorithm.

### 3 Selection of appropriate recognition mechanism

#### 3.3.3 Results

##### Original algorithms

Results are provided in the form of recognition rates corresponding to means on the different users tested, for each number of examples by class considered (1 or 2 to 9). These means are depicted in a set of graphs allowing to highlight some characteristic in order to analyze it. More precisely, three big characteristics were considered in this comparative study : the training type (one method dependent of the user and two independent of the user), the family of classes (digits, letters, action gestures and geometrical shapes), and the algorithm (Rubine, One Dollar, original Levenshtein and stochastic Levenshtein). The idea is to fix two characteristics by varying the third one in order to observe its behavior. The signatures constitute a special case and are analyzed separately. Finally the execution times are provided.



**Figure 3.16** Results of the analysis by training type for Rubine.

### 3 Selection of appropriate recognition mechanism

**Analysis by training type** Figure 3.16 shows the results obtained for Rubine's algorithm and the different families of classes by varying the training type. For the four families the training made dependently of the user clearly appears to be the best one, even if this difference is less huge for the action gestures. Furthermore, these action gestures have another particularity : unlike the other families of gestures, the first method of training independently of the user offers better recognition rates than the second one.

Another point to underline is the very low recognition rate with 2 training examples for the digits and the geometrical shapes before drastically increasing with 3 training examples. On the contrary, the curve is smoother from the beginning for letters and the action gestures. After analyses on reduced families of letters, it seems that the number of the different classes populating the families dictates the behavior of the recognition for 2 training examples. For the moment, no coherent explanation has been found and other investigations could be produced in order to find the origin of this behavior.

For the three training methods, the recognition rate stops varying in a consequent way approaching the 9 training examples. Only the rate for the letters trained independently of the user seems to still significantly increase.

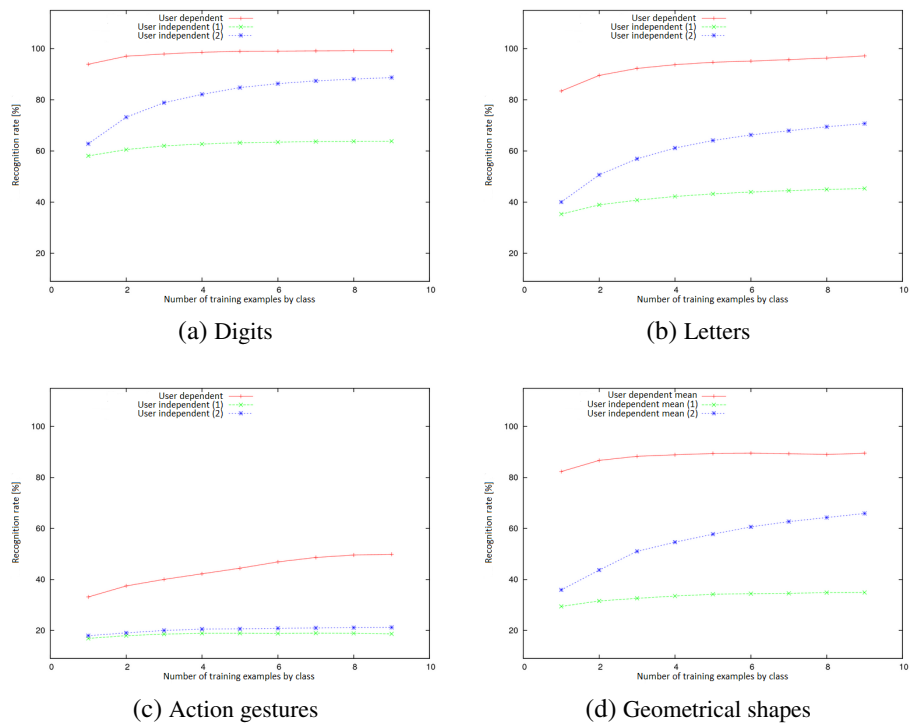
Figure 3.17 shows the results obtained for One Dollar's algorithm and the different families of classes by varying the training type. The training made dependently of the user clearly gives better results. However, for the digits with a training independent of the user (second method) the results are quite close.

Considering the tests without training, the second method offers better classification than the first one (with a very small difference for the action gestures). Furthermore, it is in constant progression when the number of training examples increases while the first method stagnates quickly.

Figure 3.18 shows the results obtained for Levenshtein's algorithm and the different families of classes by varying the training type. The lessons we can learn are the similar to those for the One Dollar's algorithm. The difference between the two algorithms concerns the results for the action gestures that are not so bad and follow the evolution of the other families of classes concerning the differences between the training methods.

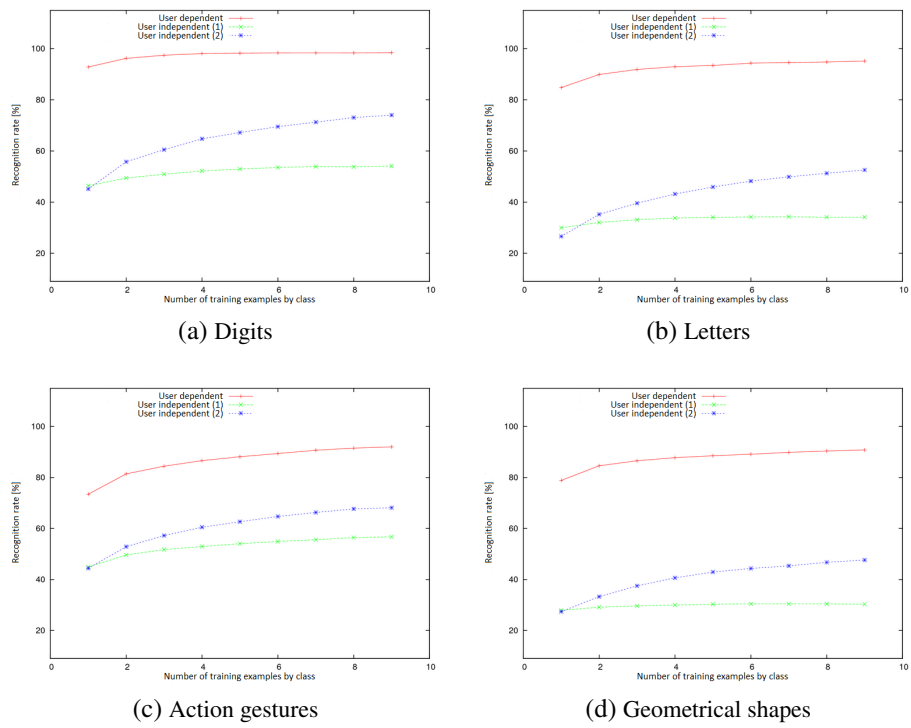
Figure 3.19 shows the results obtained for stochastic Levenshtein's algorithm and the different families of classes by varying the training type. An important limitation appeared during the tests. The algorithm is not able to work for too

### 3 Selection of appropriate recognition mechanism



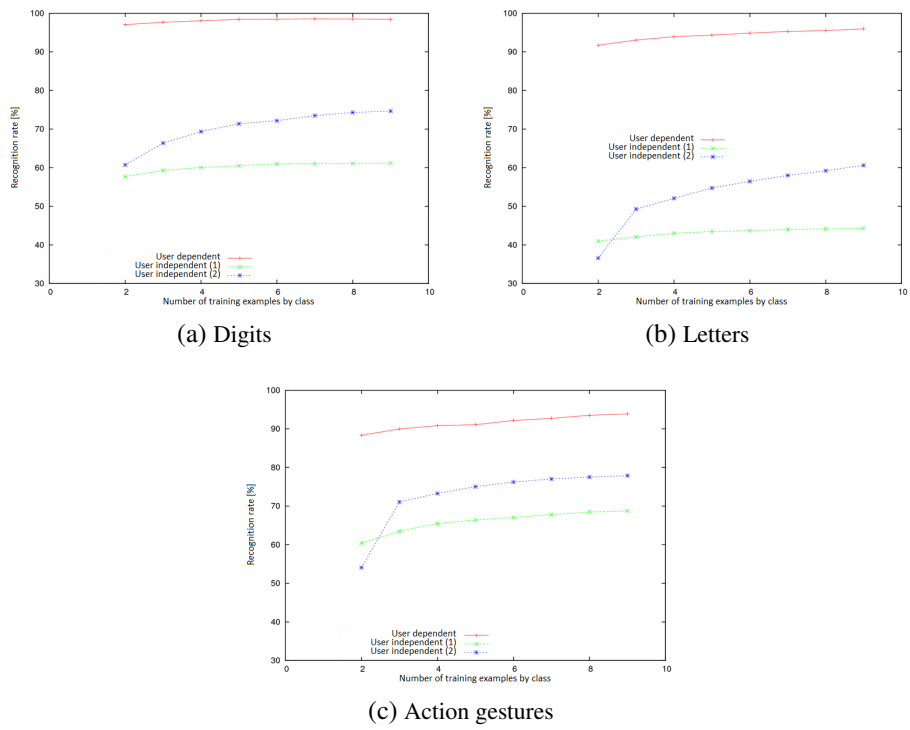
**Figure 3.17** Results of the analysis by training type for OneDollar.

### 3 Selection of appropriate recognition mechanism



**Figure 3.18** Results of the analysis by training type for Levenshtein.

### 3 Selection of appropriate recognition mechanism



**Figure 3.19** Results of the analysis by training type for stochastic Levenshtein.

### 3 Selection of appropriate recognition mechanism

long string representing the gestures. The problem occurs by using the backward and forward functions during the computation of probabilities of a string conditionally to another one. Indeed, the longer are the strings, the smaller become the probabilities. If the strings become too long, the probabilities tend toward 0. For a computer, when a value is too close of 0, then it is rounded to 0. This is not exact and causes the dysfunction of the algorithm.

Actually, it represents a well-known behavior for this type of algorithm. Some programming tricks allowing avoiding the problem exist like considering the logarithms of the probabilities instead of the probabilities themselves. The applied operations would not be multiplications anymore but rather additions, avoiding the results to tend towards 0. This amelioration is not implemented in this work since it represents a huge effort in order to modify the algorithm while other families of classes allow to study the performances of stochastic Levenshtein's algorithm.

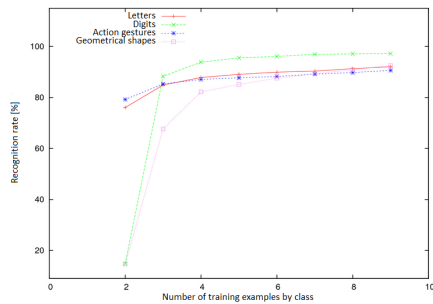
For the reasons just explained, the geometrical shapes are not considered for this algorithm. For the three other families of classes, similar tendency can be observed considering the other algorithms. The difference for the stochastic Levenshtein's algorithm is that the recognition rates for a training made independently of the user are closer to the ones for a training made dependently of the user.

**Analysis by family of classes** Figure 3.20 shows the results obtained for a training made dependently of the user and the different algorithms by varying the family of classes. For this training type, it can be observed that the recognition rates comes quickly close to their optimal value and without varying with the increase of the number of training examples. However, the action gestures seem to keep a relatively consequent increasing rate even approaching 9 numbers of examples.

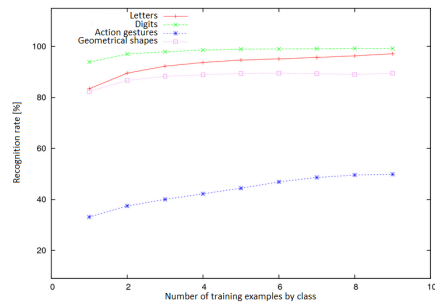
Let us also underline the better results for digits and letters. These two families of classes are very similar and differ mostly by the number of classes they hold. Because of a smaller number of classes, digits are thus better classified than letters. After these two families of classes come the geometrical shapes and then the action gestures that merge at the end with the increase of the number of training examples for Rubine and Levenshtein's algorithm while the action gestures are poorly recognized for One Dollar's algorithm.



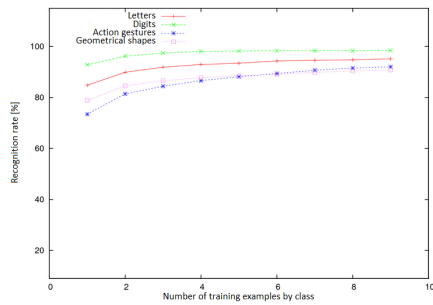
### 3 Selection of appropriate recognition mechanism



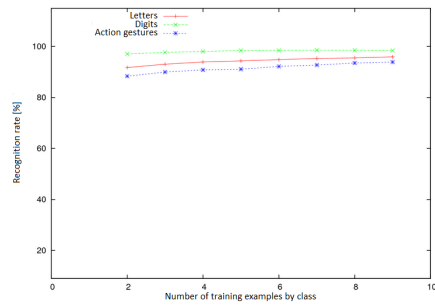
(a) Rubine



(b) One Dollar



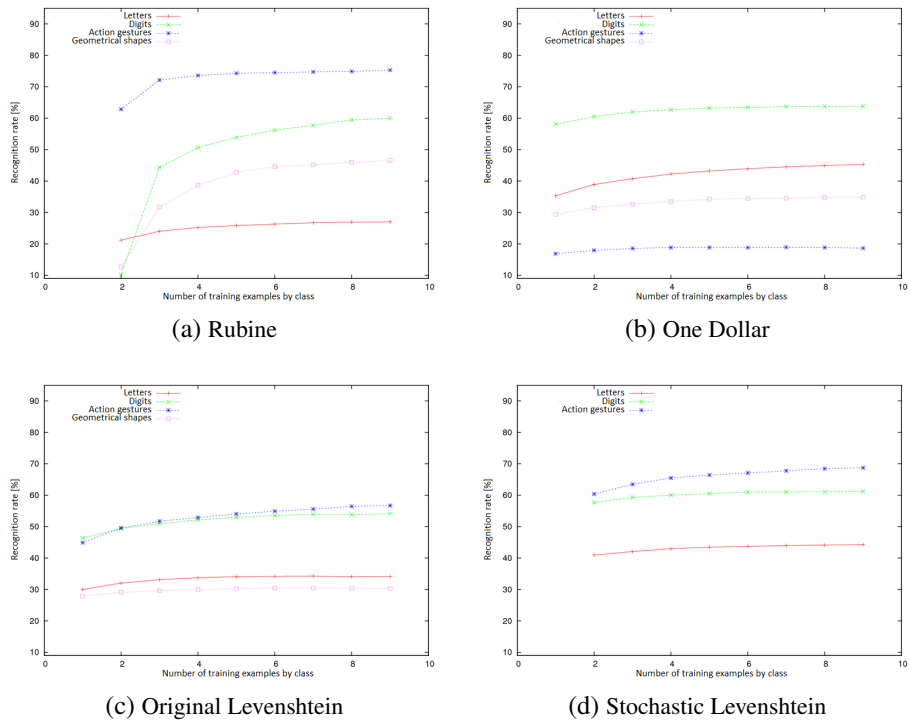
(c) Original Levenshtein



(d) Stochastic Levenshtein

**Figure 3.20** Results of the analysis by family of classes for a training made dependently of the user.

### 3 Selection of appropriate recognition mechanism



**Figure 3.21** Results of the analysis by family of classes for a training made independently of the user (first method).

### 3 Selection of appropriate recognition mechanism

Figure 3.21 shows the results obtained for a training made independently of the user (first method) and the different algorithms by varying the family of classes. In this configuration, the most remarkable constant is the classification of the action gestures. Except for One Dollar's algorithm, it is better than the classification of all the other families of classes. This behavior can be explained by the low variation between the users for sketching the action gestures. Train the algorithm on another user then the one used for the test is less faced with a problem for this family of classes. The particular case of One Dollar will be treated in the analysis by algorithm.

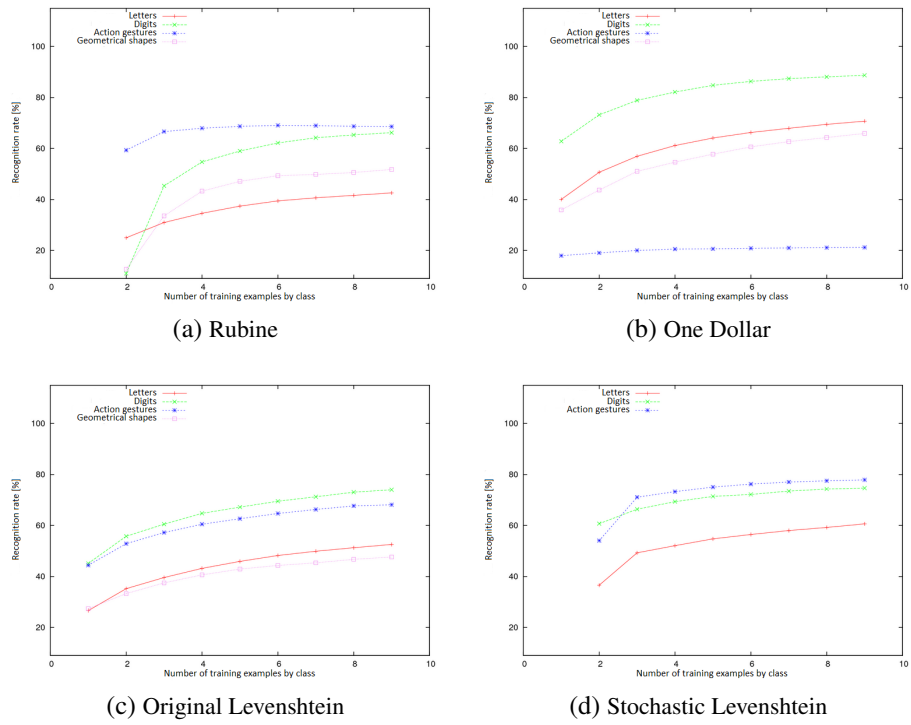
The second classified family of classes is the digits. For the algorithms of original Levenshtein — especially — and stochastic Levenshtein, the difference with the action gestures appears to be very small, which is not the case for Rubine's algorithm. The recognition rate for the geometrical shapes constitutes another distinction for Rubine's algorithm since it is better than the one for the letters. Indeed, the geometrical shapes are a particular family of classes because even if their general aspects are similar for different users, the order and sense of the strokes can strongly vary. Rubine is more robust for these features than One Dollar of Levenshtein.

Concerning the variation of the recognition rates when the number of examples increases, this first method of independent training induces a rapid stagnation for all the families of classes and algorithms. An analogy can be done with the training made dependently of the user where, provided the number of training examples is already sufficient, an increase does not offer so much additional information since the training is realized on only one user.

Figure 3.22 shows the results obtained for a training made independently of the user (second method) and the different algorithms by varying the family of classes. Except for the Levenshtein's algorithm, similar conclusions than for the first method of independent training can be drawn concerning the order of classification of the families of classes.

For the Levenshtein's algorithm, the digits are the best-classified family of classes. Furthermore, if this observation cannot yet be done for Rubine's and stochastic Levenshtein's algorithms, the rates for the other families of classes come closer to the action gestures. It can be explained by the increasing variation provided by the different users. Train the algorithms with examples from more different users induces a beneficial effect for all the families of classes, but is more pronounced for the digits and then letters and geometrical shapes.

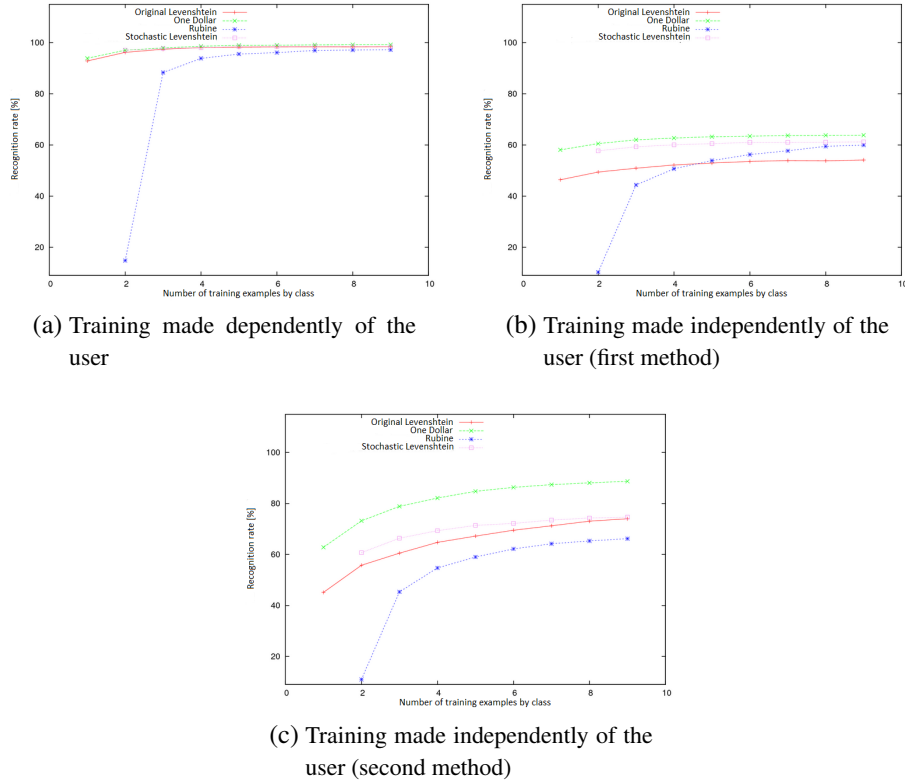
### 3 Selection of appropriate recognition mechanism



**Figure 3.22** Results of the analysis by family of classes for a training made independently of the user (second method).

### 3 Selection of appropriate recognition mechanism

The other difference with the first method of independent training concerns the variation of the recognition rates. They increase with the number of training examples, even at the end. The potential of increase is thus more important with a huger variety of users for a training made independently of the user.



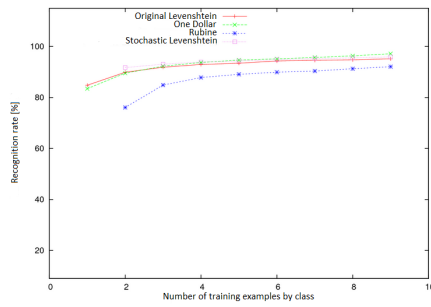
**Figure 3.23** Results of the analysis by algorithm for the digits.

**Analysis by algorithm** Figure 3.23 shows the results obtained for the digits and the different training types by varying the algorithms. For a training made dependently of the user, the recognitions merge at the end. It is difficult to underline an algorithm, even if Rubine is a little bit less performing than the other ones for a low number of training examples. When the training is made independently of the user, The One Dollar's algorithm provides the best recognition rates. If the difference is very small with the stochastic Levenshtein's algorithm for the first method, the distinction is more visible for the second method. After these two algorithms comes Levenshtein's algorithm followed by Rubine's algorithm. However, for the first method of an independent

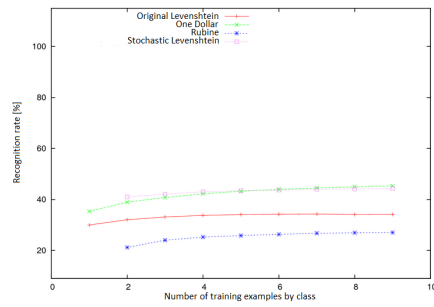
### 3 Selection of appropriate recognition mechanism

training and a sufficient number of training examples, the results for Rubine's algorithm exceed the results for Levenshtein's algorithm.

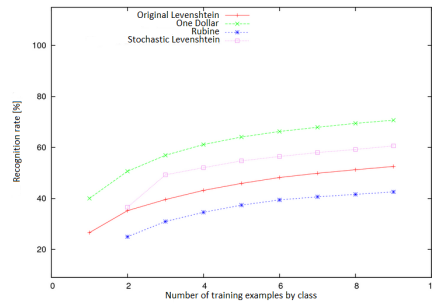
One Dollar is thus the best algorithm for the digits, thanks to its improved efficiency for the training made independently of the user. The transition from the first method of the independent training to the second one improves the performances of the four algorithms, but the difference is less huge for Rubine and stochastic Levenshtein. It can be explained by only available training example by user for the second method. Indeed, difficulties can be met in order to determine the weights, especially when only a few training examples are available by class while these two algorithms become performing with several training examples. It could be interesting to test this training with more examples by user.



(a) Training made dependently of the user



(b) Training made independently of the user (first method)

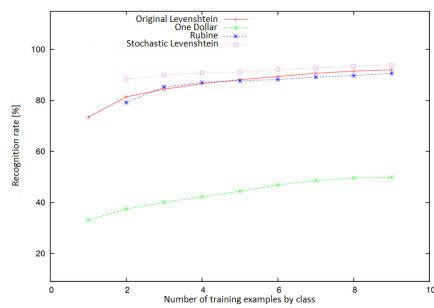


(c) Training made independently of the user (second method)

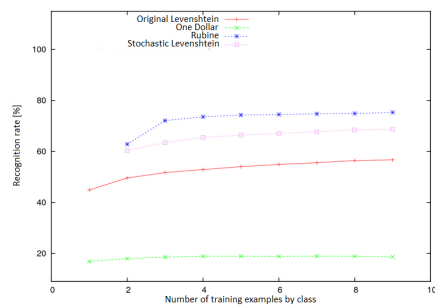
**Figure 3.24** Results of the analysis by algorithm for the letters.

Figure 3.24 shows the results obtained for the letters and the different training types by varying the algorithms. The lessons that can be learned from these graphs are similar to the one learned for the digits.

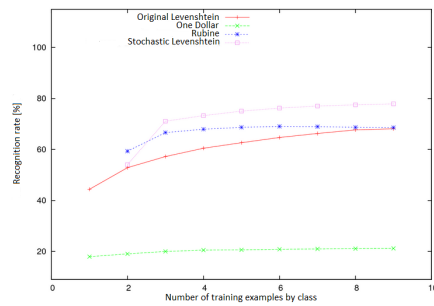
### 3 Selection of appropriate recognition mechanism



(a) Training made dependently of the user



(b) Training made independently of the user (first method)



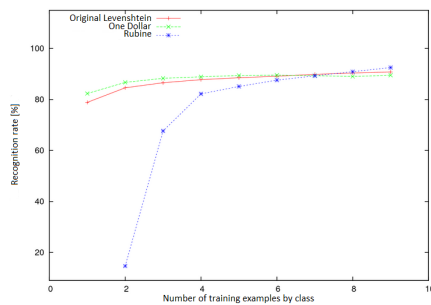
(c) Training made independently of the user (second method)

**Figure 3.25** Results of the analysis by algorithm for the action gestures.

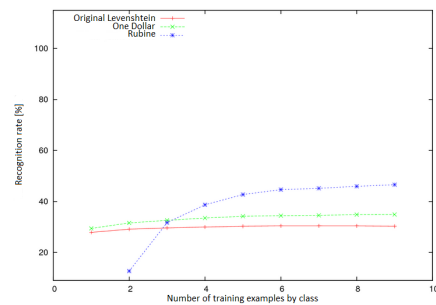
### 3 Selection of appropriate recognition mechanism

Figure 3.25 shows the results obtained for the action gestures and the different training types by varying the algorithms. The main observation of these graphs is the very bad performances of the One Dollar's algorithm for the action gestures. It confirms the difficulties met by this algorithm to distinguish gestures only different by geometrical features. In this case, a rotation is sufficient to merge the different action gestures.

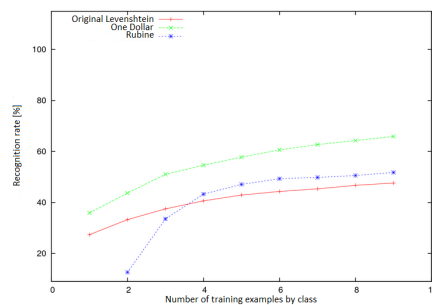
Outside the consideration of One Dollar, Rubine offers the best recognition rates for the first method of a training made independently of the user while stochastic Levenshtein is more performing for the second method since it takes advantage of the variety of the users in a better way. For a training made dependently of the user, the recognition rates are quite similar for the three algorithms.



(a) Training made dependently of the user



(b) Training made independently of the user (first method)



(c) Training made independently of the user (second method)

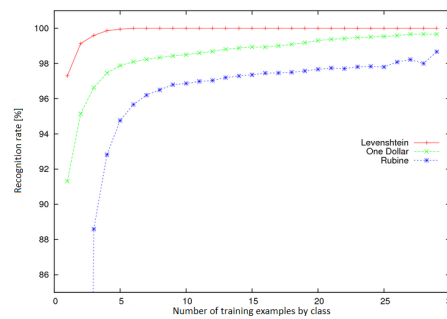
**Figure 3.26** Results of the analysis by algorithm for the geometrical shapes.

Figure 3.26 shows the results obtained for the geometrical shapes and the different training types by varying the algorithms (except stochastic Levenshtein since it is not adequate for the geometrical shapes that are composed of too



### 3 Selection of appropriate recognition mechanism

much points). For a training made dependently of the user, Rubine has one more time difficulties for a small number of examples but the three tested algorithms reach similar recognitions rates when this number increases. Concerning the first method of independent training, Rubine is more performant than Levenshtein and One Dollar. As previously said, geometrical shapes can have a lot of differences between users because of different order and sense of the strokes. This characteristic is more suitable for Rubine. On the contrary, the second method of independent training allows a significative progression of Levenshtein and One Dollar's algorithms while Rubine is hardly improved. As consequence, One Dollar's becomes the best algorithm for this method.

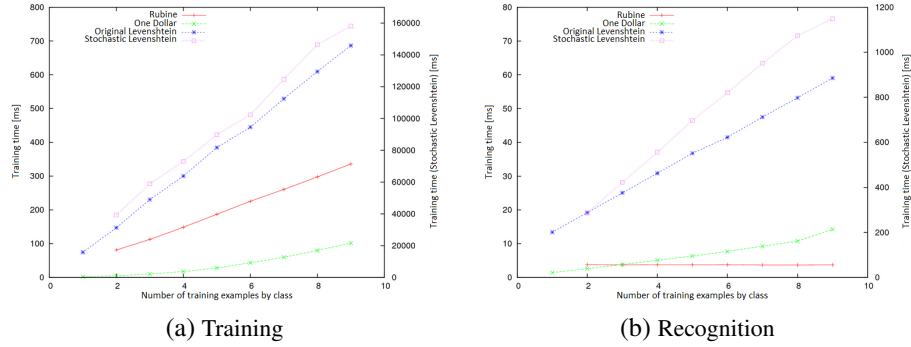


**Figure 3.27** Results for the signatures.

**Signatures** Figure 3.27 shows the results obtained for the signatures. As for the geometrical shapes, the signatures suffer from being constituted by too much points and cannot be tested for the stochastic Levenshtein's algorithm. As explained in the methodology, the goal here is to observe the ability of an algorithm to distinguish the signature of a user by considering a dataset composed of signatures of the same user as well as other ones. For this purpose, the stochastic Levenshtein's algorithm is the best since it requires only a few training examples in order to reach perfect classification. After comes One Dollar and then Rubine. These two algorithms seem to tend towards the same rate but do not reach it before the 29 training examples by class.

**Execution times** The execution times were determined for the test with a training dependent of the user and the digits. Figure 3.28(a) shows the results obtained for the times in function of the number of training examples by class for one training. These training times are of the order of magnitude (under 1 second) for Rubine, One Dollar and original Levenshtein's algorithms. How-

### 3 Selection of appropriate recognition mechanism



**Figure 3.28** Results for the execution times.

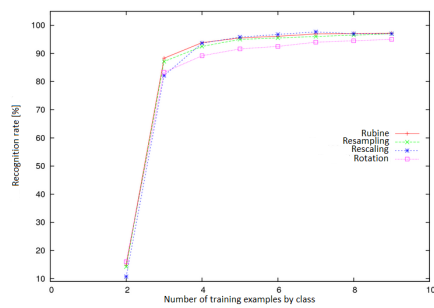
ever, for stochastic Levenshtein's algorithm the time is much more important and can reach order of magnitudes until 1000 times bigger (40 to 160 seconds). For all the algorithms, the training time increase linearly with the number of training examples by class.

Figure 3.28(b) shows the results obtained for the times in function of the number of training examples by class for one recognition. These times are smaller for all the algorithms but the big difference between the recognition time and the recognition times for the other algorithms is still present. The evolution of the times in function of the number of training examples by class is also linear, except for Rubine for which it is constant : the time needed by Rubine in order to classify an example is not dependent of the number of training examples by class.

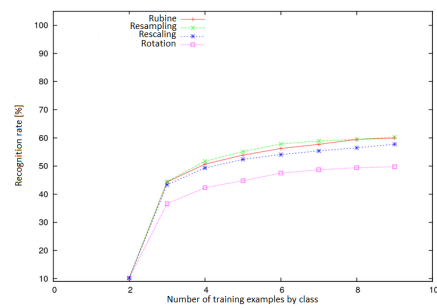
#### Refined algorithms

As for the original algorithms, results are provided in the form of recognition rates corresponding to means on the different users tested, for each number of examples by class considered (1 or 2 to 9). The behaviors of the refined algorithms are shown through graphs in which the curves of the original algorithms are also depicted in order to highlight the difference. The graphs represent the recognition rate in relation to the number of training examples by class, except for the  $k$ NN extension (One Dollar and Levenshtein) for which the recognition rate is in relation to the number of nearest neighbors.

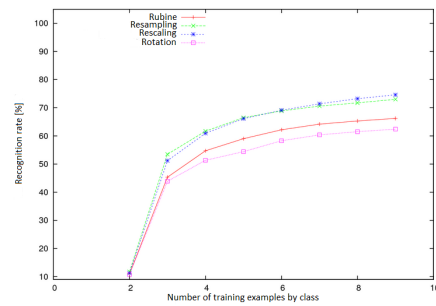
### 3 Selection of appropriate recognition mechanism



(a) Training made dependently of the user



(b) Training made independently of the user (first method)

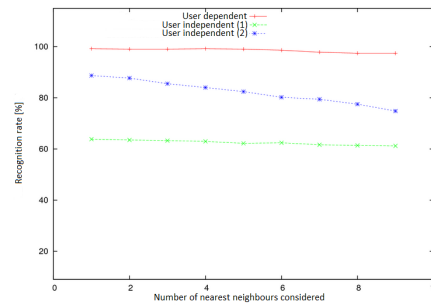


(c) Training made independently of the user (second method)

**Figure 3.29** Results for the preprocessing of Rubine.

### 3 Selection of appropriate recognition mechanism

**Rubine** Figure 3.29 shows the results obtained for the preprocessing extension of Rubine. It can be observed on the three graphs that the rotation does not produce beneficial effect since the recognition rates are deteriorated in comparison with the original Rubine's algorithm. On the contrary, the resampling seems to bring improvement. If it is difficult to observe when the training is made dependently of the user because of its merged curves, it is easily visible for the training made independently of the user. The rescaling is more enigmatic since it decreases the performances for the first method of a training made independently of the user while it increases the performances for a second method. The result is quite surprising, the variability in the sample of users used for the tests seems to impact the performances of the Rubine's algorithm when it is preprocessed with a rescaling.



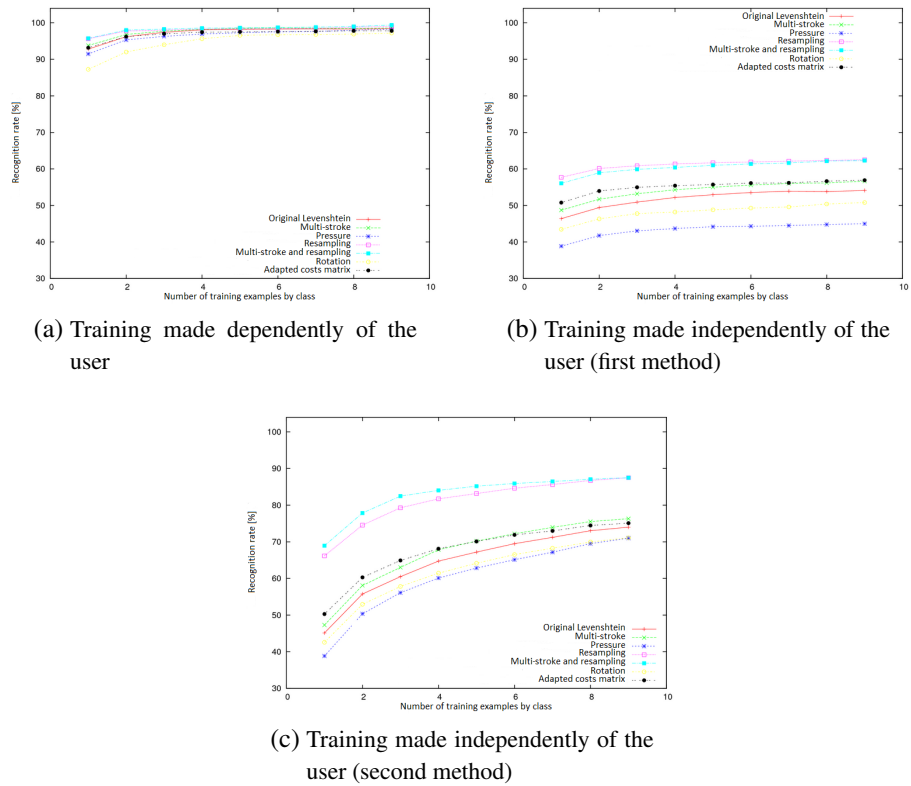
**Figure 3.30** Results for  $k$ NN of One Dollar.

**One Dollar** Figure 3.30 shows the results obtained for the nearest neighbor extension of One Dollar for a training made on 9 numbers of examples by class. The graph does not denote any improvement with this extension. On the contrary, the recognition rates decrease with the number of nearest neighbors considered.

**Original Levenshtein** Figure 3.31 shows the results obtained for the extensions of preprocessing, integration of new features, consideration of adapted costs and consideration of multi-strokes gestures of Levenshtein. The curves are too close for the training made dependently of the user. The analysis will thus concern the two graphs for the training made independently of the user.

Firstly, let us consider the combination between the difference of pressures and the point-by-point orientations. The pressure was chosen because of the bad results for the inclusion of the other dynamic features (less than 50% for the

### 3 Selection of appropriate recognition mechanism



**Figure 3.31** Results for the extensions of preprocessing, integration of new features, consideration of adapted costs and consideration of multi-strokes gestures of Levenshtein.

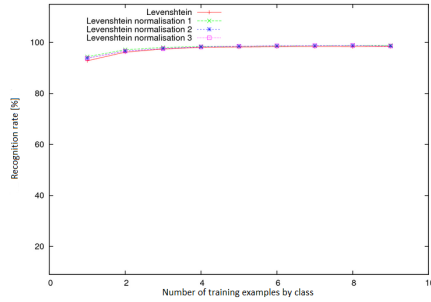
### 3 Selection of appropriate recognition mechanism

dependent training and less than 20% for the independent trainings). However, it can be observed on the graphs that including the difference of pressures still represents a degradation of the recognition rates.

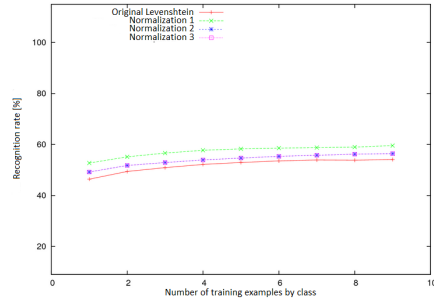
Another personalization of the Levenshtein's algorithm decreasing the performances is the reorientation of the data.

Concerning the refinements allowing to improve the algorithm, let us start by underlining the consideration of the multi-strokes and also the adapted costs matrix that slightly increases both the recognition rates in a similar way.

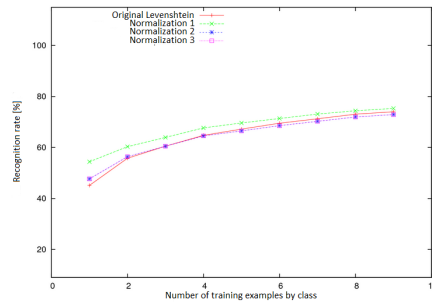
The resampling of the data, allowing a strong improvement of the performances, provides the real evolution. This improvement is a little bit more present for the second method of the training made independently of the user. A combination of this resampling with the consideration of the multi-strokes even allows to increase the performances.



(a) Training made dependently of the user



(b) Training made independently of the user (first method)



(c) Training made independently of the user (second method)

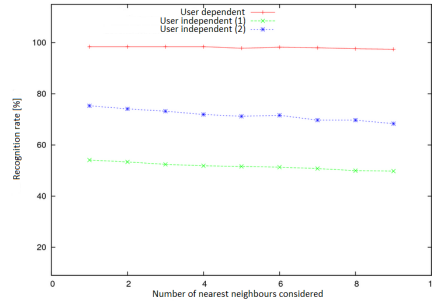
**Figure 3.32** Results for the normalizations of Levenshtein.

### 3 Selection of appropriate recognition mechanism

Figure 3.32 shows the results obtained for the normalizations of Levenshtein. Here is a brief reminder of the three normalizations. Let  $dL$  the original distance of Levenshtein,  $A$  and  $B$  the two considered strings and  $|A|$  and  $|B|$  their respective lengths :

- normalization 1:  $dL1 = \frac{dL}{\max(|A|, |B|)}$
- normalization 2:  $dL2 = \frac{dL}{|A| + |B|}$
- normalization 3:  $dL3 = \frac{2 * dL}{|A| + |B| + dL}$

It can be observed that the first normalization offers the best improvement. The two other ones provide similar results by improving a very little bit the performances for the first method of the independent training, and even slightly decrease the performances for the second method.

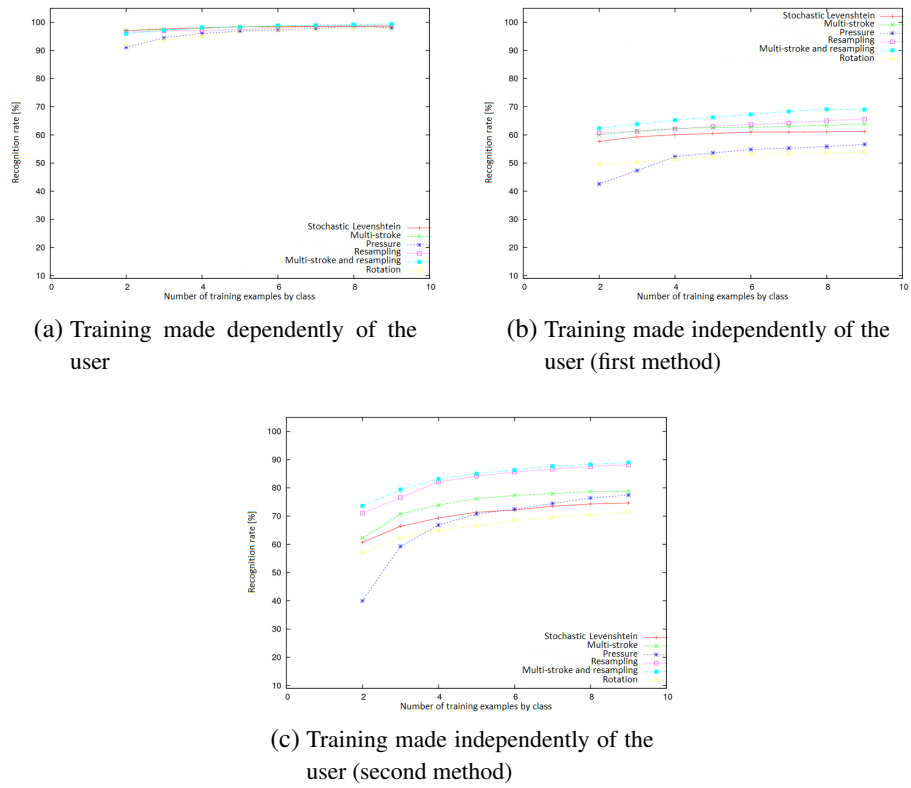


**Figure 3.33** Results for  $k$ NN of Levenshtein.

Figure 3.33 shows the results obtained for the nearest neighbor extension of Levenshtein for a training made on 9 numbers of examples by class. Conclusions similar to those for One Dollar can be drawn : this extension implies a degradation of the results.

**Stochastic Levenshtein** Figure 3.34 shows the results obtained for the extensions of preprocessing, integration of new features and consideration of multi-strokes gestures of stochastic Levenshtein. Conclusions similar to those for original Levenshtein can be drawn. The rotation decreases the performances of the algorithm while the resampling improves them in a consequent way. Taking the multi-strokes gestures into account also allows helping the algorithm, for the original gestures as well as the resampled ones. However, a difference with the original Levenshtein's algorithm appears for the inclusion

### 3 Selection of appropriate recognition mechanism

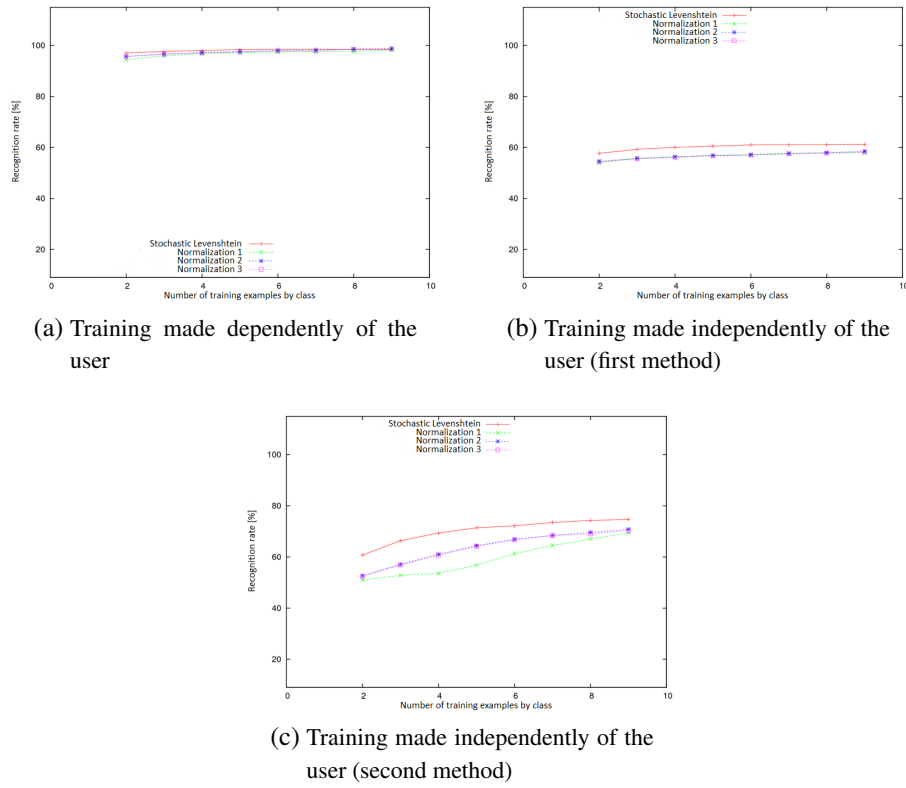


**Figure 3.34** Results for the extensions of preprocessing, integration of new features and consideration of multi-strokes gestures of stochastic Levenshtein.



### 3 Selection of appropriate recognition mechanism

of the pressure combined with the orientations. Indeed, if it does not improve, or even degrades, the performances for the training made dependently of the user and for the first method of the training made independently of the user, the recognition rates becomes better when the number of training examples by class is sufficient for the second method of the training made independently of the user. The optimization of the weights seems thus to take advantage of the integration of the pressure, provided that the number of training examples as well as the variety of different users are sufficient.

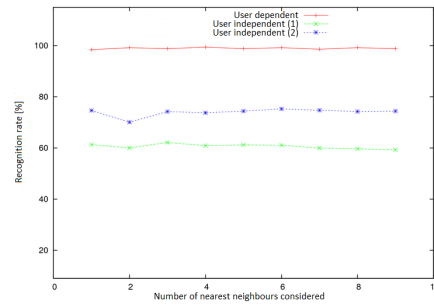


**Figure 3.35** Results for the normalizations of stochastic Levenshtein.

Figure 3.35 shows the results obtained for the normalizations of stochastic Levenshtein. On the contrary of the original Levenshtein's algorithm, these normalizations do not help and even degrade the performances of the algorithm.

Figure 3.36 shows the results obtained for the nearest neighbor extension of stochastic Levenshtein for a training made on 9 numbers of examples by class. It can be observed on the graphs that this extension does not really have any influence on the rates and brings no improvement.

### 3 Selection of appropriate recognition mechanism



**Figure 3.36** Results for  $k$ NN of stochastic Levenshtein.

#### 3.3.4 Selection of the most appropriate algorithms

Firstly, it is important to insist that all the tests of this study were produced on a reduced number of training examples for temporal feasibility reasons. The rates shown do not thus represent the maximum reachable rates by the algorithms but allow to highlight several interesting tendencies based on some criteria. These tendencies can even be considered as references for fast recognition purposes or recognition on a reduced set of training examples.

##### Original algorithms

Three big axes were visited during this experimental study of the original algorithms : 3 training types (one dependent of the user and two methods independent of the user); 4 families of classes (digits, letters, action gestures and geometrical shapes); 4 algorithms (Rubine, One Dollar, original and stochastic Levenshtein).

It appears very clearly that the best way to train the algorithm is dependently of the user, i.e. by providing training examples from the person which is used for the recognition. Nevertheless, such a training is not always possible and a database of training examples is then needed. In this case the results show that, except for the particular case of Rubine on the action gestures, a training on several users is more efficient than a training on one user. The diversity of the user has thus a beneficial effect for the training. On the one hand because it provides better recognition rates in an absolute way, on the other hand because it constitutes a better solution in term of progression since the curves show recognition rates permanently improving for a training multi-users while

### 3 Selection of appropriate recognition mechanism

they are more constant with a one-user training. Increasing the number of users trainers could be compared to increasing the dependency of the training, meaning that the effect is to tend towards a training made dependently of the user. Indeed, the more persons are used for the training, the bigger are the chances to find examples matching well the person used for the recognition.

Concerning the families of classes, two specificities occur for each training type. The first one concerns the stochastic Levenshtein's algorithm which is unable to be trained on the geometrical shapes. This explication comes from the too long string representing the gesture and the problem of the computer when dealing with probabilities close to 0. The second specificity concerns the action gestures for the algorithm of One Dollar. The recognition rates are very bad since the reorientation step causes the matching of dissimilar gestures that are initially only distinguishable by their orientation.

For a training made dependently of the user, the digits represent the best classified set. After come the letter and finally the geometrical shapes and action gestures for which the recognition rates are merged for Rubine and original Levenshtein's algorithms. The way the digits and the letters are linked is quite understandable because these two families of classes gather similar classes. Their major difference is the number of classes (10 for the digits, 26 for the letters). They offer better recognition rates than the geometrical shapes and action gestures thanks to the more important specificity characterizing their classes.

For a training made independently of the user, the action gestures represent the best classified set if we except One Dollar and the second method for stochastic Levenshtein. The conclusions can thus be drawn at the opposite for this training type and it is a normal behavior since the less specificity from one user to another one for the action gestures is an advantage in this situation. However, these conclusions should be moderated for the second method of the independent training since the classification rates between the action gestures and the other gestures are less differentiated, or even less good in the case of the stochastic Levenshtein's algorithm.

Considering the algorithms, One Dollar is the best one for the digits and the letters. It can be explained by the extreme simplicity of the algorithm that only needs a few training examples in order to work at full capacity. Reserve must be shown for this conclusion since more training examples could change it. But training with more examples has a cost that can become very expensive, in particular for the stochastic Levenshtein's algorithm for which tens and even hundreds training examples can be required in order to work at its full capacity.

### 3 Selection of appropriate recognition mechanism

For a fast recognition purpose, the One Dollar's algorithm offers an excellent price-earning ratio (except for the gestures only distinguishable by their geometrical features like action gestures). For the geometrical shapes, One Dollar gives also satisfying results even if they really surpass those for Rubine and Levenshtein only for the second method of the independent training. For the first one, Rubine is the best algorithm and for a training made dependently the different recognition rates tend to merge. For the action gestures, the algorithms of Rubine and stochastic Levenshtein offer the best classification. Finally, a tendency to underline is the behavior of the two algorithms of Levenshtein. The stochastic one always allows better recognition rates but with a higher temporal cost.

For the particular family classes of signatures, the stochastic Levenshtein's algorithm was not tested for the same reasons explained for the geometrical shapes. Considering the three other algorithms, the specificities present among the classes lead to almost perfect recognition rates for Rubine and One Dollar and even totally perfect recognition rates for Levenshtein.

Concerning the execution times, stochastic Levenshtein is about 1000 times slower than the three other algorithms, for the training phase as well as for the recognition phase. It represents its major limitation.

|         | User-dependent         | User-independent 1 | User-independent       |
|---------|------------------------|--------------------|------------------------|
| Digits  | One Dollar             | One Dollar         | One Dollar             |
| Letters | One Dollar             | One Dollar         | One Dollar             |
| Actions | Stochastic Levenshtein | Rubine             | Stochastic Levenshtein |
| Shapes  | Rubine                 | Rubine             | One Dollar             |

**Table 3.1** Original algorithms summary.

Table 3.2 summarizes the different performances for the algorithms in their initial versions by highlighting the best algorithm for each dataset.

#### Refined algorithms

Several algorithms previously studied in their original version were then modified in order to try improving them. The modification do obviously not constitute an exhaustive set of the potential possible improvement, but rather certain ideas and suggestions selected among a large panel of possible choices.

### 3 Selection of appropriate recognition mechanism

Among these possible choices, the preprocessing taking part of the One Dollar's algorithm has been tested on the other algorithms.

The resampling appears very clearly as the more efficient operation of preprocessing since it improves Rubine's algorithm as well as both Levenshtein's algorithms. Moreover, it improves the algorithm of stochastic Levenshtein in such a way that it even surpasses the One Dollar's algorithm, that was the best original algorithm for the digits. However, the price-earning ratio of One Dollar's algorithm stays still better but optimal performances purpose without temporal constraints, the stochastic Levenshtein's algorithm with resampling (and still better with multi-strokes consideration) is the ideal solution.

The reorientation of the gestures is to forgive since it decreases the performances for Rubine and the two Levenshtein's algorithms.

The rescaling does not have any effect on the Levenshtein's algorithms since it does not change the way the strings representing the gestures are built. For Rubine's algorithm, rescaling seems to bring degradation of the performances for a training made on a small number of users while it can bring an improvement when this number increases.

Finally the translation has no effect on the Levenshtein's algorithms for the same reasons explained for the rescaling. It has no effect on the Rubine's algorithm too since this operation is already performed in the original version of the algorithm.

The variation of the number of nearest neighbors were also tested for the algorithms of One Dollar and the two Levenshtein. The lessons learned are the same for the three algorithms : this extension does not bring any improvement and can even decrease the performances.

Additionally, features as pressure, inclination or orientation of the pen were considered. Their inclusions has only been tried for the Levenshtein's algorithms, because the One Dollar's algorithm is space geometrical by nature and no pertinent way to include them to the Rubine's algorithm was found.

The process used to add these features to the Levenshtein's algorithms is by building differently the strings representing the gestures, by considering the features. A preliminary study firstly allowed to learn that building the strings based on the features as themselves leads to very bad performances and is not pertinent. Instead of including the features in this way, considering the point-by-point increasing/equality/decreasing has then been tested and gave slightly better results. Unfortunately the rates were still not sufficient and a third way to integrate the features was imagined : combine the point-by-point increasing/equality/decreasing of the features with the point-by-point orientations of

### 3 Selection of appropriate recognition mechanism

the gestures resulting in a 24 possibilities alphabet. Since the pressure was the best feature, it was selected to try this refinement in combination with the point-by-point orientations to finally lead to a global degradation of the results, except for the stochastic Levenshtein's algorithm with the second method of a training independent of the user on a sufficient number of training examples.

Another modification tested is the adaptation of the algorithms of Levenshtein in order to take the multi-strokes gestures into account. That is to say that for the gaps left by the user between two points, some special symbols have been added in order to simulate the absence of points during the generation of the strings representing the gestures. It constitutes an interesting idea since the performances are improved in most cases, or stay the same at worst.

The distance of Levenshtein has also been normalized following three different ways. This extension of the stochastic Levenshtein's algorithm is to forgive because of the bad rates resulting. However, one of the three normalization brings improvement for the original Levenshtein's algorithm. Even if the improvement is quite light, it is interesting because of the easiness to implement this extension.

Finally, special substitution costs adapted to the orientation have been tried for the original Levenshtein's algorithm. During the computation of the distance of Levenshtein, instead of fixing the costs of substitution between two different symbols to 1, different costs were considered in function of the involved symbols and thus directions represented by these symbols. The results are pretty good and show an increasing of the performances.

Table 3.2 gives a performance measure for each modification of the algorithms in comparison with their initial version: '−' for hard decreasing; '-' for decreasing; '=' for similarity; '+' for improvement; '++' for hard improvement; 'X' if not applicable.

#### 3.3.5 Recommendations for gesture sets creation

Creating an application integrating pen-based gesture recognition is not only an algorithmic challenge. Choosing the right gestures from the end-user perspective is and an important and difficult task that needs to be carefully executed in order to make the application more user friendly. In the context of the application supporting their gesture recognizers, Microsoft proposes very interesting recommendations when building a gesture set (gestures):

### 3 Selection of appropriate recognition mechanism

|                           | Rubine | One Dollar | Levenshtein | Sto. Lev. |
|---------------------------|--------|------------|-------------|-----------|
| Resampling                | +      | X          | ++          | ++        |
| Rescaling                 | +      | X          | X           | X         |
| Rotation                  | -      | X          | -           | -         |
| k $\mathcal{NN}$          | X      | -          | -           | -         |
| Multi-stroke              | X      | X          | +           | =         |
| Multi-stroke + resampling | X      | X          | ++          | ++        |
| Pressure                  | X      | X          | -           | -         |
| Normalization 1           | X      | X          | +           | -         |
| Normalization 2           | X      | X          | =           | -         |
| Normalization 3           | X      | X          | =           | -         |
| Adapted costs matrix      | X      | X          | +           | X         |

**Table 3.2** Refined algorithms summary.

- Support commonly used real-world gestures such as scratch-out.
- Keep the number of gestures manageable. Recognition performance suffers as the number of gestures increases. Users may get confused if there are too many to remember.
- Choose gestures with glyphs that are distinct enough to avoid confusion by the user and the gesture recognizer. For example, don't include both the Up Left and the Up Left Long gestures in the same user interface.
- Choose gestures with glyphs that suggest the functions associated with them.
- Maintain consistency with the suggested actions for each application gesture unless using an alternate action is both important to your application and does not confuse users.

An additional very interesting criterion for good gestures is their guessability. Wobbrock et al. present an unified approach to both maximizing and evaluating the guessability of symbolic input, such as pen-based gestures (Wobbrock et al.).

Vatavu et al. present empirical results that can act as a guidance when building a gesture set. More specifically, they show different factors influencing the perceived difficulty of gestures. They show that it is highly correlated to execution time and that testing on a few examples is in general sufficient to

### 3 Selection of appropriate recognition mechanism

estimate this complexity. In a general way, complexity is also correlated to the number of constraints of the gesture (e.g. a triangle in a rectangle to represent a combobox is quite complex as the triangle must be circumscribed).

An example of gesture set is the gesture set associated with navigation commands. Navigating back or forward are commonly represented by the left and right gestures as they are simple, easy to remember and represent real-world gestures that would naturally be used for such actions.

Choosing the right gesture set is therefore the first task to perform when selecting a good recognition mechanism. On this basis, the algorithm chosen to support the dataset will have to be selected in a way to maximize the recognition rate. As the experiment has shown, creating a new training gesture set is a long and fastidious but necessary task to train and select the best algorithms. An interesting possibility to ease this step is to follow the approach proposed by Gestures à Go Go (Leiva et al. [2016]): instead of collecting all the gestures from users, only one gesture by class needs to be provided and variations of the gestures are programmatically created based on the kinematic theory of rapid human movements.

## 3.4 Conclusion

This chapter presented a platform allowing to conduct comparative studies and benchmarking of recognition algorithms. It then provided a comparative study for different gesture datasets, recognition algorithms, and testing conditions. The results of this comparative study will be reused in the second tool presented in Chapter 5. Next chapter will describe the modelling aimed at supporting user interfaces integrating pen-based gesture recognition, and hence components representing the assets as algorithms and gesture datasets discussed in this chapter.



# **Chapter 4** Conceptual modelling of graphical user interfaces incorporating pen-based gesture recognition

The previous chapter presented the recognition mechanism part of user interfaces integrating pen-based gesture recognition. The goal of this chapter is to propose a modelling aimed at supporting the creation of such user interfaces. This modelling is mainly composed of two parts: the first one describing the classical graphical user interface part, and the second one describing the gesture part, including the assets discussed in the previous chapter.

## **4.1 Introduction**

The Task and Abstract User Interface (AUI) meta models are beyond the scope of this work. The AUI model is independent of the modality and the Task model even more abstract are indeed not suitable to describe the incorporation of gesture in graphical user interfaces. The thesis focuses on the Concrete User Interface model describing modalities and more specifically the graphical modality.

We propose here a CUI model to describe the graphical interfaces, along with a gesture model supporting the incorporation of the recognition mechanisms in the interfaces. They are both presented in the rest of this chapter, and the details are available in Appendix D.

## 4 Conceptual modelling of graphical user interfaces incorporating pen-based gesture recognition

### 4.2 Concrete User Interface

User Interface eXtensible Markup Language (UsiXML) is a formal Domain-Specific Language (DSL) used in Human-Computer Interaction (HCI) and Software Engineering (SE) in order to describe any user interface of any interactive application independently of any implementation technology. UsiXML is fully compliant with the four levels of abstraction defined in the Cameleon Reference Framework.

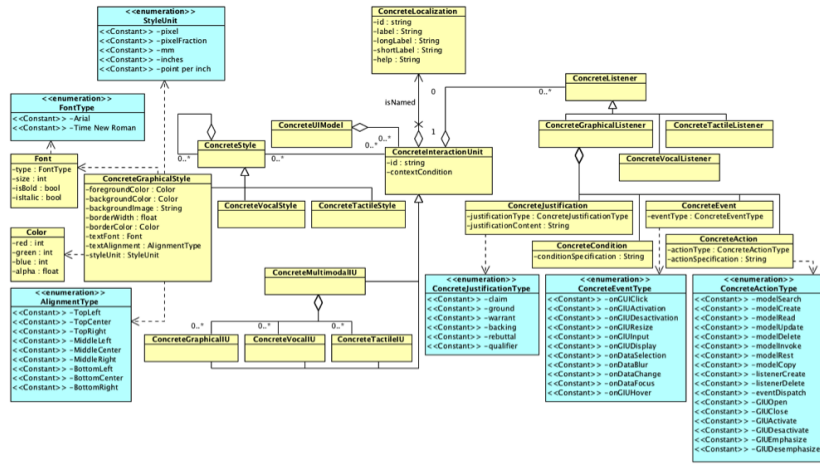


Figure 4.1 CUI Overview.

Figure 4.1 displays the overview of the CUI meta-model of UsiXML. For the complete description of all the entites and the relationships, please refer to Appendix D.1. As the author of this meta-model, we are the main contributor for this part of UsiXML but this work was realized in colloboration with the UsiXML consortium.

The central unit of the CUI meta-model is the *ConcreteInteractionUnit*. It represents any unit that can be found in an interface, for any modality. It can be graphical (e.g. a button), vocal (e.g. a speech input), tactile (e.g. a trackpad). These thrree modalities are the currently supported modalities but any other one can be added. Additionally, the multimodal modality is an aggregation of components with different modalities.

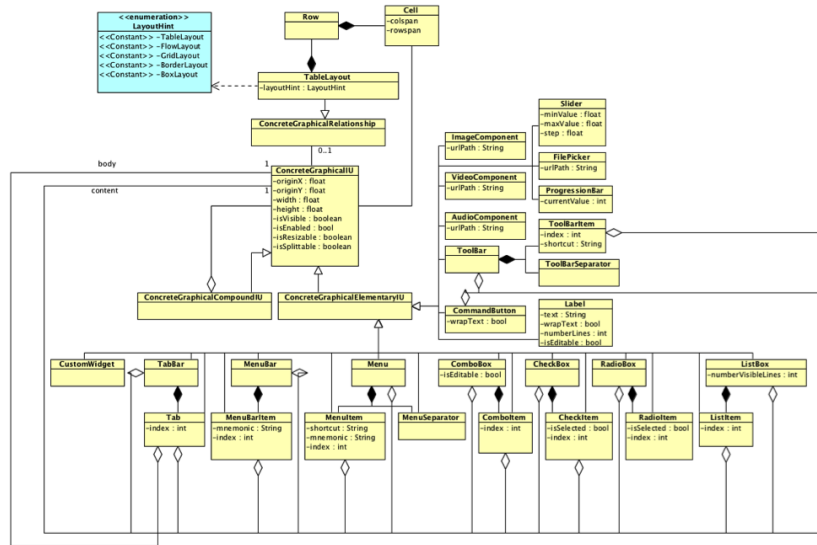
Any unit can be associated with a *ConcreteStyle* defining its style. At the moment, the graphical modality is the most specified one. The CUI meta-model proposes different possibilities in order to stylize the concrete interaction units: color (foreground and background), background image, border

#### 4 Conceptual modelling of graphical user interfaces incorporating pen-based gesture recognition

(width and color), text font, text alignment and the measurement unit considered (i.e. pixel, mm, inches, etc.).

The *ConcreteLocalization* is useful to name the interaction unit and confer it labels (default, short, long) and a help text.

Along with the static attributes and properties, an interaction unit must be associated with behaviors. This is the role of the *ConcreteListener* which allows to specify rules in order to describe behaviors. For the graphical part, these rules take the form of JECA rules, standard for Justification - Event - Condition - Action. The justification is not directly involved in the definition of how the rule is actually executed but has more a role of documentation for the motivation. The rule becomes then: according to {Justification}, when {Event}, if {Condition} then {Action}. The condition is expressed via a string containing boolean expressions. The events and actions can be multiple and are listed through enumerations in the meta-model. As an example, a button could be associated with a rule defining the following: when the button is clicked, if the input fields in the page are not modified, then the application can navigate to another interaction unit.



**Figure 4.2** CUI ConcreteGraphicalIU.

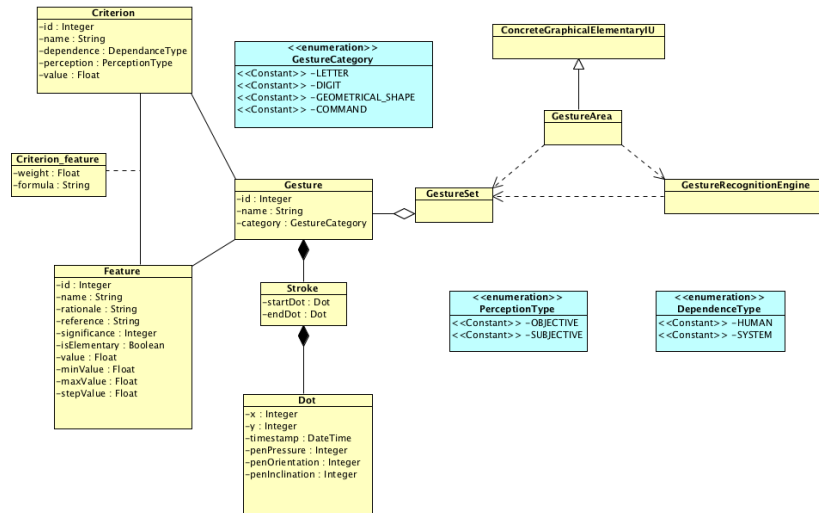
Figure 4.2 shows the specification of the concrete interaction unit for the graphical modality. It is articulated around a composite pattern allowing to express the recursive properties of this unit. It is indeed refined either in a *ConcreteGraphicalElementaryIU* or *ConcreteGraphicalCompoundIU*. The

#### 4 Conceptual modelling of graphical user interfaces incorporating pen-based gesture recognition

graphical elementary interaction unit represents any smallest subdivision of a graphical interface and takes the form of many different kind of units: tab bar, combobox, checkbox, image component, and many more. The graphical compound interaction unit is an aggregation of these elementary units and other compound units. With this kind of relationships, it is then possible to express the most simple graphical user interfaces to the most complex ones.

The *ConcreteGraphicalRelationship* allows to describe the way sub-elements are organized in their parent. Depending on the layout hint, the compound interaction unit can be organized like a table with rows and columns, like a flow layout with content placed from left to right with justification, etc.

### 4.3 Gesture System Modeling



**Figure 4.3** Gesture system modeling.

Figure 4.3 shows the modeling describing the gesture system. For the complete description of all the entites and the relationships, please refer to Appendix D.2.

The gesture system meta-model is linked to the concrete user interface meta-model through a specific *ConcreteGraphicalElementaryIU*: the *GestureArea*. This area is a drawing area with the ability to understand the gesture the user sketch. For this purpose, it is linked to two high level components: the *GestureRecognitionEngine* and the *GestureSet*. The gesture recognition engine

#### 4 Conceptual modelling of graphical user interfaces incorporating pen-based gesture recognition

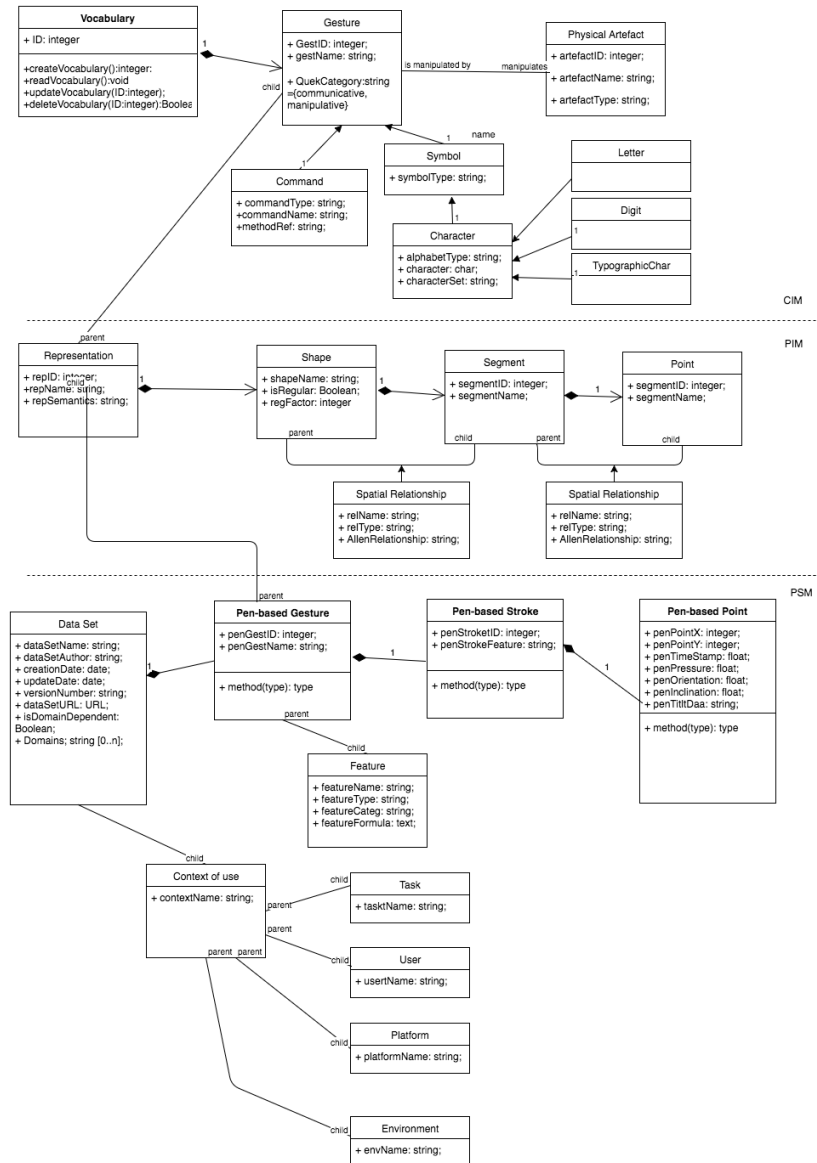
mainly represents the different algorithms possibly used to perform the gesture recognition. It is voluntarily left unconstrained as the possibilities of representation are very various and numerous. The gesture set represents a set of gestures, each gesture is itself composed of a set of strokes, and a stroke is itself composed of a set of dots.

### 4.4 Gesture ontology

In order to further detail the gesture modelling, the ontology depicted in Figure 4.4 is proposed. It is divided into three levels:

- the Computing Independent Model (CIM) represents the real world in which any gesture may occur independently of any implementation in any computer-based system. This level therefore defines gestures in semantic terms. For instance, a “Stop” gesture may convey a special meaning of stopping an action, therefore being represented as a command. Gestures are gathered in a vocabulary that may differ depending on the user and its environment. Different people may indeed have different vocabularies of gestures, even for the same goal. Each gesture may be associated to either a command or represents a symbol (e.g., an icon), of which it could be any character, such as a letter (e.g., “A” or “ $\mu$ ”), a digit (e.g., 0..9) or any other typographic character (e.g., “%”).
- the Platform Independent Model (PIM) expresses how a semantic gesture could be represented in the computer world, but independently of any implementation on any platform. Each gesture may have one to many representations, each representation being further decomposed into shapes that consist of segments linked with relationships. For instance, the gesture for the “7” digit may have different representations, such as the European one (with a bar in the middle) or the American one (without any bar inside) or extra (e.g., a rounded seven). Each such representation is associated to a shape, which is decomposed into segments.
- the Platform Specific Model (PSM) expresses how a gesture representation could be effectively expressed in a particular implementation on a particular platform (including devices). For instance, a European “7” could be captured via different input devices, e.g., a pen on a surface (which is the case that we consider in this thesis), a touch screen or a

#### 4 Conceptual modelling of graphical user interfaces incorporating pen-based gesture recognition



**Figure 4.4** Gesture ontology.

#### 4 Conceptual modelling of graphical user interfaces incorporating pen-based gesture recognition

mid-air 3D gesture. Each of these captures will lead to a particular characterization, the sum of which being gathered in a data set, which is a particular gesture set represented on a particular platform. For instance, digits may hold different captures depending on the context of use in which they are captured and used. In this thesis, each pen-based gesture is further decomposed into strokes, which are in turn decomposed into a series of points, whose sampling depends on the input device. For instance, an Apple FinePoint will capture a gesture with a higher sampling than, perhaps, a Wacom pen on a large SmartBoard.

The CIM consists in a *Vocabulary* around a topic (e.g. navigation) gathering different *Gesture*. The *Gesture* is itself either a *Command* (e.g. “Go to next page”, or “Go to previous page”) or a *Symbol*, that is subclassed here by a *Character* (e.g. the “R” letter to reload the page). Additionally, a *Gesture* is manipulated by a *Physical Artefact* (e.g. the hand or a pen).

The PIM specifies the CIM *Gesture* with a computing *Representation*. This representation consists in a *Shape* that is divided into many instances of *Segment*. A *Segment* gathers many instances of *Point*. The segments as well as the points are linked through *Spatial Relationship*.

The PSM then specifies the PIM *Representation* with a specific platform equivalent, in our case the *Pen-based Gesture* (e.g. the “R” letter drawn with a pen). It is itself divided into many *Pen-Based Stroke* and the pen-based stroke is then divided into many *Pen-based Point*. The spatial relationships are expressed thanks to the positions of the points. The *Dataset* gathers different *Gesture* and represents the CIM *Vocabulary*. Furthermore, it is associated with a context of use.

## 4.5 Conclusion

This chapter proposed a meta-model to describe concrete user interfaces (and more specifically graphical ones) and a meta-model to capture pen-based gestures and include them in a graphical user interface. These meta-models are purely conceptual and require an instantiation specific to the use case belonging to this thesis. The next chapter will present the proposed UsiGesture tool integrating this instantiation.





## Chapter 5 Approach and tool

This chapter first proposes an approach aimed at fostering the collaboration between the actors involved in the process of creation of a user interface integrating pen-based gesture recognition. It then creates an instantiation of the modelling proposed in the previous chapter. It finally describes a tool to support the full process of pen-based user interfaces creation supporting this modelling and approach. Together, they form the fully integrated and structured UsiGesture method. The chapter ends by giving an example of a creation of such interface through the method.

### 5.1 Approach

Creating a user interface is a complex task requiring many skills. In the perfect case, all these skills would belong to a single developer that would then be able to conduct the whole process of creating the user interface alone. In practice, it is rarely the case since it is difficult for a single developer to acquire all the needed knowledge for each sub-process involved in the user interface creation. The rest of this section describes the different actors involved in this process and proposes a methodology fostering their collaboration, partly inspired by the SCRUM methodology described in Chapter 2.

The usual process of creating a user interface can be divided in two distinct parts: during the conception phase, the engineer/architect analyzes the user preferences, environment parameters, and miscellaneous requirements, and elaborates the specifications of the interface with the help of the designer. Based on these specifications, the programmer actually codes the user interface during the implementation phase.

We are here interested by the specification of user interfaces including gesture or sketch manipulation. Taking such a feature into account would impact the

## 5 Approach and tool

two phases of the user interface creation. A gesture specialist helps the engineer/architect and the designer during the conception phase. The programmer codes the gesture mechanism based on the specifications during the implementation phase, or is helped by a second programmer specialist in this domain.

Many observations of UI creation processes could be realized in different organizations (Universities of Louvain-la-Neuve, Valencia, Grenoble and companies like Telefonica, Thales, Defimedia, Namahn, etc.) through different projects (UsiXML, FEDER eHealth, First Enterprise). Based on these empirical observations, we thereby define two phases and four roles characterizing the creation of gesture-based user interfaces:

1. *Conception phase*: outputs a UI description.

- The **engineer/architect** analyzes the different requirements elicited by the user, the environment, or any other input and identifies the different parts to be included in the user interface (widgets) as well as the behavior enabling interaction with the user and between the widgets. He is taking care of the ergonomics of the system.
- The **designer** is in charge of the aesthetics of the user interface. His role is to choose the right layout parameters (size, color, font, etc.) for each part of the user interface, and follow aesthetics rules based on metrics such as density or balance. He helps the engineer/architect ensuring good ergonomics.
- The **gesture specialist** is devoted to the recognition mechanism specification with its different parameters.

2. *Implementation phase*: based on the UI description elaborated in conception phase, outputs the final code of the UI.

- The **programmer(s)** are the builders of the user interface. Based on the specifications of the conception phase, they actually code it. This includes the recognition mechanism, i.e. the algorithms and the gesture datasets.

The roles defined in the conception phase are well delimited, but in practice, they are more interfering and lead to a real cooperation. Additionally, if the conception and implementation phases remain independent, they will rarely

## 5 Approach and tool

be performed one time each. After the conception and the implementation, the generated user interface is tested, often leading to a conception refinement. The different phases are then repeated until convergence on a satisfying user interface. Our goal is to provide a system supporting the conception and implementation phases, enacting the interaction between the different actors.

At this point, we have considered providing a system that can be **used** by engineers/architects, designers, gesture specialists and programmers to create a gesture-based user interface. But how to tune such a system? What widgets, behavior, algorithms, elements of design, etc. should be proposed? We are not specialists and we do not have competences to make any assumptions on what is important or not. Furthermore, technologies are constantly evolving, how to update the system in order to reflect them?

The system should be flexible and extensible to be able to easily acquire new knowledge from skilled people. These people are engineers/architects, designers and gesture specialists, helped by programmers. Instead of being users of the system, they become **feeders**.

At this level, the role of the programmer(s) is slightly different since it consists in programming the engines allowing the automatic generation of the user interface specification (i.e. every element of knowledge determined by engineers/architects, designers and gesture specialists) in final code, rather than directly programming the user interface by transforming the specifications “manually” (thanks to the direct mapping between the XML specifications and Java code provided by WindowBuilder). In this view, the role of the programmer(s) as user(s) of the system is not useful anymore and may then be suppressed for this level.

Around these different descriptions, we are now able to formalize a method enacting the creation of gesture-based user interfaces. For this purpose, we define three different roles acting at three different levels of interaction:

- The **Interface Users (IU)**: end users of the interface.
- The **System Users (SU)**: first group of engineer(s)/architect(s), designer(s) and gesture specialist(s) using the system in order to produce the user interface for the Interface Users.
- The **System Feeders (SF)**: second group of engineer(s)/architect(s), designer(s), gesture specialist(s) and programmer(s) feeding the system with knowledge allowing SU creating user interfaces.

## 5 Approach and tool

The roles are not exclusive. People acting as System Feeders may be System Users (e.g., a gesture specialist that creates an algorithm and then tunes and includes it in the user interface) or Systems Users may be Interface Users (e.g., engineer(s)/architect (s), designer (s) and gesture specialist(s) wanting to use the interface they have created with the system), etc.

The method is then defined in seven steps:

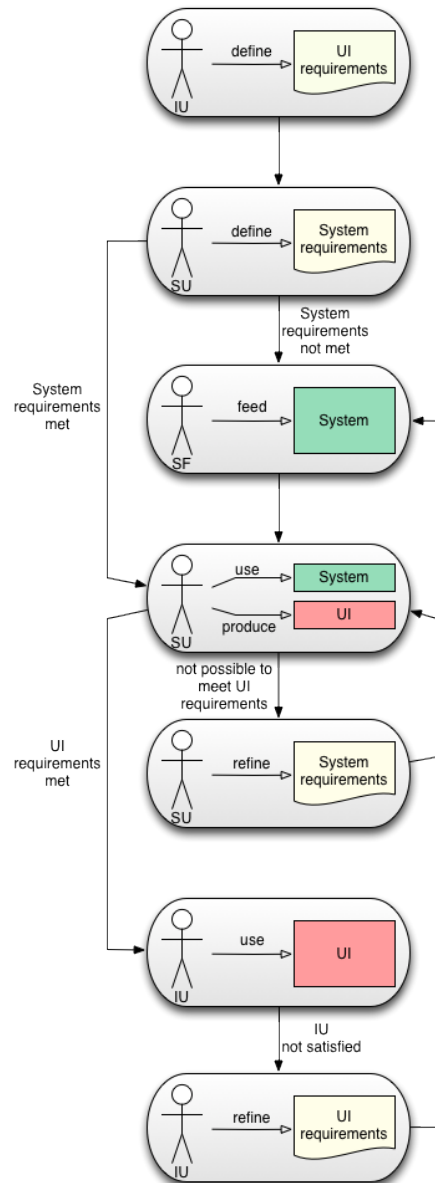
1. Interface Users define user interface requirements ;
2. Based on the UI requirements, System Users define system requirements ;
3. If system requirements not met: based on the system requirements, System Feeders feed the system ;
4. Based on UI requirements, System Users use the system to produce the user interface ;
5. If UI requirements not met: System Users refine system requirements, then go back to step 3 ;
6. Interface Users use the produced user interface ;
7. If Interface Users not satisfied: Interface Users refine UI requirements, then go back to step 4.

When the interface users are satisfied, the method stops iterating. The whole process is illustrated in Figure 5.1.

The main advantage of such a system is its ability to evolve through the different user interfaces creations. It may be viewed as a structured repository of gesture-based user interfaces knowledge. The more user interfaces are produced, the more the system is fed with knowledge, allowing any system users taking profit of all the experience acquired during previous work.

Next section presents the choices we made to model the system on which the approach is relying.

## 5 Approach and tool



**Figure 5.1** Method.

## 5.2 Model

The Cameleon Reference Framework (Calvary et al. [2003], UsiXML-D1.3) defines the description of a user interface at different levels of abstraction: Tasks and Concepts, Abstract User Interface (AUI), Concrete User Interface (CUI) and Final User Interface (FUI). Each level is formalized by a User Interface Description Language (UIDL) representing the user interface for the specific level. Different mechanisms of forward and reverse engineering allow translating one level into another one.

The user interface description elaborated during the conception phase of the method proposed in previous section actually corresponds to the CUI UIDL. CUI is then transformed in FUI in the implementation phase. This is the modelling approach chosen to support the method.

The CUI UIDL we decided to adopt is Eclipse XML Window Toolkit (XWT) and constitutes a derivation of the meta-models presented in Chapter 4. This language allows specifying user interfaces with XML representation by relying on Standard Widget Toolkit (SWT), which is an open-source widget toolkit for Java. XWT provides a very interesting piece of work, including the description of an important set of SWT widgets as well as a tool supporting the XWT and graphical edition of user interface (see next section). The chosen FUI language is naturally Java since it is directly supported by XWT.

Although XWT constitutes a very useful basis, it is not sufficient to achieve the targeted method. SWT proposes many widgets with design and behavior features. As aforementioned, XWT provides an XML description of an important part of them, but this description only concerns the widgets themselves with their elements of design. The behavior part is not fully described by XWT: although it is possible to specify listening to events, launching an action provided some condition is not possible with XML syntax. Furthermore, no mechanism is devoted to gesture interaction. We then chose XWT as our core model, extended to meet our requirements.

Figure 5.2 depicts the model including the XWT package, with the behavior extension part (bottom block) and the sketch extension part (top-left block). The three next subsections respectively detail XWT, behavior and gesture parts.

## 5 Approach and tool

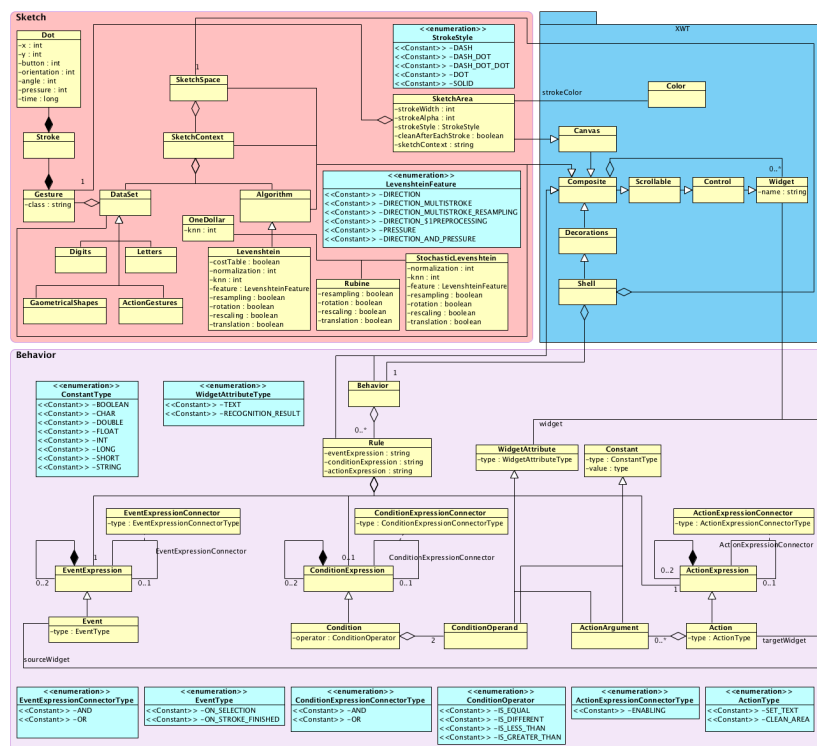


Figure 5.2 Model.

### 5.2.1 XWT Package

Only a subset of the XWT model is represented since the whole model is too big and contains useless information at this level. The attributes are not shown for the same reasons.

The main principle of this model is that most entities are derived from the concept of Widget, which defines a set of methods and attributes. Control and Scrollable both add other methods and attributes to Widget to increase the available functionalities. These three classes are sub-classed by many concrete classes as Button, Label, etc. Among them, three important widgets will be the starting points for the behavior and gesture extensions:

- Composite: container of widgets.
- Canvas: special container defining a paintable zone.
- Shell: root container representing the user interface.

Additionally, Color entity is represented since it will be useful for the gesture block.

### 5.2.2 Behavior extension

The extension proposed for the behavior is articulated around two main entities:

- *Behavior*: main behavior entity enclosing all the needed information in a set of Rule. It is unique for a user interface and belongs directly to the Shell.
- *Rule*: represents an Event-Condition-Action rule (ECA rule). “eventExpression”, “conditionExpression” and “actionExpression” attributes are the string representations of the rules elements described in the followings.

An ECA rule in its basic form allows specifying an action to be launched when an event occur, under a certain condition. In order to add more expressivity, we considered event expressions, condition expressions and action expressions



## 5 Approach and tool

instead of events, conditions and actions. The corresponding entities in the model are *EventExpression*, *ConditionExpression* and *ActionExpression*.

An event expression is a set of events interconnected by connectors. Additionally, subsets of events may be interconnected with other subsets of events, or simple events. *EventExpression* is either a simple event, or a group of two *EventExpression* interconnected with an *EventExpressionConnector*, which allows modelling the described event expression. The simple event is modelled with *Event* entity, which has a source widget and an event type. There is no explicit constraint between the source widget and the type of event, but the programmer must take care of associating the possible combinations during the implementation phase.

The condition expressions and the action expressions follow, with different types of connectors, the pattern adopted for the event expressions. However, conditions and actions are different from events. A condition is represented by *Condition* entity, which is a couple of two operands linked through an operator. These operands may be either a constant (integer, string, etc.) or a widget attribute (represented by *WidgetAttribute* entity). An action is modelled with *Action* entity, which has a target widget on which the action applies, an action type and a list of arguments to be taken by the action type. There is no explicit constraint between the target widget, type of action and the arguments, but the programmer must take care of associating the possible combinations during the implementation phase.

Let's finally notice that both *Behavior* and *Rule* entities are sub-classing *Composite* entity. With this way of modelling – strongly suggested by common practices in XWT and partly imposed by the tool (presented in next section) – the behavior and the rules are considered as being entirely part of the interface. It is usual in XWT, since it allows reusing already defined entities with useful attributes and functionalities. In this case, *Behavior* and *Rule* take profit of the *Composite* functionalities, but have a visibility set to false since they do not need graphical representation in the interface. We only chose these two elements to be explicitly part of the interface because we think specifying the event, condition and action expressions as strings is less constraining, more flexible and more straightforward.

### 5.2.3 Gesture Extension

Similarly to behavior extension, gesture extension relies on entities subclassing *Composite*:

- *SketchSpace*: main sketch entity enclosing all the needed information in a set of *SketchContext*. It is unique for a user interface and belongs directly to the Shell.
- *SketchContext*: defines a sketch context gathering different algorithms and datasets.
- *DataSet*: collection of gestures, themselves composed of strokes, themselves composed of dots. Dots gather different information of position, time, and stylus state (if can be applied).
- *Algorithm*: represents the engine able to classify a new gesture, provided the datasets of the sketch context parent.
- *SketchArea*: paintable area linked to sketch context in order to provide recognition feature to the gesture being drawn on it. “*sketchContext*” attribute is the widget name of the sketch context it is linked to. “*strokeWidth*”, “*strokeAlpha*” and “*strokeStyle*” provide control on the stroke aspect, and “*cleanAfterEachStroke*” indicates if the area must be cleaned each time a stroke is drawn.

As *Behavior* and *Rule*, *SketchSpace*, *SketchContext*, *DataSet*, and *Algorithm* are not intended to be visible in the FUI, while *SketchArea* is a visible component. The other entities are not part of the core description of the gesture extension and will be explained in the next subsection.

### 5.2.4 Feeding and Using the System

At this point, we have presented the entities of the core model describing the system aimed at supporting the method. In other words, without more specification this model is “naked”, it describes a system ready to acquire any knowledge, but currently empty of knowledge. Increasing this knowledge is the role of the System Feeders. At model level, it corresponds in refining and/or increasing some specific entities in order to describe the features the system supports. The System Users will then use this model as a kind of documentation

## 5 Approach and tool

of the system. The way we expect the System Feeders to extend the model is different for each part.

We think the behavior extension allows an advanced expressivity for many rules or composition of rules. However, the types of events, conditions, actions and widgets attributes are missing, as well as the connector types between expressions. For this part, *EventType*, *ConditionOperator*, *ActionType*, *WidgetAttributeType*, *EventExpressionConnectorType*, *ConditionExpressionConnectorType*, and *ActionExpressionConnectorType* are the enumerations to be increased in order to specify the possible events, conditions and actions, and the way they are connected. We already provide some possibilities for each enumeration. They are given as examples and must not be considered as mandatory attributes.

For the gesture extension, the way of adding new knowledge is different. The sketch contexts must actually contain sub-classes of *DataSet* and *Algorithm*, representing specific datasets composed of several Gesture, and specific algorithms with their options. Again, we provide some datasets and algorithms examples that we elaborated during the experimental study.

The datasets we provide are the datasets described previously: the digits, letters, action gestures and geometrical shapes.

In addition to the behavior and the gesture extensions, System Feeders may add other new functionalities. The XWT widgets may be extended in order to add any new component with a specific purpose. This new component may then be linked to the behavior extension or even the gesture extension if needed.

Next section shows the tool we developed in order to implement the system. It is described by the model presented in this section and supports the method.

### 5.3 Tool

WindowBuilder (WindowBuilder) is a powerful and easy to use bi-directional Java GUI designer built as an Eclipse plug-in. It supports different toolkits, including SWT and its XML representation XWT. It may be viewed as a visual CUI editor outputting XWT user interface specifications. Moreover, it provides a Java rendering engine for the XWT package described in the previous section.

## 5 Approach and tool

We decided to choose this tool since it fits our requirements by providing a CUI editor and a Java rendering engine to produce the FUI from the CUI. Nevertheless, it must be improved in order to support the behavior and gesture extensions. This section first shows how to use the tool to create user interfaces integrating pen-based gesture recognition, and then shows how to feed the tool to enrich it with new knowledge supporting more use cases of such interfaces.

### 5.3.1 Using the Tool

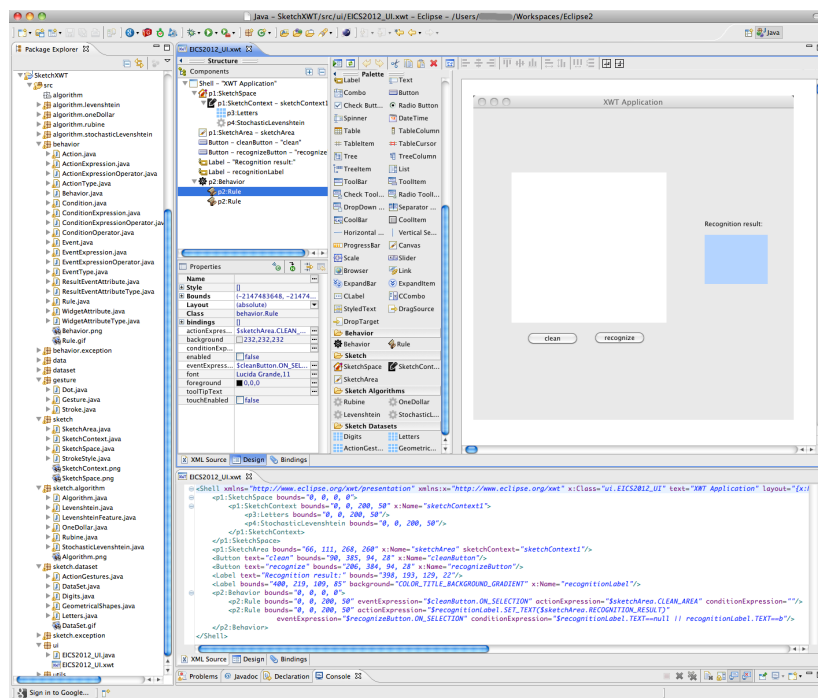


Figure 5.3 Tool.

Figure 5.3 shows a screenshot of the extended WindowBuilder environment (it may also be seen in action at the following addresses: <http://dai.ly/GJduWS> and <http://bit.ly/1YIqtzr>) The top-right part is the visual CUI editor allowing, through the widget palette, manipulating standard SWT widgets as well as behavior and gesture widgets, in order to build the description of a pen-based user interface. A tree representing the interface is available along with a properties panel useful to specify any option or attribute of a widget. The bottom part of the tool shows the XWT sources of the specification of the interface.

For editing the user interface specifications, System Users have three choices:

## 5 Approach and tool

through the visual CUI editor, by editing the tree representation, or directly by updating the XWT description. In the three cases, the changes are reflected in the other representations.

To build an interface, System Users place widget elements in the visual editor, in the tree, or directly write them in the XWT specification. They define the behavior with one and only one *Behavior* widget in the *Shell* and add a set of *Rule* to it. Each Rule may be specified with event, condition and action expressions as String attributes, as described in the model.

The tool adds an interesting feature on the behavior by providing validation. It is used to throw an exception when the System Users specify a malformed or invalid event, condition or action expression. The validation may be turned off via an attribute in the *Behavior* widget in order to let the possibility of describing a different syntax than the one expected by the system. In this case, the running interface provided by WindowBuilder may not work, but in some cases it may be useful to create a specification based on a custom syntax. Concerning the gesture part, System Users are expected to place one and only one *SketchSpace* widget in the Shell and add a set of *SketchContext* to it. These *SketchContext* gather one or many *Algorithm* and *DataSet* that can be tuned with the different options described in the model. The *SketchArea* is the sketch element with a visible representation in the interface, allowing the user drawing on it and performing recognition on the gesture drawn. The *SketchArea* has many options, and must be linked to a *SketchContext* through one of them.

The specifications of the user interface are described in the *.xwt* file generated when a new XWT application is created. In order to actually produce the user interface, the System Users have to run the *.java* associated file, which will parse the CUI specifications and render the FUI in Java.

### 5.3.2 Feeding the Tool

The left column of the tool provides a view on the sources of the system implementation. Each entity described in the model has a class corresponding to its implementation.

System Feeders may add enumeration constants to *EventType*, *ConditionOperator*, *ActionType*, *WidgetAttributeType*, *EventExpressionConnectorType*, *ConditionExpressionConnectorType*, and *ActionExpressionConnectorType* classes

## 5 Approach and tool

in order to increase the behavior possibilities. The four first ones allow increasing respectively the set of events, conditions, actions and widget attributes, while System Feeders may add new connectors respectively between the events, the conditions and the actions through the three last ones.

The schema of the aforementioned validation feature is set in *EventType*, *ConditionOperator*, *ActionType* and *WidgetAttributeType* classes. Indeed, each constant of these enumerations can optionally take one or several arguments in its constructor, indicating the schema of validation:

- *EventType*: takes a table of Class objects as source specification in order to specify the allowed widget types from which the event may be issued.
- *ConditionOperator*: takes a first Class object as left operand specification and a second Class object as right operand specification to specify the allowed types than may be compared.
- *ActionType*: takes a Class object as target specification in order to specify the allowed widget type on which the action may be applied, and a table of Class objects as argument specification in order to specify the allowed types for each action argument.
- *WidgetAttributeType*: takes a table of Class objects as specification in order to specify the allowed widget types from which the widget attribute may be called.

For the constants of the four enumerations, no schema specification means no restriction. Validation exceptions are thrown if the System Users do not respect these specifications, use a malformed syntax, or try to use inexistent widgets, events, actions, etc.

Feeding the gesture recognition mechanism is done by providing new algorithms or datasets. A new algorithm is added by sub-classing the *Algorithm* class and providing it all the wanted options with getters and setters. It is the way *WindowBuilder* works to show an attribute in the visual editor. New datasets are added by sub-classing *DataSet* class and providing option attributes with getters and setters. The new algorithms and datasets must then be added to the widget palette of the tool (simply by adding a new component to the palette through the contextual menu, and then selecting the newly created class).

## 5 Approach and tool

At this point, no recompilation of the tool or any further action is required from the System Feeders: the System Users are able to use the new elements of behavior or gesture mechanism in the user interface specification. However, new elements are not implemented for working during the runtime yet. The rendering engines have still to be updated in consequence by the programmer(s) to reflect the new types of events, conditions, actions and widget attributes, the new types of connectors or the new algorithms and datasets. For the behavior part, the programmer(s) must edit the *EventExpression*, *ConditionExpression*, *ActionExpression*, *Event*, *Condition*, *Action* and *WidgetAttribute* classes. Each new feature (new types of event, condition, action, or new types of connections between them) must be handled in one specific method of one of these classes.

For the gesture part, the programmer(s) must modify each new sub-class of *Algorithm* and of *DataSet*. For each algorithm sub-class, train and recognize methods must be implemented in order to train the algorithm and return the string class of the gesture to recognize. For each dataset sub-class, dataset method must be implemented in order to return set of *Gesture*. For the implementation we provide, all the logic allowing fulfilling the methods are contained in *algorithm* and *dataset* packages. All the used gestures are provided as text files and located in the records folder.

This section has given a description of the few pieces of code to update in order to increase the tool possibilities, the rest of the platform is expected to remain unchanged. It globally shows how the platform is articulated, but does not constitute an exhaustive documentation, which is present in the code. Next section presents an example showing how the method, model and tool are used to build a pen-based user interface.

### 5.4 Example

The user interface chosen for the example targets document manipulation, and more specifically document navigation. The goal is to show how the method, the model and the tool can be used in order to create a system such as Papier-Craft (Liao et al.), which allows using gesture as commands for manipulating a paper-based interface. Additionally, at the beginning we assume that the system is empty of any knowledge, i.e. without sub-class of *DataSet* and *Algorithm*, and with empty enumerations in the behavior part. We will start from user interface requirements produced by Interface Users and show how the

## 5 Approach and tool

System Users with the help of the System Feeders will build the user interface by iterating through the different steps of the method (Figure 5.1).

### ■ Step 1 – IU define UI requirements

The user interface is a viewer for a document containing several pages. The viewer must have the capability of changing the current page to previous or next page by using gestures.

### ■ Step 2 – SU define system requirements

The system must propose a component on which it is possible to specify a background corresponding to the current page of the document, and triggering the recognition of left and right action gestures as the actions for going to previous and next pages. The system requirements are not met then go to step 3.

### ■ Step 3 – SF feed the system

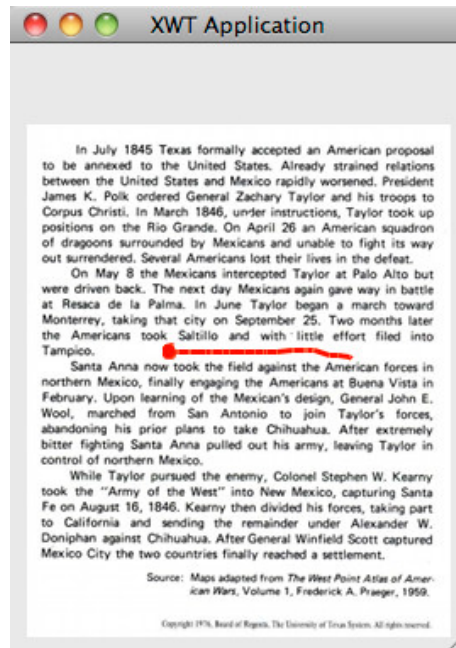
Engineer(s)/architect(s), designer(s) and gesture specialist(s) update the CUI UIDL by creating an *ActionGestures* dataset and a *Stochastic Levenshtein* algorithm with options. They create a new *PageViewer* component on top of a sketch area allowing specifying different pages. They additionally update *EventType*, *ActionType*, and *WidgetAttributeType* by providing them respectively *ON\_STROKE\_FINISHED* (for monitoring the end of an action gesture on the document viewer); *PREVIOUS\_PAGE* and *NEXT\_PAGE* (for modifying the current page on document viewer); *RECOGNITION\_RESULT* (for getting recognition result on the document viewer). They add also a *IS\_EQUAL ConditionOperator*. The programmer(s) then fulfill the dataset method of *ActionGestures*, the train and recognize methods of *Stochastic Levenshtein*, the perform method of *Action* and the *getWidgetAttribute* of *WidgetAttribute* in order to implement the rendering engines reflecting the knowledge introduced in the CUI UIDL.

### ■ Step 4 – SU use the system to produce the UI

SU put the document viewer component in the UI and specify the different pages. For the behavior, they add a first rule triggered for the event expression “\$documentViewer.ON\_STROKE\_FINISHED” and, if the condition “\$documentViewer.RECOGNITION\_RESULT == LEFT”, launching action expression “\$documentViewer.PREVIOUS\_PAGE”. This rule express the action for



## 5 Approach and tool



**Figure 5.4** Example: first version.

going back to the previous page. Similarly, they add a second rule for handling the possibility for going to next page. This second rule is triggered by event expression “\$documentViewer.ON\_STROKE\_FINISHED” and, if the condition “\$documentViewer.RECOGNITION\_RESULT == RIGHT”, launching action expression “\$documentViewer.PREVIOUS\_PAGE”. The produced user interface is depicted in Figure 5.4 (the red line represents a right action gesture).

For the SU, the UI requirements are met then go to step 6.

- Step 6 – IU use the UI

IU are not satisfied with produced UI then go to step 7.

- Step 7 – IU refine the UI requirements

For the IU, even if the recognition is good, it is too slow and should be fastened. They also would like receiving a feedback indicating the current page they are watching.

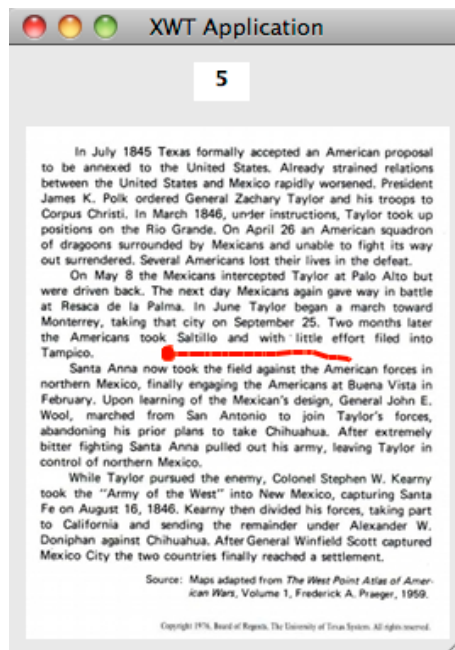
- Step 4 – SU use the system to produce the UI

## 5 Approach and tool

With the current system, SU are unable to create the user interface respecting the new UI requirements.

- Step 5 – SU refine the system requirements

The system needs to propose a faster algorithm. Additionally, a label is required with the possibility to specify a text, corresponding the page currently visualized.



**Figure 5.5** Example: second version.

- Step 3 – SF feed the system

SF will add a label component to the CUI UIDL and implement the original *Levenshtein* algorithm (less good performances but less time consuming), and a *SET\_TEXT ActionType* on the label component.

- Step 4 – SU use the system to produce the UI

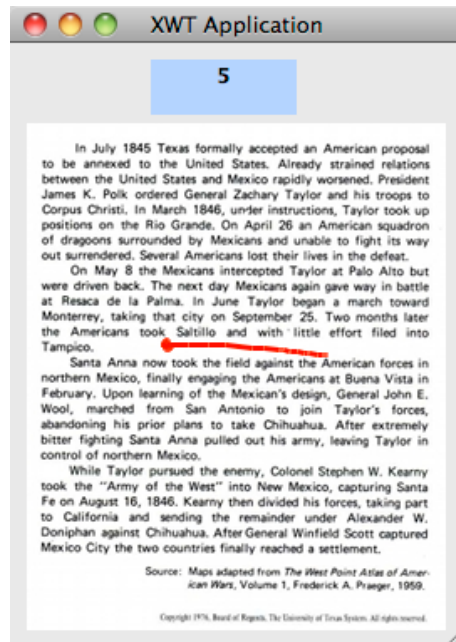
SU are now able to add a label on top of the document viewer in order to indicate the current page. The behavior possibilities have been increased in order to allow linking the page viewer with the label. The generated UI is depicted in Figure 5.5.

## 5 Approach and tool

For the SU, the UI requirements are met then go to step 6.

- Step 6 – IU use the UI

IU are not satisfied with produced UI then go to step 7.



**Figure 5.6** Example: third version.

- Step 7 – IU refine the UI requirements

IU finally want the label in blue and bigger because they think it is more beautiful.

- Step 4 – SU use the system to produce the UI

System already allows addressing this new UI requirement. The color of the label is then changed and the produced user interface is proposed to the IU, which are finally satisfied. This user interface specification may be viewed in the tool screenshot (Figure 5.3)), with the corresponding XWT. The final generated user interface is depicted in Figure 5.6.

## **5.5 Conclusion**

This chapter proposed an approach and a tool aimed at supporting the process of creation of user interfaces integrating pen-based gesture recognition. The next chapter will report two experiments conducted with the help of this tool and give some insights on the benefits of using UsiGesture method.

## Chapter 6 Validation

An ideal test case to validate the method would have been to observe a team of gesture specialists, engineers and designers creating a pen-based user interfaces first without the help method and then with the help of the method, and then compare the improvements brought by the method we propose. Unfortunately, such a team is not easy to find and it goes beyond the scope of this thesis.

Instead, two different experiments were conducted with the help of the method. They constitute a preliminary study aimed at providing first insights on the benefits of using UsiGesture method.

Additionally, an experiment led with the first platform presented in this work could assess new algorithms to be considered for gesture recognition.

### 6.1 Woke experiment

This section first presents the methodology followed during the woke experiment, and then describes and discusses the obtained results.

#### 6.1.1 Methodology

Hypotheses

This experiment is based on two hypotheses:

- H1: UsiGesture method can improve the process of development of user interfaces integrating pen-based gesture recognition;
- H2: by extension of H1, the user experience of the end-users will be improved.

## 6 Validation

### Task

The task for the developers is to create a classical application for food ordering in a woke restaurant and its equivalent integrating pen-based gesture recognition. Based on these two representations, the goal is to ask end-users to test them and give their opinion on their usability by rating them.

### Measures

The measures for this experiment are qualitative. The participants were asked to rate the 6 ergonomics criteria of Bastien and Scapin (Bastien and Scapin):

1. Coherence: the solution was thought to make it as easy as possible for the users, with recognizable actions;
2. Utility: the number of functionalities available. It is maybe a drawback of these interfaces, only a subset of them were implemented in the scope of this work;
3. Workload: the number of actions by task is not too high;
4. Adaptation: this kind of interface (gestural) can be re-used in other contexts. This criterion is only used for gesture interface;
5. Representativity: each possible action has a label associated to it. Those labels are light and don't increase the workload;
6. Guidance: the solution was thought to avoid losing users in what they have to do.

Additionally, the participants were asked to provide a general grade between 0 and 5, by considering three levels: "bad" (0 – 2), "average" (3) and "good" (4-5).

### Participants

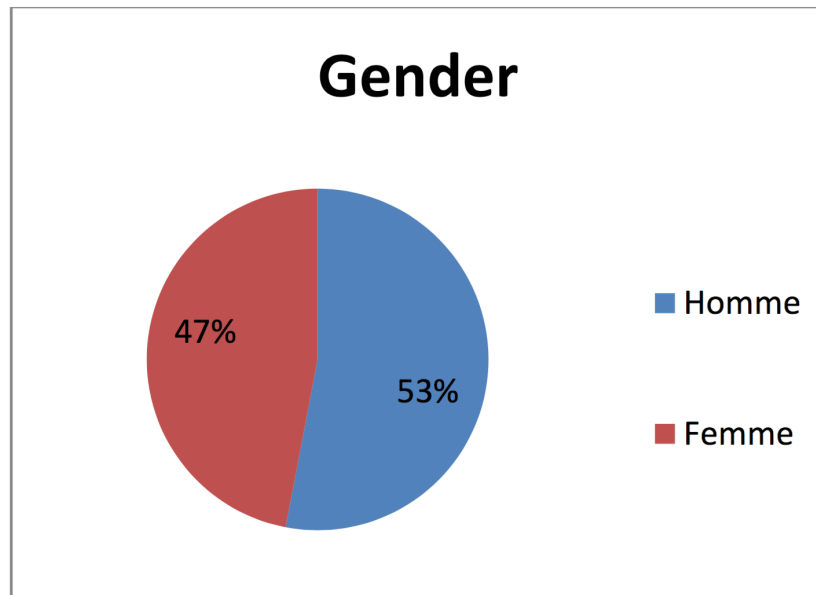
The experiment and the survey were realized with 15 participants (Figure 6.1), almost evenly distributed between men and women (Figure 6.2) and among

## 6 Validation

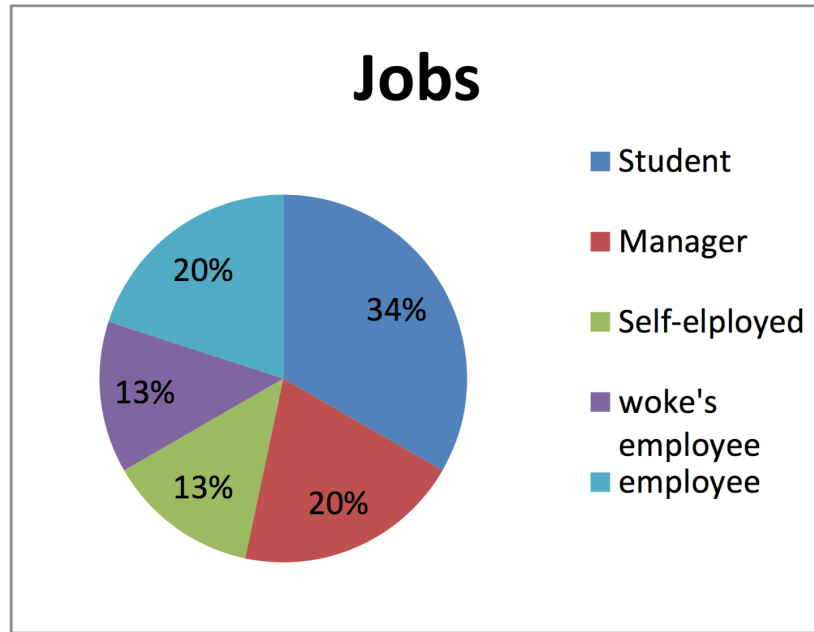
different jobs: students, managers, self-employed, employees and Woke employees (Figure 6.3). The participants have a mean age of 32,5 with a standard deviation of 11,2. They all have a university or high school qualification in fields ranging from computing science, economy or management sciences to goods transport, food, secretaryship or medicine.

| #  | Sex | Job             | Area of activity    | Age | qualification | Affinity with graphical interface (grades /10) |
|----|-----|-----------------|---------------------|-----|---------------|------------------------------------------------|
| 1  | M   | Student         | Computer science    | 23  | University    | 8                                              |
| 2  | W   | Student         | Computer science    | 24  | University    | 9                                              |
| 3  | W   | Student         | Computer science    | 21  | University    | 8                                              |
| 4  | M   | Student         | Economy             | 22  | University    | 6                                              |
| 5  | M   | Student         | Economy             | 19  | University    | 5                                              |
| 6  | W   | Manager         | Management sciences | 48  | University    | 6                                              |
| 7  | W   | Manager         | Management sciences | 41  | University    | 4                                              |
| 8  | M   | Manager         | goods transport     | 53  | high school   | 5                                              |
| 9  | W   | self-employed   | /                   | 34  | high school   | 2                                              |
| 10 | W   | self-employed   | /                   | 49  | high school   | 3                                              |
| 11 | M   | woke's employee | Food                | 24  | University    | 7                                              |
| 12 | M   | woke's employee | Food                | 25  | University    | 6                                              |
| 13 | M   | Employee        | Secretaryship       | 32  | high school   | 5                                              |
| 14 | W   | Employee        | Medicine            | 36  | high school   | 4                                              |
| 15 | M   | Employee        | Medicine            | 37  | Others        | 3                                              |

**Figure 6.1** Woke experiment: participants statistics.



**Figure 6.2** Woke experiment: participants gender shares.



**Figure 6.3** Woke experiment: participants jobs shares.

### Procedure

Before actually performing their tasks, end-users were first asked to fulfill a questionnaire on their profile and background, along with a consent form.

The goal of this work was to create a user interface simulating the system of a woke restaurant ([www.woke.be](http://www.woke.be)). It allows users to order food by following a step-by-step vegetable and meat picker. This system was created in two different ways (task and abstract user interface models are available in Appendices in Figure E.1 and E.2):

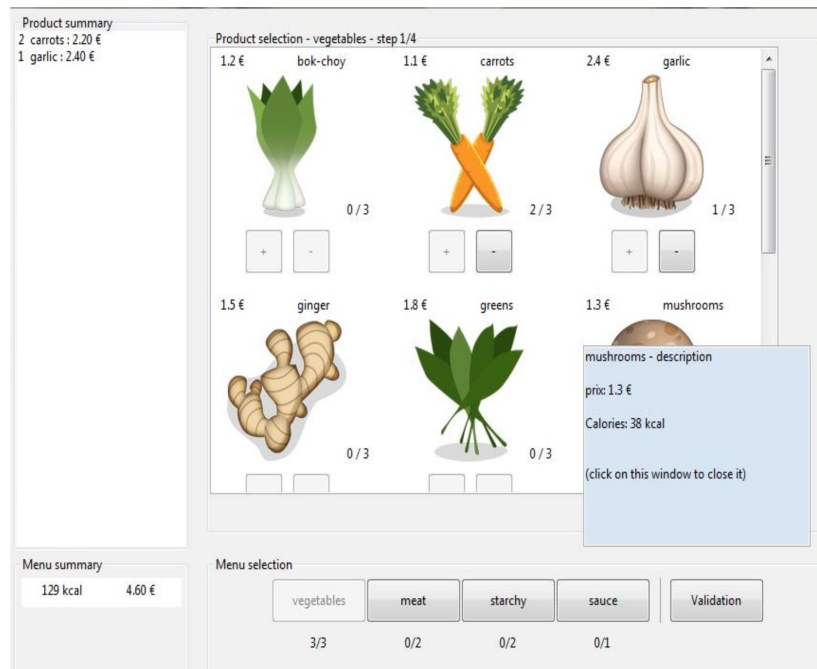
- A more usual way where the actions are mapped to buttons, like minus and plus to specify the amount of items (see Figure 6.4).
- A gestural system where actions can be executed with gestures, like directly writing the amount of wanted items on the item itself.

The interfaces constituting the solution can be divided into 6 parts:

- A list displaying the summary of all ingredients chosen by the user, with the amount of each and the whole price.



## 6 Validation



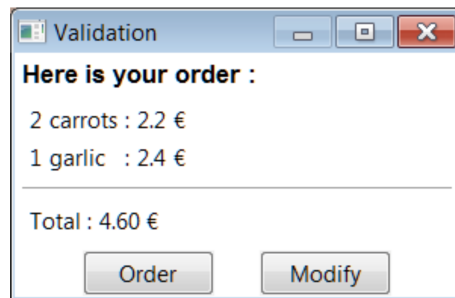
**Figure 6.4** Woke experiment: general interface.

- Another list displaying all available ingredients with the category.
- A label summarizing the number of calories, the items selected and the price of the menu.
- A button array allowing the user to browse in the categories and to confirm selection (confirmation window).
- A confirmation window.
- For the guidance, the interface shows a number indicating to the user the current step. That number is at the top of the product panel.

The main difference between classical and gestural interfaces is the way the number of items can be selected. With the classical ones, two buttons (labeled with “+” and “-”) are used to increase/decrease the quantity. With the gestural ones, the quantity can be drawn on the pictures. For both interfaces, the number of each available ingredient is limited to 5.

After a demonstration of the working of the interfaces, the end-users were asked to play with the user interface in a controlled environment. More specifically, they had to select the ingredients for their woke, validate the menu and

## 6 Validation

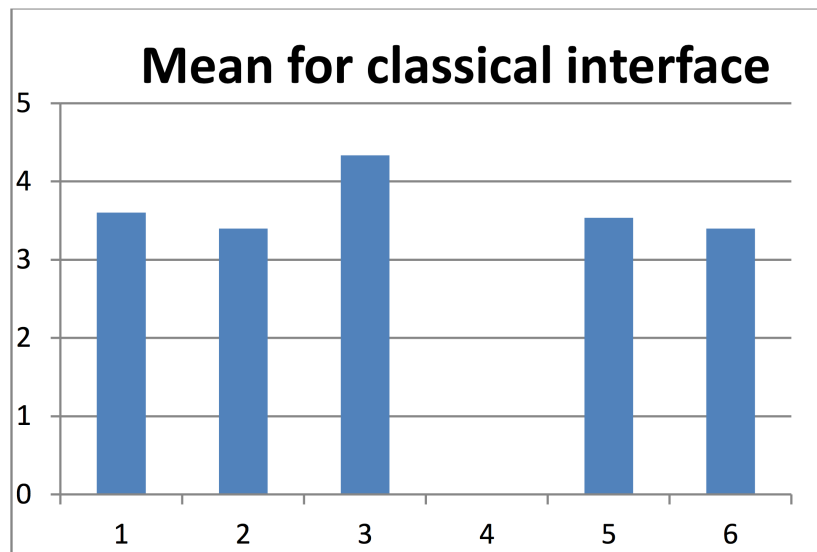


**Figure 6.5** Woke experiment: order interface.

wait for the bill. After their task, the participants were invited to fulfill a new questionnaire to rate the 6 ergonomic criteria of Bastien and Scapin regarding their feelings about the interfaces.

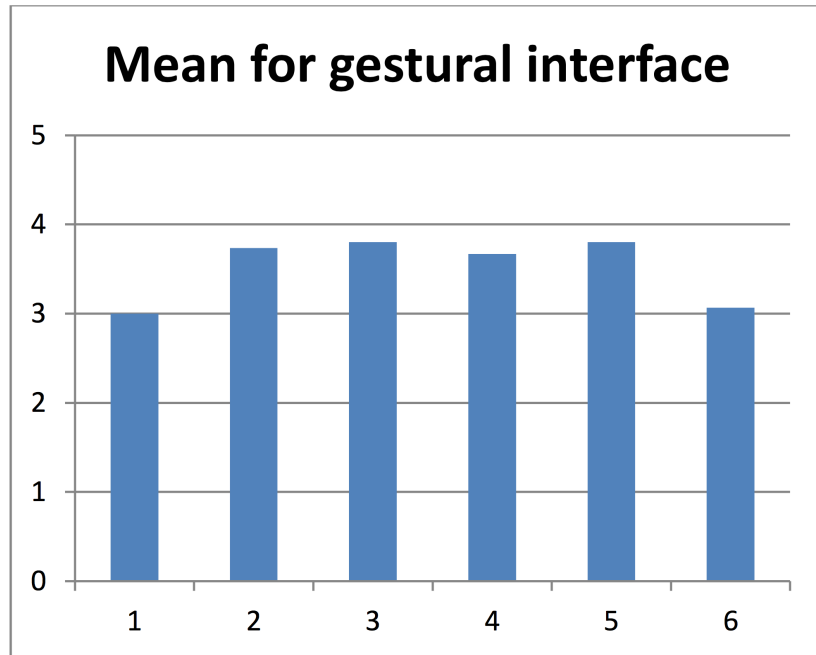
### 6.1.2 Results and discussion

The results of the rating of the 6 ergonomic criteria of Bastien and Scapin are depicted in Figure 6.6 and Figure 6.7.



**Figure 6.6** Woke experiment: classical interface results.

In addition to these survey results, the general grades obtained are the following:



**Figure 6.7** Woke experiment: gestural interface results.

- Classical interfaces: 9 good, 5 average, 1 bad;
- Gestural interfaces: 8 good, 6 average, 2 bad.

The results are quite comparable with a slight preference for the classical interfaces. The general feeling about the gestural interfaces was that they are more straightforward and intuitive, but the lack of recognition mechanism for some digits was decreasing the level of usability.

In addition to user interface considerations, some tests have been conducted to select the best recognition engine. In order to reuse the mechanisms already provided by the platform, the algorithm to be used had to be selected among Rubine, stochastic Levenshtein and One Dollar. For the woke use case, the most important gestures to be used are the digits and some action gestures. The tests have then been conducted on the 10 digits (0 to 9) and four geometrical shapes (square, triangle, circle, rectangle). It appears the One Dollar algorithm was the most accurate for this case.

## 6.2 Restaurant experiment

This section first presents the methodology followed during the restaurant experiment, and then describes and discusses the obtained results.

### 6.2.1 Methodology

#### Hypotheses

This experiment is based on two hypotheses:

- H1: UsiGesture method can improve the process of development of user interfaces integrating pen-based gesture recognition;
- H2: by extension of H1, the user experience of the end-users will be improved.

#### Task

The task is similar to the task for the woke experiment. For the developers, it is to create a classical application for food ordering in a restaurant and its equivalent integrating pen-based gesture recognition. Based on these two representations, the goal is to ask end-users to test them and give their opinion on their usability.

#### Measures

Similarly to the woke restaurant, the measures collected for the participants are qualitative. They had to rate the interfaces experiences depending on their feelings when using them.

#### Participants

The participants were 19 and were all students at university.

## 6 Validation

### Procedure

The application is divided in two main sections: the drinks and the burgers. Each page of the application shows a set of choices to the client and these pages are divided into three parts as shown in Figure 6.8 and Figure 6.9.



**Figure 6.8** Restaurant experiment: drinks.

The first part of the application is a toggle menu and is positioned at the top of the page. This menu shows the next and the previous page of the application, both the drinks and food menu, and can be triggered by the two gestures, left and right arrows.

The second part contains the user's choice of hamburgers, self-made hamburgers, beers, soft drinks, hot drinks and is the biggest part of the application. It allows the user or the waiter to write a number on each choice, this will then be registered in the third part of the application.

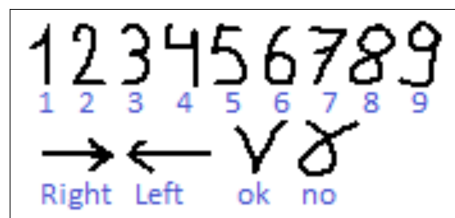
The third and last part is the client order list and is positioned on the right side of the second part. It allows the client or the waiter to clean or validate the order list with the two gestures, Ok and No. These two gestures have been added to the original set of gestures and are represented in Figure 6.10 with the other numerical and arrow gestures used.

The participants were asked to play with the interfaces in a controlled environment. The application was made so that the user can quickly change the

## 6 Validation



**Figure 6.9** Restaurant experiment: food.



**Figure 6.10** Restaurant experiment: gesture set.

## 6 Validation

number of drinks he ordered. The value of each menu or drink is overwritten when the user draws a new digit on it. This method provides more usability to the user because a mistake can quickly be changed but it limits the maximum amount of each menu and drink to nine. This is because only digits from zero to nine can be recognized.

As an experiment, two same interfaces with different recognition engines (Rubine and One Dollar) were firstly created. Several people were asked to play with these two applications so that results could be collected separately for each algorithm. With these results, Rubine and One Dollar algorithms could be analyzed on every gesture and at the same time an improved version mixing both algorithms could be created.

During the experiment, people could use the mouse, a touchpad or an electronic pen with a tablet to test the application. Each person was asked to do the experiment with every restaurant application made to evaluate the two different recognition algorithms. The experiments were conducted either on a UNIX or Windows operating system, and can be visualized in Figure 6.11 where all numbers were drawn by an anonymous tester to test the effectiveness of gesture recognition.



**Figure 6.11** Restaurant experiment: gestures handling.

### 6.2.2 Results and discussion

From the graphical user interface point of view, the nineteen users gave a lot of different feedbacks. Fourteen people found that the use of gestures for this application was really fun; three people found that it was interesting but they prefer the WIMP (Windows, Icons, Menus, Pointers) paradigm. The last two people are completely opposed to the use of gestures such as digits because they think it is neither intuitive nor accurate. Almost every user has the same issue: they are not used to draw with a mouse and are not accustomed to work with a tablet. We think that this issue is more important than it seems because it is the habit of WIMP that will curb some people to use the application. Another point of the graphical user interface is that the use of the gesture “ok” and “no” is not so intuitive. If people do not know these two gestures they cannot clean or order something. In consequence we think that the application is more suitable for a waiter or a waitress than a simple client because they will have a learning curve for the two gestures to use.

It was also noticed that 75% of the people prefer to use their fingers or a digital pen than the mouse when using our application, and that it was more intuitive for most people to draw a line going to the left in the upper menu in order to go to the right side of the menu. It may be explained by the young age of participants and their ability to use their smartphone.

Along with the qualitative results on the interfaces usability, data was collected to compare One Dollar and Rubine algorithms. The results presented in Table 6.1 show that One Dollar algorithm is better for the digits, the “ok” and the “no” gesture. The Rubine algorithm seems to be perfect for the arrows. We can see that the digits have some troubles but it depends on the people drawing these digits. For eight persons the results for the digit “2” were equal to 100%. Another interesting notice is that the result of the recognition was often the digit three when the result was wrong. These discoveries allowed to understand that by adding in the recognition environment records closer to the writing of the eleven other persons, better results can be obtained. For the “ok” and “no” gestures, the Rubine algorithm works strangely as it recognizes well the “ok” when the gesture is small and the “no” when the gesture is big.



## 6 Validation

| Gesture     | Rubine | One dollar |
|-------------|--------|------------|
| 1 (one)     | 72%    | 94%        |
| 2 (two)     | 77%    | 55,5%      |
| 3 (three)   | 100%   | 98%        |
| 4 (four)    | 0%     | % 98       |
| 5 (five)    | 0%     | 93%        |
| 6 (six)     | 11%    | 98%        |
| 7 (seven)   | 0%     | 80%        |
| 8 (eight)   | 100%   | 100%       |
| 9 (nine)    | 0%     | 98%        |
| 10 (ten)    | 72%    | 100%       |
| Right arrow | 100%   | 2%         |
| Left arrow  | 100%   | 2%         |
| Ok          | 50%    | 100%       |
| No          | 50%    | 100%       |

**Table 6.1** Restaurant experiments: results.

### 6.3 Algorithms comparison

This section first presents the methodology followed during the algorithms comparison experiment, and then describes and discusses the obtained results.

#### 6.3.1 Methodology

##### Hypothesis

The experiment is based on the hypothesis (H1) that UsiGesture method can improve the process of development of user interfaces integrating pen-based gesture recognition.

##### Task

The task is here essentially destined to the developers. Their goal is to create a recognition system to ask participants to reproduce gestures leading to gesture set collection. Thanks to this gesture set, the developers can perform gesture recognition to test new algorithms. The first algorithm is a variation of \$1 algorithm using a new distance for the point-to-point comparison: the Mahalanobis distance (Mahalanobis [1936]) instead of the Euclidean distance. The second algorithm is a variation of the Levenshtein's algorithm using a new distance for the string comparison: the Jaro-Winkler distance (Winkler [1999]) instead of the edit distance. The third algorithm is \$n algorithm (Wobbrock and

## 6 Validation

Anthony [2010]) that adds multi-stroke capabilities to \$1 algorithm. Please refer to corresponding literature for more details.

### Measures

The measures are quantitative for the algorithms recognition rates. Additionally, qualitative measures were asked to the developers to assess their experience with the platform.

### Participants

The participants were 16 and were all students at university.

### Procedure

Three roles were defined during the experiment:

- The user: the person who passes the test
- The supervisor : the person who launches the recognition process with different algorithms
- The scribe: the person who shows to the user the gesture, performs and writes down the results

Before starting the test phase, the supervisor is responsible for training the algorithms. In this context, training an algorithm means giving it some examples of gestures to permit it to recognize future gestures. The reference gestures were directly taken from UsiGesture platform.

The rules that the user must respect during the test are simple:

- A gesture can be done only once. The user cannot ask to restart the test or to redraw an already drawn gesture.
- The user must write as naturally as possible. He must use the pen on the tablet as a normal pencil on a sheet of paper.

## 6 Validation

Defining the right datasets was not an easy task. Indeed, the \$n algorithm is optimized to recognize multistrokes gestures. If the input of the \$n algorithm is a unistroke, the result will be very likely the same than the \$1 algorithm. In the meanwhile, the two others algorithms can only recognize unistroke gestures.

The first set of gestures that the users were asked to perform can be unistroke or multistroke and is composed of five geometrical shapes and five action commands. Each of those ten gestures was analysed by the three newly implemented algorithms.

The second set of gestures is composed of five lower-case letters of the alphabet (unistroke). The difference with the first set is that these five gestures were only analysed by Jaro-Winkler and Mahalanobis algorithms.

The last set of gestures is composed of five upper-case letters of the alphabet (multistroke). As with the second set, the third one was not analysed by all algorithms but only with nDollar. Every time a gesture is performed by the user, the supervisor launches the corresponding algorithms and the scribe records the result returned by the algorithm.

At the end of this process, fifteen gestures have been analysed by each algorithm.

### 6.3.2 Results and discussion

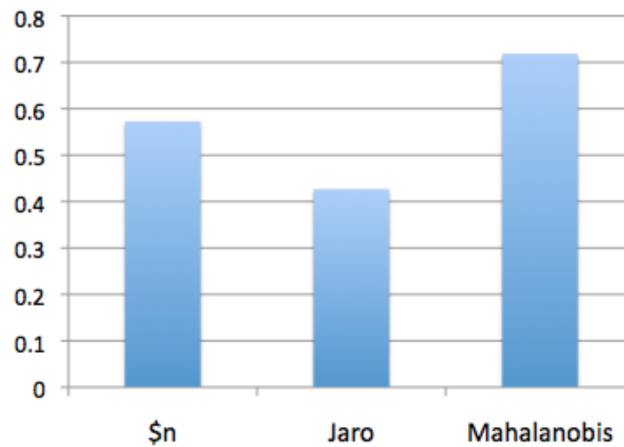
The results are presented by type of gestures.

#### Geometrical shapes

In order to evaluate the performance of the three algorithms on geometrical shapes, the percentage of correct recognition for each algorithm is computed. It corresponds to the number of correctly recognized shapes divided by the number of shapes. The average percentage of recognition on all tests for each algorithm is depicted in Figure 6.12.

The Mahalanobis algorithm has the highest rate of recognition: 72%. Also, the results reveal the poor efficiency of the Jaro technique on geometrical shapes (42,5%) which is worse than the \$n algorithm (57,5%). This is due to the fact

## 6 Validation



**Figure 6.12** Algorithms comparison: geometrical shapes.

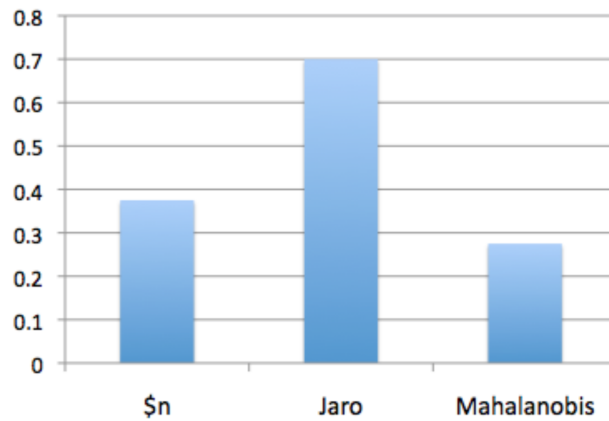
that the Jaro Wrinkler algorithm has been designed to recognise words and searches for letter inversions. In this case the string given as input is not a word but rather a set of direction values. Trying to search for inversions inside those strings is thus not very relevant as this would mean the user has inverted two directions while drawing the shape. Furthermore the Wrinkler modification which boosts the score when detecting a similar prefix has no real reason to be used in this context as it would only try to recognize the beginning of the gesture, which is often irrelevant due to the stylus making unwanted movements when first touching the tablet. Mahalanobis has a good recognition rate for geometrical forms because it uses an improved Euclidian distance algorithm which has a good efficiency for common shapes recognition.

### Arrows

The same principle is applied in this section to evaluate the efficiency of the algorithms but on the arrows. The average recognition rate for the algorithm is represented in Figure 6.13.

It reveals very different performances. Mahalanobis is significantly worse than Jaro technique unlike results computed on the geometrical forms. Also, the Jaro method has the best recognition rate (70,2%) and is significantly better than the \$n method. Jaro-Wrinkler algorithm achieves the best performances on this test. This is probably due to the fact that the algorithm takes directions and not points as input and is thus more able to recognize the arrow directions.

## 6 Validation

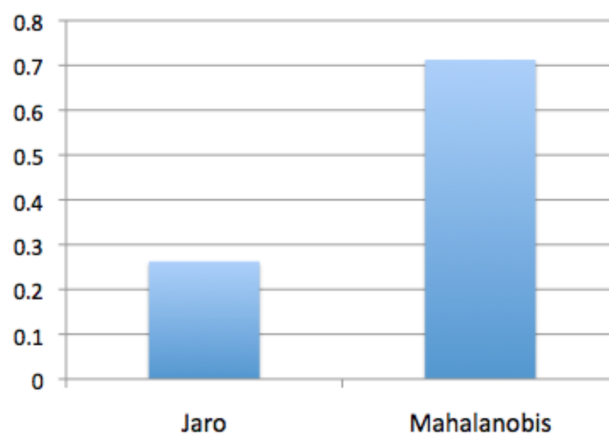


**Figure 6.13** Algorithms comparison: arrows.

Mahalanobis has a bad recognition rate for arrows because it does not take rotation and direction of the strokes into account. Indeed, some of the stored arrow templates are simple rotation of each others. Since the direction of the stroke is not taken into account, it is not possible to distinguish some of them and the recognition is ambiguous.

### Lowercase letters

Again, the same principle is applied for the lowercase letters set to evaluate the efficiency of the algorithms. The average recognition rate for the algorithm is represented in Figure 6.14.



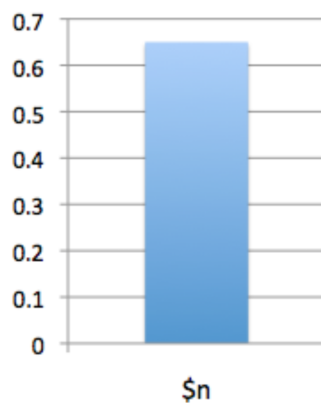
**Figure 6.14** Algorithms comparison: lowercase letters.

## 6 Validation

Since the test on lower case letters is only applied on the Jaro-Winkler method and the Mahalanobis method as explain previously, there is no possible comparison with the \$n method. The result reveals a poor recognition rate for the Jaro-Winkler method while Mahalanobis method has a much better rate of 70% as compared to Jaro-Winkler method. As seen during the tests on the Geometrical forms, Jaro-Wrinkler algorithm is not an algorithm adapted to shape recognition which explains its low score. Mahalanobis has a good recognition rate for lower case letters for the same reasons as for geometrical shapes.

### Uppercase letters

The last evaluation concerns the \$n method on the upper case letters. The average recognition rate for the algorithm is represented in Figure 6.15.



**Figure 6.15** Algorithms comparison: uppercase letters.

The recognition rate of the \$n method is not as good as expected (66%) but is still encouraging.

The goal of this experiment was to evaluate the ability of three different algorithms to recognize shapes and gestures. During the first stage they were implemented in the UsiGesture platform. In the second phase test users were asked to use those algorithms in order to recognize what they wrote. From those tests can be concluded that some algorithms are not adapted to recognize handwriting and others are not adapted to recognize directions, or at least not in the way their inputs are currently modeled. For instance the Winkler modification in the Jaro-Wrinkler algorithm only makes sense to compare words but not to compare strings containing directions values representing gestures.

On the other hand algorithms such as Mahalanobis are not really adapted to recognize movements. The experiment shows that the algorithm performances is very dependent of the type of gesture to recognize.

In addition to the algorithms performances results, developers have assessed the UsiGesture platform and appreciated the ease for adding new algorithms and create interfaces on top of them. Moreover, UsiGesture method could bring structuration during the development workflow of their experiment.

### 6.4 Discussion and Conclusion

One important criterion to take into account when designing such interfaces is the compatibility between the real world and the system. Gestures need to be chosen adequately in order to be as representative as possible of what human being would expect. This point has a big impact on coherence criterion from Bastien and Scapin. Different criteria of SQuaRE (Software Product Quality Requirements and Evaluation, SQuaRE-iso) are or can be impacted by UsiGesture method. Here are the most important ones:

- **Interoperability:** Interoperability is the ability of making systems and organizations work together (inter-operate). UsiGesture improves collaboration between different kind of stakeholder (gesture specialists, developers, designers, end-users, ...).
- **Suitability:** Suitability is the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions. UsiGesture tool is targeted to help to the integration of gestures and improves this process if used according the proposed method.
- **Resource utilization:** Resource utilization is the ability to improve available resources. UsiGesture improves human resources utilization by allowing a good separation of tasks assigned to the suitable stakeholders.
- **Changeability:** Changeability is the degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality. Adding new gestures algorithms or datasets in UsiGesture is quite straightforward. For adding gesture, the

## 6 Validation

feeders need to create the recognition mechanism itself and then plug it to the platform by creating one new class and modifying two methods. The plugin would require an effort estimated to of a few hours. Extending the datasets represents a similar effort. For the user point of view, changing an algorithm or dataset on a previously created interface is very simple it can be changed through an option in the WYSIWYG editor.

UsiGesture tool and modelling were helpful as they could enact better and faster developments of the use cases. The approach can bring a good structure for the development lifecycle of a project, but it may not be necessary in small applications. The developers for the woke use case made the decision to not use it and develop the interface independently of the end users with a result not very convincing. However, the developers of the restaurant use case tried to take end users into account by selecting the right algorithms for the right gestures and then improving user experience.

As shown in the third experiment, selecting the best algorithms to improve recognition is an example of many possibilities to improve the user experience, like selecting the appropriate gestures for the right situation, increasing the gesture set or the number of existing algorithms. The platform is already designed to easily support the improvement of algorithms and gestures set knowledge by capitalizing on the different application coded with it. Additionally to the developers knowledge, UsiGesture can still be improved by adding the ability to automatically capitalize on end users knowledge. For example, the platform could compute success rates between algorithms and types of gestures (and possibly by taking context into account) thanks to all previous recognitions done through the system. It may then support developers in their choices of algorithms and gestures by suggesting the best combinations. Another possibility would be to propose a survey for querying the user at the end of interfaces manipulations. This could be useful to aggregate quality information about Bastien and Scapin criteria and guide the developers during the creation process.



# Chapter 7 Conclusion

The goal of this thesis is to provide a structured method for incorporating pen-based gesture recognition in graphical user interfaces. The work described in this report contributes in this field by providing inputs at three levels: conceptual, methodological and implementational. The conclusion is divided in two parts: it first summarizes these contributions before discussing the limitations and the resulting potential future work.

## 7.1 Summary of contributions

Chapter 2 gave an overview of the existing related work in the field targeted by this thesis. Based on this work, it allowed to identify the shortcomings and elicit the requirements from them. This section details the different contributions elaborated to fulfill these requirements, classified in three categories: conceptual, methodological and implementational.

### 7.1.1 Conceptual

On the conceptual point of view, the following contributions have been brought:

- UsiXML Concrete User Interface meta-model
- Gesture meta-model extension of UsiXML Concrete User Interface meta-model

A set of meta-models were provided in Chapter 4. The first meta-model describes a graphical user interface at a concrete level: the different components appearing in the user interface along with the relationships between them

and the dynamic ruling their behavior. The second meta-model is devoted to the gesture recognition mechanism: the recognition algorithmic, the gesture dataset specification and the way it is linked to the concrete user interface meta-model.

### 7.1.2 Methodological

On the methodological point of view, the following contributions have been brought:

- Method to enact collaboration between end-users, developers, engineers and gesture specialists to create user interfaces integrating pen-based gesture recognition
- Comparison of recognition mechanisms depending on the applicative context

Creating user interfaces integrating pen-based gesture recognition is not an easy task and requires more skills compared to the creation of classical user interfaces. More specifically, this process involves gesture specialists able to identify and parametrize the different assets required to design good recognition mechanisms, while it still allows the regular user interface developers, engineers, designers and end-users involvement. Chapter 5 described a methodology supporting the process of creation of user interfaces integrating pen-based gesture recognition by defining the different roles involved in such a task, with the interactions between these roles during the whole development lifecycle.

Additionally, Chapter 3 provided a comparative study aimed at helping to select the right recognition mechanism (recognition algorithm and gesture dataset) for the right use case. This comparative study can provide guidance to the gesture specialist during the phase devoted to their task in the approach.

### 7.1.3 Implementational

On the implementational point of view, the following contributions have been brought:

## 7 Conclusion

- Platform to perform comparative study of recognition mechanisms assets
- Recognition mechanism assets: various gesture datasets and classical algorithms implementations
- Extensions of algorithms: Rubine, One Dollar, Levenshtein, stochastic Levenshtein
- UsiGesture tool allowing user interfaces integrating pen-based gestures recognition, supporting the elaborated models and approach
- Validation experiments

Two different tools were provided through this work. Chapter 3 presented the first one supporting the benchmarking and comparative study of gesture recognition mechanisms: recognition algorithms and gestures. The platform also supports gesture dataset acquisition and integrates various implementations of existing algorithms, as well as extensions of these algorithms created for the specific needs of this work.

Chapter 5 presented the second tool, UsiGesture, supporting the whole method elaborated in this thesis. It allows the different roles involved in the creation of user interfaces integrating pen-based gesture recognition to work together by concretizing the approach defined in the same chapter. It also supports the UsiXML Concrete User Interface meta-model with its extension for gestures provided in Chapter 4. Additionally, UsiGesture tool integrates the implementation of the various algorithms (and extensions) and gesture datasets described in this work. These assets are provided as out-of-the-box components and can be extended following the structure established in the chapter.

Finally, Chapter 6 provided two experiments conducted with the help of UsiGesture. They do not represent a strong evidence for the improvement of the quality of the produced user interfaces, but rather constitute a preliminary study aimed at providing first insights on the benefits of using the elaborated method.

### 7.2 Gestures pertinence

Before creating a user interface, it would be appropriate to initiate a reflection on the pertinence to integrate gesture recognition or not. Even if this can

## 7 Conclusion

look fancy, every application has its own context with its own constraints and integrating gesture recognition may increase but also decrease the end-user perceived quality.

Choosing to incorporate or not gesture recognition in an application is highly correlated with the ability to constitute a good gesture set for the context of the application (see section 3.3.5). But even with a good gesture set, this decision should be function of the complexity of performing action with the help of the gestures and without. In the ideal case, a study would be led to test the perception of end-users with two same applications, one including gestures and the other not. Vatavu and Wobbrock propose a very interesting framework to rate the level of agreement that can be used to understand better the end-users' preferences for interacting with different input devices, including pens (Vatavu and Wobbrock [2015]). This framework could be used to assess the differences between classical and gestural interfaces.

### 7.3 Limitations and future work

UsiGesture currently supports the creation of user interfaces integrating pen-based gesture recognition through a WYSIWYG editor and provides the corresponding XML specification of the produced interface. This specification can then be translated in a final user interface in Java. Even if this programming language is widely distributed and adopted for creating user interfaces, many other languages would be helpful in such a context and would be worth to be supported, like Objective-C for iOS or C# for Windows.

Different recognition algorithms were considered and implemented during this thesis. Research and development in the area of gesture recognizers has been subject to a continuous effort, as reflected in several other recognizers: the extension of \$1 (Wobbrock et al. [2007]) into other recognizers such as \$n (Wobbrock and Anthony [2010]) and \$P (Vatavu et al. [2012]) creating the so-called \$-family of recognizers, 1¢ (Herold and Stahovic [2012]), Penny Pincher (Taranta and Jr. [2015]), etc. This demonstrates a vivid area of research still today. These recognizers could be an opportunity to increase UsiGesture capabilities.

The first platform presented in this thesis allows to perform comparative studies for recognition mechanisms benchmarking. Gesture specialists can use

## 7 Conclusion

it to select the right algorithms and gestures for their specific use cases, and integrate them to the user interface through UsiGesture. It would be useful to bridge the gap between the two platforms by providing a way to directly reuse the results of the benchmarking in UsiGesture. More specifically, we think fuzzy logic rules could be created and integrated in UsiGesture. These rules could receive input requirements captured by UsiGesture by analysing the produced interface (and maybe asking simple questions to developers). Based on this input requirements, the rules would output propositions of right recognition algorithms with adequate parametrization. Additionally, the system would output the gestures discriminating in the best way the current chosen gestures, providing the opportunity to optimize the recognition.

Two different experiments conducted with the help of the method elaborated in this work constitute a preliminary study and provide a first validation on its benefits for the process of creation of user interfaces integrating pen-based gesture recognition. However, even if it is an expected result, it does not prove the improvement of the quality of the produced user interfaces. Conducting a broader validation study comparing the produced user interfaces would be insightful.

This thesis is scoped to pen-based gestures. Many devices as mobile phones and tablets offer the ability to capture finger-based gestures. This kind of gestures hence represents an interesting field of investigation. Theoretically, the concepts and tools provided in this work are still largely valid for the finger-based gestures, even if it would require some adaptations. For example, it would maybe require some algorithmic adaptation to take the stroke thickness into account. Experiments in such a direction would be valuable.

### 7.4 Wrap up

UsiGesture is a method aimed at contributing to the field of gesture recognition, and more specifically gesture recognition applied to pen-based user interfaces. Even if different actors bring useful assets at different levels, some assets are still missing and to our knowledge, none of them define models, approaches and tools provided in a fully integrated method. We hope this thesis will help future actors of this field to structure and improve their creations.



# Bibliography

L. Anthony and J.O. Wobbrock. \$n-protractor: A fast and accurate multistroke recognizer. *Proceedings of Graphics Interface (GI '12)*. Toronto, Ontario (May 28-30, 2012), 2012.

L. Anthony, R.-D. Vatavu, and J.O. Wobbrock. Relative accuracy measures for stroke gestures. *Proceedings of the ACM International Conference on Multimodal Interfaces (ICMI '13)*. Sydney, Australia (December 9-13, 2013), 2013a.

L. Anthony, R.-D. Vatavu, and J.O. Wobbrock. Understanding the consistency of users' pen and finger stroke gesture articulation. *Proceedings of Graphics Interface (GI '13)*. Regina, Saskatchewan (May 29-31, 2013), 2013b.

Thierry Artieres, Sanparith Marukatat, and Patrick Gallinari. Online handwritten shape recognition using segmental hidden markov models. *IEEE Transaction Pattern Analysis and Machine Intelligence*, 29(2):205–217, Feb 2007.

J.M. Christian Bastien and Dominique L. Scapin. Ergonomic criteria for the evaluation of human-computer interfaces.

W. Buxton, E. Fiume, R. Hill, A. Lee, and C. Woo. Continuous hand-gesture driven input. *Proceedings of Graphics Interface '83, 9th Conference of the Canadian Man-Computer Communications Society*, pages 191–195, 1983.

W. Buxton, R. Sniderman, W. Reeves, S. Patel, and R. Baecker. The evolution of the sssp score-editing tools. *Foundations of Computer Music, C. Roads and J. Strawn (Eds.)*, MIT Press, pages 387–392, 1985.

Cadoz and Wanderley. Gesture-music. *Trends in Gestural Control of Music*, 2000.

G. Calvary and G. Meixner. Introduction to model-based user interfaces. *W3C Working Group Note 07 January 2014*, accessible at <https://www.w3.org/TR/mbui-intro/>.

## Bibliography

G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, Vol. 15, No. 3, June 2003, 2003.

Stéphane Chatty and Patrick Lecoanet. Pen computing for air traffic control. *ACM, Computer-Human Interface, CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems, Vancouver, British Columbia, Canada*, pages 87–94, 1996.

A. Coyette, S. Faulkner, M. Kolp, Q. Limbourg, and J. Vanderdonckt. Sketchixml: towards a multi-agent design tool for sketching user interfaces based on usixml. *Proceedings of the 3rd annual conference on Task models and diagrams, Prague, Czech Republic*, 86:75–82, 2004.

Adrien Coyette, Sascha Schimke, Jean Vanderdonckt, and Claus Vielhauer. Trainable sketch recognizer for graphical user interface design. *Proceedings of 11th IFIP TC 13 International Conference on Human-Computer Interaction INTERACT'2007 (Rio de Janeiro, 10-14 September 2007)*, pages 123–135, 2007.

Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. Wiley Interscience, 1993.

Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis*. Cambridge University Press, April 1998.

D. Efron. Gesture and environment. *Morningside Heights, New York: King's Crow Press*, 1941.

T. O. Ellis, J. F. Heafner, and W. L. Sibley. The grail language and operations. *The Rand Corporation, RM-6001-ARPA*, 1969.

J.M. Cantera Fonseca, J.M. Gonzalez Calleros, G. Meixner, F. Patterno, J. Pullmann, D. Raggett, D. Schwabe, and J. Vanderdonckt. Model-based ui xg final report. *W3C Incubator Group Report, 4 May 2010*, <http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/>.

M. J. Fonseca, C. Pimentel, and J. A. Jorge. An online scribble recognizer for calligraphic interfaces. *2002 Association for the Advancement of Artificial Intelligence Spring Symposium*, 2002.

Microsoft gestures. URL [https://msdn.microsoft.com/en-us/library/ms697513\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms697513(v=vs.85).aspx).



## Bibliography

- W. L. Goh, D. P. Mital, and H. A. Babri. An artificial neural network approach to handwriting recognition. *First International Conference on Knowledge-Based Intelligent Electronic Systems*, 1:132–136, 1997.
- J. A. Graham and M. Argyle. A cross-cultural study of the communication of extra-verbal meaning by gestures. *International Journal of Psychology*, Vol. 10, No. 1, pages 57–67, 1975.
- M. Grosjean and C. Kerbrat-Orecchioni. Acte verbal et acte non verbal, ou comment le sens vient aux actes. *Actes du colloque: les relations intersemiotiques*, 1999.
- Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, January 1997. URL <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0521585198>.
- C. Hardyck and L. F. Petrinovich. Left-handedness. *Psychological Bulletin*, 84:385–404, 1977.
- H. Rex Harston and Deborah Boehm-Davis. User interface development processes and methodologies. *Behaviour and Information Technology*, Volume 12, Issue 2, 1993.
- J. Herold and T.F. Stahovic. The 1¢ recognizer: a fast, accurate, and easy-to-implement handwritten gesture recognition technique. *SBIM '12 Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, 2012.
- K. Hinckley, G. Ramos, F. Guimbretiere, P. Baudisch, and M. Smith. Stitching: pen gestures that span multiple displays. *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '04)*, ACM, pages 23–31, 2004.
- J. Hong, J. Landay, A. C. Long, and Jennifer Mankoff. Sketch recognizers from the end-user's, the designer's, and the programmer's perspective. *IAAA 2002 Spring Symposium: Sketch Understanding Workshop*, Stanford, 2002.
- J. I. Hong and J. A. Landay. Satin: A toolkit for informal ink-based applications. *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 63–72, 2000.
- Shah Rukh Humayoun, Tiziana Catarci, Massimiliano de Leoni, Andrea Marrella, Massimo Mecella, Manfred Bortenschlager, and Renate Steinmann. The

## Bibliography

workpad user interface and methodology: Developing smart and effective mobile applications for emergency operators. *Proceedings of the 5th International Conference on Universal Access in Human-Computer Interaction*, 2009.

IAM database. URL <http://www.iam.unibe.ch/~zimmerma/iamdb/iamdb.html>.

A. C. Long Jr., J. A. Landay, L. A. Rowe, and J. Michiels. Visual similarity of pen gestures. *Conference on Human Factors in Computing Systems, Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 360–367, 2000.

Allan Christian Long Jr., James A. Landay, and Lawrence A. Rowe. Implication for a gesture design tool. *ACM, Computer-Human Interface, CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems, Pittsburgh, Pennsylvania, United States*, pages 40–47, 1999.

JWinTab. URL <http://www.sonycs1.co.jp/person/rekimoto/java/jwintab/>.

V. Kellen. Business performance measurement. at the crossroads of strategy, decision-making, learning and information visualization. *DePaul University, Chicago, 2003*, accessible at <http://www.kellen.net/bpm.htm>.

A. Kendon. How gestures can become like words. *Crosscultural Perspectives in Nonverbal Communication*, 1988.

Haruka Kikuchi, Shinji Kimura, Shinzo Ohkubo, Hiroshi Inamura, and Atsushi Takeshita. User interface development from conceptualization to prototype evaluation through ucd processes.

L.A. Leiva, D. Martin-Albo, and Réjean Plamondon. Gestures à go go: Authoring synthetic human-like stroke gestures using the kinematic theory of rapid movements. *ACM Transactions on Intelligent Systems and Technology (TIST) - Special Issue on Causal Discovery and Inference, Vol. 7, issue 2, Jan. 2016*, 2016.

Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

Chunyuan Liao, François Guimbretière, Ken Hinckley, and Jim Hollan. Papiercraft: A gesture-based command system for interactive paper. *ACM Trans. Computer-Human Interaction*.

## Bibliography

- A. C. Long, J. A. Landay, and L. A. Rowe. *quill: A Gesture Design Tool for Pen-based User Interfaces*. PhD thesis, University of California at Berkeley, 2001.
- I. Scott Mackenzie and Shawn X. Zhang. The immediate usability of graffiti. *Proceedings of the conference on Graphics interface '97, Kelowna, British Columbia, Canada*, pages 129–137, 1997.
- P.C. Mahalanobis. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India*, 1936.
- Dimitri Mahieux and Patrick Rácz. Analyse de données microarray - Conception d'un protocole d'analyse à balayages multiples. Master's thesis, Université catholique de Louvain, Juin 2007.
- B. Michotte and J. Vanderdonckt. Grafixml, a multi-target user interface builder based on usixml. *ICAS 2008: Fourth International Conference on Autonomic and Autonomous Systems*, pages 15–22, 2008.
- Tom Michael Mitchell. *Machine Learning*. MCGRAW-HILL INTERNATIONAL EDITIONS, 1997.
- Mozilla. URL <http://optimoz.mozdev.org/gestures/>.
- Brad A. Myers and Mary Beth Rosson. Survey on user interface programming. *Proceeding CHI 1992 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 195–202, 1992.
- Daiki Okumura, Seiichi Uchida, and Hiroaki Sakoe. An hmm implementation for on-line handwriting recognition - based on pen-coordinate feature and pen-direction feature. *ICDAR '05: Proceedings of the Eighth International Conference on Document Analysis and Recognition*, pages 26–30, 2005.
- Jose Oncina and Marc Sebban. Learning stochastic edit distance : Application in handwritten character recognition. *Pattern Recognition*, 39:1575–1587, 2006.
- Palm. URL <http://www.palm.com/support/>.
- Brandon Paulson, Aaron Wolin, Joshua Johnston, and Tracy Hammond. Sousa: Sketch-based online user study applet. *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2008.

## Bibliography

- V. I. Pavlovic, R. Sherma, and T. S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, No. 7, pages 677–695, 1997.
- William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1992.
- F.K.H. Quek. Eyes in the interface. *Image and Vision Computing*, Volume 13, Issue 6, pages 511–525, 1995.
- B. Rime and L. Shiaratura. Gesture and speech. *Fundamentals of Nonverbal Behavior*, Press Syndicate of the University of Cambridge, pages 239–281, 1991.
- Eric Sven Ristad and Peter N. Yianilos. Learning string edit distance. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 20(5):522–532, May 1998.
- Dean Rubine. Specifying gestures by example. *SIGGRAPH Computer Graphics*, pages 329–337, Jul 1991.
- Rubine3D.
- Paul Schmieder, Beryl Plimmer, and Rachel Blagojevic. Automatic evaluation of sketch recognizers. *Proceeding SBIM 2009 Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 85–92, 2009.
- SIGeR. URL [http://www.igesture.org/algo\\_siger.html](http://www.igesture.org/algo_siger.html).
- B. Signer, U. Kurmann, and M. C. Norrie. igesture: A general gesture recognition framework. *Document Analysis and Recognition, 2007. ICDAR 2007*, 2007.
- R. William Soukoreff and I. Scott MacKenzie. Measuring errors in text entry tasks : An application of the levenshtein string edit distance. *Computer-Human Interaction : Extended Abstracts of CHI 2001*, pages 319–320, 2001.
- SQuaRE-iso. URL [http://www.iso.org/iso/home/store/catalogue\\_tc/catalogue\\_detail.htm?csnumber=64764](http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=64764).
- I. E. Sutherland. Sketchpad : a man-machine graphical communication system. Ph.d. thesis, Massachusetts Institute of Technology, Lincoln Lab, Cambridge, 1963.

## Bibliography

SWT. URL <https://www.eclipse.org/swt>.

W. van Melle T. P. Moran, P. Chiu. Pen-based interaction techniques for organizing material on an electronic whiteboard. *Proceedings of the 10th Annual ACM Symposium on User interface Software and Technology (UIST '97)*, ACM, pages 45–54, 1997.

E.M. Taranta and J.J. Laviola Jr. Penny pincher: a blazing fast, highly accurate \$-family recognizer. *GI '15 Proceedings of the 41st Graphics Interface Conference*, 2015.

UCI UJIPen. URL <http://archive.ics.uci.edu/ml/datasets/UJI+Pen+Characters>.

Unipen. URL <http://hwr.nici.kun.nl/unipen/>.

UsiXML. URL <http://www.usixml.org/>.

UsiXML-D1.3. URL <https://itea3.org/project/workpackage/document/download/1583/08026-UsiXML-WP-1-D13v3UsiXMLDefinition.pdf>.

E. van den Hoven and A. Mazalek. Grasping gestures: Gesturing with physical artifacts. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 25:255–271, 2011. URL [http://synlab.gatech.edu/data/papers/hoven\\_aiedam2011\\_grasping-gestures.pdf](http://synlab.gatech.edu/data/papers/hoven_aiedam2011_grasping-gestures.pdf).

R.-D. Vatavu and J.O. Wobbrock. Formalizing agreement analysis for elicitation studies: New measures, significance test, and toolkit. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '15)*. Seoul, Korea (April 18-23, 2015), 2015.

R.-D. Vatavu, D. Vogel, G. Casiez, and L. Grisoni. Estimating the perceived difficulty of pen gestures. *Proceeding INTERACT'11 Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Volume Part II*.

Radu-Daniel Vatavu, Jacob O. Wobbrock, and Lisa Anthony. Gestures as point clouds: A \$p recognizer for user interface prototypes. *Proceedings of the ACM International Conference on Multimodal Interfaces (Santa Monica, California, 22-26 October 2012)*, pages 273–280, 2012.

Wikipedia. URL [http://en.wikipedia.org/wiki/Gesture\\_recognition](http://en.wikipedia.org/wiki/Gesture_recognition).

## Bibliography

WindowBuilder. URL <https://eclipse.org/windowbuilder>.

W.E. Winkler. The state of record linkage and current research problems. *Statistics of Income Division, Internal Revenue Service Publication R99/04*, 1999.

J. O. Wobbrock, M. R. Morris, and A. D. Wilson. User-defined gestures for surface computing. *Conference on Human Factors in Computing Systems, Proceedings of the 27th international conference on Human factors in computing systems*, pages 1083–1092, 2009.

Jacob O. Wobbrock and Lisa Anthony. A lightweight multistroke recognizer for user interface prototypes. *Proceedings of Graphics Interface 2010 (Ottawa, Canada, 31 June 2010)*, pages 242–252, 2010.

Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without libraries, toolkits or training : A \$1 recognizer for user interface prototypes. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '07). Newport, Rhode Island (October 7-10, 2007). New York: ACM Press.*, 2007.

J.O. Wobbrock, H.H. Aung, B. Rothrock, and B.A. Myers. Maximizing the guessability of symbolic input. *Proceeding CHI EA '05 CHI Extended Abstracts on Human Factors in Computing Systems*.

C. G. Wolf, J. R. Rhyne, and H. A. Ellozy. The paper-like interface. *Proceedings of the Third international Conference on Human-Computer interaction on Designing and Using Human-Computer interfaces and Knowledge Based Systems (2nd Ed.)*, G. Salvendy and M. J. Smith (Eds.), Elsevier Science, pages 494–501, 1989.

XWT. URL [https://wiki.eclipse.org/XWT\\_Documentation](https://wiki.eclipse.org/XWT_Documentation).

Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:1091–1095, 2007.

PART I

Appendices

# Appendix A Publications

## Conference papers

Beuvs, F. and Vanderdonckt, J. UsiGesture: an Environment for Integrating Pen-Based Interaction in User Interfaces. Proc. of RCIS'12 (2012), 339-350.

Beuvs, F. and Vanderdonckt, J. Designing Graphical User Interfaces Integrating Gestures. Proc. of SigDoc'12 (2012), 313-322.

Sangiorgi, U.B., Beuvs, F., and Vanderdonckt, J. User Interface Design by Collaborative Sketching. Proceedings of the Designing Interactive Systems Conference - DIS'12, ACM Press (2012), 378.

Sangiorgi, U.B., Motti, V.G., Beuvs, F., and Vanderdonckt, J. Assessing lag perception in electronic sketching. Proceedings of the 7th Nordic Conference on Human-Computer Interaction - NordiCHI'12, (2012).

## Journal papers

Beuvs F. and Vanderdonckt J. UsiGesture: Test and Evaluation of an Environment for Integrating Gestures in User Interfaces - Romanian Journal of Human-Computer Interaction - RRIOC'14 (2014), 161-194.

## Additional links

UsiXML reference: <http://www.usixml.org/en/usigesture.html?IDC=239&IDD=1333>



## Appendix B Gestures set

### B.1 Letters

|                                   |                                   |                                   |                                   |                                   |
|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|
| <div>a   a</div> <div>a   a</div> | <div>b   b</div> <div>b   b</div> | <div>c   c</div> <div>c   c</div> | <div>d   d</div> <div>d   d</div> | <div>e   e</div> <div>e   e</div> |
| <div>f   f</div> <div>f   f</div> | <div>g   g</div> <div>g   g</div> | <div>h   h</div> <div>h   h</div> | <div>i   i</div> <div>i   i</div> | <div>j   j</div> <div>j   j</div> |
| <div>k   k</div> <div>k   k</div> | <div>l   l</div> <div>l   l</div> | <div>m   m</div> <div>m   m</div> | <div>n   n</div> <div>n   n</div> | <div>o   o</div> <div>o   o</div> |
| <div>p   p</div> <div>p   p</div> | <div>q   q</div> <div>q   q</div> | <div>r   r</div> <div>r   r</div> | <div>s   s</div> <div>s   s</div> | <div>t   t</div> <div>t   t</div> |
| <div>u   u</div> <div>u   u</div> | <div>v   v</div> <div>v   v</div> | <div>w   w</div> <div>w   w</div> | <div>x   x</div> <div>x   x</div> | <div>y   y</div> <div>y   y</div> |
|                                   |                                   | <div>z   z</div> <div>z   z</div> |                                   |                                   |

## B Gestures set

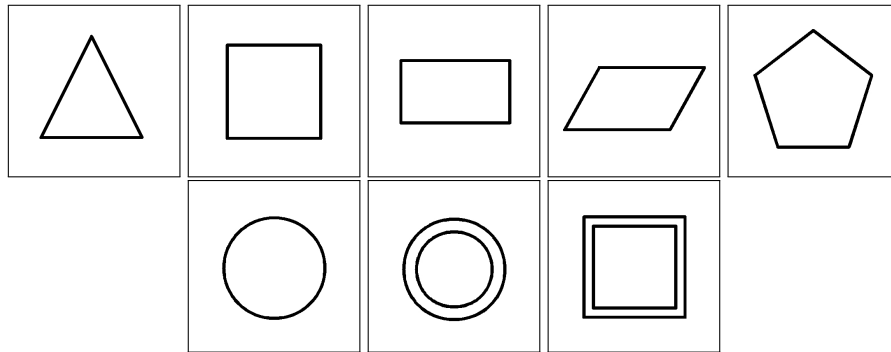
### B.2 Digits

|                                                               |                                                               |                                                               |                                                               |                                                               |
|---------------------------------------------------------------|---------------------------------------------------------------|---------------------------------------------------------------|---------------------------------------------------------------|---------------------------------------------------------------|
| $\begin{array}{c c} 0 & 0 \\ \hline \text{O} & 0 \end{array}$ | $\begin{array}{c c} 1 & 1 \\ \hline \text{1} & 1 \end{array}$ | $\begin{array}{c c} 2 & 2 \\ \hline \text{2} & 2 \end{array}$ | $\begin{array}{c c} 3 & 3 \\ \hline \text{3} & 3 \end{array}$ | $\begin{array}{c c} 4 & 4 \\ \hline \text{4} & 4 \end{array}$ |
| $\begin{array}{c c} 5 & 5 \\ \hline \text{5} & 5 \end{array}$ | $\begin{array}{c c} 6 & 6 \\ \hline \text{6} & 6 \end{array}$ | $\begin{array}{c c} 7 & 7 \\ \hline \text{7} & 7 \end{array}$ | $\begin{array}{c c} 8 & 8 \\ \hline \text{8} & 8 \end{array}$ | $\begin{array}{c c} 9 & 9 \\ \hline \text{9} & 9 \end{array}$ |

### B.3 Action gestures

|                                                                                     |                                                                                     |                                                                                     |                                                                                      |                                                                                       |
|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
|  |  |  |  |  |
|  |  |  |   |  |
|  |  |  |  |  |
|  |  |                                                                                     |                                                                                      |                                                                                       |

## B.4 Geometrical shapes



## B.5 Signatures

Votre  
signature

## Appendix C Set of volunteers

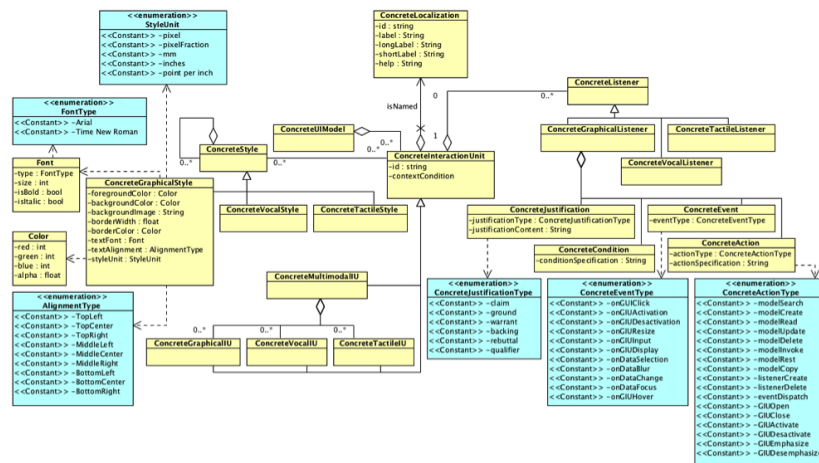
| Age | Sex | Right/left handed | Work                                       | Certificates                                                 | Computer use | PDA, pocket use | PC, tablet use | Graphic tablet use | Duration (minutes) |
|-----|-----|-------------------|--------------------------------------------|--------------------------------------------------------------|--------------|-----------------|----------------|--------------------|--------------------|
| 28  | W   | R                 | Professor and research assistant at UCL    | Civil Engineer in electronics                                | 1            | 1               | 1              | 1                  | 13                 |
| 24  | W   | R                 | Assistant                                  | Sociology                                                    | 1            | 0               | 0              | 0                  | 38                 |
| 20  | W   | R                 | Student                                    | Primary school teacher                                       | 1            | 0               | 0              | 0                  | 33                 |
| 56  | W   | R                 | IT business analyst                        | Bac in mathematics                                           | 1            | 0               | 0              | 0                  | 29                 |
| 27  | M   | R                 | Student                                    | Master in computer sciences                                  | 1            | 0               | 0              | 0                  | 28                 |
| 24  | W   | R                 | Assistant                                  | Master in management                                         | 1            | 0               | 0              | 0                  | 26                 |
| 23  | W   | R                 | Research assistant                         | Master in management                                         | 1            | 0               | 0              | 0                  | 45                 |
| 30  | W   | R                 | Assistant                                  | Master in psychology with complementary studies in marketing | 1            | 0               | 0              | 0                  | 23                 |
| 21  | M   | L                 | Student                                    | Master in computer sciences                                  | 1            | 0               | 0              | 0                  | 32                 |
| 24  | M   | L                 | Student                                    | Civil engineer in computer sciences                          | 1            | 1               | 0              | 0                  | 31                 |
| 55  | M   | R                 | Computer sciences teacher                  | Sociology                                                    | 1            | 0               | 0              | 0                  | 28                 |
| 27  | M   | R                 | Site manager                               | Civil engineer                                               | 1            | 1               | 0              | 0                  | 20                 |
| 53  | W   | R                 | Receiver of children (ONE)                 | Executive secretary                                          | 1            | 0               | 0              | 0                  | 27                 |
| 23  | M   | R                 | Consultant in computer sciences            | Civil engineer in computer sciences                          | 1            | 0               | 0              | 0                  | 19                 |
| 23  | W   | R                 | Interior designer                          | Interior designer                                            | 1            | 0               | 0              | 0                  | 24                 |
| 24  | M   | R                 | Student in architecture                    | School leaving certificate                                   | 1            | 0               | 0              | 0                  | 33                 |
| 24  | M   | R                 | Student                                    | Teacher                                                      | 1            | 0               | 0              | 0                  | 41                 |
| 23  | W   | R                 | Student                                    | Master in Roman philology                                    | 1            | 0               | 0              | 0                  | 39                 |
| 20  | W   | R                 | Student in languages and modern literature | School leaving certificate                                   | 1            | 0               | 0              | 0                  | 35                 |
| 23  | M   | R                 | Interim                                    | School leaving certificate                                   | 0            | 0               | 0              | 0                  | 29                 |
| 24  | M   | L                 | Computer scientist                         | Civil engineer in computer sciences                          | 1            | 0               | 0              | 0                  | 48                 |
| 23  | M   | R                 | Laboratory technician                      | Bac in biology                                               | 1            | 0               | 0              | 0                  | 35                 |
| 24  | M   | R                 | Project Engineer                           | Civil engineer in computer sciences                          | 1            | 1               | 0              | 0                  | 23                 |
| 19  | W   | R                 | Student civil engineer                     | School leaving certificate                                   | 1            | 1               | 0              | 0                  | 43                 |
| 23  | W   | L                 | Student                                    | Master in Roman philology                                    | 1            | 0               | 0              | 0                  | 26                 |
| 23  | M   | R                 | Assistant / PdD student                    | Civil engineer in computer sciences                          | 1            | 0               | 0              | 0                  | 16                 |
| 24  | W   | R                 | Student in economic sciences               | Bac en economic sciences                                     | 1            | 1               | 0              | 0                  | 25                 |
| 30  | M   | R                 | IT analyst                                 | Civil engineer in electro-mechanic                           | 1            | 1               | 0              | 0                  | 26                 |
| 27  | M   | R                 | Student                                    | Civil engineer in computer sciences                          | 1            | 1               | 1              | 1                  | 25                 |
| 53  | M   | R                 | IT analyst                                 | Bac in architecture and bac in computer sciences             | 1            | 1               | 0              | 0                  | 32                 |

**Table C.1** Table summarizing the characteristics of the volunteers.

## Appendix D Meta-models details

## D.1 Concrete User Interface

### D.1.1 Overview



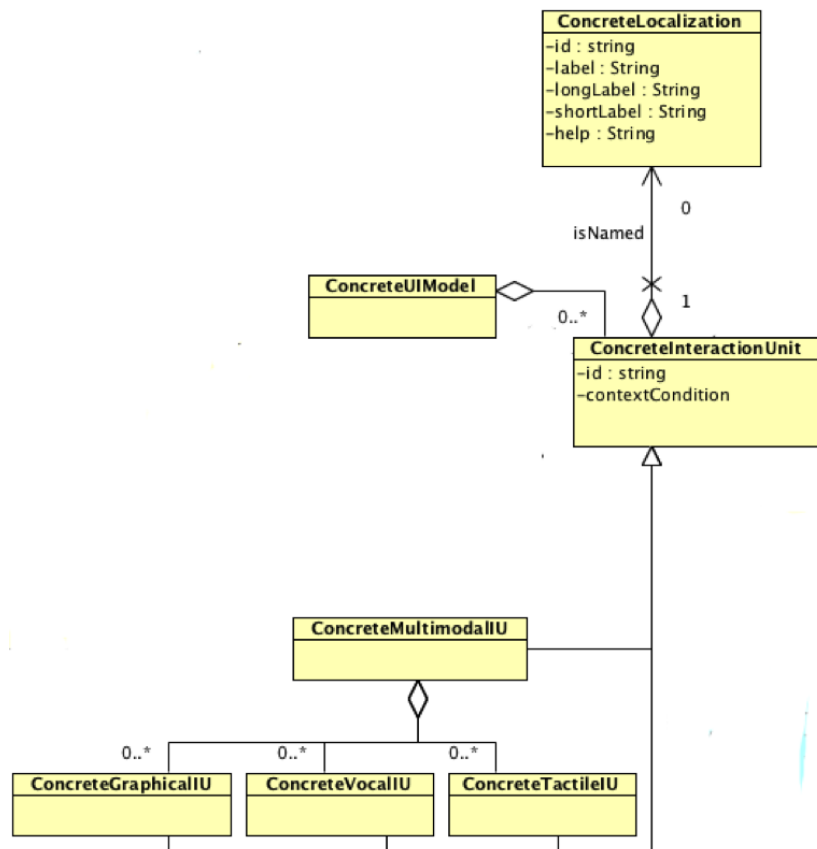
**Figure D.1** CUI Overview.

Figure D.1 displays the overview of the CUI meta-model. The chosen modelling is to compose the *ConcreteUIModel* by a set of *ConcreteInteractionUnit*: every concrete object is then such an interaction unit. These interaction units may own *ConcreteStyles* allowing to describe a style for each concrete interaction unit. Additionally, each interaction unit may have one or more *ConcreteListeners*. These listeners allow to define the dynamic of the model, the way the interaction units will react to the different events.

### D.1.2 Main entities

Figure D.2 shows the main entities of the CUI meta-model.

## D Meta-models details



**Figure D.2** CUI main entities.

## D Meta-models details

### ConcreteUIModel

Last level of abstraction (before the code). Describes user interfaces independently of the toolkit or the execution environment (Java-Swing, Web, VoiceXML ...). The modality is known.

### ConcreteInteractionUnit

Main entity of the model, every concrete object is a *ConcreteInteractionUnit*. For the moment, it is refined in three simple modalities (graphical, vocal, tactile) and compound modality (multimodal) and only the graphical modality is refined (see *ConcreteGraphicalIU* description).

#### Attributes

- id (Integer): Identification number of the *ConcreteInteractionUnit*.
- contextCondition : Link to the Context model.

### ConcreteLocalization

Allows specifying locales in different languages for the concrete interaction unit.

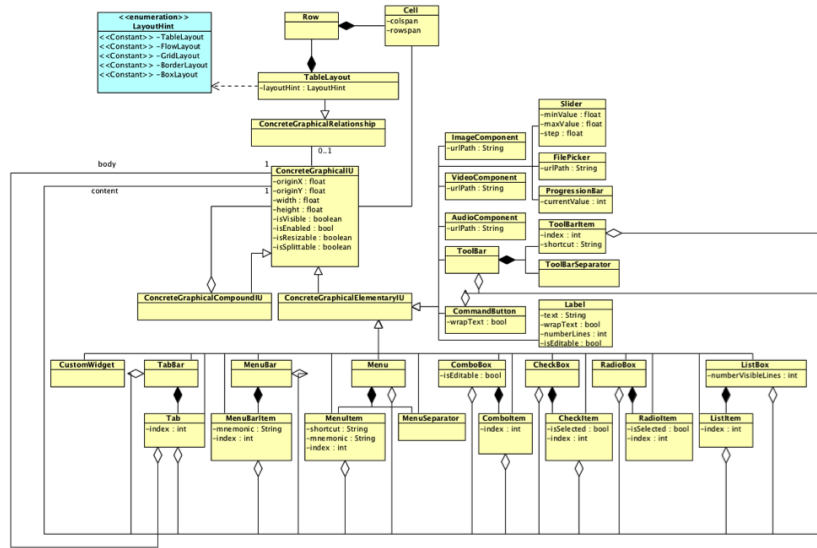
#### Attributes

- id (Integer): Identification number of the *ConcreteLocalization*.
- label (String): Description label of the *ConcreteInteractionUnit*.

### D.1.3 ConcreteGraphicalIU

Figure D.3 shows the graphical modalities. Other modalities are beyond the scope of this thesis and are not detailed.

## D Meta-models details



**Figure D.3** CUI ConcreteGraphicalIU.

### ConcreteGraphicalIU

Main graphical entity of the model, every graphical entity is a *ConcreteGraphicalIU*. It can be of two types: *ConcreteGraphicalCompoundIU* or *ConcreteGraphicalElementaryIU*. The elementary entities represent the atomic graphical widgets while the compound entities behave as containers gathering other compound entities and elementary entities. The way the entities are organized is described in the *ConcreteGraphicalRelationship*.

### Attributes

- originX (Float): x coordinate of the origin point of the *ConcreteGraphicalIU*.
- originY (Float): y coordinate of the origin point of the *ConcreteGraphicalIU*.
- width (Float): width of the *ConcreteGraphicalIU*.
- height (Float): height of the *ConcreteGraphicalIU*.
- isVisible (Boolean): specifies if the *ConcreteGraphicalIU* is visible.



## D Meta-models details

- isEnabled (Boolean): specifies if the *ConcreteGraphicalIU* is enabled.
- isResizable (Boolean): specifies if the *ConcreteGraphicalIU* is resizable.
- isSplittable (Boolean): specifies if the *ConcreteGraphicalIU* is splittable.

### ConcreteGraphicalCompoundIU

Entity allowing to gather combinations of other *ConcreteGraphicalCompoundIU* and *ConcreteGraphicalElementaryIU*.

### ConcreteGraphicalElementaryIU

Atomic graphical entity.

### TabBar

Bar gathering tab entries to click in order to show the tab.

### Tab

Entity linking the tab bar entry and the content of the Tab (that is a *ConcreteGraphicalCompoundIU*).

### Attributes

- text (String): text to display as entry for the tab in the tab bar.
- Index (Integer): index of the tab in the tab bar.

### MenuBar

Bar gathering different menu bar items.

## D Meta-models details

### MenuBarItem

Item linking an item of the menu bar with a menu.

#### Attributes

- mnemonic (String): combination of touches to press in order to uncollapse the associated menu.
- index (Integer): index of the menu bar item in the menu bar.

### Menu

Organizer of separators and items allowing launching specific menu actions.

### MenuItem

Item allowing launching specific menu action.

#### Attributes

- shortcut (String): combination of touches to press in order to launch the menu item action.
- mnemonic (String): combination of touches to press in order to launch the menu item action after having uncollapsed the parent menu.
- index (Integer): index of the menu item in the menu.

### MenuSeparator

Separator to be used in menus.

## D Meta-models details

### ComboBox

Combination of a text label and a list box: only one entry is visible and a button on the right allows showing the entire list for selection.

#### **Attributes**

- `isEditable` (Boolean): specifies if the user can type its own entry.

### ComboItem

Item of the combo box list.

#### **Attributes**

- `index` (Integer): index of the combo item in the combo box.

### CheckBox

Entity allowing the user to make multiple selections from a number of options.

### CheckItem

Item representing an option of the check box.

#### **Attributes**

- `isSelected` (String): specifies if the current item is selected.
- `index` (Integer): index of the check item in the check box.

### RadioBox

Entity allowing the user to make single selection from a number of options.

## D Meta-models details

### RadioItem

Item representing an option in the radio box.

#### Attributes

- **isSelected** (String): specifies if the current item is selected. Setting this attribute to 'true' causes setting to false the isSelected attribute of all the other radio items of the radio box.
- **index** (Integer): index of the radio item in the radio box.

### ListBox

Entity allowing to select one or more items from a list.

### ListItem

Item of the list box.

#### Attributes

- **index** (Integer): index of the list item in the list.

### ToolBar

Bar allowing organizing buttons and separators for specific commands that are not belonging to the menu.

### ToolBarItem

Item of the toolbar.

## D Meta-models details

### Attributes

- index (Integer): index of the toolbar item in the toolbar.
- shortcut (String): combination of touches to press in order to launch the toolbar button action.

### ToolBarSeparator

Separator to be used in toolbars.

### CommandButton

Free button allowing to launch a command.

### Attributes

- wrapText (Boolean): specifies if the button has to automatically resize in order to wrap the text.

### Label

Simple component allowing to insert or display text.

### Attributes

- text (String): text of the label.
- wrapText (Boolean): specifies if the label has to automatically resize in order to wrap the text.
- numberLines (Integer): maximum number of lines of the label (0 for infinite)
- isEditable (Boolean): specifies if the text can be changed.

## D Meta-models details

### ImageComponent

Component allowing displaying an image.

#### Attributes

- `urlPath (String)`: path to the image.

### VideoComponent

Component allowing playing a video.

#### Attributes

- `urlPath (String)`: path to the video.

### AudioComponent

Component allowing playing an audio file.

#### Attributes

- `urlPath (String)`: path to the audio file.

### Slider

Bar with a cursor that the user can move in order to specify a value.

#### Attributes

- `minValue (Float)`: beginning value of the bar.
- `maxValue (Float)`: ending value of the bar.
- `step (Float)`: ): interval between two successive values of the bar, by starting by `minValue` and ending by `maxValue`).

### FilePicker

Component allowing uploading a file.

#### Attributes

- `urlPath` (String): path to the file.

### ProgressionBar

Bar that begins to 0 and finishes to 100 in order to represent a progression in percentages.

#### Attributes

- `currentValue` (Integer): value between 0 and 100 representing a percentage.

### ConcreteGraphicalRelationship

Entity linked to a `ConcreteGraphicalIU` in order to describe the way it is organized. If no `ConcreteGraphicalRelationship` is specified, then the organization is the following:

- For a *ConcreteGraphicalCompoundIU*: the components contained in the *ConcreteGraphicalCompoundIU* are organized by placing their top-left corner to their specified origin, by taking the origin and the size/height of the considered *ConcreteGraphicalCompoundIU* as space reference.
- For a *ConcreteGraphicalElementaryIU*: the items of the bars (TabBar, MenuBar, ToolBar) are organized from left to right, the items of boxes (ComboBox, CheckBox, RadioBox, ListBox) and Menu for top to bottom, by following the indexes.

## D Meta-models details

### TableLayout

Defines the layout of the *ConcreteGraphicalIU*.

#### Attributes

- layoutHint (LayoutHint): Gives a hint on how the layout is structured.

### Row

If TableLayout hint is chosen, represents a row in the table.

### Cell

If TableLayout hint is chosen, represents a cell in the row of the table. This entity encloses a *ConcreteGraphicalIU*.

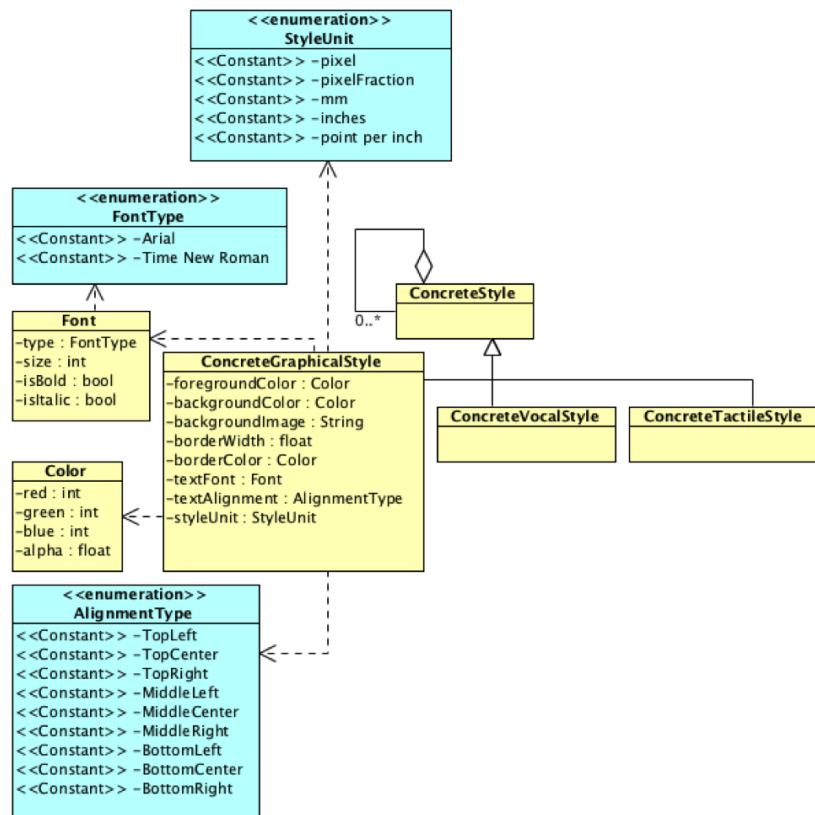
### LayoutHint

#### Attributes

- TableLayout: Allows organizing in rows and cells, like in HTML.
- FlowLayout: Allows organizing the content from left to right by specifying a justification.
- GridLayout: Allows organizing the content as a table with cells of same sizes. By following their indexes, the components are added in the cells by left to right, by beginning at the top line until the bottom line.
- BorderLayout: Allows organizing the content in 5 zones: North, East, South, West, and center.
- BoxLayout: Allows to organize the content in a box of one line, where the direction of the line can be specified.



## D Meta-models details



**Figure D.4** CUI ConcreteStyle.

### D.1.4 ConcreteStyle

Figure D.4 shows the part related to the styling of the interface.

#### ConcreteStyle

Allows to specify a style for each *ConcreteInteractionUnit*. For the moment, it is refined in three simple modalities (graphical, vocal, tactile) and only the graphical modality is refined (see *ConcreteGraphicalStyle* description).

#### ConcreteGraphicalStyle

Entity allowing specifying the style of a *ConcreteGraphicalIU*.

#### Attributes

- **foregroundColor** (Color): color of the foreground.
- **backgroundColor** (Color): color of the background.
- **backgroudImage** (String): url to the image for the background.
- **borderWidth** (Float): width of the border (0 for no border).
- **borderColor** (Color): color of the border.
- **textFont** (Font): font of the text, if text is present.
- **textAlignment** (AlignmentType): alignment of the text, if text is present.
- **styleUnit** (StyleUnit): style of the units used for measuring sizes.

## D Meta-models details

### StyleUnit

#### Attributes

- Pixel
- PixelFraction
- Mm
- Inches
- Point per inch

### Font

Entity describing font text.

#### Attributes

- type (FontType): type of font, described in FontType description.
- size (Integer): size of the font.
- isBold (Boolean): specifies if the font is bold.
- isItalic (Boolean): specifies if the font is italic.

### FontType

#### Attributes

- Arial
- Times New Roman

### Color

Entity representing a RGB color.

## D Meta-models details

### Attributes

- red (Integer): red component of the color (between 0 and 255)
- green (Integer): green component of the color (between 0 and 255)
- blue (Integer): blue component of the color (between 0 and 255)
- alpha (Float): transparency of the color (0 = translucent, 1 = opaque)

### AlignmentType

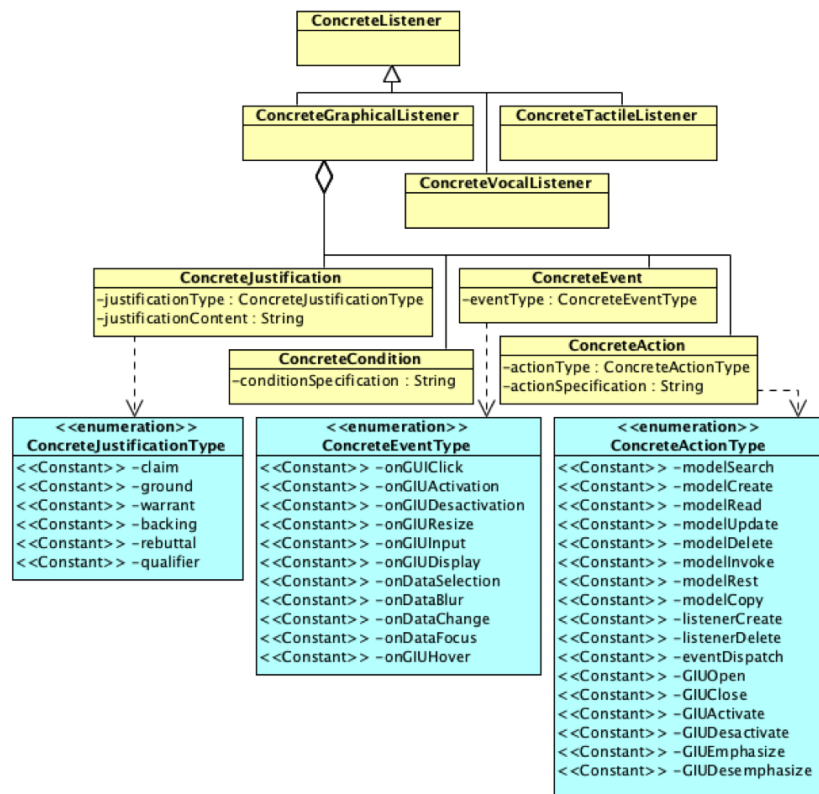
#### Attributes

- topLeft
- topCenter
- topRight
- middleLeft
- middleCenter
- middleRight
- bottomLeft
- bottomCenter
- bottomRight

### D.1.5 ConcreteListener

Figure D.5 shows the modelling of the listener part implemented through a JECA (Justification - Event - Condition - Action) structure.

## D Meta-models details



**Figure D.5** CUI ConcreteListener.

## D Meta-models details

### ConcreteListener

Allows specifying the behavior of the AbstractInteractionUnits. For the moment, it is refined in three simple modalities (graphical, vocal, tactile) and only the graphical modality is refined (see ConcreteGraphicalListener description).

### ConcreteGraphicalListener

Entity used to describe the behavior of a ConcreteGraphicalIU by using Event-Condition-Action (ECA) rules.

### ConcreteJustification

The justification is a kind of motivation for the ECA rules, it is not used for describing the interface itself but more for documentation purposes.

#### Attributes

- justificationType (ConcreteJustificationType): type of justification, the different possibilities are described in the ConcreteJustificationType description.
- justificationContent (String): text representing the justification itself.

### ConcreteJustificationType

#### Attributes

- claim: assertion put forward publicly for general acceptance with the implication that there are underlying 'reasons' that could show it to be 'well founded' and therefore entitled to be generally accepted. Example: "Our organization should cut costs next quarter"
- ground: the term 'ground' refers to the specific facts relied on to support a given claim. Example: "Our organization has lost money the last 3 quarters"

## D Meta-models details

- warrant: assertion that entitles you to interpret or link the grounds (facts) as support of the claim. Example: “When losing money, organizations should cut expenses as much as possible”
- backing: the ‘backing’ consists of a very general set of background assumptions which, in effect, legitimize the basis for believing in the warrant. That is, if the warrant is not accepted on its surface, then the backing is called into play to add deeper support to the argument. Example: “A business can’t continue to lose money and stay in business”
- rebuttal: the ‘rebuttal’ presents the possible exceptions or objections as to why the claim, the grounds, the warrants, or the backing may not hold for the situation under discussion. Example: “That may be true in general, but not with the customers of our company. Besides that, times have changed; the economy has changed; the dollar has fallen in value.
- qualifier: word that indicates how strongly the claim is being asserted, or how likely that something might occur. Example: “probably”, “certainly”, “very likely”, etc.

### ConcreteEvent

Specifies the signal that triggers the invocation of the rule.

#### Attributes

- eventType (ConcreteEventType): type of event, the different possibilities are described in the ConcreteEventType description.

### ConcreteEventType

#### Attributes

- onGIUClick: a ConcreteGraphicalIU has been clicked
- onGIUActivation: a ConcreteGraphicalIU has been activated
- onGIUDeactivation: a ConcreteGraphicalIU has been deactivated

## D Meta-models details

- onGIUResize: a ConcreteGraphicalIU has been resized
- onGIUInput: data has been entered in a ConcreteGraphicalIU
- onGIUDisplay: a ConcreteGraphicalIU has been displayed
- onDataSelection: a data entry element has been selected
- onDataBlur: a data entry element has lost focus
- onDataChange: a data entry element has lost focus after its content has been modified
- onDataFocus: a data entry element has got focus
- onGIUHover: a mouse pointer has been moved over an element

### ConcreteCondition

Logical test that, if satisfied or evaluates to true, causes the action to be carried out.

#### Attributes

- conditionSpecification (String): logical formula representing the condition to evaluate, for the moment under the form of a String.

### ConcreteAction

Consists of updates or invocations on the Domain Model data, or modifications of the concrete entities themselves.

#### Attributes

- actionType (ConcreteActionType): type of action, the different possibilities are described in the ConcreteActionType description.
- actionSpecification (String): kind of argument that, used in conjunction with type, allows to specify the action to launch. Example: for an ConcreteActionType 'modelUpdate', the actionSpecification could specify which field to change and what is the new value.



## ConcreteActionType

### Attributes

- modelSearch: search for model elements based on logical formula
- modelCreate: create a new model in the Domain Model
- modelRead: read a specified field in the Domain Model
- modelUpdate: update a field in the Domain Model
- modelDelete: delete a field in the Domain Model
- modelInvoke: perform a query external to the current Domain Model
- modelReset: reset the Domain Model with the initial parameters
- modelCopy: copy the Domain Model
- listenerCreate: create a new listener on a specified ConcreteGraphicalIU
- listenerDelete: delete a listener of a specified ConcreteGraphicalIU
- eventDispatch: dispatch the event to another ConcreteGraphicalIU
- GIUOpen: open a specified ConcreteGraphicalIU
- GIUClose: close a specified ConcreteGraphicalIU
- GIUActivate: activate a specified ConcreteGraphicalIU
- GIUDeactivate: deactivate a specified ConcreteGraphicalIU
- GIUEmphasize: focus in a specified ConcreteGraphicalIU
- GIUDeemphasize: focus out a specified ConcreteGraphicalIU

## D Meta-models details

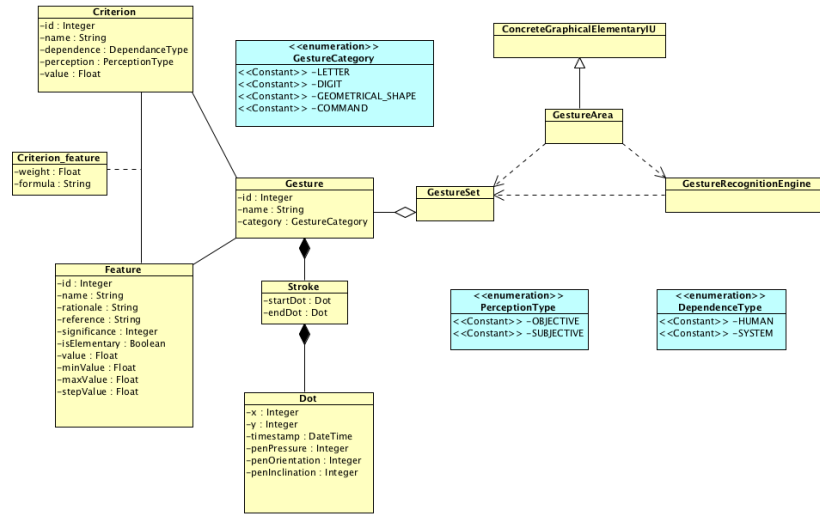


Figure D.6 Gesture system modeling.

## D.2 Gesture System Modeling

Figure D.6 shows the modeling describing the gesture system. It is centered around the *Gesture* representing the main entity. The *Gesture* is composed of *Stroke* entities, themselves composed of *Dot*. The gestures are linked to the CUI through the *GestureSet* which represents a set of gestures. It is used by the *GestureRecognitionEngine* and the *GestureArea*, which represents a *ConcreteGraphicalElementaryIU*. The gesture recognition engine is a high level entity corresponding to the recognition mechanism (mainly the algorithms). As it can be described in a lot of different ways, it is voluntarily let unconstrained.

Gesture

Set of *Stroke* representing a pen-based gesture.

### Attributes

- id (Integer): Identification number of the *Gesture*.
- name (String): Name of the Gesture (e.g. "a", "1" or "left\_stroke").
- category (GestureCategory): Type of the gesture, the different possibilities are described in the GestureCategory description.

## D Meta-models details

### GestureCategory

Describes four possible categories for the gestures (not exhaustive). See Appendix B for graphical description.

#### Attributes

- LETTER: the 26 alphabet letters
- DIGIT: the 10 digits
- GEOMETRICAL\_SHAPE: geometrical shapes such as triangle or square
- COMMAND: action gestures like left stroke or right stroke

### Stroke

Set of *Dot* recorded between two pen lifts.

- startDot (Dot): first *Dot* of the stroke.
- endDot (Dot): last *Dot* of the stroke.

### Dot

- x (Integer): x coordinate on the tablet.
- y (Integer): y coordinate on the tablet.
- timestamp (Datetime): creation date of the dot.
- penPressure (Integer): pressure of the pen on dot creation (between 0 and 1).
- penOrientation (Integer): angle of the pen in the tablet plan (between 0 and 360 degrees).
- penInclination (Integer): angle between the pen and the table plan (between 0 and 90 degrees).

# Appendix E Woke experiment models

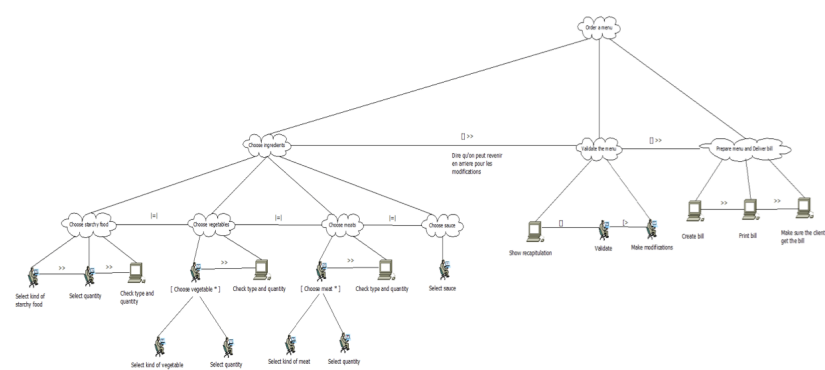


Figure E.1 Woke experiment task model.

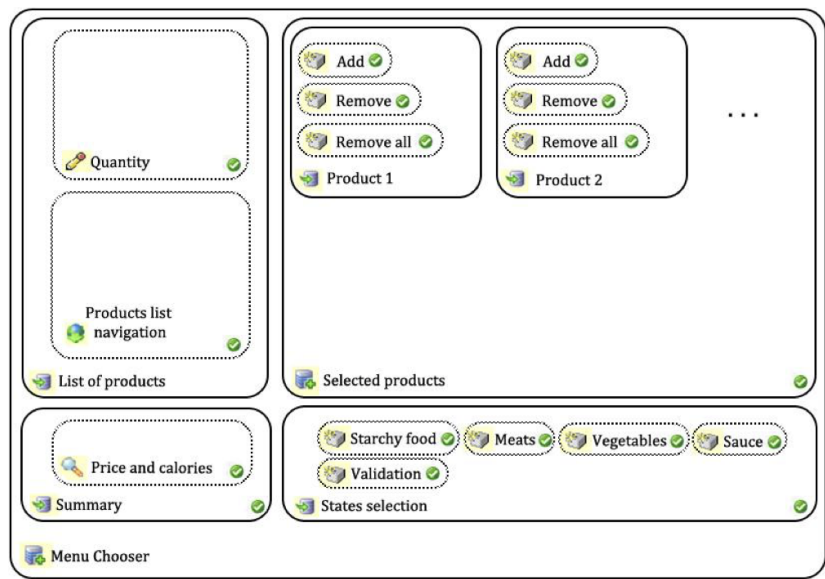


Figure E.2 Woke experiment AUI model.